

LNCS

LNCS



Springer

Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

263

A. M. Odlyzko (Ed.)

Advances in Cryptology – CRYPTO '86

Proceedings



Springer-Verlag

Berlin Heidelberg New York London Paris Tokyo

Editorial Board

D. Barstow W. Brauer P. Brinch Hansen D. Gries D. Luckham
C. Moler A. Pnueli G. Seegmüller J. Stoer N. Wirth

Editor

Andrew M. Odlyzko
AT&T Bell Laboratories
Murray Hill, NJ 07974-2070, USA

CR Subject Classification (1987): E.3

ISBN 3-540-18047-8 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-18047-8 Springer-Verlag New York Berlin Heidelberg

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in other ways, and storage in data banks. Duplication of this publication or parts thereof is only permitted under the provisions of the German Copyright Law of September 9, 1965, in its version of June 24, 1985, and a copyright fee must always be paid. Violations fall under the prosecution act of the German Copyright Law.

© Springer-Verlag Berlin Heidelberg 1987
Printed in Germany

Printing and binding: Druckhaus Beltz, Hemsbach/Bergstr.
2145/3140-543210

Preface

This book is the proceedings of CRYPTO 86, one in a series of annual conferences devoted to cryptologic research. They have all been held at the University of California at Santa Barbara. The first conference in this series, CRYPTO 81, organized by A. Gersho, did not have a formal proceedings. The proceedings of the following four conferences in this series have been published as:

Advances in Cryptology: Proceedings of Crypto 82, D. Chaum, R. L. Rivest, and A. T. Sherman, eds., Plenum, 1983.

Advances in Cryptology: Proceedings of Crypto 83, D. Chaum, ed., Plenum, 1984.

Advances in Cryptology: Proceedings of CRYPTO 84, G. R. Blakley and D. Chaum, eds., Lecture Notes in Computer Science #196, Springer, 1985.

Advances in Cryptology - CRYPTO '85 Proceedings, H. C. Williams, ed., Lecture Notes in Computer Science #218, Springer, 1986.

A parallel series of conferences is held annually in Europe. The first of these had its proceedings published as

Cryptography: Proceedings, Burg Feuerstein 1982, T. Beth, ed., Lecture Notes in Computer Science #149, Springer, 1983.

Eurocrypt 83, held in March of 1983 in Udine, Italy, and Eurocrypt 86, held in May of 1986 in Linköping, Sweden, did not have formal proceedings, while the '84 and '85 conference proceedings have appeared as

Advances in Cryptology: Proceedings of EUROCRIPT 84, T. Beth, N. Cot, and I. Ingemarsson, eds., Lecture Notes in Computer Science #209, Springer, 1985.

Advances in Cryptology - EUROCRYPT '85, F. Pichler, ed., Lecture Notes in Computer Science #219, Springer, 1986.

Papers in this volume are presented in seven sections containing most of the papers presented in the regular program, and a final section based on some of the informal presentations at the "Rump Session" organized by W. Diffie. Several of the regular papers presented at the conference are not included in this volume. There was a special session on integer factorization, and the three papers in that section will be published in journals:

C. Pomerance, J. W. Smith, and R. Tuler, A pipeline architecture for factoring large integers with the quadratic sieve algorithm, SIAM J. Comp. (to appear).

T. R. Caron and R. D. Silverman, Parallel implementation of the quadratic sieve, J. Supercomputing (to appear).

M. C. Wunderlich and H. C. Williams, A parallel version of the continued fraction integer factoring algorithms, J. Supercomputing (to appear).

Also, the paper

J. G. Osborn and J. R. Everhart, A large community key distribution protocol,

was not revised in time for publication.

It is my pleasure to thank all those who make these proceedings possible: the authors, organizers, and all the attendees. Special thanks are due to M. Janssen, Y. Cohen, and the Springer staff for their help in the production of this volume.

Murray Hill, New Jersey

Andrew M. Odlyzko

CRYPTO 86

A Conference on the Theory and Applications of Cryptographic Techniques

held at the University of California, Santa Barbara,
through the cooperation of the
Computer Science Department

August 11-15, 1986

sponsored by:

The International Association for Cryptologic Research

in co-operation with

*The IEEE Computer Society Technical Committee
on Security and Privacy*

Organizers

General Chairman: D. Coppersmith (IBM)

Program Committee: T. A. Berson (Anagram Laboratories)
E. F. Brickell (Bell Communications Research)
S. Goldwasser (MIT)
A. M. Odlyzko (AT&T Bell Laboratories, Chairman)
C. P. Schnorr (U. Frankfurt)

Local Arrangements: O. Egelcioglu (UCSB)

TABLE OF CONTENTS

SECTION 1: DATA ENCRYPTION STANDARD

Structure in the S-boxes of the DES (Extended abstract)	3
<i>E. F. Brickell, J. H. Moore, and M. R. Purtill</i>	
Cycle structure of the DES with weak and semi-weak keys	9
<i>J. H. Moore and G. J. Simmons</i>	

SECTION 2: PUBLIC-KEY CRYPTOGRAPHY

Private-key algebraic-coded cryptosystems	35
<i>T. R. N. Rao and K.-H. Nam</i>	
Some variations on RSA signatures and their security	49
<i>W. de Jonge and D. Chaum</i>	
Breaking the Cade cipher	60
<i>N. S. James, R. Lidl, and H. Niederreiter</i>	
A modification of a broken public-key cipher	64
<i>J. J. Cade</i>	
A pseudo-random bit generator based on elliptic logarithms	84
<i>B. S. Kaliski, Jr.</i>	
Two remarks concerning the Goldwasser-Micali-Rivest signature scheme	104
<i>O. Goldreich</i>	
Public-key systems based on the difficulty of tampering (Is there a difference between DES and RSA?) (Extended abstract)	111
<i>Y. Desmedt and J.-J. Quisquater</i>	

A secure and privacy-protecting protocol for transmitting personal information between organizations	118
<i>D. Chaum and J.-H. Evertse</i>	

SECTION 3: CRYPTOGRAPHIC PROTOCOLS AND ZERO-KNOWLEDGE PROOFS

How to prove all NP-statements in zero-knowledge, and a methodology of cryptographic protocol design (Extended abstract)	171
<i>O. Goldreich, S. Micali, and A. Wigderson</i>	
How to prove yourself: Practical solutions to identification and signature problems	186
<i>A. Fiat and A. Shamir</i>	
Demonstrating that a public predicate can be satisfied without revealing any information about how	195
<i>D. Chaum</i>	
Demonstrating possession of a discrete logarithm without revealing it	200
<i>D. Chaum, J.-H. Evertse, J. van de Graaf, and R. Peralta</i>	
Cryptographic capsules: A disjunctive primitive for interactive protocols	213
<i>J. Cohen Benaloh</i>	
Zero-knowledge simulation of Boolean circuits	223
<i>G. Brassard and C. Crépeau</i>	
All-or-nothing disclosure of secrets	234
<i>G. Brassard, C. Crépeau, and J.-M. Robert</i>	
A zero-knowledge poker protocol that achieves confidentiality of the players' strategy or How to achieve an electronic poker face	239
<i>C. Crépeau</i>	

SECTION 4: SECRET-SHARING METHODS

Secret sharing homomorphisms: keeping shares of a secret secret (Extended abstract)	251
<i>J. Cohen Benaloh</i>	
How to share a secret with cheaters	261
<i>M. Tompa and H. Woll</i>	
Smallest possible message expansion in threshold schemes	266
<i>G. R. Blakley and R. D. Dixon</i>	

SECTION 5: HARDWARE SYSTEMS

VLSI implementation of public-key encryption algorithms	277
<i>G. A. Orton, M. P. Roy, P. A. Scott, L. E. Peppard and S. E. Tavares</i>	
Architectures for exponentiation in GF (2^n)	302
<i>T. Beth, B. M. Cook, and D. Gollmann</i>	
Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor	311
<i>P. Barrett</i>	

SECTION 6: SOFTWARE SYSTEMS

A high speed manipulation detection code	327
<i>R. R. Jueneman</i>	
Electronic Funds Transfer Point of Sale in Australia	347
<i>R. Gyoery and J. Seberry</i>	

SECTION 7: SOFTWARE PROTECTION, PROBABILISTIC METHODS, AND OTHER TOPICS

The notion of security for probabilistic cryptosystems (Extended abstract)	381
<i>S. Micali, C. Rackoff, and B. Sloan</i>	
Large-scale randomization techniques	393
<i>N. R. Wagner, P. S. Putter, and M. R. Cain</i>	
On the linear span of binary sequences obtained from finite geometries	405
<i>A. H. Chan and R. A. Games</i>	
Some constructions and bounds for authentication codes	418
<i>D. R. Stinson</i>	
Towards a theory of software protection (Extended abstract)	426
<i>O. Goldreich</i>	

SECTION 8: INFORMAL CONTRIBUTIONS

Two observations on probabilistic primality testing	443
<i>P. Beauchemin, G. Brassard, C. Crépeau, and C. Goutier</i>	
Public-key registration	451
<i>S. M. Matyas</i>	
Is there an ultimate use of cryptography? (Extended abstract)	459
<i>Y. Desmedt</i>	
Smart card, a highly reliable and portable security device	464
<i>L. C. Guillou and M. Ugon</i>	

THOMAS - A complete single chip RSA device	480
<i>G. Rankine</i>	
AUTHOR INDEX	489

Structure in the S-Boxes of the DES
(extended abstract)

by

E. F. Brickell
Bell Communications Research
Morristown, NJ 07960

J. H. Moore*
Sandia National Laboratories
Albuquerque, NM 87185

M. R. Purtill**
Massachusetts Institute of Technology
Cambridge, MA 02139

ABSTRACT

The *S*-boxes used in the DES are the major cryptographic component of the system. Any structure which they possess can have far reaching implications for the security of the algorithm. Structure may exist as a result of design principles intended to strengthen security. Structure could also exist as a "trapdoor" for breaking the system. This paper examines some properties which the *S*-boxes satisfy and attempts to determine a reason for such structure to exist.

INTRODUCTION

The DES (Data Encryption Standard) was certified by the NBS in 1975 [NBS1]. A complete description of the DES can also be found in either [D] or [K]. The major nonlinear component of the DES is a function f that involves the *S*-boxes. f is a function that takes as input 32 bits of partially enciphered message and 48 bits of key and produces 32 bits of partially enciphered message as output. f uses eight *S*-boxes. Each *S*-box is a function from 6 bits into 4 bits. To be more precise, let $a = (a_1 \dots a_{32})$ be 32 bits of partially enciphered message and let $k = (k_1 \dots k_{48})$ be 48 bits of key. Then to form $f(a, k)$, a is expanded to a 48 bit b by duplicating the bits that have an index that is 0 or 1 mod 4 in the following manner. Let

$$b = (b_1 \dots b_{48}) = (a_{32} a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9 a_8 a_9 \dots a_{28} a_{29} a_{28} a_{29} a_{30} a_{31} a_{32} a_1).$$

Let $c_i = b_i + k_i$ for $1 < i < 48$. (+ will refer to addition mod 2 throughout this paper.) Let $c = (c_1 \dots c_{48})$. $c_{8(i-1)+1} \dots c_{8(i-1)+8}$ will be the 8 input bits into the i 'th *S*-box. Let $d_{4(i-1)+1} \dots d_{4(i-1)+4}$ be the output bits from the i 'th *S*-box. Let $d = (d_1 \dots d_{32})$. Then $f(a, k) = d$.

*This work performed at Sandia National Laboratories supported by the U.S. Department of Energy under contract number DE-AC04-76DP00789.

**This work performed while the author was visiting Bell Communications Research

The eight *S*-boxes used in the DES are listed in Table 1. Each *S*-box, S_i , is described by a matrix, M_i , with 4 rows and 16 columns. Then $S_i(c_1 \dots c_8) = M_i(c_1c_6, c_2c_3c_4c_5)$. It is clear from examining the *S*-boxes in Table 1 that each row of an *S*-box is a permutation of the integers 0 to 15. We will list this as property P0 of the *S*-boxes.

P0. Each row of an *S*-box is a permutation of the integers 0 to 15.

There have also been other properties found [K],[L],[S] that the *S*-boxes satisfy that would not likely be satisfied if the boxes were chosen randomly. We want to study these properties to see if some of them are related. The purpose of this paper is to attempt to find a minimal set of properties satisfied by the *S*-boxes. That is, a set of properties such that if we generate random boxes designed to satisfy these properties, then these boxes will satisfy all of the unusual properties of the *S*-boxes that we have been able to find.

S-BOX DESIGN PRINCIPLES

We would like to know what properties the *S*-boxes were designed to satisfy. This information has never been published and in fact, the only source for specific "design principles" appears to be responses from the NSA to a study of the DES made by the Lexar Corporation [L]. These were included in the report of the second workshop on the DES held by the NBS in 1976 [BGK]. In their comments, the NSA labelled the following as "design criteria" for the *S*-boxes.

P1. No *S*-box is a linear or affine function of the input.

P2. Changing 1 input bit to an *S*-box results in changing at least 2 output bits.

P3. $S(x)$ and $S(x + 001100)$ must differ in at least 2 bits.

The following were labelled by the NSA as "caused by design criteria."

P4. $S(x) \neq S(x + 11EF00)$ for any choice of e and f .

P5. The *S* boxes were chosen to minimize the difference between the number of 1's and 0's in any *S*-box output when any single input bit is held constant.

In this paper, an attempt is made to link any structure found in the *S*-boxes to these design principles. A basic tool used in this study was the generation of new boxes designed to meet a subset of these properties, but chosen randomly subject to the constraints imposed by the properties. These new boxes were compared with the DES *S*-boxes in order to identify further structure.

RELATION BETWEEN PROPERTIES

When we generated boxes satisfying P0, P2, and P3, we found that P5 also held. P5 can be stated in a more obvious manner. For an *S*-box S , let $p_i S$ be the projection of S onto the i 'th output bit. i.e. if $S(x) = (d_1d_2d_3d_4)$, then $p_i S(x) = d_i$. Since each row of the *S*-box is a permutation, the list of $p_i S(x)$ over all 6 bit inputs, x , will contain exactly 32 1's and 32 0's. Consider the same list with one of the input bits fixed. For example, consider the list over all 6 bit inputs $x = x_1 \dots x_6$ such that $x_i = 0$. If $i = 1$ or 6, then the list will contain exactly 16 1's

and 16 0's. If $i = 2, 3, 4$, or 5, the list will not necessarily contain the same number of 1's and 0's. P5 states that the S -boxes were chosen to minimize this difference between the number of 1's and 0's. In Table 2, we tabulated the number of 1's in each of the lists $p_j S_i(x)$ where x ranges over all 6 bit inputs satisfying $x_k = 0$ for $2 < k < 5$, $1 < i < 8$, $0 < j < 3$. If the S -boxes satisfied only the row permutation property, then we would expect the distribution of the number of 1's to look like the distribution in Table 3. Table 4 contains the distribution of 100 boxes generated to satisfy only P0, P2, and P3. The similarity with Table 2 is apparent.

Given an S -box S , we can define R_i to be the permutation defined by the i 'th row of S . Each row of an S -box is a Boolean vector function from 4 input bits to 4 output bits. Each such output bit can be viewed as a function of the 4 input bits and each input bit can be viewed as a function of the 4 output bits. These 256 functions belong to the set F of all Boolean functions from 4 bits to 1 bit having the property that exactly 8 inputs are mapped to 0 and 8 are mapped to 1. Define an equivalence relation on F by $f \sim g$ if there is a function α which is a permutation on 4 elements and also allows for complementation of those elements, so that $f \circ \alpha = g$ or $f \circ \alpha^{-1} = g$. The elements of F fall into 58 equivalence classes under this relation. However, the subset F_s of F used in the rows of the S -boxes, fall into only 22 of these classes.

In the boxes we generated satisfying P0, P2, P3, and P4, there were two classes of functions that frequently appeared that were not in F_s . One class contained the function $f(w, x, y, z) = w + x + y + z$. The other class contained the function $g(w, x, y, z) = w + x + y$. It is possible that these two classes were prohibited in the S -boxes because of property 1. We then generated new boxes which did not contain these two classes of functions. The distribution of classes of functions in F used in these new boxes was found to be similar to that of the DES S -boxes. Thus we define property P1'.

P1'. None of the four bit to one bit functions used in a row of an S -box is equivalent to the sum of four bits or to the sum of three bits.

We now return to the function f defined in the introduction. If a vector K of 48 bits is fixed, then $f|_K$ is a function from 32 bits to 32 bits. ($f|_K(a) = f(a, K)$.) However, $f|_K$ is not one-to-one or onto, so an investigation of the image of $f|_K$ was initiated. A set X of one thousand 32-bit vectors was randomly generated and for several different key vectors K , the integers $\{y : f|_K(y) = z\}$ were calculated for each $z \in X$. For the S -boxes, for all keys tested, about $1/2$ of the elements of X had exactly one pre-image and about $1/3$ of them were not in the image set. However, these two statistics were reversed in boxes that satisfied only P0, P1', P2, and P3. Since it seems desirable to make the image set as large as possible, it appears that this shift in the distribution may be due to some design principle.

This characteristic of the image set appears to be caused by property P4. Random boxes were generated which satisfied properties P0, P1', P2, P3, and also P4. The tabulation of $\{y : f|_K(y) = z\}$ was then repeated for these boxes. The distribution obtained from these new boxes was not significantly different from that of the DES S -boxes.

The relationship between property 4 and the image set becomes clearer with a deeper look at the implications of property 4. To be precise, some notation is required. For a 32-bit vector X , let $S_i(X)$ denote the 6-bits of X used in the input to the i -th S -box. Also, S_i and S_j will be called *consecutive* if $|i - j| = 1$ or if $\{i, j\} = \{1, 8\}$. Using property 4 and the expansion operator, the following result can be proven.

Theorem 1

If X and Y are 32-bit vectors for which $f|_K(X) = f|_K(Y)$, for some key K , then $S_i(X) \neq S_i(Y)$ for at least 3 consecutive S_i 's and the Hamming distance between X and Y is at least 4.

Without property 4, the smallest distance between such an X and Y would be 2 and would involve differences in only 2 consecutive S_i 's. Thus property 4 tends to make the possibility that two different inputs would be mapped onto the same output less likely. This both links the property to the shift in the distribution and appears to be a desirable cryptographic result.

At Crypto'85, Shamir [S] presented a paper in which he pointed out some unusual patterns in the S -boxes and questioned why such patterns exist. In particular, if the 6-bit input to an S -box is labelled as $ABCDEF$, then these patterns demonstrated an extremely high correlation between even hamming weight outputs and either $B = 0$ or $B = 1$. The boxes that we generated that satisfied P0 - P4 also tended to exhibit the same patterns, although the correlations were not as strong. This does indicate that the probabilities given in Shamir's talk do not fairly evaluate the chances of these patterns occurring, since the acknowledged design criteria seem to make them likely to exist.

CONCLUSION

All of the structure of the S -boxes that we have described appears to be the result of design principles. The question that remains is whether this is a complete list of the design principles used in creating the S -boxes. This question could be answered in the negative if further structure was discovered in the S -boxes that did not occur in the boxes created using these design principles.

References

[BGK].

Dennis K. Branstead, Jason Gait, and Stuart Katzke, "Report of the Workshop on Cryptography in Support of Computer Security," National Bureau of Standards, September 21-22, 1976, NBSIR 77-1291, September 1977.

[D]. Dorothy E.R. Denning, "Cryptography and Data Security," Addison-Wesley, Menlo Park, California, 1983.

[K]. Alan G. Konheim, "Cryptography, A Primer," John Wiley, New York, 1981.

[L]. Lexar Corporation, "An Evaluation of the NBS Data Encryption Standard," unpublished report, Lexar Corporation, 11611 San Vicente Blvd., Los Angeles, 1976.

[NBS1].

National Bureau of Standards, "Encryption Algorithm for Computer Data Protection," Federal Register, 40, March 17, 1975, pp. 12134-12139.

[S]. Adi Shamir, "On the Security of DES," Advances in Cryptology, Proceedings of Crypto 85, pp.280-281.

<u>column</u>	0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111
<u>row</u>	S-box 1
00	1110 0100 1101 0001 0010 1111 1011 1000 0011 1010 0110 1100 0101 1001 0000 0111
01	0000 1111 0111 0100 1110 0010 1101 0001 1010 0110 1100 1011 1001 0101 0011 1000
10	0100 0001 1110 1000 1101 0010 1011 1111 1100 1001 0111 0011 1010 0101 0000 1000
11	1111 1100 1000 0010 0100 1001 0001 0111 0101 1011 0011 1110 1010 0000 0110 1101
	S-box 2
00	1111 0001 1000 1110 0110 1011 0011 0100 1001 0111 0010 1101 1100 0000 0101 1010
01	0011 1101 0100 0111 1111 0010 1000 1110 1100 0000 0001 1010 0110 1001 1011 0101
10	0000 1110 0111 1011 1010 0100 1101 0001 0101 1000 1100 0110 1001 0011 0010 1111
11	1101 1000 1010 0001 0011 1111 0100 0010 1011 0110 0111 1100 0000 0101 1110 1001
	S-box 3
00	1010 0000 1001 1110 0110 0011 1111 0101 0001 1101 1100 0111 1011 0100 0010 1000
01	1101 0111 0000 1001 0011 0100 0110 1010 1010 1000 0101 1110 1000 1011 1111 0001
10	1101 0110 0100 1001 1000 1111 0011 0000 1011 0001 0010 1100 0101 1010 1110 0111
11	0001 1010 1101 0000 0110 1001 1000 0111 0100 1111 1110 0011 1011 0101 0010 1100
	S-box 4
00	0111 1101 1110 0011 0000 0110 1001 1010 0001 0010 1000 0101 1011 1100 0100 1111
01	1101 1000 1011 0101 0110 1111 0000 0011 0100 0111 0010 1100 0001 1010 1110 1001
10	1010 0110 1001 0000 1100 1011 0111 1101 1111 0001 0011 1110 0101 0010 1000 0100
11	0011 1111 0000 0110 1010 0001 1101 1000 1001 0100 0101 1011 1100 0111 0010 1110
	S-box 5
00	0010 1100 0100 0001 0111 1010 1011 0110 1000 0101 0011 1111 1101 0000 1110 1001
01	1110 1011 0010 1100 0100 0111 1101 0001 0101 0000 1111 1010 0011 1001 1000 0110
10	0100 0010 0001 1011 1010 1101 0111 1000 1111 1001 1100 0101 0110 0011 0000 1110
11	1011 1000 1100 0111 0001 1100 0010 1101 0110 1111 0000 1001 1010 0100 0101 0011
	S-box 6
00	1100 0001 1010 1111 1001 0010 0110 1000 0000 1101 0011 0100 1110 0111 0101 1011
01	1010 1111 0100 0010 0111 1100 1001 0101 0110 0001 1101 1110 0000 1011 0011 1000
10	1001 1110 1111 0101 0010 1000 1100 0011 0111 0000 0100 1010 0001 1101 1011 0110
11	0100 0011 0010 1100 1001 0101 1111 1010 1011 1110 0001 0111 0110 0000 1000 1101
	S-box 7
00	0100 1011 0010 1110 1111 0000 1000 1101 0011 1100 1001 0111 0101 1010 0110 0001
01	1101 0000 1011 0111 0100 1001 0001 1010 1110 0011 0101 1100 0010 1111 1000 0110
10	0001 0100 1011 1101 1100 0011 0111 1110 1010 1111 0110 1000 0000 0101 1001 0011 0010
11	0110 1011 1101 1000 0001 0100 1010 0111 0001 0000 1111 1110 0010 0010 0011 1100
	S-box 8
00	1101 0010 1000 0100 0110 1111 1011 0001 1010 1001 0011 1110 0101 0000 1100 0111
01	0001 1111 1101 1000 1010 0011 0111 0100 1100 0101 0110 1011 0000 1110 1001 0010
10	0111 1011 0100 0001 1001 1100 1110 0010 0000 0110 1010 1101 1111 0011 0101 0001
11	0010 0001 1110 0111 0100 1010 1000 1101 1111 1100 1001 0000 0011 0101 0010 1011

Table 1: DES S-boxes

8			
9		.1	
10		.3	
11		1.0	
12		3.0	.3
13	.8	6.8	1.1
14	6.3	12.1	5.7
15	25.8	17.1	20.5
16	35.2	19.2	40.9
17	25.0	17.1	22.5
18	6.3	12.1	7.2
19	.8	6.8	1.6
20		3.0	.2
21		1.0	
22		.3	
23		.1	
24			

Table 2a: S boxes Table 2b: Expected value Table 2c:
 boxes satisfying
 P_0, P_2, P_3

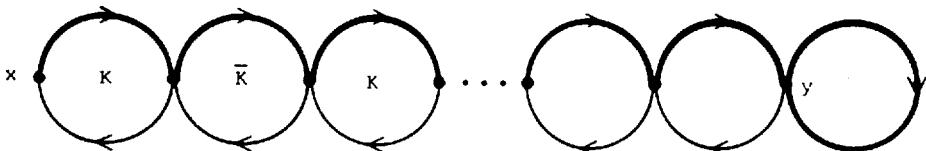
Table 2: Property P5

distribution of number of 1's in output with 1 input bit fixed

Cycle Structure of the DES with Weak and Semi-Weak Keys*

Judy H. Moore and Gustavus J. Simmons
Sandia National Laboratories
Albuquerque, NM 87185

As part of a report on cycling experiments with DES, Rivest [1] announced at Crypto'85 that a small cycle had been found when alternately encrypting with the all zeroes key and the all ones key. This cycle contained approximately 2^{33} points. Later in the same meeting, Coppersmith [2] explained this phenomenon by noting that if a fixed point occurred in the cycle, since with these keys encryption is the same as decryption, the successive encryptions would actually be decryptions and would retrace the steps to the starting point. We can picture this as follows:



where x is the starting point, y is the fixed point and K and \bar{K} represent the keys used. He also argued that since there are 2^{32} such fixed points for each of these keys, the apparently small size of the cycle reported was not actually surprising. Intrigued by these observations, we began an in-depth study of the cycle structure of DES using weak and semi-weak keys. The results presented in this paper outline the current status of that study.

* This work performed at Sandia National Laboratories supported by the U. S. Dept. of Energy under contract no. DE-AC04-76DP00789.

Notation

A complete description of the DES algorithm will not be given here, but since we will use a nonstandard notation, introduced by Grossman and Tuckerman [3], we begin with the specifics of that notation. Omitting the initial and final permutations, the DES transformation can be viewed as a sequence of 32-bit vectors

$$m_0, m_1, m_2, \dots, m_{16}, m_{17}$$

defined recursively by

$$m_{i+1} = m_{i-1} \oplus f(K_i, m_i)$$

where K_i is the i^{th} round key and f is the nonlinear DES function described in the original FIPS Publication 45 [4]. The concatenation, $m_0 m_1$, represents the 64-bit input after the initial permutation, while $m_{17} m_{16}$ represents the output before the inverse of that permutation. This notation is much better suited to our purposes than the original description of the DES. For all of the work reported in this paper, the initial and final permutations are irrelevant, so we will routinely omit them.

Some details of the nonlinear function f are required for our discussion. The function takes as input a 32-bit vector X and expands it to a 48-bit vector $E(X)$. The 48-bit round key K_i is then exclusive-ored with $E(X)$. The resulting vector is used as input to the S-boxes, yielding a 32-bit vector. The output of f is a permuted form of this 32-bit vector. This process is shown in the following figure.

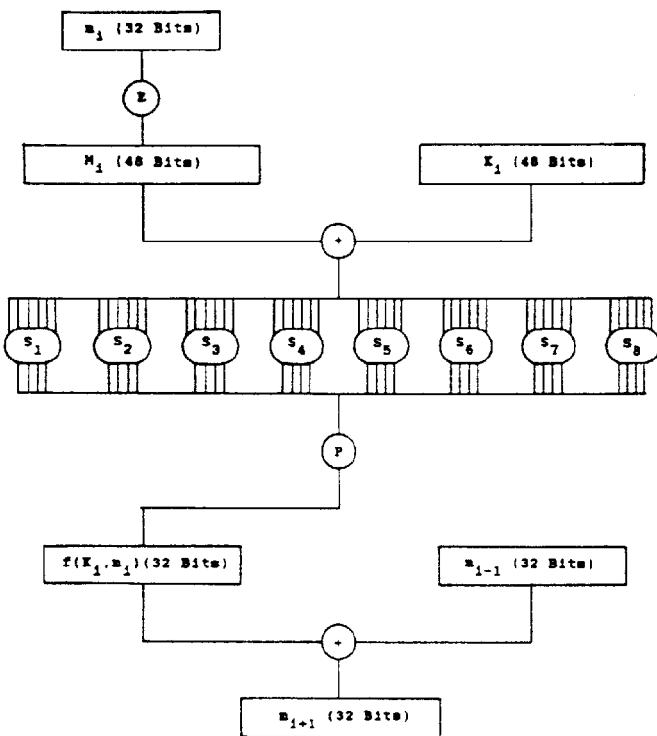


Figure 1.

The descriptions of E,P and the S-boxes can be found in FIPS Publication 46 [4]. The complete DES encryption of a 64-bit vector Y with a key K will be denoted in this paper by $E(K, Y)$, while decryption with K will be denoted by $D(K, Y)$.

The Keys

We begin with a review of what is known about the weak and semi-weak keys. Davies [5] and Jueneman [6] have studied the structure of these keys and some of the results have also appeared

in FIPS Publication 74 [7]. The approach in this paper basically follows that of Moore and Simmons [8].

The keys used in this study fall into two classes. The first class, the weak keys, consists of four keys distinguished by the fact that all 16 round keys are the same. This means that decryption is identical to encryption since reversing the calling sequence for the round keys has no effect.

The second class consists of the semi-weak keys. A key K is a semi-weak key if there exists another key K^* so that the round keys for these keys satisfy $K_i^* = K_{17-i}$ for $0 < i < 17$. This property has the effect of providing "inverse keys" in the sense that decryption with K is the same as encryption with K^* .

The following theorem verifies that the weak and semi-weak keys are the only such keys having this kind of inverse keys.

Theorem 1

A DES key K has an inverse key K^* satisfying

$$K_i^* = K_{17-i}$$

if and only if K is one of the 16 keys in which all 14 of the bits in each of the four subsets A, B, C, and D of K listed below are alike.

A	(1	2	3	17	18	19	33	34	35	36	49	50	51	52)
B	(4	5	6	7	20	21	22	23	37	38	39	53	54	55)
C	(9	10	11	25	26	27	41	42	43	44	57	58	59	60)
D	(12	13	14	15	28	29	30	31	45	46	47	61	62	63)

Proof:

The proof is a tedious but straightforward bit tracing of the round keys. ■

The sets A, B, C, and D in the previous theorem also give rise to a labeling technique for the keys used in this study. This label consists of a four-bit number, the most significant bit of which identifies the value for the bits in set A. The next bits identify the values for sets B, C and D. For example, K(3) is the key in which the bits in sets A and B are zero and the bits in sets C and D are one since the binary representation of 3 is 0011. That is,

$$\begin{aligned} K(3) = & \text{ 0000000(1) } 1111111(0) \text{ 0000000(1) } 1111111(0) \\ & \text{ 0000000(1) } 1111111(0) \text{ 0000000(1) } 1111111(0) \end{aligned}$$

The bits in parentheses are the parity bits and are set by the rule that each byte must have odd parity. The 16 keys mentioned in the previous theorem are listed below using this notation with their corresponding inverse keys identified.

K	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
K^*	0	4	8	12	1	5	9	13	2	6	10	14	3	7	11	15

The following easy lemma is a consequence of the fact that the expansion operator E is a homomorphism.

Lemma 2

For any 32-bit vectors U and M, and any 48-bit vector K,

$$f(K, M) = f(K \oplus E(U), M \oplus U).$$

Proof:

$$\begin{aligned} f(K \oplus E(U), M \oplus U) &= K \oplus E(U) \oplus E(M \oplus U) \\ &= K \oplus E(U) \oplus E(M) \oplus E(U) \\ &= K \oplus E(M) \\ &= f(K, M) . \blacksquare \end{aligned}$$

The argument used to count the number of fixed points of a weak key can be captured in a more general statement in the next theorem.

Theorem 3

Suppose that for some key K , the round keys satisfy

$$K_i \oplus K_{17-i} = E(U) ,$$

where $E(U)$ is the 48-bit expansion of some 32-bit vector U . Then the following are equivalent:

$$1) m_8 \oplus m_9 = U,$$

$$2) m_i \oplus m_{17-i} = U, \text{ for } 0 < i < 17 ,$$

$$3) m_0 \oplus m_{17} = U \text{ and } m_1 \oplus m_{16} = U .$$

Proof:

First note that 2) \Rightarrow 3) and 2) \Rightarrow 1) are obvious. From the definition of m_j and m_{17-j+2} , we have

$$\begin{aligned} m_j \oplus m_{17-j} &= m_{j-2} \oplus f(K_{j-1}, m_{j-1}) + m_{17-j+2} \oplus f(K_{17-j+1}, m_{17-j+1}) \\ &= m_{j-2} \oplus m_{17-j+2} \oplus f(K_{j-1}, m_{j-1}) \oplus f(K_{j-1} \oplus E(U), m_{17-j+1}). \quad (*) \end{aligned}$$

For 3) \Rightarrow 2) the above equation yields

$$\begin{aligned}
 m_2 \oplus m_{15} &= m_0 \oplus m_{17} \oplus f(K_1, m_1) \oplus f(K_1 \oplus E(U), m_1 \oplus U) \\
 &= m_0 \oplus m_{17} \\
 &= U .
 \end{aligned}$$

Then if we assume that for all $j < i$, $m_j \oplus m_{17-j} = U$, then (*) above can be used to show that $m_i \oplus m_{17-i} = U$. Thus by induction 2) is established.

For 1) \Rightarrow 2) we have

$$m_{10} \oplus m_7 = m_9 \oplus m_8 \oplus f(K_9, m_9) \oplus f(K_9 \oplus E(U), m_9 \oplus U) = U .$$

The induction argument used above applies again to complete the proof. ■

For the all zeroes or the all ones key, the hypothesis of Theorem 3 is satisfied for U equal to the all zeroes vector. Hence, fixed points for these keys coincide with those messages in which $m_8 = m_9$ during the encryption process. Since there are 2^{32} such possible equations, there are precisely 2^{32} fixed points for each key.

This theorem appears to be quite powerful, so the next issue is the identification of those keys which satisfy the hypothesis of the theorem.

Theorem 4

If for some key K , the round keys satisfy

$$K_1 \oplus K_{17-i} = E(U) ,$$

where U is the 48 bit expansion of some 32 bit vector U , then U is either the all zeroes or the all ones vector.

Proof:

The proof is again rather tedious and we refer the reader to [8] for its details. ■

This theorem states that the only keys satisfying the hypothesis of Theorem 3 are those in which the round keys either form a palindromic sequence:

$$K_i = K_{17-i} \quad . \text{ or}$$

an antipalindromic sequence

$$K_i = \bar{K}_{17-i} \quad .$$

The following theorem connects these conditions with the weak and semi-weak keys.

Theorem 5

A DES key K has a palindromic round key sequence or an antipalindromic round key sequence if and only if K is one of $K(0)$, $K(5)$, $K(10)$ or $K(15)$ in the first case or one of $K(3)$, $K(6)$, $K(9)$ or $K(12)$ in the second case.

Proof: See [8].

To end this section, we give a theorem which will be useful in studying the cycles structure of weak and semi-weak keys. One definition is required first. A point x is an antifixed point of a key K if $E(K,x) = \bar{x}$. ■

Theorem 6

For each of the keys with a palindromic round key sequence there are precisely 2^{32} fixed points and for each of the keys with an antipalindromic round key sequence there are precisely 2^{32} antifixed points.

Proof:

The fixed point argument was given earlier. For the anti-fixed point argument, the keys with an antipalindromic round key sequence satisfy Theorem 3 with U being the all ones vector. Hence the antifixed points for these keys will coincide with those messages in which $m_g = \bar{m}_g$ during the encryption process. Since there are 2^{32} such possible equations, there are 2^{32} possible antifixed points for each such key. ■

The DES Engines

Two special-purpose hardware devices were designed and built at Sandia as part of this study. These devices will be referred to in this paper as the DES Engine and the Micro DES Engine.

The DES Engine was designed to perform several types of cycle testing. It consists of 16 identical PC boards, each running a DES chip, the AM 9568, at high speed without changing keys. An IBM PC is used to provide communication with the user and to count the number of encryptions performed.

There are two basic modes of operation for this machine. In the first mode, each board performs a cycle test experiment using its preset key, independently of all other boards. In the second mode, the boards are paired so that cycle testing using alternating keys may be performed. The output of one board in the pair is used as the input to the other board. By this means, the pair can

perform an experiment with alternating keys, while each DES chip keeps its preset key unchanged.

Without describing the hardware in detail, the rudiments of its operation will be discussed. As part of the initialization of an experiment, several variables are set. These include a key, two starting points, SA and SB, and two other values, HA and HB, called "hit values". During the first step of the counter, SA is encrypted with the set key and compared to HA. If there is a match, the machine stops to report this result. It also stops if the encrypted value of SA is the same as SA, i.e., a fixed point has been found, or if a specified number of steps have been taken. The encrypted value of SA then is stored in the place of SA. During the next step of the counter, SB is encrypted with the preset key and compared to HB. The stop conditions described above are checked and, if not met, the encrypted value of SB replaces the original SB. This process continues until a condition for a machine halt is met or the operator intervenes. The machine will complete about 2^{32} encryptions per cycle per day.

The Micro DES Engine is a very specialized piece of hardware which was designed to take advantage of the internal structure of DES to find specific examples. In order to explain its operation, we need some notation. Suppose we are given two keys, K1 and K2, and two 32 bit vectors, m_0 and m_1 . Let M be the concatenation of m_0 and m_1 . To compute $E(E(M, K1), K2)$ we would calculate

$$m_{i+1} = m_{i-1} \oplus f(m_i, K1_i) \quad \text{for } 0 < i < 17 .$$

Then letting $n_0 = m_{17}$ and $n_1 = m_{16}$, we would calculate

$$n_{i+1} = n_{i-1} \oplus f(n_i, K2_i) \quad \text{for } 0 < i < 17 .$$

The resulting concatenation of n_{17} and n_{16} would be the result. The Micro DES Engine allows us to specify two keys, two integers, i and j, and two 32 bit vectors U and V. It then allows m_i to

assume all 2^{32} possible values. For each such value, m_{j+1} is set to $m_j \oplus U$. Proceeding through the rounds of DES using the one of the keys, m_{1+2}, \dots, m_{17} are calculated. Now setting $n_0 = m_{17}$ and $n_1 = m_{16}$, the engine calculates the rounds of DES using the second key until it has found n_{j+1} . If $n_{j+1} = n_j \oplus V$, the result is reported before changing the value of m_j . In other words, the Micro DES Engine starts at some specified round of encryption with the first key and some linear relationship between adjacent terms of the sequence $\{m_j\}$ and stops at another specified round of encryption with the second key to check for another linear relationship between adjacent terms of the sequence $\{n_j\}$. There is a technical restriction that the combined number of rounds of DES in one of these steps cannot exceed 16. The complete experiment, trying all 2^{32} choices for m_j , requires approximately 13 hours of operation.

The Weak Key Cycle Structure

Before proceeding with the details of the cycle structures for any of the keys, we need to make the observation that the complement of any cycle is also a cycle since

$$\overline{E(K, x)} = E(\bar{K}, \bar{x}) .$$

This complementary cycle will also be called the dual cycle.

We are now ready to consider the cycle structure for weak keys. Several important properties of the weak keys, which have already been discussed, will now come to bear on the cycle structure. These are:

1. There are 4 weak keys $K(0)$, $K(5)$, $K(10)$ and $K(15)$,
2. Each key is its own inverse,
3. Each key has 2^{32} fixed points.

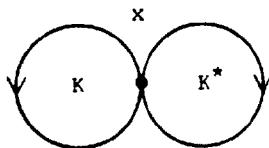
Since each key is its own inverse, a cycle of repeated encryptions with a weak key will either consist of one point, a fixed point for that key, or two points. These cycles, of course, are rather trivial.

However, alternately encrypting with a weak key and its complement has already produced some interesting results. We will call cycles of this type Coppersmith cycles. To be specific, a Coppersmith cycle is a cycle obtained by alternately encrypting with a weak key and its complement in which a fixed point is eventually encountered.

Since the complement of a cycle is a cycle, Coppersmith cycles could conceivably be self-dual or occur in isomorphic pairs. However, in [8], it was shown that only the latter case is possible. Hence, Coppersmith cycles can never contain both a point and its complement.

The Coppersmith cycles traced thus far range in size from 1 point to 12,605,533 points. Those cycles with one point are the "degenerate" Coppersmith cycles and were found with the use of the Micro DES Engine.

To find a one point Coppersmith cycle, we must find a point which is fixed by both a weak key K and its complementary key K^* . Pictorially this cycle will be



Hence, we initialize the Micro DES Engine with these keys, K and K^* , set i and j equal to 8, and let U and V be the all zero vectors. The engine will then produce a list of all possible values for m_8 , so that $m_8 = m_9$ in the encryption with K and $n_8 = n_9$ in the encryption with K^* . From this we can produce a list of all fixed points of K which are also fixed points of K^* . There is

exactly one degenerate pair of complementary cycles for the key pair K(0), K(15) and one for the key pair K(5), K(10). These are

$$x = 74080FA36E793E74 \text{ (Hex)}$$

and \bar{x} fixed by K(0) and K(15) and

$$y = 1BDAFF22E4BDAA52 \text{ (Hex)}$$

and y fixed by K(5) and K(10).

Excluding these degenerate cases, the remaining Coppersmith cycles traced thus far range in size from 12,605,533 points ($\approx 2^{23.6}$) to 26,717,619,870 points ($\approx 2^{34.6}$). We have traced 174 such cycles and find that these appear to end in fixed point cycles on the same key or on different keys with equal probability. We give one example in each case:

$$x = A1E1751167FED858 \text{ (Hex)} \quad \text{fixed by } K(15)$$

and

$$y = 07CDA64B52C48D2F \text{ (Hex)} \quad \text{fixed by } K(0)$$

with a cycle length of 12,605,633 points ($\approx 2^{23.6}$) and

$$x = 0A60B8BCFB7F4116 \text{ (Hex)} \quad \text{fixed by } K(15)$$

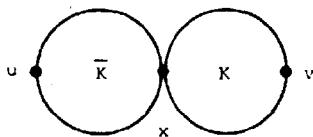
and

$$y = C4D9A9A9EDC0988C \text{ (Hex)} \quad \text{fixed by } K(15)$$

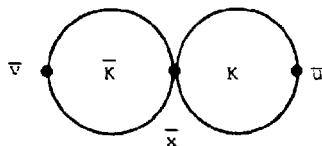
with a cycle length of 158,461,212 points ($\approx 2^{27.2}$).

The process of alternately encrypting with a weak and a semi-weak key may never encounter a fixed point. Cycles of this type

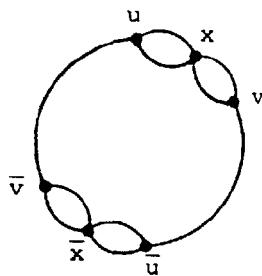
type will be called non-Coppersmith. These seem to naturally divide into two classes depending upon whether or not a point and its complement occur in the same cycle. The cycle containing a point x and the cycle containing \bar{x} , which may be disjoint, have the local structure



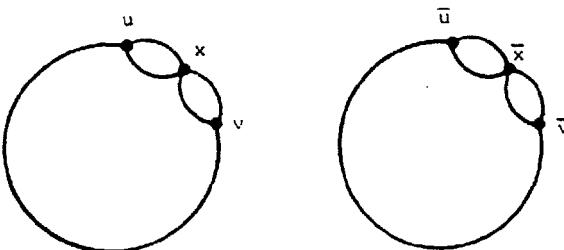
and



since $E(\bar{K}, \bar{x}) = E(\bar{K}, \bar{\bar{x}}) = \bar{v}$, etc. Analysis of the structure of such cycles leads to the discovery that non-Coppersmith cycles occur either as self-dual, centrally symmetric, cycles of the form



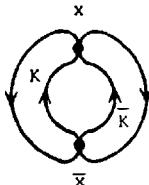
or as isomorphic pairs of the form.



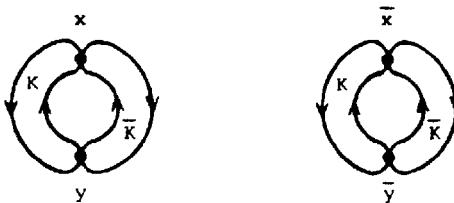
The central symmetry of both keys and points on the self-dual non-Coppersmith cycles means that the size of such a cycle must be congruent to 2 mod 4. The details of these theoretical results are in [8].

However, in an extensive computer search, no instance of either of these types of non-Coppersmith cycles has been found. Since there are exactly 2^{32} Coppersmith cycles and 2^{64} points in all, a reliable estimate of the size of Coppersmith cycles could be used to infer the likelihood of the existence of non-Coppersmith cycles. The best that can be said based on the 174 known cycles is that with a confidence of 99.9%, the fraction of the points in Coppersmith cycles is at least 96%. In other words, if 96% or fewer of the points are actually in Coppersmith cycles, 174 random selections would all be in Coppersmith cycles only one time in a thousand. This type of statistical argument can never prove the non-existence of non-Coppersmith cycles, but it can (as the number of unsuccessful tries increases) quantify the futility of continuing to search for them by a brute force random selection of starting points. If these exist, degenerate forms are also possible and would have the following structures:

Degenerate self-dual non-Coppersmith



Complementary pair of degenerate non-Coppersmith

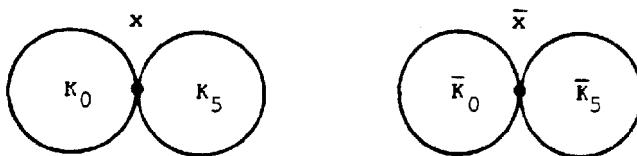


Unfortunately, we have no easy way to find these degenerate cases, if they exist, so producing one appears to be a 2^{64} search problem.

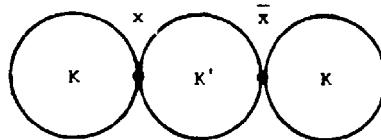
The final variety of cycle for the weak keys which we will consider consists of those obtained by alternate encryptions with any two weak keys. Obviously the cycles already discussed are special cases of these, in which the two keys are actually the same or one is the complement of the other. However, the remaining pairings give rise to some new cycle structures.

In this new setting, a cycle very much like the Coppersmith cycle is encountered in that it has a fixed point at either end of a chain of beads as in the Coppersmith cycles. However, there is no reason to believe that such a cycle would not contain both a point and its complement. Therefore, these new cycles have an extra possible class to consider. Of course, just as for the alternation of a weak key and its complement, a cycle alternating between any two weak keys might never encounter a fixed point. Thus structures corresponding to the non-Coppersmith cycles above appear to be possible. At this time, no results are available

except for those degenerate cases which could be found with the Micro DES Engine. No points were found which were fixed simultaneously by $K(0)$ and $K(10)$, however two points were found for $K(0)$ and $K(5)$. Hence, we obtain two degenerate pairs of complementary cycles, for this key pair, of the form:



If we consider the cycles in which fixed points are encountered and in which both a point and its complement are found, the degenerate case would be of the form:



This is not possible since we would have to find a point x for which $E(K,x) = x$ and $E(K,\bar{x}) = \bar{x}$. The last equation requires that $E(\bar{K},x) = x$ so that x would have to be a point fixed by K and its complement \bar{K} . The complete list of such points is available and for each such point x and each choice of a weak key K' , we have verified that $E(K',x) \neq \bar{x}$.

The cycles in which a fixed point does not occur seem to once again require the solution of a 2^{64} search problem to locate degenerate cases, so that no such cycles have been produced.

The Semi-weak Keys

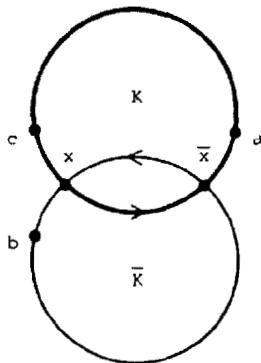
The semi-weak keys will be considered in two stages. There are four keys which have an antipalindromic sequence of round keys, as was discussed earlier. The remaining eight semi-weak keys have a different structure for their round keys. The discussion of these keys will be delayed until later in this section.

We begin by summarizing the properties, which were developed in the previous sections, of the keys with an antipalindromic sequence of round keys.

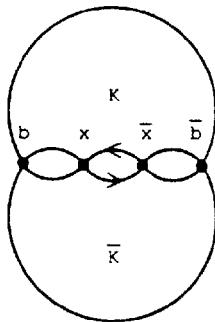
- 1) There are 4 of these keys $K(3)$, $K(6)$, $K(9)$ and $K(12)$.
- 2) The complement of one of these keys is its inverse key.
- 3) Each key has 2^{32} antifixed points.

Once again two cases seem to occur. A cycle which contains a point x may either contain its complementary point \bar{x} or not. We will consider the first case now.

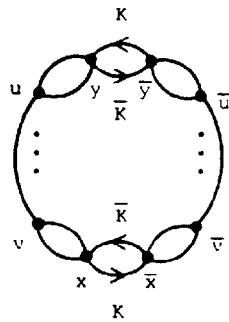
Suppose that x is an antifixed point of a key K . Thus x and \bar{x} are in the cycle for K , but because $E(K, x) = E(\bar{K}, \bar{x})$, these points are also in the cycle for \bar{K} . Schematically, we have



Now consider the points $a = E(K, \bar{x})$ and $b = E(\bar{K}, x)$. Notice that $\bar{b} = E(\bar{K}, x) = E(K, \bar{x}) = a$. Also, since \bar{K} is the inverse key for K , we have that $c = D(K, x) = E(\bar{K}, x) = b = \bar{a}$. Hence the structure shown in the last diagram can be replaced by



By repeating this argument, we see that the points in the cycle all occur as complementary points with one of the antifixed point pairs at each of the antipodal points as shown in the following diagram.



Of course, this means that these cycles are self-dual and have diametrical symmetry, i.e., every point, u , in the cycle is reflected in the diameter drawn through the centers of the antipodal antifixed point pairs into its complement, \bar{u} . Since each of these cycles must have precisely two antifixed point pairs

in it and there are precisely 2^{32} such points for each key, there are exactly 2^{31} such cycles for each of the pairings of these semi-weak keys.

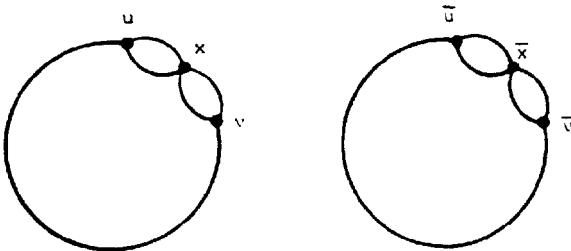
An example of a cycle of this type using $K = K(3)$, has as the antipodal antifixed points:

$$\begin{aligned}x &= 9EDB66CF776212B8 \text{ (Hex)} \\y &= 4B659E4C304032BF \text{ (Hex)}\end{aligned}$$

The cycle has a length of $6,236,877,706 \approx 2^{32.5}$.

We have not traced sufficiently many cycles of this type to permit a reliable estimate of the expected cycle size. It would appear to be in the vicinity of 2^{32} , which, since there are only 2^{31} such cycles in all, would suggest that only half of the total number of points are in these self-dual (under complementation) cycles.

The other cycles for these keys must occur in complementary pairs. The form of these pairs of cycles is pictured below.



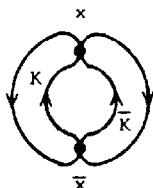
An example of a complementary pair of such cycles using $K = K(3)$ are those on x and \bar{x} where

$$x = 51F25587495909A5 \text{ (Hex)}$$

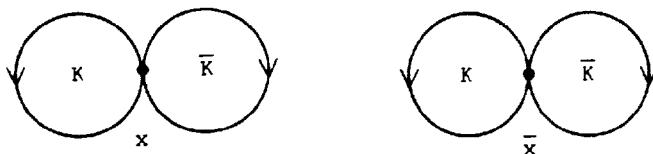
which has a cycle length of $6,671,292,514 \approx 2^{32.6}$.

Given that both types of cycles occur for the semi-weak keys, i.e., self-dual and isomorphic pairs of cycles, an intriguing

question is whether degenerate cycles exist or not. A degenerate self-dual cycle would be of the form



while a degenerate isomorphic pair would be of the form



The Micro DES Engine allows us to answer the first half of the question. By letting the two keys be a complementary pair of the keys with antipalindromic round key sequences, choosing $i = j = 8$, and letting U and V be the all ones vectors, the set of all points which are antifixed by both of the chosen keys can be found. After trying all possible key pairs, we found that there is exactly one degenerate cycle for the key pair $K(3)$, $K(12)$ and one for the pair $K(6)$, $K(9)$. These are

$$x = 2046CAC677DCA40F \text{ (Hex)}$$

for $K(6)$ and $K(9)$ and

$$x = 5A77FF65EC179215 \text{ (Hex)}$$

for $K(3)$ and $K(12)$. Unfortunately, Theorem 3 does not (so far as we can see) provide a means to reduce the 2^{64} search for degenerate isomorphic pairs. We therefore do not know how many, if any, degenerate cycles of this type exist for the semi-weak keys.

We will now turn our attention to the remaining semi-weak keys. Listed below are the facts known about these keys:

- 1) There are 8 such keys,
- 2) The inverse of any key in the set is also in the set,
- 3) The complement of any key in the set is also in the set.

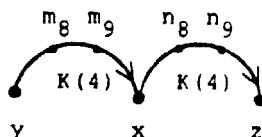
The round keys for a key K in this collection satisfy

$$K_i \oplus K_{17-i} = V$$

where V is either the vector consisting of 24 ones followed by 24 zeroes or the vector consisting of 24 zeroes followed by 24 ones. Of course, the round keys do not satisfy the hypothesis of Theorem 3, since $V \neq E(U)$ for any 32 bit vector U.

At this time, we do not know about the cycle structure for these keys, but some intriguing experiments have been completed on the Micro DES Engine. We offer a few of them here simply as tantalizing bits of information. For the description of these experiments, let U_1 be the vector of 16 zeroes followed by 16 ones; U_2 be the vector of 32 zeroes; and U_3 be the vector of 32 ones.

The first experiment used the key $K(4)$ in both key positions of the Micro DES Engine and the values of i and j were both set to 8. There were three stages to this experiment and in each stage U and V were set to be equal. When the value of U was set to U_1 the engine found 2 values for m_8 and when $U = U_3$, the engine found 1 value for n_8 . However, none were found when U was equal to U_2 . Pictorially we have



where the arrows from y to x and from x to z show encryption with key $K(4)$. The points marked along the arrow show the middle step in the rounds, that is, the position of m_8 , m_9 and n_8 , n_9 . The results of the experiment show that there exists a value for y in this diagram for which $m_8 \oplus m_9 = U = n_8 \oplus n_9$ when U is equal to U_1 or U_3 but not when $U = U_2$.

A similar experiment was performed with $K(4)$ and $K(11)$ as the keys in the Micro DES Engine. We found that a value for y existed for which $m_8 \oplus m_9 = U = n_8 \oplus n_9$ when U is equal to U_1 or U_2 but not when $U = U_3$.

Perhaps these strange results will point to new directions in this study of cycles of cycles for these semi-weak keys.

New Directions

The results reported here are part of a study which is not yet complete. We will continue to collect statistics on the cycles obtained by alternate encryptions using two weak keys and also on the cycles using semi-weak keys which have antipalindromic sequences of round keys. The remaining semi-weak keys seem to be an open area of discussion with many possible avenues to pursue.

References

1. B. S. Kaliski, Jr., R. L. Rivest, and A. T. Sherman, "Is DES a Pure Cipher? (Results of More Cycling Experiments on DES)," *Proceedings of Crypto'85*, Santa Barbara, CA, August 18-22, 1985, in *Advances in Cryptology*, Ed. by H. C. Williams, Springer-Verlag, Berlin (1986), pp. 212-222.
2. D. Coppersmith, "The Real Reason for Rivest's Phenomenon", *Proceedings of Crypto'85*, Santa Barbara, CA, August 18-22, 1985, in *Advances in Cryptology*, Ed. by H. C. Williams, Springer-Verlag, Berlin (1986), pp. 535-536.

3. E. K. Grossman and B. Tuckerman, "Analysis of a Weakened Feistel-like Cipher," IBM Research Report, RC 6375, January 31, 1977; also, Proceedings ICC'78.
4. "Data Encryption Standard," U. S. Dept. of Commerce, National Bureau of Standards, FIPS Pub. 46, January 15, 1977.
5. D. W. Davies, "Some Regular Properties of the 'Data Encryption Standard; Algorithm," Proceedings of Crypto'82, Santa Barbara, CA, August 23-25, 1982, in Advances in Cryptology, Ed. by D. Chaum, R. L. Rivest, and A. T. Sherman, Plenum Press, New York (1983) pp. 89-96.
6. R. R. Jueneman, Privately circulated letter to American cryptologists, March 1, 1983.
7. "Guidelines for Implementing and Using the NBS Data Encryption Standard," U. S. Dept. Of Commerce, National Bureau of Standards, FIPS Pub. 74, April 1, 1981.
8. J. H. Moore and G. J. Simmons, "Cycle Structure of the DES for Keys Having Palindromic (or Antipalindromic) Sequences of Round Keys," Proceedings of Eurocrypt'86, Linköping, Sweden, May 20-22, 1986.

**PRIVATE-KEY
ALGEBRAIC-CODED
CRYPTOSYSTEMS ***

T. R. N. Rao

The Center for Advanced Computer Studies
University of Southwestern Louisiana
Lafayette, Louisiana 70504

Kil-Hyun Nam **

National Defense College
Seoul, Korea

ABSTRACT

Public-key cryptosystems using very large distance algebraic codes have been studied previously. Private-key cryptosystems using simpler codes have also been subject of some study recently. This paper proposes a new approach to the private-key cryptosystems which allows use of very simple codes such as distance 3 and 4 Hamming codes.

This new approach gives not only very efficient encoding/decoding and very high information rates but also appears to be secure even under chosen-plaintext attacks.

Keywords : cryptosystems, public-key cryptosystems, private-key cryptosystems, Algebraic codes, crypto-complexity, chosen-plaintext attack, Joint Encryption and Error-control Coding

* This research is supported by a grant from National Security Agency grant # MDA 904-84-H-005.

** This author's research was performed while he was with the Center for Advanced Computer Studies at University of Southwestern Louisiana.

1. INTRODUCTION

McEliece introduced a public-key cryptosystem based on algebraic coding theory using t -error correcting Goppa codes [McEliece '78]. But McEliece Public-key Cryptosystem (MPBC) requires large block lengths with capabilities to correct large number of errors ($n \approx 1000$ bits, $t \approx 50$ bits) to be effective. This involves very large computational (encryption and decryption) overhead to be practical in computer communications.

Private-key Algebraic-coded Cryptosystems (PRAC) were suggested by Rao [Rao '84b] using the same techniques as MPBC but keep the public generator matrix as private. PRAC provides better security with simpler error correcting codes, hence, requires relatively low computational overhead. However, we show that PRAC can be broken easily by a chosen-plaintext attack. Both MPBC and PRAC are classified as Algebraic-Coded Cryptosystems (ACC) here.

This paper introduces a new approach to PRAC, which requires simple error correcting codes (i.e. distance 3 codes) and also provides much higher security level.

1.1. McEliece Public-key Cryptosystems (MPBC)

Encryption

Let G be a t -error correcting $k \times n$ generator matrix of a linear code over GF(2) capable of t -error correction. The rate of the code is $\frac{k}{n}$. We can select a random $k \times k$ nonsingular matrix S called scrambler and a random $n \times n$ permutation matrix P . Having G , S and P , we can compute the public generator matrix G' such that $G' = SGP$, which is combinatorially equivalent to G .

Then the encryption is done by:

$$C = MG' + Z$$

where C : ciphertext of length n ,

M : plaintext message of length k ,

Z : random error vector of length n with weight t .

Note that the vectors are italic lettered, and weight means Hamming weight.

Decryption

The decryption is very straight forward.

From the encryption equation

$$G' = SGP$$

$$C = MG' + Z$$

$$= MSGP + Z$$

$$= M' GP + Z \quad \text{where } M' = MS$$

Hence, we can recover M as given by the following steps.

Step 1 compute C' :

$$\begin{aligned} C' &= CP^T = M' G + ZP^T \\ &= M' G + Z' \quad \text{where } Z' = ZP^T \end{aligned}$$

(Note: Z' has same weight as Z since

P and P^T are permutation matrices)

Step 2 Decoding and error correction:

(Patterson Algorithm [MCEL 77]).

Step 3 recover plaintext M :

$$M = M' S^{-1}$$

Cryptanalysis of MPBC

As suggested by McEliece in his paper [McEliece '78], there could be two kinds of basic attacks for the cryptanalyst to try.

(a) Factoring S , G and P from G'

Since the number of codes which are combinatorially equivalent to a given code is astronomical, it is hopeless task to find out exact keys S , G and P used for G' . However, the cryptanalyst needs only some

S_i, G_i and P_i such that $S_i G_i P_i = G'$ and G_i is t-error correcting code. For the given G' , the cryptanalyst can obtain S_j, G_s and P_j satisfying the equation $S_j G_s P_j = G'$, where G_s is a generator in systematic form. G_s is obtained from G' by elementary row operations (row canonical reduction) and column operations. G', G_s and G are all said to be combinatorially equivalent. Where as G corresponds directly to a Goppa code which has well understood and well-known decoding algorithms, no such would be possible for G_s . Trial and error manipulation to obtain a G_s coinciding with an equivalent Alternant code generator would require an astronomically large work factor.

(b) Recovering M from C directly without keys

Another approach involves solving a set of k -unknowns from n simultaneous equations for all possible Z values.

Let M and C be a plaintext pair

$$\begin{aligned} M &= m_1 m_2 m_3 \dots m_k \\ C &= c_1 c_2 c_3 \dots c_k \dots c_n \\ Z &= z_1 z_2 z_3 \dots z_k \dots z_n \end{aligned}$$

$$\begin{aligned} G' &= [G_{ij}'] , \quad i = 1, \dots, k \\ &\quad j = 1, \dots, n \\ &\quad (\text{t-error correcting algebraic code}) \end{aligned}$$

Then, for $j = 1, \dots, n$

$$\begin{aligned} c_1 &= m_1 G_{11}' + m_2 G_{21}' + \dots + m_k G_{k1}' + z_1 \\ c_2 &= m_1 G_{12}' + m_2 G_{22}' + \dots + m_k G_{k2}' + z_2 \\ &\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\ c_n &= m_1 G_{1n}' + m_2 G_{2n}' + \dots + m_k G_{kn}' + z_n \end{aligned}$$

To solve k unknowns (m_1, m_2, \dots, m_k), k^3 operations are required because k equations are sufficient to solve the equations if the code is maximal distance separable (MDS) code. Otherwise, at most $k' = n-d+1$ equations are required to solve for k -unknowns [Pless '82].

Since t is smaller than $n-k$, it is possible that the cryptanalyst could select k equations containing no errors from n equations. Therefore, the cryptanalyst could repeat solving equations by selecting arbitrary k equations from n simultaneous equations with the assumption of no errors in selected equations until a meaningful plaintext is obtained.

The probability of no errors in k equations, P_k is:

$$P_k = \prod_{i=0}^{k-1} \left(1 - \frac{t}{n-i}\right)$$

and the average number of repetition is P_k^{-1} .

Hence, the average work factor, T is:

$$T = k^3 * P_k^{-1}$$

However, this does not include the work factor to check whether the plaintext M obtained by solving equations is correct (i.e., meaningful) or not. It is assumed that the plaintexts are from a source such as natural language or a programming language which contains an enormous amount of redundancy [Denning '82]. Redundancy in M helps to determine the validity of the plaintext derived.

1.2. Private-key Algebraic-coded Cryptosystems (PRAC)

To increase information rate and to reduce computational (encryption and decryption) overhead of MPBC, Private-key Algebraic-coded Cryptosystems (PRAC) were suggested [Rao '84b]. PRAC can provide better security with simpler error correcting codes, hence, require relatively low computational overhead compared to MPBC.

PRAC keeps G' private as well as S , P and G to provide higher security level. A known-plaintext attack to PRAC is feasible by solving matrices for each column vector of G' independently but this method requires a very large set of known (M, C) pairs. Hence, this attack can be foiled by periodic change or modification of the keys by the cryptographer. However, the analysis given below shows that PRAC still requires large t to be secure from a chosen-plaintext attack.

Chosen-Plaintext Attack

The cryptanalyst is required to go through two steps.

Step 1 : Solve for G' from a large set of (M, C) pairs.

Step 2 : Determine M from C using G' obtained in Step 1 (same work factor as in MPBC).

It can be safely assumed that a chosen plaintext of the form $M = (00 \dots 010 \dots 0)$ with only one 1 in i^{th} position (for $i = 1, \dots, k$) is not allowed by the cryptosystem. However, a chosen-plaintext attack may proceed as follows.

Let M_1 and M_2 are two plaintext differing in one position only, that is,

$$M_1 - M_2 = (00 \dots 0 \underset{i^{\text{th}}}{\overset{|}{1}} 0 \dots 0)$$

i^{th} position for $i = 1, \dots, k$

then,

$$C_1 - C_2 = g_i' + (Z_1 - Z_2) \quad (\text{Eq. 1})$$

where g_i' is the i^{th} row vector of G' .

The Hamming weight of $(Z_1 - Z_2)$ is at most $2t$. Since t is much smaller than n , the majority of the bits of the vector $C_1 - C_2$ correspond directly with g_i' . We can let $C_1 - C_2$ represent one estimate of g_i' . By repeating the step several times a number of estimates of g_i' can be obtained. From these estimates of g_i' and by majority voting for each position, the vector g_i' can be correctly determined. This step repeated for all $i = 1, 2, \dots, k$ will give us G' , which can be used to break the code by step 2. This step 2 will require a relatively small work factor because t is small.

However, a chosen-plaintext attack of the above nature can succeed only when $\frac{t}{n}$ is small and it will not if $t \approx \frac{n}{2}$.

2. MODIFIED CRYPTOSYSTEMS

2.1. Introduction

Our intent here is to obtain private-key cryptosystems using simple algebraic codes such as Hamming codes or distance 5 BCH codes. Furthermore, we would still want the Z vector to have a weight t sufficiently large to provide good security. By a clever design we will show that we could obtain $t \approx \frac{n}{2}$. Obviously it would not be possible unless we change or modify the original encryption method.

Here we develop such a modification and show that it is indeed possible to use simple (i.e., short distance) algebraic codes for PRAC which are very secure from chosen-plaintext attacks. Clearly a system that is secure from such an attack is also secure from other attacks including known-plaintext attacks.

2.2. Encryption of Modified PRAC

This approach uses a minimum distance 3 code generator G (as an example) and uses specific error patterns for the random error vector Z of which the average Hamming weight is approximately $\frac{n}{2}$. Encryption method is modified as follow.

Let $G' = SG$

where $S : k*k$ nonsingular matrix

$G : k*n$ distance 3 code generator matrix

$G' : k*n$ encryption matrix

Then

$$C = (MG' + Z)P \quad (\text{Eq. 2})$$

where $M : \text{plaintext of length } k$

$C : \text{ciphertext of length } n$

$P : n*n$ permutation matrix

$Z : \text{a random ATE (Method 1)}$

or an entry of the Syndrome-error table (Method 2)

(Method 1 and 2 are described below.)

Since the security of PRAC crucially depends on the weight of Z , the selection of Z is very important. We introduce two kinds of error patterns.

Method 1 : Use adjacent t errors for Z .

Definition 1 : Adjacent t Errors (ATE)

An ATE is a vector of length n with t ($\leq \frac{n}{2}$) adjacent errors, i.e., an ATE consists of $n-t$ 0's and t consecutive 1's. ATE must not be a codeword.

A random ATE can be used for Z . There exist exactly $n-t+1$ ATE's for the given n and t (and n ATE's for cyclic codes).

Method 2 : Use of predetermined set of vectors (Syndrome-error table).

A predetermined set of vectors consisting one from each coset of the standard array decoding table can be used for Z . Each coset has a distinct syndrome and there are exactly 2^{n-k} cosets [Blahut '83, Lin '83]. Therefore, we could select any set of vectors one from each of the 2^{n-k} cosets. The set is predetermined in the sense the decryptor knows the Syndrome-error table used for Z . Fig.1 shows an example of standard array and Syndrome-error table. The vectors in the rectangular boxes are selected as Z -vectors.

$$H = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Coset leader	Syndrome								
000000	001110	010101	100011	011011	101101	110110	111000	000	
000001	001111	010100	100010	011010	101100	110111	111001	001	
000010	001100	010111	100001	011001	101111	110100	111010	010	
000100	001010	010001	100111	011111	101001	110010	111100	100	
001000	000110	011101	101011	010011	100101	111110	110000	110	
010000	011110	000101	110011	001011	111101	100110	101000	101	
100000	101110	110101	000011	111011	001101	010110	011000	011	
001001	000111	011100	101010	010010	100100	111111	110001	111	

Fig. 1. Standard array for the (6, 3, 3) code

G' and P are secret encryption keys, and the Syndrome-error table is also secret in the Method 2.

2.3. Decryption of Modified Cryptosystems

From the encryption algorithm (Eq. 2)

$$\begin{aligned} C &= (MG' + Z)P \\ &= MSGP + ZP \\ &= M' GP + ZP. \quad (M' = MS) \end{aligned}$$

Decryption can be done using secret keys S^{-1} , H^T ($GH^T = 0$) and P^T through following steps.

Step 1 Obtain C' :

$$C' = CP^T = M' G + Z$$

Step 2 : Find the error pattern and recover M' :

$$\begin{aligned} C' H^T &= M' GH^T + ZH^T \\ &= ZH^T \quad (\text{Syndrome}) \end{aligned}$$

Identify the error pattern.

(use the Syndrome-error table look-up for the Method 2).

Recover M' by correcting for the error pattern.

Step 3 Recover plaintext M :

$$M = M' S^{-1}$$

Note : It appears that this approach requires long keys (S , P , G and the Syndrome-error table for the Method 2). However, the keys could be generated by using a pseudo-random number generator algorithm. In that case the user may require only short seeds for keys S , P and the Syndrome-error table. This problem is not addressed here and it would be a topic for future work.

2.4. Application to JOEEC

Recently Joint Encryption and Error-control Coding (JOEEC) was suggested [Rao '84a]. This approach combines data encryption and error-control coding steps into one step to gain speed and efficiency in implementation.

The modified cryptosystems could also be implemented as JOEEC by using higher distance codes. But the application to JOEEC of this approach is presently being studied.

3. CRYPTANALYSIS OF MODIFIED CRYPTOSYSTEMS

The encryption algorithm (Eq. 2) can be rewritten as follows.

$$C = (MG' + Z)P$$

$$= MG'' + ZP$$

where $G'' = G'P = [g_i'']$ for $i = 1, \dots, k$,
and g_i'' is a row vector.

The following lemmas help us to establish the high level of security provided by this new approach.

Lemma 1 : The number of P's that transform ATE's into non-ATE's is at least $(n - \lfloor \frac{n}{t} \rfloor - 1)!$ if $2 < t \leq \frac{n}{2}$, where n is the length of ATE and t is the length of adjacent errors.

Outline of Proof: Let vector V be an ATE of length n. We select a set of positions, $\{1, 2, t, 2t, \dots, bt\}$, from V where $b = \lfloor \frac{n}{t} \rfloor$. We reorder these positions as an ordered set, $B = \{1, t, 2t, \dots, bt, 2\}$. This mapping is illustrated in the figure below.

$$V = | +-----+-----+-----+-----+-----+-----| \quad (\text{ATE}) \\ 1 \quad 2 \quad t \quad 2t \quad 3t \quad \dots \quad bt \quad n \quad b = \lfloor \frac{n}{t} \rfloor$$

$$B = \{1, t, 2t, \dots, bt, 2\}$$

$$V' = |-----+----B----+-----| \quad (\text{non-ATE})$$

We consider a permutation map of vector V to V' with B embedded in V'. The purpose is to make V' a non-ATE. This is achieved because B

contains at least one '1' separated by '0's. Strictly B could start from any position of V' and therefore, we have $n-b-1$ choices. In addition the number of permutations possible for $V \rightarrow V'$ of the remaining positions is $(n-b-2)!$. Thus the total number of permutations of transforming an ATE vector V to non-ATE vector V', N_p can be shown to be at least

$$\begin{aligned} N_p &= (n-b-1) * (n-b-2)! \\ &= (n-b-1)! \end{aligned}$$

QED.

This formula gives a lower bound for N_p of $(n-3)!$ when $t = \lfloor \frac{n}{2} \rfloor$.

Lemma 2 : The number of code generators combinatorially equivalent to a $(n, k, 3)$ code generator is at least $k!$.

Proof: Let G be a $(n, k, 3)$ code generator in systematic form.

$$G = [I_k \ P_{k,n-k}]$$

where I_k is an identity matrix and

$P_{k,n-k}$ is a parity check matrix.

Then, there are $k!$ row combinations of parity check matrix, which are distinct $(n, k, 3)$ code generators also. All of these code generators can be obtained by row exchange and column permutation of G, and hence, are combinatorially equivalent to G [Peterson '72].

Lemma 3: The number of $k \times k$ non-singular matrices over GF(2), N_s is given by

$$N_s = \prod_{i=0}^{k-1} (2^k - 2^i) > 2^{k^2 - k} \quad (\text{Eq. 3})$$

Proof: We can start with any non-zero vector for the first row of non-singular matrix S and we have $2^k - 1$ choices. The second row must be linearly independent of the first. That is we have $2^k - 2$ choices for the second row. For the third row the choice is any vector linearly independent of the first two. Clearly it has $(2^k - 2^2)$ choices. Continuing this way, the number of non-singular matrices are given by the equality (Eq. 3). Since there are k terms in the product, the smallest of which is 2^{k-1} ,

the inequality is easily proved.

An attack by exhaustive search for S, G and P is considered hopeless task due to the results of above Lemmas. The previously described method of the chosen-plaintext attack (described in Section 1.2.1) can not be applied here because the average Hamming weight of $(Z_1 - Z_2)P$ is about $\frac{n}{2}$, which is very large. Therefore, we have to look for a different method to cryptanalysis and it could be as follows.

Let C_j and C_k be two distinct ciphertexts obtained for the same plaintext M .

$$C_j = MG'' + Z_j P$$

$$C_k = MG'' + Z_k P$$

$$C_j - C_k = (Z_j - Z_k)P$$

The above step provides one value for $(Z_j - Z_k)P$. This step needs to be repeated until all possible pairs of Z 's are used. The number of distinct Z 's is given by

$$N = \frac{n}{2} \text{ for the Method 1,}$$

$$\geq n \text{ for the Method 2;}$$

and the number of possible distinct values of $(Z_i - Z_j)P$ is $\frac{N^2 - N}{2}$.

An expression for g_i'' by a computation as described in Section 1.2.1 is given by

$$\begin{aligned} C_1 - C_2 &= g_i'' + (Z_1 - Z_2)P \\ g_i'' &= C_1 - C_2 - (Z_1 - Z_2)P. \end{aligned} \quad (\text{Eq. 4})$$

Hence, every possible value of $(Z_i - Z_j)P$ should be tested for $(Z_1 - Z_2)P$ of Eq. 4. Since the correctness of each row vector of G'' , g_i , can not be verified independently, the complete solution of G'' should be obtained and verified.

This involves on the average work factor, T given by

$$T \geq \frac{1}{2} \left(\frac{N^2}{2} \right)^k.$$

Substituting for N , T can be shown to be $\Omega(n^{2k})$. Thus we establish the following.

Claim : To determine G'' from a chosen-plaintext attack (as discussed above) has a work factor $T = \Omega(n^{2k})$.

It can be easily shown that the above step, namely, the determination of G'' is the really dominant factor. Determination of P and Z vectors are straight forward after that. As of now, the analysis and procedure explained seems to be the only possible approach to break the code and it requires an enormous work factor $\Omega(n^{2k})$.

4. CONCLUSION

We have introduced a new approach to the private-key algebraic-coded cryptosystems requiring only simple codes such as distance 3 codes. These systems will be very efficient because of high information rates and low overhead for encoding and decoding logic. The chosen-plaintext attack given here appears to be the only plausible approach for cryptanalyst.

It requires a work factor $\Omega(n^{2k})$ and is therefore, computationally secure even for small $k \approx 50$. It will be a challenge to find alternate methods of attack which can be successful.

REFERENCES

- [Blahut '83] Richard E. Blahut, *Theory and Implementation of Error Correcting Code*, Addison-Wesley, 1983.
- [Denning '82] Dorothy E. Denning, *Cryptography and Data Security*, Addison Wesley, 1982.
- [Lin '83] Shu Lin, Daniel J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, Prentice-Hall, 1983.
- [McEliece '77] McEliece R. J. "The theory of Information and coding," (vol. 9 of the *encyclopedia of mathematics and its Applications*) Reading, Mass Addison-Wesley, 1977.

[McEliece '78] R.J. McEliece, "A Public-Key Cryptosystem Based on Algebraic Coding Theory," *DSN Progress Report*, Jet Propulsion Laboratory, CA., Jan. & Feb. 1978, pp 42 - 44.

[Peterson '72] W. Wesley Peterson and E. J. Weldon, Jr., *Error-Correcting Codes*, Second edition, The MIT Press, 1972.

[Rao '84a] T.R.N. Rao, "Joint Encryption and Error Correction Schemes," *Proc. 11th Intl. Symp. on Comp. Arch.*, Ann Arbor, Mich., May 1984.

[Rao '84b] T.R.N. Rao, "Cryptosystems Using Algebraic Codes," *Intl. Conf. on Computer Systems & Signal Processing*, Bangalore, India, Dec. 1984.

Some Variations on RSA Signatures & their Security

*Wiebren de Jonge & David Chaum**

Subfaculteit Wiskunde en Informatica
Vrije Universiteit Amsterdam the Netherlands

*Centre for Mathematics and Computer Science
Kruislaan 413 1098 SJ Amsterdam the Netherlands

Abstract: The homomorphic structure of RSA signatures can impair security. Variations on a generalization of RSA signatures are considered with the aim of obviating such vulnerabilities. Of these variations, which involve a function of the message in the exponent, several are shown to have potential weaknesses similar to those of RSA.

No attacks have been found for one of the variations. Its security does not depend on redundancy present in or artificially combined with messages. The same holds for a well-known use of RSA that relies on a one-way compression function. A comparison between the schemes is given.

Introduction

The RSA signature function is a homomorphism with respect to multiplication. This multiplicative property can be useful, since it allows various powerful techniques, such as blind signatures [Chaum85]. However, the property also means a potential weakness for RSA signatures used in other applications, since it prevents some redundancy schemes from securing RSA signatures against attacks based on the property [DeJCh85].

One solution would be to find redundancy schemes that are able to resist such attacks. Another solution, which has been well-known in the folklore and offers some advantages, is to make use of a one-way compression function. A third approach is to try to find signature schemes that do not have such unwished for properties. This last approach is the main subject of this paper.

The RSA digital signature scheme is extensively reviewed in section 2. Particularly, that section treats in some detail the aspects relevant to this paper: redundancy, chosen signature attacks, multiplicative attacks, chaining and compression.

In section 3 a generalization of RSA signatures is introduced. This generalization

encompasses, besides RSA, various other digital signature schemes. The properties of one of these schemes are analyzed in section 4 and compared with RSA in section 5.

1. RSA signatures

In the signature scheme of Rivest, Shamir and Adleman [RSA78], now widely known as RSA, each user chooses a large number n , which is a product of two large primes p and q , and a pair of numbers (e, d) such that $e \cdot d$ is congruent with 1 modulo $\phi(n)$. Here, ϕ denotes Euler's totient function and, since n is $p \cdot q$, $\phi(n)$ is equal to $(p - 1) \cdot (q - 1)$. Each user has to publish his n and e , but should keep secret his d .

In RSA, a user A can construct a signature S_A on a suitable numeric message M by computing $S_A(M) = M^{d_A} \bmod n_A$, where d_A is A 's secret and n_A is A 's published product.

Since for other persons finding d_A is as difficult as factoring n_A (which is, as far as we know, infeasible for large, appropriately chosen n_A), only user A can compute $S_A(M)$ in practice, so long as he keeps his d_A secret. As a consequence, anyone can substantiate a claim that A signed M if he can come up with $S_A(M)$.

However, if somebody comes up with a number S and claims that it is M signed by A , one should be able to check whether indeed S is equal to $S_A(M)$. Since only A can produce $S_A(M)$, this check cannot be performed directly. But anyone can compute $S^{e_A} \bmod n_A$ using the public numbers e_A and n_A . If the result of this computation is not equal to $M \bmod n_A$, then S cannot have been equal to $S_A(M)$, since

$$(S_A(M))^{e_A} \bmod n_A = M^{d_A \cdot e_A} \bmod n_A = M \bmod n_A.$$

Thus, if somebody comes up with a number S for which $S^{e_A} \bmod n_A$ is equal to M , then one should be convinced that A signed M , otherwise not.

In RSA messages must be restricted to natural numbers less than n . (How to deal with larger messages will be explained in section 2.3.) Without this restriction, messages that are equal modulo n would have the same signature. Since n is publicly known, fraud would be too easy. One should not even use all numbers M with $0 \leq M < n$ for messages to be signed.

1.1 Redundancy to prevent a chosen signature attack

Rivest, Shamir and Adleman recommended that n be about 200 decimal digits long, which amounts to about 664 bits. For concreteness and convenience, while retaining an ample margin of safety, we will assume for the rest of this paper that n is 800 bits long.* Thus, a message can

* One cannot choose n arbitrarily large, since the practicability of RSA decreases as n increases, due to the increasing computational cost of exponentiation.

comprise as much as 100 bytes.

A forger can choose a number S_c , with $S_c < n_A$, and compute $M_c = (S_c)^{e_A} \bmod n_A$ from it. Subsequently, he could claim successfully that S_c is the message M_c signed by A. Since exponentiation modulo n acts as a one-way function when $\phi(n)$ is unknown, this *chosen signature attack* can be used for finding signatures on “random” (i.e., unpredictable) messages only. (In other words, nobody but A can make the signature on a chosen message, but anybody can determine which message corresponds to a chosen signature.)

To prevent such unpredictable messages from having a reasonable chance of being meaningful, it is necessary to have *redundancy* in the messages. Thus, a distinction will be made throughout the paper between *messages* and *valid messages*. All numbers M with $0 \leq M < n$ are messages, but only a very small fraction of these are valid messages. For example, if 100 bits of redundancy is used, a chosen signature will have only a chance of 2^{-100} of corresponding to a valid message. Thus, finding a *false signature* (i.e., a signature on a valid message not actually signed by A) will cost 2^{99} trials on the average, which makes a successful chosen signature attack infeasible.

Finally, notice that messages need redundancy against a chosen signature attack because RSA is a *readable signature* scheme; i.e., a scheme whereby anyone can derive the message signed from the signature.

1.2. Multiplicative attacks

The need for redundancy in RSA messages has been established. To prevent a chosen signature attack only the *quantity* of the redundancy present in valid messages is of interest. Still, the *nature* of the redundancy is also important, since RSA signatures have the property of being *multiplicative*.

For example, suppose that person B can construct three valid messages M_1 , M_2 and M_3 such that $M_3 = (M_1 \cdot M_2) \bmod n_A$. Then, if he succeeds in getting M_1 and M_2 signed by A, he can form the product (modulo n_A) of these signatures to get a false signature on M_3 , since

$$\begin{aligned} S_A(M_3) &= (M_1 \cdot M_2)^{d_A} \bmod n_A \\ &= ((M_1^{d_A} \bmod n_A) \cdot (M_2^{d_A} \bmod n_A)) \bmod n_A \\ &= (S_A(M_1) \cdot S_A(M_2)) \bmod n_A. \end{aligned}$$

B can also use the inverse M^{-1} or the opposite $-M$ of a message M of which he knows the corresponding signed version, as a factor in a product forming a new message, for, $S_A(M^{-1} \bmod n_A) = (S_A(M))^{-1} \bmod n_A$ and $S_A((-M) \bmod n_A) = (-S_A(M)) \bmod n_A$. (This last equation is true, because d_A is known to be odd.)

Thus, if B knows A’s signature on one or more valid messages M_i , he can easily forge a signature for any new valid message which he can discover how to rewrite as a product of

message(s) M_i , their opposite(s) $-M_i$, or their inverse(s) M_i^{-1} (all modulo n_A). (Notice that a message and/or its opposite and/or its inverse may occur in such a product more than once.) Therefore, the redundancy should prevent feasibility of discovering such valid messages.

As already mentioned, the protection that a redundancy scheme offers against a chosen signature attack depends only on the amount of redundancy. For protection against multiplicative attacks, however, the nature of the redundancy is also important, because the difficulty of finding valid messages that are products of other valid messages or their inverses can be different for two redundancy schemes, even if both use the same amount of redundancy. Indeed, for some redundancy schemes it seems feasible to find such combinations even though these schemes command a considerable amount of redundancy [DeJCh85].

For example, in two simple redundancy schemes each message must start (respectively end) with a sequence of, say, 100 zero-bits to be accepted as a valid message. Although these two simple and well-known techniques to add redundancy make a chosen signature attack infeasible, they do not generally provide sufficient protection against multiplicative attacks [DeJCh85].

1.3. Chaining or compression

Since our RSA only provides signatures on messages of at most 800 bits, some method is needed to make it also useful for larger messages.

An obvious approach is to split messages up into appropriate parts, and then to sign each part separately. To ensure that the parts are not re-ordered, they should be *chained* in some way; i.e., each part should contain extra information linking it unambiguously to (an)other part(s).

Another solution is to use a suitable, publicly known, one-way, *compression* function F_c (see also section 4.4), which maps a message of *any* size to some 800-bit number, before applying the RSA signing function. Thus, A 's signature on a message M then will be $(F_c(M))^{d_A} \bmod n_A$. Note that the function F_c indeed needs the one-way property. Otherwise, a person having acquired A 's signature on some message could determine some (or all) other messages having the same signature, and could claim successfully that he got such messages from A .

The one-way property of F_c implies that the signatures have become *unreadable* (which means that the message cannot be derived from the signature), which has two implications. First, the message itself must be delivered together with the particular number which constitutes its signature; i.e., a signed message consists of the pair $(M, (F_c(M))^{d_A} \bmod n_A)$. Second, redundancy no longer has to be present in M to prevent the chosen signature attack. (Clearly, a chosen signature attack does not work if it is infeasible to find a message for which the chosen bit pattern is a true signature.) Thus, any bit pattern may represent a valid message, and a signed message needs only 100 bytes more than the *actual* message (i.e., the original message without added redundancy) itself. Therefore, this scheme may be more (storage-)efficient than RSA with chaining, particularly for large messages.

In the case of RSA with chaining, each part of the message must be signed and/or checked separately. Therefore, the costs of signing and/or checking a large message is linear in its size. (Thus, if a message is, for example, three times as long as another large message, signing it costs three times as much as signing the other one.) When RSA is used with compression, after the compression only one part of 800 bits is still involved. Thus, if the computation required for the one-way compression can be made relatively cheap, RSA with compression will be more (time-)efficient than RSA with chaining (which subsequently will also be called: basic RSA).

Although redundancy is no longer required to prevent chosen signature attack, some well-chosen redundancy may still be necessary to prevent a multiplicative attack. For example, this would be the case when the compression function F_c is a homomorphism with respect to multiplication modulo n on a considerable part of its domain; i.e., if for many M_1 and M_2 :

$$F_c(M_1 \cdot M_2) \bmod n = (F_c(M_1) \cdot F_c(M_2)) \bmod n.$$

2. Generalized Exponentiation Signatures

As explained above, the multiplicative property of RSA means in particular that one should be very careful in choosing a redundancy scheme. Instead of looking for a suitable redundancy scheme, we will try to solve the problem by finding a signature scheme that does not have such unwished for structure. Thereto, we introduce a generalization of RSA which uses functions of M and n in the base as well as in the exponent, while preserving the idea of choosing n as a public product of secret primes to keep $\phi(n)$ secret. Thus, in this Generalized Exponentiation Signature (GES) scheme, signatures look like:

$$F_1(M, n)^{F_2(M, n)} \bmod n.$$

Since only knowledge of $\phi(n)$ should provide the ability to make signatures corresponding to n , function F_2 is supposed to comprise at least the computation of an inverse in modulo $\phi(n)$ arithmetic.

Obviously, RSA is a special case of GES whereby F_2 is chosen to be a constant function always mapping to $d = e^{-1} \bmod \phi(n)$, and whereby the function F_1 comprises, for example, the redundancy mapping and/or the compression function. In the following, three other signature schemes, which are special cases of GES, will be investigated. The first two of these will be shown to give problems similar to those of RSA. The third variation of GES seems a more promising alternative to RSA.

2.1 A first variation

In our first example, F_1 and F_2 are chosen to be $M \bmod n$ and $(M \cdot d) \bmod \phi(n)$, respectively. (Note that it makes no difference at all whether F_1 is chosen to be M or $M \bmod n$. Similarly, it is equivalent to have just $M \cdot d$ for F_2 .) Thus, a signature looks like:

$$M^{M \cdot d} \bmod n.$$

In this scheme the combination of a multiplicative property in the base (like in RSA) and an additive property in the exponent may seem to make it more difficult to tamper with signatures to produce a false one. However, the following counterexample shows that this scheme does not offer a significant improvement.

Suppose one has two messages M_1 and M_2 signed by A. By raising the signature on M_1 to the power M_2 (in modulo n_A arithmetic) one gets:

$$(M_1^{M_1 \cdot d} \bmod n_A)^{M_2} \bmod n_A = (M_1^{M_1 \cdot M_2 \cdot d}) \bmod n_A.$$

Similarly, one can raise the signature on M_2 by M_1 . Multiplying both results gives A's signature on $M_1 \cdot M_2$, for,

$$(M_1^{M_1 \cdot M_2 \cdot d} \bmod n_A) \cdot (M_2^{M_1 \cdot M_2 \cdot d} \bmod n_A) = (M_1 M_2)^{(M_1 M_2) \cdot d} \bmod n_A.$$

Thus, if one knows A's signature on one or more valid messages M_i , one can easily forge a signature for any new valid message that one can discover how to write as a product of message(s) M_i . Consequently, the only achievement is that the inverses of these M_i cannot be used as factors in such products, as was the case with RSA (see section 2.2).

2.2 A second variation

In a second variation of GES, F_1 and F_2 are chosen to map to a constant number C and to the inverse of M in modulo $\phi(n)$ arithmetic, respectively. In this variation A's signature function is:

$$S_A(M) = C^{M^{-1} \bmod \phi(n_A)} \bmod n_A.$$

(How to guarantee that the inverse needed in the exponent does in fact exist will be treated shortly.)

At first sight this signature scheme might seem to offer excellent protection against tampering, since the additive property in the exponent does no harm, because the inverse of a sum is, in general, not equal to the sum of the inverses. Thus, the following inequality usually holds:

$$S_A(M_1) \cdot S_A(M_2) \neq S_A(M_1 + M_2).$$

However, this signature scheme is open to the following attack. Suppose one has A's signature on a valid message M which can be written as the normal integer product of some factors m_1, m_2, \dots, m_k . Thus,

* This first variation of GES can also be considered to be an "RSA with compression" signature using the compression function $F_c(M) = M^M \bmod n$.

$$S_A(M) = C^{(m_1 m_2 \dots m_k)^{-1} \bmod \phi(n_A)} \bmod n_A.$$

By raising $S_A(M)$ to the power m_1 in modulo n_A arithmetic, one gets $S_A(m_2 m_3 \dots m_k)$. Similarly, one is able to get a false signature on any other valid message that can be obtained from M by erasing one or more of its factors.

Another point to consider with this signature scheme, already mentioned, was how to guarantee that the inverse modulo $\phi(n)$, which is used as exponent, does in fact exist. For any number x , its inverse modulo $\phi(n)$ exists only if x is co-prime with $\phi(n)$; i.e., if $\gcd(x, \phi(n)) = 1$. However, this restriction poses no serious problems, even though $\phi(n)$ is known to always contain the factor 4.

For example, A can choose his public product n_A as follows. First he searches for two large primes p'_A and q'_A (roughly 400 bits each) such that $2 \cdot p'_A + 1$ and $2 \cdot q'_A + 1$ are also prime. Then A takes $n_A = p_A \cdot q_A$ with $p_A = 2 \cdot p'_A + 1$ and $q_A = 2 \cdot q'_A + 1$. As a consequence, $\phi(n_A)$ will be $4 \cdot p'_A \cdot q'_A$, and thus almost all odd numbers will be co-prime with $\phi(n_A)$. To be more precise, the chance that an odd number is not co-prime with $\phi(n_A)$ is $(p'_A + q'_A - 1) / (p'_A \cdot q'_A)$, which is negligibly small (roughly 2^{-400}). (Acceptably small probabilities might also be achieved when p'_A and q'_A have only large factors.) To get the number to be inverted to be odd, one could append a 1-bit to M before computing its inverse modulo $\phi(n_A)$. Note that not only $2M + 1$ is guaranteed to be odd, but also $(2M + 1) \bmod \phi(n_A)$, since $\phi(n_A)$ is even.

The change of A 's signature function to $S_A(M) = C^{(2M+1)^{-1} \bmod \phi(n_A)} \bmod n_A$ does not result in much more security against tampering. Suppose that $2M + 1$ is known to be a product of some factors m_1, m_2, \dots, m_k . The product of any subset of these factors then will be odd too, and thus will correspond to some other message M' . For example, if $k > 2$ one is able to get a false signature on the message $M' = (m_1 m_2 - 1) / 2$.

A safer signature scheme results if a one-way function F_o is used to change the signature function to $S_A(M) = C^{(2 \cdot F_o(M) + 1)^{-1} \bmod \phi(n_A)} \bmod n_A$. Making the base number also depend on the message to be signed seems to be another way to improve safety. This approach will be investigated below.

2.3. A third variation

Trying to prevent the attack that appeared to be possible in the previous section, we now choose $F_1(M, n) = M \bmod n$. For the exponent we use again $F_2(M, n) = (2M + 1)^{-1} \bmod \phi(n)$, assuming that n is chosen appropriately as described in the previous section. This time, the F_2 chosen is not only suited for solving the problems resulting from the fact that not every M has an inverse in $\bmod \phi(n)$ arithmetic, but also prevents the following attack that still (!) would be possible in case the signature function would be just $M^{M^{-1} \bmod \phi(n)} \bmod n$.

Suppose one would like to get a false signature on the message M . First, one uses some P and Q to construct three messages $M_1 = MP$, $M_2 = MPQ$ and $M_3 = M^2PQ$. If one succeeds in getting A to sign M_1 , M_2 and M_3 , one can forge A 's signature on M as follows.

Computing $(S_A(M_2))^Q \bmod n_A$ gives $(MPQ)^{(MP)^{-1}}$. Multiplying this with the mod n_A inverse of $S_A(M_1)$ gives $Q^{(MP)^{-1}}$. In a similar way, one can compute $(MQ)^{(MP)^{-1}}$ from $S_A(M_1)$ and $S_A(M_3)$. Multiplying $(MQ)^{(MP)^{-1}}$ with the mod n_A inverse of $Q^{(MP)^{-1}}$ gives $M^{(MP)^{-1}}$. This last number exponentiated with P gives $S_A(M) = M^{M^{-1}}$.

In the next section we will examine in some detail the properties of the more promising variation which uses $F_1(M, n) = M \bmod n$ and $F_2(M, n) = (2M + 1)^{-1} \bmod \phi(n)$. For convenience, this last scheme will be called DJ.

3. Some properties of DJ signatures

3.1. Fixed storage costs

Naturally any GES scheme that uses a function F_2 which is really dependent on M is unreadable. Thus, DJ signatures are unreadable. Therefore, to give a person a message signed by A , one has to send him both the message M and A 's signature on it; i.e., one has to send the pair $(M, M^{(2M+1)^{-1} \bmod \phi(n_A)} \bmod n_A)$. As a consequence, a signed message requires only 100 bytes more than the message itself (independent of the size of the message, as explained below). This is the same as for RSA with compression.

3.2. No need for redundancy in messages

DJ signatures being unreadable also implies that messages need no redundancy to protect against a chosen signature attack. Since DJ signatures are not multiplicative, messages do not need some well-chosen redundancy to prevent a multiplicative attack either. DJ appears to have no other unwanted properties such as, for example, being additive. And so, it currently seems to be secure against other, similar attacks.

3.3. No chaining required for large messages

With DJ it is not necessary to restrict M to, for example, numbers less than n , as in case of RSA. Of course, all messages that are congruent modulo n and modulo $\phi(n)$ will have the same signature. But this gives no problem, since $\phi(n)$ is supposed to be secret. So, the *implicit* compression that results from the reduction modulo $\phi(n)$, has the required one-way property. As a consequence, it is not necessary with the DJ scheme to use chaining or a *separate* compression function for signing large messages. Furthermore, it is likely that implicit compression is much easier (cheaper) to perform than separate compression, since a separate compression function is likely to involve a much more complex computation than just a reduction modulo $\phi(n)$ of $2M + 1$.

3.4. Computational costs

Because of the above mentioned implicit compression, forming a signature costs one exponentiation in modulo n arithmetic to an exponent of at most 800 bits. Thus, signing large messages is hardly more expensive than signing short ones. However, the same is not true for checking a signature. Since only the signer knows his $\phi(n)$, checking a signature has to be done by first raising (in modulo n arithmetic) the given signature to the full $2M + 1$ and then checking whether the result is indeed equal to $M \bmod n$. Thus, the cost of checking a signature is linear in the size of the message. (In the sense that checking a twice as large message cost twice as much work.) Recall that when RSA is used with chaining, the costs of signing and of checking are *both* linear in the size of the message, since each part of the message must be signed or checked separately.

Of course, as with RSA, it is also possible to use a separate, publicly known, one-way, compression function F_c . Then A 's signature on a message M will be

$$M^{(2F_c(M)+1)^{-1} \bmod \phi(n_A)} \bmod n_A. *$$

If the computation required for such a separate compression can be made relatively inexpensive, it is advantageous to use it for DJ as well, since the second phase of checking then also consists of only one exponentiation to a number of at most 800 bits. (The first phase encompasses the computation of $F_c(M)$.)

To show the different demands that RSA and DJ make upon the one-way function, we will digress now somewhat into the subject of one-way functions. A usual definition is that a function is called one-way if it is not generally feasible to find, for a given y , an x such that $F(x)=y$. However, in the context of cryptographic applications one often adds the requirement that, given some x , it should also be infeasible to find an x' with $x' \neq x$ such that $F(x')=F(x)$.

Since we have chosen to use the function F_c only in the exponent, the only requirement to be imposed on the compression function is that, when knowing some pair (x,y) for which $F_c(x)=y$, it must be infeasible to find an x' for which $x' \neq x$, $x' \bmod n = x \bmod n$ and $F_c(x')=y$. Thus, with DJ the compression function has to be “one-way” only in a more restricted sense. As a consequence, it may be much easier to find suitable compression functions for DJ than for RSA.

4. Comparison of DJ with RSA

Both the RSA and the DJ digital signature scheme are a special case of a GES scheme. The basic form of RSA has several unpleasant properties. Since it only works for messages of less than $\log_2(n)$ bits, large messages must be divided into small parts. To prevent forgeries, it is necessary to include in each of these parts some bits comprising chaining information and redundancy, to prevent a chosen signature attack. Furthermore, since RSA signatures are

* We prefer not to use $F_c(M)$ in the base as well.

multiplicative, the redundancy scheme to be used should be chosen very carefully to prevent successful multiplicative attacks. Thus, the security of basic RSA signatures (i.e., RSA with chaining) depends heavily on the quality of the redundancy scheme chosen.

The following properties of DJ signatures give them considerable advantages over basic RSA signatures:

- Messages and signatures are kept separate.
- Since DJ signatures are unreadable and not multiplicative, messages need no redundancy to prevent a multiplicative attack or a chosen signature attack.
- DJ signatures cost $\log_2(n)$ bits beyond the size of the message. In practice, this means less than 100 bytes for a signature.
- The costs for signing a message are more or less constant, because the implicit compression means that only one exponentiation to a number of at most $\log_2 \phi(n)$ bits has to be performed, even for very large messages. (Thus, signing large messages is much cheaper with DJ than with basic RSA, since in the latter case such an exponentiation has to be performed for each part of the message.)
- Every person has to publish only his product n . (The same holds for RSA if one agrees on everybody using the same public exponent e .)

Another way to circumvent the disadvantages of basic RSA signatures is to use RSA with compression. This requires a publicly known, one-way compression function that also destroys the multiplicative structure. With respect to RSA with compression, DJ has only two advantages. The first is that DJ even works without using any compression function. (With DJ, compression is only useful for bounding the costs for checking signatures on large messages.) Second, a compression function has to meet less requirements in case of DJ than in case of RSA; for example, it is not crucial for DJ whether the compression function destroys multiplicative properties or not. On the other hand, RSA with compression has the advantage that signatures can be checked more economically if everybody uses a relatively small number e as the public exponent. It may be concluded that RSA with compression and DJ both seem to be good digital signature schemes, and that both are better than RSA with chaining.

Summary

It has been shown in [DeJCh85] that RSA signatures may be vulnerable to so-called multiplicative attacks. In this paper we have shown a similar potential weakness in the special cases of the generalizations of RSA presented that use the functions $M^{M \cdot d} \bmod n$, $C^{(2M+1)^{-1} \bmod \phi(n)} \bmod n$, and $M^{M^{-1} \bmod \phi(n)} \bmod n$, respectively.

A further variation is presented that, although it does not rely on the use of a one-way function, does not seem to be vulnerable to multiplicative or other attacks known to the authors.

References

- [Chaum85] Chaum, D., Security Without Identification: Transaction Systems to Make Big Brother Obsolete, Communications of the ACM, Vol. 28, No. 10, October 1985, pp. 1030-1044.
- [DeJCh85] de Jonge, W. and Chaum, D., Attacks on some RSA Signatures, in: Hugh C. Williams (ed.), Advances in Cryptology - Crypto '85, Lecture Notes in Computer Science, No. 218, Springer Verlag, pp. 18-27.
- [RSA78] Rivest, R.L., Shamir, A., and Adleman, L., A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, Communications of the ACM, Vol. 21, No. 2, February 1978, pp. 120-126.

BREAKING THE CADE CIPHER

N.S. James
Department of Mathematics
University of Tasmania
Hobart, Tasmania 7001
Australia

R. Lidl
Department of Mathematics
University of Tasmania
Hobart, Tasmania 7001
Australia

H. Niederreiter
Kommission für
Mathematik
Austrian Academy of
Sciences
A-1010 Vienna
Austria

Abstract: A cryptanalysis is given of a cryptosystem introduced by J.J. Cade, which is based on solving equations over finite fields.

In 1985 J.J. Cade [1] introduced a new public-key cryptosystem. The Cade cryptosystem is a public-key cipher in which each block is a string of n binary digits or equivalently an element of the finite field IF_{2^n} . Because of the design of the system n must be a multiple of 3, say $n = 3c$. The blocks are enciphered by a permutation of IF_{2^n} induced by a polynomial $P \in \text{IF}_{2^n}[x]$ of the following form,

$$P(x) = p_{00}x^2 + p_{10}x^{q+1} + p_{11}x^{2q} + p_{20}x^{q^2+1} + p_{21}x^{q^2+q} + p_{22}x^{2q^2}$$

where $q = 2^c$ and $p_{00}, \dots, p_{22} \in \text{IF}_q[x]$. The six coefficients p_{00}, \dots, p_{22} are the public-key. The trapdoor information is a decomposition

$$P(x) \equiv S \circ M \circ T(x) \pmod{(x^{q^3} - x)}. \quad (1)$$

S and T are both linearized polynomials,

$$T(x) = a_0x + a_1x^q + a_2x^{q^2},$$
$$S(x) = b_0x + b_1x^q + b_2x^{q^2},$$

where $a_0, \dots, b_2 \in \text{IF}_q$ are the private key.

S and T are linear mappings of IF_q , considered as a vector space over IF_q and are both chosen to be invertible. A necessary and sufficient condition for a linearized

Research by the first two authors partially supported by Australian Research Grants Scheme grant No. F8415183.

polynomial $L(x) = \sum_{i=0}^{r-1} d_i x^{q^i} \in \text{IF}_{q^r}[x]$ to be invertible is that $\det A \neq 0$, where

$$A = \begin{pmatrix} d_0 & d_{r-1}^q & d_{r-2}^{q^2} & \cdots & d_1^{q^{r-1}} \\ d_1 & d_0^q & d_{r-1}^{q^2} & \cdots & d_2^{q^{r-1}} \\ \vdots & \vdots & \vdots^2 & \ddots & \vdots^{r-1} \\ d_{r-1} & d_{r-2}^q & d_{r-3}^{q^2} & \cdots & d_0^{q^{r-1}} \end{pmatrix}.$$

In the Cade cipher we have $q = 2^c$ and $r = 3$. The set of linearized polynomials over IF_{q^r} forms a group under composition mod $(x^{q^3} - x)$, called the Betti-Mathieu group, which is isomorphic to the general linear group of nonsingular r by r matrices with entries in IF_q , see [3] for details on linearized polynomials.

We note that $P(x)$ in (1) is obtained mod $(x^{q^3} - x)^3$. Therefore polynomial decomposition algorithms for finding the secret composition factors S , M and T are not applicable. M is the special monomial $M(x) = x^{q+1}$ which is invertible because $(q+1, q^3-1) = 1$ for $q = 2^c$. T , M and S are easy to invert and so $p^{-1} = T^{-1} \circ M^{-1} \circ S^{-1}$ is easy to calculate if one knows the private key.

We now give a method for finding the private key a_0, \dots, b_2 in terms of the public key p_{00}, \dots, p_{22} .

From (1) we have

$$\begin{aligned} P \circ T^{-1}(x) &\equiv S \circ M(x) \pmod{(x^{q^3} - x)} \\ &\equiv b_0 x^{q+1} + b_2 x^{q^2+1} + b_1 x^{q^2+q}. \end{aligned} \quad (2)$$

Because T is a linearized polynomial T^{-1} will have the same form as T . In fact

$$T^{-1}(x) = \alpha_0 x + \alpha_1 x^q + \alpha_2 x^{q^2}$$

where

$$\begin{aligned} \alpha_0 &= (a_0^{q^2+q} + a_1^q a_2^q)/A, \\ \alpha_1 &= (a_2^{q^2+1} + a_0^q a_1)/A, \\ \alpha_2 &= (a_1^{q+1} + a_0^q a_2)/A, \end{aligned} \quad (3)$$

$$\text{and } A = a_0^{q^2+q+1} + a_1^{q^2+q+1} + a_2^{q^2+q+1} \\ + a_0 a_1 a_2^{q^2} + a_0^{q^2} a_1 a_2 + a_0^{q^2} a_1 a_2^q .$$

We may then calculate $P \circ T^{-1}(x)$. This has six terms and comparison of the coefficients of these terms with those in (2) yields the following equations:

$$\left. \begin{aligned} b_0 &= p_{10}(\alpha_0^{q+1} + \alpha_1 \alpha_2^q) + p_{20}(\alpha_1^{q^2+1} + \alpha_0 \alpha_2^q) \\ &\quad + p_{21}(\alpha_2^{q^2+q} + \alpha_0^q \alpha_1^q) , \end{aligned} \right\} \quad (4)$$

$$\left. \begin{aligned} b_2 &= p_{10}(\alpha_2^{q+1} + \alpha_0 \alpha_1^q) + p_{20}(\alpha_0^{q^2+1} + \alpha_1^q \alpha_2^2) \\ &\quad + p_{21}(\alpha_1^{q^2+q} + \alpha_0^q \alpha_2^q) , \end{aligned} \right\}$$

$$\left. \begin{aligned} b_1 &= p_{10}(\alpha_1^{q+1} + \alpha_0^q \alpha_2) + p_{20}(\alpha_2^{q^2+1} + \alpha_0^q \alpha_1^2) \\ &\quad + p_{21}(\alpha_0^{q^2+q} + \alpha_1^q \alpha_2^q) , \end{aligned} \right\}$$

$$\left. \begin{aligned} p_{00} \alpha_0^2 + p_{11} \alpha_2^{2q} + p_{22} \alpha_1^{2q} &+ p_{10} \alpha_0 \alpha_2^q \\ &+ p_{20} \alpha_0^q \alpha_1^2 + p_{21} \alpha_1^q \alpha_2^q = 0 , \end{aligned} \right\}$$

$$\left. \begin{aligned} p_{11} \alpha_0^{2q} + p_{22} \alpha_2^{2q} &+ p_{00} \alpha_1^{2q} + p_{21} \alpha_0^q \alpha_2^q \\ &+ p_{10} \alpha_0^q \alpha_1 + p_{20} \alpha_1^q \alpha_2^q = 0 , \end{aligned} \right\}$$

$$\left. \begin{aligned} p_{22} \alpha_0^{2q} &+ p_{00} \alpha_2^{2q} + p_{11} \alpha_1^{2q} + p_{20} \alpha_0^q \alpha_2 \\ &+ p_{21} \alpha_0^q \alpha_1^q + p_{10} \alpha_1^q \alpha_2 = 0 . \end{aligned} \right\} \quad (5)$$

Now if we raise the second and third equations of (5) to the powers q^2 and q respectively and put $\alpha = \alpha_0$, $\beta = \alpha_2^q$, $\gamma = \alpha_1^{q^2}$ then we obtain

$$\left. \begin{aligned} p_{00} \alpha^2 + p_{11} \beta^2 + p_{22} \gamma^2 &+ p_{10} \alpha \beta + p_{20} \alpha \gamma + p_{21} \beta \gamma = 0 , \\ p_{11}^{q^2} \alpha^2 + p_{22}^{q^2} \beta^2 &+ p_{00}^{q^2} \gamma^2 + p_{21}^{q^2} \alpha \beta + p_{10}^{q^2} \alpha \gamma + p_{20}^{q^2} \beta \gamma = 0 , \\ p_{22}^{q^2} \alpha^2 + p_{00}^{q^2} \beta^2 &+ p_{11}^{q^2} \gamma^2 + p_{20}^{q^2} \alpha \beta + p_{21}^{q^2} \alpha \gamma + p_{10}^{q^2} \beta \gamma = 0 . \end{aligned} \right\} \quad (6)$$

If one of α , β or γ is zero then the equations are easy to solve. This can be detected a priori, e.g. if $\gamma = 0$ then necessarily

$$\det \begin{pmatrix} p_{00} & p_{11} & p_{10} \\ p_{11}^q & p_{22}^q & p_{21}^q \\ p_{22}^q & p_{00}^q & p_{20}^q \end{pmatrix} = 0.$$

Thus assume $\alpha\beta\gamma \neq 0$. Because the equations in (6) are homogeneous we may assume $\gamma = 1$. Using two of the equations in (6) to eliminate the α^2 term we obtain

$$\alpha(c_1\beta + c_2) + c_3\beta^2 + c_4\beta + c_5 = 0 \quad (7)$$

for some $c_1, \dots, c_5 \in \mathbb{F}_{q^3}$.

If $c_1 = c_2 = 0$ then we have a quadratic equation for β . Such an equation can be solved by treating this case as an affine polynomial and use of the method described in [4, p.103], or alternatively use the method of Exercise 4.44 in [4, p.161].

Otherwise we may substitute for α in one of the equations in (6) and so obtain a quartic equation for β . A quartic equation over \mathbb{F}_{2^n} may be solved by the method described in Chen [2]. Equations (3) and (4) then give the values of a_0, a_1, a_2 and b_0, b_1, b_2 respectively.

We understand from the originator of the Cade cipher that S. Berkovits has developed an alternative method of breaking the cipher. An improved version of the cipher has been presented at CRYPTO 86.

REFERENCES

1. Cade, J.J.; A public key cipher which allows signatures. Paper presented at 2nd SIAM Conference on Applied Linear Algebra, Raleigh 1985.
2. Chen, Chin-Long; Formulas for the solutions of quadratic equations over $GF(2^m)$, IEEE Trans. Inform. Theory 28, 792-794 (1982).
3. Lidl, R. and Niederreiter, H.; Finite Fields. Addison-Wesley, Reading, Mass. 1983. Now distributed by Cambridge University Press.
4. Lidl, R. and Niederreiter, H.; Introduction to Finite Fields and Their Applications. Cambridge University Press, Cambridge, 1986.

A MODIFICATION OF A BROKEN PUBLIC-KEY CIPHER

John J. Cade
24 Ginn Rd.
Winchester, MA 01890

Abstract

A possible public-key cipher is described and its security against various cryptanalytic attacks is considered.

1. Introduction

In this paper, we describe a possible public-key cipher. It is a modification of the public-key cipher that was proposed by the author [2] in April 1985, was broken by Berkovits [1] in August 1985, and was broken independently by James, Lidl, and Niederreiter [3] in October 1985.

This modified cipher, like the original, is a block substitution cipher that operates on binary messages. With this cipher, for a suitably large value of n , n -blocks of binary digits are identified with elements of the finite field $GF(2^n)$, and elements of $GF(2^n)$ are enciphered by means of a permutation of $GF(2^n)$ whose public description is as a polynomial function on $GF(2^n)$ which has a very high degree but only a few terms.

We consider several possible cryptanalytic attacks against the cipher. The most obvious attack consists of solving the polynomial equations of high degree over $GF(2^n)$ which relate corresponding n -blocks of plaintext and ciphertext. Another possible attack consists of solving the system of polynomial equations of high degree over $GF(2^n)$ that expresses the public key for the enciphering permutation in terms of secret trapdoor information about this permutation.

For each cryptanalytic attack that we consider, we give an estimate of the amount of computation required as a function of the cipher's block-length n . The estimates for all but one of the attacks are based on fairly complete and satisfying analyses of the attacks in question. Unfortunatley, however, for the attack by solving the system of equations that expresses the public key in terms of trapdoor information, the estimate is based only on indirect evidence obtained by an analysis of a simpler related system of equations. This attack will require further study, perhaps with the aid of a computer algebra system. On the basis of the estimates of the amounts of computation required by the various cryptanalytic attacks, it appears that the cipher provides adequate security with a block-length of $n \geq 150$.

This paper is organized as follows. In section 2 below, we describe our modified cipher. In section 3, we prove that the enciphering and deciphering permutations used in the cipher are indeed mutually inverse permutations. In sections 4 - 6, we describe various methods of cryptanalyzing the cipher and we estimate the amounts of computation required by these methods. Finally, in section 7, we summarize these estimates and use them to determine a suitable block-length for the cipher.

2. Description of the cipher

Our cipher is designed to encipher binary messages. Each such message is enciphered one n -block at a time, for a specified block-length n , by substituting for each plaintext n -block x a corresponding ciphertext n -block y which is given by $y = P(x)$, where P is a certain kind of permutation of the set of all binary n -blocks.

Because of the particular form of the enciphering permutations used in the cipher, the block-length n must be an integer for which there exist integers δ , γ , and β such that $n = 2\delta$ and $\delta = 2\gamma = 3\beta$. Note that an integer n satisfies this requirement if and only if n is

a multiple of 12. In the following, n , δ , γ , and β are understood to be as just described.

For the operation of the cipher, the set of all binary n -blocks must be identified in some specified way with the finite field $GF(2^n)$. Then the public description of the enciphering permutation P consists of a 16-term polynomial formula for P having the form

$$P(x) = \sum_{g=0}^3 \sum_{h=0}^3 p_{gh} x^{2^{\gamma g + \beta} + 2^{\gamma h}}. \quad (2.1)$$

The coefficients p_{gh} in this formula are publicly revealed elements of $GF(2^n)$ which constitute the public key for P .

Although P is a polynomial function of very high degree, $P(x)$ can nevertheless be computed quite efficiently for each $x \in GF(2^n)$. One way to do this is to use formula (2.1) written in the form

$$P(x) = \sum_{h=0}^3 \left(\sum_{g=0}^3 p_{gh} x^{2^{\gamma g + \beta}} \right) x^{2^{\gamma h}}$$

and to compute the powers of x of the form x^{2^k} appearing in this formula by doing k successive squarings. Computing $P(x)$ this way requires a total of just $(11/12)n$ squarings, 20 multiplications, and 15 additions in $GF(2^n)$.

$P(x)$ can be computed even more efficiently by using matrix-vector multiplication to compute various quantities which are the values of linear functions on $GF(2^n)$, where $GF(2^n)$ is regarded as a vector space over its smallest subfield $GF(2)$. To compute $P(x)$ this way, first compute the quantities u_0, \dots, u_3 and v_1, v_2, v_3 given by

$$u_h = \sum_{g=0}^3 p_{gh} x^{2^{\gamma g + \beta}}, \text{ for } h = 0, \dots, 3,$$

and $v_h = x^{2^{\gamma h}}$, for $h = 1, 2, 3$. Each of these quantities is a $GF(2)$ -linear function of x , and so can be computed by doing a single matrix-vector multiplication involving an $n \times n$ matrix over $GF(2)$ and an n -element vector over $GF(2)$. Then compute $P(x)$ by using the formula

$$P(x) = u_0x + \sum_{h=1}^3 u_h v_h.$$

Computing $P(x)$ this way requires a total of just 7 matrix-vector multiplications over $GF(2)$, together with 4 multiplications and 3 additions in $GF(2^n)$.

For the construction of enciphering permutations, $GF(2^n)$ and its subfield $GF(2^\delta)$ are regarded as vector spaces, of dimensions 4 and 2 respectively, over their common subfield $GF(2^\gamma)$. To construct an enciphering permutation, one first chooses at random two secret bases a_1, \dots, a_4 and b_1, \dots, b_4 of $GF(2^n)$ over $GF(2^\gamma)$. One also chooses a basis e_1, e_2 of $GF(2^\delta)$ over $GF(2^\gamma)$. This last basis need not be kept secret and can be chosen to be whatever is most convenient. The sequence $a_1, \dots, a_4, b_1, \dots, b_4, e_1, e_2$ formed by these three bases constitutes secret trapdoor information about an enciphering permutation P that is specified by this sequence. We will call this sequence a trapdoor sequence for the permutation P .

This permutation is constructed as follows. First, let S_1 and S_2 be the $GF(2^\gamma)$ -linear functions from $GF(2^\delta)$ into $GF(2^n)$ such that $S_1(e_j) = a_j$ and $S_2(e_j) = a_{j+2}$, for $j = 1, 2$. Next, let T_1 and T_2 be the $GF(2^\gamma)$ -linear functions from $GF(2^n)$ into $GF(2^\delta)$ such that

$$T_1(b_j) = \begin{cases} e_j, & \text{for } j = 1, 2 \\ 0, & \text{for } j = 3, 4 \end{cases}$$

and

$$T_2(b_j) = \begin{cases} 0, & \text{for } j = 1, 2 \\ e_{j-2}, & \text{for } j = 3, 4. \end{cases}$$

Finally, let M be the permutation of $GF(2^\delta)$ given by

$$M(x) = x^{2^\delta + 1}. \quad (2.2)$$

Then the enciphering permutation P specified by the trapdoor sequence $a_1, \dots, a_4, b_1, \dots, b_4, e_1, e_2$ is the function from $GF(2^n)$ into $GF(2^n)$ given by

$$P(x) = S_1 M T_1(x) + S_2 M T_2(x). \quad (2.3)$$

Here and in the following, we denote the composition of two or more functions by the juxtaposition of their symbols. Thus, for $i = 1, 2$,

$$S_1 \circ M \circ T_1(x) = S_1 \circ M \circ T_1(x) = S_1(M(T_1(x))).$$

We note that the enciphering permutation P just described does not determine a unique trapdoor sequence which specifies it. Indeed, it can be shown that for each enciphering permutation, there are a very large number of trapdoor sequences which specify it.

For the public description of the enciphering permutation P described above, P must be expressed as a polynomial function. To do this, first the functions S_1 and T_1 are expressed as polynomial functions. The functions S_1 are given by the polynomial formulas

$$S_1(x) = a_{10}x + a_{11}x^{2^Y}, \quad (2.4)$$

where the coefficients a_{1k} are the elements of $GF(2^n)$ uniquely determined by the system of linear equations

$$a_{10}e_j + a_{11}e_j^{2^Y} = S_1(e_j), \text{ for } j = 1, 2.$$

The functions T_1 are given by the polynomial formulas

$$T_1(x) = \sum_{k=0}^3 b_{1k}x^{2^{Yk}}, \quad (2.5)$$

where the coefficients b_{1k} are the elements of $GF(2^n)$ uniquely determined by the system of linear equations

$$\sum_{k=0}^3 b_{1k}b_j^{2^{Yk}} = T_1(b_j), \text{ for } j = 1, \dots, 4.$$

Once the elements a_{1k} and b_{1k} have been determined, the enciphering permutation P is given by the polynomial formula (2.1), where the coefficients p_{gh} are given by

$$p_{gh} = \sum_{i=1}^2 \sum_{k=0}^1 a_{1k}(b_{i,g-k})^{2^{Yk+\beta}} (b_{i,h-k})^{2^{Yk}}, \quad (2.6)$$

where $b_{i,-1} = b_{i,3}$, for $i = 1, 2$.

We note that this polynomial formula for P can be derived by substituting the polynomial formulas (2.4), (2.5), and (2.2) for the functions S_1 , T_1 , and M into formula (2.3) and expanding the resulting

expression for $P(x)$ as a polynomial in x , taking into account that repeated squarings are automorphisms of $GF(2^n)$, and using the identity $x^{2^n} = x$ to reduce the degree of this polynomial to less than 2^n . We also note that the coefficients a_{ik} and b_{ik} in the polynomial formulas (2.4) and (2.5) for the functions S_1 and T_1 must be kept secret because a trapdoor sequence for P can be computed from them quite easily.

To decipher a message which has been enciphered using the enciphering permutation P , each ciphertext n -block y is replaced by the corresponding plaintext n -block x which is given by $x = P^{-1}(y)$, where P^{-1} is the inverse of the permutation P . To obtain a formula for the deciphering permutation P^{-1} , one must know a trapdoor sequence $a_1, \dots, a_4, b_1, \dots, b_4, e_1, e_2$ for P . The permutation P^{-1} is specified by this trapdoor sequence as follows. Let U_1 and U_2 be the $GF(2^{\gamma})$ -linear functions from $GF(2^{\delta})$ into $GF(2^n)$ such that $U_1(e_j) = b_j$ and $U_2(e_j) = b_{j+2}$, for $j = 1, 2$. Let V_1 and V_2 be the $GF(2^{\gamma})$ -linear functions from $GF(2^n)$ into $GF(2^{\delta})$ such that

$$V_1(a_j) = \begin{cases} e_j, & \text{for } j = 1, 2 \\ 0, & \text{for } j = 3, 4 \end{cases}$$

and

$$V_2(a_j) = \begin{cases} 0, & \text{for } j = 1, 2 \\ e_{j-2}, & \text{for } j = 3, 4. \end{cases}$$

Finally, let M^{-1} be the inverse of the permutation M of $GF(2^{\delta})$, which means that M^{-1} is given by

$$M^{-1}(y) = y^{\epsilon}, \quad (2.7)$$

where $\epsilon = 2^{\beta-1}(2^{2\beta} + 2^{\beta} - 1)$. Then the deciphering permutation P^{-1} is given by

$$P^{-1}(y) = U_1 M^{-1} V_1(y) + U_2 M^{-1} V_2(y). \quad (2.8)$$

Like the functions S_1 and T_1 , the functions U_1 and V_1 can be expressed as polynomial functions. The functions U_1 are given by the polynomial formulas

$$U_1(y) = c_{10}y + c_{11}y^{2^{\gamma}}, \quad (2.9)$$

where the coefficients c_{ik} are the elements of $GF(2^n)$ uniquely determined by the system of linear equations

$$c_{10}e_j + c_{11}e_j^{2^{\gamma k}} = U_1(e_j), \text{ for } j = 1, 2.$$

The functions V_1 are given by the polynomial formulas

$$V_1(y) = \sum_{k=0}^3 d_{ik} y^{2^{\gamma k}}, \quad (2.10)$$

where the coefficients d_{ik} are the elements of $GF(2^n)$ uniquely determined by the system of linear equations

$$\sum_{k=0}^3 d_{ik} a_j^{2^{\gamma k}} = V_1(a_j), \text{ for } j = 1, \dots, 4.$$

The coefficients c_{ik} and d_{ik} in the polynomial formulas (2.9) and (2.10) for the functions U_1 and V_1 can be regarded as a secret private key for the deciphering permutation P^{-1} .

$P^{-1}(y)$ can be computed for each $y \in GF(2^n)$ by using formula (2.8) together with the polynomial formulas (2.9), (2.10), and (2.7) for the functions U_1 , V_1 , and M^{-1} . An efficient way of doing this is based on the following formula:

$$\begin{aligned} M^{-1}V_1(y) &= V_1(y)^{2^{3\beta-1}} V_1(y)^{2^{2\beta-1}} / V_1(y)^{2^{\beta-1}} \\ &= \frac{\left(\sum_{k=0}^3 (d_{1,k-1})^{2^{3\beta-1}} y^{2^{\gamma k+\gamma-1}}\right) \left(\sum_{k=0}^3 (d_{1,k-1})^{2^{2\beta-1}} y^{2^{\gamma k+\alpha-1}}\right)}{\sum_{k=0}^3 (d_{1k})^{2^{\beta-1}} y^{2^{\gamma k+\beta-1}}}, \end{aligned}$$

where $d_{1,-1} = d_{1,3}$ and $\alpha = n/12$. To compute $P^{-1}(y)$ efficiently using this formula, first compute the quantities z_1 and z_2 given by $z_1 = M^{-1}V_1(y)$ by using the above formula and computing the powers of y of the form y^{2^k} appearing in this formula by doing k successive squarings. Then compute the quantities $U_1(z_1)$ by using the polynomial formulas (2.9) for the functions U_1 and again computing powers of the z_1 by repeated squaring. Finally, compute $P^{-1}(y)$ by adding $U_1(z_1)$ and $U_2(z_2)$. Computing $P^{-1}(y)$ this way requires a total of just $(3/2)n - 1$ squarings, 30 multiplications, 2 divisions, and 21 additions in $GF(2^n)$.

$P^{-1}(y)$ can be computed even more efficiently by making use of

matrix-vector multiplication. To compute $P^{-1}(y)$ this way, first compute the quantities t_1 , u_1 , and v_1 for $i = 1, 2$, where these quantities are given by $t_1 = V_1(y)^{2^{3\beta-1}}$, $u_1 = V_1(y)^{2^{\beta-1}}$, and $v_1 = V_1(y)^{2^{\beta-1}}$. Each of these quantities is a GF(2)-linear function of y , and so can be computed by doing a single matrix-vector multiplication over GF(2). Next, compute the quantities w_1 and w_2 given by $w_1 = M^{-1}V_1(y) = t_1 u_1 / v_1$. Then compute $U_1(w_1)$ and $U_2(w_2)$. For each i , the quantity $U_i(w_i)$ is a GF(2)-linear function of w_i , and so can be computed by doing a single matrix-vector multiplication over GF(2). Finally, compute $P^{-1}(y)$ by adding $U_1(w_1)$ and $U_2(w_2)$. Computing $P^{-1}(y)$ this way requires a total of just 8 matrix-vector multiplications over GF(2), together with 2 multiplications, 2 divisions, and 1 addition in $GF(2^n)$.

For the security of the cipher, the trapdoor sequences used should be such that all the coefficients p_{gh} , a_{ik} , b_{ik} , c_{ik} , and d_{ik} in the polynomial formulas (2.1), (2.4), (2.5), (2.9), and (2.10) for the functions P , S_1 , T_1 , U_1 , and V_1 are nonzero. It can be shown that, given any basis e_1, e_2 of $GF(2^\delta)$ over $GF(2^\gamma)$, if elements a_1, \dots, a_4 , b_1, \dots, b_4 are chosen at random from $GF(2^n)$, then it is virtually certain that a_1, \dots, a_4 and b_1, \dots, b_4 will both form bases of $GF(2^n)$ over $GF(2^\gamma)$ and that the sequence $a_1, \dots, a_4, b_1, \dots, b_4, e_1, e_2$ will form a trapdoor sequence that satisfies the security requirements just stated.

3. Invertibility of the enciphering and deciphering permutations

We now show that the enciphering and deciphering permutations given by formulas (2.3) and (2.8), respectively, are indeed mutually inverse permutations of $GF(2^n)$.

Since the invertibility of these functions depends on the invertibility of the function M given by formula (2.2), we first indicate why this function is a permutation of $GF(2^\delta)$ and why M^{-1} is given by formula (2.7). Using the Euclidean algorithm and the relation $\delta = 3\beta$,

it can be calculated that $\gcd(2^8 - 1, 2^\beta + 1) = 1$. Hence there exist numbers ϵ satisfying the congruence $(2^\beta + 1)\epsilon \equiv 1 \pmod{2^8 - 1}$. If ϵ is any positive solution of this congruence, then it follows from the identity $x^{2^8-1} = 1$, which is satisfied by all nonzero $x \in GF(2^8)$, that $M(x)^\epsilon = x^{(2^\beta+1)\epsilon} = x$ for all $x \in GF(2^8)$. Thus M is a permutation of $GF(2^8)$, and M^{-1} is given by $M^{-1}(y) = y^\epsilon$, where ϵ is any positive solution of the above congruence. It follows that M^{-1} is given by formula (2.7) provided that the number ϵ appearing in this formula satisfies the condition just given. The Euclidean algorithm calculations mentioned above can be used to find all the solutions of the congruence above. Of these solutions, the least positive one is exactly the number $\epsilon = 2^{\beta-1}(2^{2\beta} + 2^\beta - 1)$ appearing in formula (2.7). Thus M^{-1} is indeed given by formula (2.7).

Proposition. The enciphering function P given by formula (2.3) is a permutation of $GF(2^n)$ and the inverse of this permutation is the deciphering function given by formula (2.8).

Proof. Let Q denote the function on $GF(2^n)$ defined by formula (2.8). To prove the proposition, it suffices to show that $QP(x) = x$ for all $x \in GF(2^n)$. Let $a_1, \dots, a_4, b_1, \dots, b_4, e_1, e_2$ be a trap-door sequence for P that specifies the $GF(2^Y)$ -linear functions S_1, T_1, U_1 , and V_1 appearing in formulas (2.3) and (2.8). Let X_1 and X_2 be the $GF(2^Y)$ -subspaces of $GF(2^n)$ spanned by b_1, b_2 and by b_3, b_4 , respectively, and let Y_1 and Y_2 be the $GF(2^Y)$ -subspaces of $GF(2^n)$ spanned by a_1, a_2 and by a_3, a_4 , respectively. Then $GF(2^n) = X_1 \oplus X_2 = Y_1 \oplus Y_2$. Now suppose that $x \in GF(2^n)$ is given, and let x_1 and x_2 be the unique elements of X_1 and X_2 , respectively, such that $x = x_1 + x_2$. Then, for $i = 1, 2$,

$$T_1(x) = T_1(x_1 + x_2) = T_1(x_1) + T_1(x_2) = T_1(x_1),$$

where the last equality holds because $T_1(x_2) = T_2(x_1) = 0$ by the definition of the functions T_1 . Also T_1 maps X_1 one-to-one onto $GF(2^S)$,

M is a permutation of $GF(2^{\delta})$, and S_1 maps $GF(2^{\delta})$ one-to-one onto Y_1 , so S_1MT_1 maps X_1 one-to-one onto Y_1 . Thus, letting $y_1 = S_1MT_1(x_1)$, we have $P(x) = y_1 + y_2$, with $y_1 \in Y_1$. Next, to compute $QP(x)$, we note that, for $i = 1, 2$,

$$V_1P(x) = V_1(y_1 + y_2) = V_1(y_1) + V_1(y_2) = V_1(y_1),$$

where the last equality holds because $V_1(Y_2) = V_2(Y_1) = 0$ by the definition of the functions V_i . Hence

$$\begin{aligned} QP(x) &= U_1M^{-1}V_1(y_1) + U_2M^{-1}V_2(y_2) \\ &= U_1M^{-1}V_1S_1MT_1(x_1) + U_2M^{-1}V_2S_2MT_2(x_2). \end{aligned}$$

Also both V_1S_1 and $M^{-1}M$ are the identity map on $GF(2^{\delta})$, and U_1T_1 is the identity map on X_1 , so $U_1M^{-1}V_1S_1MT_1(x_1) = x_1$. Hence, for all $x \in GF(2^n)$, $QP(x) = x_1 + x_2 = x$. Thus P is a permutation of $GF(2^n)$, and $P^{-1} = Q$. Q.E.D.

4. Cryptanalysis by solving the equation $P(x) = y$

In this section and the next two sections, we describe some possible methods of cryptanalyzing our cipher by using public information about the enciphering permutation. For each method that we consider, we give an estimate of the amount of computation needed.

The first cryptanalytic attack that we consider consists of solving a given ciphertext message, enciphered using a known enciphering permutation P , by solving the equation $P(x) = y$ for each ciphertext n -block y to find the corresponding plaintext n -block x . We consider two methods of solving the equation $P(x) = y$. The first method is an exhaustive search procedure, while the second method is algebraic in nature.

The exhaustive search procedure that we consider for solving the equation $P(x) = y$ depends on the easily proved identity $P(wz) = M(w)P(z)$, which holds for all $w \in GF(2^Y)$ and $z \in GF(2^n)$. In view of this identity, if a nonzero $z \in GF(2^n)$ can be found such that $y/P(z) \in GF(2^Y)$, then the desired n -block x such that $P(x) = y$ is given

by $x = M^{-1}(y/P(z))z$. A nonzero $z \in GF(2^n)$ has the property just described if and only if $(y/P(z))^{2^r} = y/P(z)$. Such an element z can be found by an exhaustive search in which elements of $GF(2^n)$ are tested one-by-one until one is found that satisfies this last condition. A minimal subset of $GF(2^n)$ that is certain to contain an element z of the desired kind contains exactly one element of each different subset of $GF(2^n)$ of the form $\{wt: w \in GF(2^r), w \neq 0\}$, where t is a nonzero element of $GF(2^n)$. There are approximately $2^{(3/4)n}$ such subsets of $GF(2^n)$, so the desired element z can be found after at most $2^{(3/4)n}$ trials. We will regard each trial needed to find this element z as a single operation. Then it follows that at most approximately $2^{(3/4)n}$ operations are required to solve the equation $P(x) = y$ by the exhaustive search procedure just described.

The second method that we consider for solving the equation $P(x) = y$ is to regard this equation as a polynomial equation in x and to solve this equation algebraically. It appears that the most efficient way of doing this is to use the Euclidean algorithm to compute the polynomial in x which is the greatest common divisor of the polynomials $P(x) - y$ and $x^{2^n} - x$. To see what this accomplishes, note that, since P is a permutation of $GF(2^n)$, the polynomial $P(x) - y$ has a unique root $x = r$ in $GF(2^n)$, and hence has a unique linear factor $x - r$ over $GF(2^n)$. On the other hand, the polynomial $x^{2^n} - x$ is the product of all the linear factors $x - a$, with $a \in GF(2^n)$. Hence the greatest common divisor of $P(x) - y$ and $x^{2^n} - x$ is exactly the linear factor $x - r$ such that $x = r$ is the desired solution of the equation $P(x) = y$. Thus to solve the equation $P(x) = y$, it is only necessary to compute this greatest common divisor. Using the Euclidean algorithm to do this, the required number of multiplications and divisions in $GF(2^n)$ is at most approximately $(\deg(P))^2/2$. Thus we conclude that the equation $P(x) = y$ can be solved algebraically using the method just described by doing at most approximately $2^{(11/6)n-1}$ operations.

5. Cryptanalysis by determining a polynomial or rational formula for P^{-1}

Next, we consider a method of cryptanalyzing the cipher that consists of determining a formula for the deciphering permutation P^{-1} by using public information about the enciphering permutation P . We describe two formulas for P^{-1} that can be determined this way. The first formula expresses P^{-1} as a polynomial function, while the second formula expresses P^{-1} as a rational function, that is, as a quotient of two polynomial functions. We describe how each of these formulas can be obtained and we give estimates of the amounts of computation needed to do this.

First, we describe how a polynomial formula for P^{-1} can be obtained. It can be shown that P^{-1} can be expressed as a polynomial function of the form

$$P^{-1}(y) = \sum_{k \in K} w_k y^k,$$

where the coefficients w_k are elements of $GF(2^n)$, the index set K is a subset of the set $\{0, \dots, 2^n - 1\}$ which can be completely specified, and the number of elements in the set K satisfies $2^{n/3} \leq |K| \leq 2^{n/3+2}$. This formula for P^{-1} can be regarded as a system of 2^n linear equations which uniquely determines the coefficients w_k in the formula. By making the substitution $y = P(x)$ in this formula, an equivalent system of 2^n linear equations can be obtained which have the form

$$\sum_{k \in K} w_k P(x)^k = x.$$

Note that this second system of equations can be formulated using only public information about the enciphering permutation P . Since the rank of this second system is the same as the rank of the original system, which is $|K|$, and since $|K| < 2^n$, it follows that this second system can be reduced to a smaller system formed from it by choosing any subset of $|K|$ independent equations. We will assume that such a smaller system can be obtained without any significant computational effort, which may well be the case. Then the determination of the

coefficients w_k in the polynomial formula for P^{-1} reduces to solving this smaller system of equations. This system consists of $|K|$ equations in $|K|$ unknowns, so to solve it requires at most approximately $|K|^3/3$ operations consisting of multiplications and divisions in $GF(2^n)$. Hence, since $|K| \geq 2^{n/3}$, we conclude that it takes at most approximately $(2^n)/3$ operations to solve for the coefficients w_k , and thus to determine a polynomial formula for P^{-1} .

Next, we describe how a rational formula for P^{-1} can be obtained. The rational formula that we consider has the same form as the rational formula for P^{-1} that is obtained by expanding formula (2.8) for $P^{-1}(y)$ as a rational function of y , making use of the polynomial formulas (2.9) and (2.10) for the functions U_1 and V_1 described in section 2, and expressing the function M^{-1} by the rational formula $M^{-1}(y) = y/y$, where $\zeta = 2^{\beta-1}(2^{2\beta} + 2^\beta)$ and $\eta = 2^{\beta-1}$. The rational formula for P^{-1} just described has the form $P^{-1}(y) = Q(y)/R(y)$, where Q and R are both nonconstant polynomial functions, $Q(0) = 0$, and $R(y) \neq 0$ for all non-zero $y \in GF(2^n)$. Furthermore, it can be shown that Q and R are given by polynomial formulas having the forms

$$Q(y) = \sum_{k \in K_Q} w_Q(k)y^k$$

and

$$R(y) = \sum_{k \in K_R} w_R(k)y^k,$$

where the coefficients $w_Q(k)$ and $w_R(k)$ are elements of $GF(2^n)$, the index sets K_Q and K_R are subsets of the set $\{0, \dots, 2^n - 1\}$ which can be completely specified, and the numbers of elements in the sets K_Q and K_R satisfy $2^{n/3} \leq |K_Q| \leq 2^{n/3+3} + 64$ and $4 \leq |K_R| \leq 16$. Now if the formula $P^{-1}(y) = Q(y)/R(y)$ is rewritten as $P^{-1}(y)R(y) - Q(y) = 0$, if the substitution $y = P(x)$ is made, and if the above polynomial formulas for the functions Q and R are used, then the result is the equation

$$\sum_{k \in K_R} xw_R(k)P(x)^k - \sum_{k \in K_Q} w_Q(k)P(x)^k = 0$$

which holds for all $x \in GF(2^n)$. This equation can be regarded as a system of 2^n homogeneous linear equations that are satisfied by the elements $w_Q(k)$ and $w_R(k)$ and that can be formulated using only public information about the enciphering permutation P . Conversely, if a set of elements $w_Q(k)$ and $w_R(k)$ of $GF(2^n)$ forms a nonzero solution of this system of equations and if the functions Q and R on $GF(2^n)$ are defined by the polynomial formulas given above, then the function R is not identically zero and P^{-1} is given by the rational formula $P^{-1}(y) = Q(y)/R(y)$ for all $y \in GF(2^n)$ such that $R(y) \neq 0$. Thus a rational formula for P^{-1} can be obtained by finding a nonzero solution of the system of linear equations given above, and furthermore such solutions exist.

Since the rank of this system of 2^n equations is at most $|K_Q| + |K_R| - 1$, which is less than 2^n , this system can be reduced to a smaller system which has the same rank and consists of equations chosen from the original system. We will assume that such a smaller system consisting of $|K_Q| + |K_R| - 1$ equations can be obtained from the original system without any significant computational effort. Then the determination of the coefficients $w_Q(k)$ and $w_R(k)$ in a rational formula for P^{-1} reduces to solving this smaller system of $|K_Q| + |K_R| - 1$ linear equations in $|K_Q| + |K_R|$ unknowns, which takes at most approximately $(|K_Q| + |K_R|)^3/3$ operations. Hence, since $|K_Q| + |K_R| > 2^{n/3}$, we conclude that it takes at most approximately $(2^n)/3$ operations to determine a rational formula for P^{-1} of the kind described above.

6. Cryptanalysis by finding a trapdoor sequence

The last method of cryptanalysis that we consider consists of using the public key for a given enciphering permutation P to determine a trapdoor sequence for it. We consider two ways of finding such a sequence: first by exhaustive search, and second by solving the

system of equations (2.6) algebraically. We describe how each of these approaches might be carried out and we give estimates of the amounts of computation required.

The most efficient exhaustive search procedure for finding a trapdoor sequence for P appears to be as follows. First, choose the elements e_1, e_2 of the sequence to be any convenient basis of $GF(2^\delta)$ over $GF(2^r)$. Next, test one-by-one bases b_1, \dots, b_4 of $GF(2^n)$ over $GF(2^r)$ until a basis is found which is the b_1, \dots, b_4 part of a trapdoor sequence for P whose e_1, e_2 elements are the ones just chosen. To test a given basis b_1, \dots, b_4 for this property, let the $GF(2^r)$ -linear functions T_1 and T_2 be defined in terms of $b_1, \dots, b_4, e_1, e_2$ as described in section 2, and solve for the coefficients a_{ik} in the polynomial formulas for these functions given by equation (2.5). Next, find all the solutions for the elements a_{ik} in the system of equations (2.6). Note that these solutions can be found by linear algebra, since this system is linear in the a_{ik} . The solutions, if any, of this system are then tested one-by-one to determine whether any of them is such that $GF(2^n)$ can be expressed as $GF(2^n) = S_1(GF(2^\delta)) + S_2(GF(2^\delta))$, where S_1 and S_2 are the $GF(2^r)$ -linear functions from $GF(2^\delta)$ into $GF(2^n)$ defined in terms of the elements a_{ik} by formula (2.3). Now the basis b_1, \dots, b_4 , which is being tested for the property of being the b_1, \dots, b_4 part of a trapdoor sequence for P whose e_1, e_2 elements are the ones chosen, has this property if and only if there exists a set of elements a_{ik} that satisfies the system of equations (2.5) and that satisfies the condition stated above. As soon as such a basis b_1, \dots, b_4 and a set of elements a_{ik} has been found, a complete trapdoor sequence for P can be produced. The $b_1, \dots, b_4, e_1, e_2$ part has already been obtained, and the a_1, \dots, a_4 part of the sequence is given by $a_j = S_1(e_j)$, for $j = 1, 2$, and by $a_j = S_2(e_{j-2})$, for $j = 3, 4$, where the functions S_i are as described above.

A minimal set of bases b_1, \dots, b_4 that is certain to contain a

basis of the desired kind includes, for each different enciphering permutation, exactly one basis that is the b_1, \dots, b_4 part of a trapdoor sequence for the permutation whose e_1, e_2 elements are the ones chosen. It can be shown that such a set of bases contains approximately 2^{3n-3} bases, so at most approximately 2^{3n-3} trials are required to find a trapdoor sequence for P by the exhaustive search procedure described above. It appears likely that, for each basis b_1, \dots, b_4 tested, either there is no solution at all for the elements a_{ik} , or else the basis is the b_1, \dots, b_4 part of a trapdoor sequence for P of the desired kind and there is only one solution for the elements a_{ik} . In view of this, we will consider the testing of a single basis as being a single operation. Thus we conclude that at most approximately 2^{3n-3} operations are required to find a trapdoor sequence for P by the exhaustive search procedure described above.

Finally, we consider finding a trapdoor sequence for a given enciphering permutation P by solving algebraically for a set of elements a_{ik} and b_{ik} of $GF(2^n)$ satisfying the system of equations (2.6). First, we note the connection between solutions of this system of equations and trapdoor sequences for P . If a set of elements a_{ik} and b_{ik} of $GF(2^n)$ satisfies this system of equations and if $GF(2^{\gamma})$ -linear functions S_i and T_i from $GF(2^n)$ into $GF(2^n)$ are defined in terms of these elements by equations (2.4) and (2.5), respectively, then P can be expressed in terms of these functions by equation (2.3). Furthermore, there exists a trapdoor sequence for P which specifies these functions if and only if these functions satisfy the conditions

$$GF(2^n) = S_1(GF(2^{\delta})) \oplus S_2(GF(2^{\delta})) = \ker(T_1) \oplus \ker(T_2)$$

and $GF(2^{\delta}) = \text{range}(T_1) = \text{range}(T_2)$. If the functions S_1 and T_1 satisfy these conditions and if e_1, e_2 is any basis of $GF(2^{\delta})$ over $GF(2^{\gamma})$, then a trapdoor sequence for P which specifies these functions is given by $a_1, \dots, a_4, b_1, \dots, b_4, e_1, e_2$, where, for $j = 1, 2$, $a_j = S_1(e_j)$, and, for $j = 3, 4$, $a_j = S_2(e_{j-2})$, and where, for $j = 1, 2$, b_j is the

unique element of $\ker(T_2)$ satisfying $T_1(b_j) = e_j$, and, for $j = 3, 4$, b_j is the unique element of $\ker(T_1)$ satisfying $T_2(b_j) = e_{j-2}$. It follows that the system of equations (2.6) has many solutions for the elements a_{ik} and b_{ik} , since there is a different solution arising from each different trapdoor sequence for P having fixed e_1, e_2 elements, and there are perhaps other solutions as well that do not arise from any trapdoor sequence for P . We will assume that all solutions for the elements a_{ik} and b_{ik} do in fact arise from trapdoor sequences for P . Then, to find a trapdoor sequence for P , it suffices to find a single solution of the system of equations (2.6) for the elements a_{ik} and b_{ik} .

In order to estimate the amount of computation required to solve this system of equations algebraically, it is first necessary to determine the most efficient method of algebraic solution. As already noted, this system of equations is linear in the elements a_{ik} . Hence it appears that the most efficient way to solve this system is to first simplify it as much as possible by eliminating these unknowns. This is exactly the method that was used by Berkovits and by James, Lidl, and Niederreiter to solve the corresponding system of equations associated with the original version of our cipher. It was in this way that they broke that cipher.

For the system of equations (2.6), there are many possible ways in which the unknowns a_{ik} can be eliminated, and each of these ways must be tried in order to find the best way of simplifying the system. Unfortunately, to try all these ways would require a forbidding amount of computation, although it could probably be done fairly easily using a suitable computer algebra system. To get around these difficulties in analyzing this system of equations, we consider instead a different system of equations that presumably requires less computation to solve. This system of equations is associated with a class of permutations of $GF(2^n)$ that are somewhat simpler than the enciphering permutations used

in our cipher but which have the same general structure. These simpler permutations are obtained by modifying the enciphering permutation construction described in section 2 by changing the relationship between δ and γ from $\delta = 2\gamma$ to $\delta = \gamma$. The effect of this change is to convert the polynomial formulas (2.4) and (2.5) for the functions S_1 and T_1 from 2 terms to 1 term and from 4 terms to 2 terms, respectively. The resulting permutation P is then given by a polynomial formula having just 4 terms, rather than 16 terms as in our cipher. The system of equations that corresponds to the system of equations (2.6) and that relates the polynomial coefficients p_{gh} of P to the polynomial coefficients a_{ik} and b_{ik} of the functions S_1 and T_1 has the form

$$p_{gh} = a_{10} b_{1g}^{2^\beta} b_{1h} + a_{20} b_{2g}^{2^\beta} b_{2h}, \text{ for } g, h = 0, 1.$$

Now we consider how this system of equations can be solved. Note that, like the more complicated system of equations (2.6), the above system of equations is linear in the unknowns a_{10} and a_{20} . Hence it appears that the most efficient way to solve this system is to first simplify it as much as possible by eliminating these unknowns. Of the various ways to do this, the best way appears to be one that leads fairly directly to a single polynomial equation $R(B_1) = 0$ of degree $2^{2^\beta} + 1$ in the single unknown $B_1 = b_{10}/b_{11}$. It appears that the amount of computation required to solve this equation is at least the amount required to compute the greatest common divisor of the polynomials $R(B_1)$ and $B_1^{2^n} - B_1$. This requires approximately $\deg(R(B_1))^2/2$ operations, which is approximately $2^{(2/3)n-1}$ operations. We will take this amount as our estimate of the amount of computation required to find a trapdoor sequence by solving the system of equations (2.6) algebraically.

An obvious question now arises. Since the estimate just given is based solely on the properties of the corresponding system of equations for the simpler permutations described above, why not use these simpler permutations as enciphering permutations? Unfortunately, this cannot

be done. The reason for this is that, for such enciphering permutations, the deciphering permutations can be expressed by a rational formula corresponding to the rational formula described in section 5 for the deciphering permutations used in our cipher, and there are at most 12 terms in this formula. Thus, as indicated in section 5, the coefficients in this formula can be determined by doing at most approximately $12^{3/2}$ operations. This number of operations is far too small to provide any security, and hence the simpler permutations described above cannot be used as enciphering permutations.

7. Summary of the cryptanalytic attacks and conclusions

The following table summarizes the estimates of the amounts of computation required by the various cryptanalytic attacks discussed in sections 4 - 6.

<u>method of attack</u>	<u>maximum number of operations required</u>
1. solving the equation $P(x) = y$:	
a. by exhaustive search	$2^{(3/4)n}$
b. algebraically	$2^{(11/6)n-1}$
2. finding a formula for P^{-1} :	
a. polynomial	$(2^n)/3$
b. rational	$(2^n)/3$
3. finding a trapdoor sequence:	
a. by exhaustive search	2^{3n-3}
b. algebraically	$2^{(2/3)n-1}$

According to the above table, the most effective attack against our cipher is to solve algebraically for a trapdoor sequence for the enciphering permutation. This attack is estimated to require at most $2^{(2/3)n-1}$ operations, so the block-length n of the cipher must be chosen so that this amount of computation is unfeasible. We will

assume, somewhat arbitrarily, that the maximum feasible amount of computation is the number of operations performed by a computer that does 10^9 operations per second for a period of 10 years. This amounts to a total of 3×10^{17} operations. We multiply this by a safety factor of 10^{12} to arrive at the figure of 3×10^{29} operations as an unfeasible amount of computation. Hence the block-length n must be such that $2^{(2/3)n-1} \geq 3 \times 10^{29} \approx 2^{98}$. Thus we conclude that a suitable block-length for our cipher is $n \geq 150$.

References

1. Shimshon Berkovits (Uni. of Lowell, Dept. of Computer Science), private communication, Aug., 1985.
2. John J. Cade, A new public-key cipher which allows signatures, talk given at the Second S.I.A.M. Conference on Applied Linear Algebra, Raleigh, NC, Apr. 30 - May 2, 1985.
3. N. S. James, R. Lidl, and H. Niederreiter, Breaking the Cade cipher, preprint, 1986.

A Pseudo-Random Bit Generator Based on Elliptic Logarithms¹

Burton S. Kaliski, Jr.
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139

Abstract

Recent research in cryptography has led to the construction of several *pseudo-random bit generators*, programs producing bits as hard to predict as solving a hard problem. In this paper,

1. We present a new pseudo-random bit generator based on *elliptic curves*.
2. To construct our generator, we also develop two techniques that are of independent interest:
 - (a) an algorithm that computes the order of an element in an arbitrary Abelian group; and
 - (b) a new oracle proof method for demonstrating the simultaneous security of multiple bits of a discrete logarithm in an arbitrary Abelian group.
3. We present a new candidate hard problem for future use in cryptography: the *elliptic logarithm problem*.

1 Introduction

This paper describes a method for producing pseudo-random bits based on the elliptic logarithm problem. The paper contains background on elliptic curves and pseudo-random bit generation, two new results of independent interest, and the construction and proof of a pseudo-random bit generator. This section gives an overview of the paper.

1.1 Motivation and overview

Recently considerable progress has been made in formalizing the theory of pseudo-random number generation based on computational difficulty [BM84] [Yao82] [GGM84] [Lev85]. However, the generality of this theory (finally based on one-way functions in a weak sense [Lev85]) is in sharp contrast with the very few concrete candidates for one-way functions. Discrete logarithm and integer factorization (of which quadratic residuosity and inverting RSA are special cases) are essentially the only hard problems on which to build one-way functions. One of the main contributions of this paper is that it introduces a new hard problem *different than those previously studied*. Since cryptography stimulates mathematical research, it is interesting to note that ours is one of the first cryptographic tools based on 20th century mathematics.

In simplest form, an *elliptic curve* is the set of solutions (x, y) to an equation

$$y^2 = x^3 + Ax + B \quad (1)$$

over the finite field with p elements, where p is a prime. A well-known result is that the *points on an elliptic curve form an abelian group* under an additive composition operation called "tangents and chords." We use the group structure to apply the main ideas of the Blum-Micali generator in an entirely new context.

In the Blum-Micali case, the hard problem is "discrete logarithms" modulo p : given $g, y \in \mathbb{Z}_p^*$, find a such that $y \equiv g^a$ modulo p . In our case, the hard problem is "elliptic logarithms" on an elliptic curve modulo p : given points G, Y , find a such that $Y = aG$.

Despite the similarity of the statements and of the names, we are dealing with two very distinct problems. First, the structure of elliptic curve groups differs greatly from those groups previously studied. Second, the representation differs: points on elliptic curves require two coordinates. Third, while there

¹This research was supported by NSF grant MCS-8006938.

are closed formulas for computing the order of \mathbb{Z}_p^* and other groups, there are no such formulas for elliptic curves. To summarize, elliptic logarithms involve entirely different mathematics. (They are also conjectured to be harder to compute than discrete logarithms [Mil85a].)

To construct a pseudo-random bit generator based on elliptic curves, and to prove that the bits it outputs are as hard to predict as solving the elliptic logarithm problem, is not a straightforward generalization of previous work. The differences pointed out above make previous constructions and proofs inadequate. In developing our construction and proofs, we also develop several related results.

1. We construct a novel method of finding the order of an element of an Abelian group.
2. We also introduce a new proof technique that generalizes proofs of bit security to abelian groups of arbitrary structure.
3. Furthermore, we lay the foundation for the development of cryptosystems using elliptic curves directly.

We make use of Lenstra's new factoring algorithm based on elliptic curves [Len85] [Bac85] (which, at first glance, would seem more suitable to *break* cryptosystems than to *construct* them!). Lenstra's algorithm has the remarkable property that its running time depends on the size of the *smallest prime factor* of its input. This allows us, for instance, to find elements that generate an abelian group quickly with negligible probability of error. This solves a major problem encountered in earlier constructions of pseudo-random bit generators. We believe ours is the first cryptographic application that exploits the special features of Lenstra's algorithm.

2 Background

2.1 Number theory

Let $\varphi(N)$ be the *Euler totient function*, the number of integers N or smaller relatively prime to N . Rosser and Schoenfeld [RS62] give a lower bound for $N > 3$ of

$$\frac{\varphi(N)}{N} \geq \frac{1}{6 \ln \ln N}. \quad (2)$$

2.2 Groups

A group G is *additive* if its composition operation is written $+$; multiplicative if written \times (or implied). *Abelian* is a synonym for commutative. In an additive group, we write ax to denote repeated composition of x with itself a times; in a multiplicative group, x^a . Given an element x in G , we say the *order* of x , written $\text{order}_G(x)$, is the smallest positive integer a such that $ax = 0$, where 0 is the identity element of G .

Definition 1 Let G be an additive Abelian group. A *generating set* for G is a set

$$\{(g_1, N_1), \dots, (g_k, N_k)\}, \quad g_i \in G, N_i > 1, \quad (3)$$

such that every element $x \in G$ can be written uniquely as

$$x = a_1 g_1 + \dots + a_k g_k, \quad 0 \leq a_i < N_i. \quad (4)$$

We say a generating set is *canonical* if N_{i+1} divides N_i for $1 \leq i < k$. Every Abelian group has canonical generating sets. Furthermore, the sequence N_1, \dots, N_k is the same for all canonical generating sets. The *rank* of G is the cardinality of its canonical generating sets. Thus we may speak of the r -tuple (g_1, \dots, g_r) , where r is the rank, as a *generating tuple* for G . If the rank is 2, then we have a *generating pair* (g_1, g_2) ; and if the rank is 1, then we have simply a *generator* g .

Also observe the isomorphism

$$G \cong \mathbb{Z}_{N_1} \times \dots \times \mathbb{Z}_{N_r}. \quad (5)$$

To every $x \in G$ there corresponds a unique r -tuple. We call this the *index tuple*, and it is defined as

$$\underset{G, g_1, \dots, g_r}{\text{index}}(x) = (a_1, \dots, a_r) \iff x = a_1 g_1 + \dots + a_r g_r. \quad (6)$$

For groups with rank 2 and 1, we have the *index pair* and *index*. In any group, we also write

$$\underset{G,y}{\text{index}}(x) = a \iff x = ay, \quad (7)$$

for arbitrary y , but this may not be defined for all x .

The following are results of the lower bound on $\varphi(N)$ (equation 2).

Lemma 2 The probability that an element x has maximum order in G is at least

$$\frac{1}{6 \ln \ln N_1}, \quad (8)$$

where $N_1 > 3$ is the maximum order.

Lemma 3 Let G be a group with N elements, and let r be its rank. Then the probability that (x_1, \dots, x_r) is a generating tuple is at least

$$\frac{1}{6^r (\ln \ln N)^r} \quad (9)$$

assuming each $N_i > 3$.

3 Elliptic curves

Elliptic curves, like many other topics in number theory and algebraic geometry, enjoy a rich history as well as recent applications to computer science. As Cassels writes, “It has exercised a fascination throughout the centuries and the number of isolated results is immense” [Cas66]. The study of elliptic curves has led to a solution of the Congruence problem [Kob84], and a “Riemann hypothesis” [Cho65] [Jol73]. More recently, Lenstra has proposed a novel factoring algorithm using a group law that relates the points of an elliptic curve [Bac85] [Len85]. This same group law is the basis for Miller’s “elliptic logarithm” adaptation [Mil85a] of the Diffie-Hellman key exchange protocol [DH76], and for the primality certification of Goldwasser and Kilian [GK86].

3.1 Definition and notation

In simplest form, an *elliptic curve* is the set of (x, y) solutions over a field K to the equation

$$E : y^2 = x^3 + Ax + B, \quad (10)$$

where $A, B \in K$, $(x, y) \in K^2$, and $4A^3 + 27B^2 \neq 0$. Most elliptic curves can be expressed this way, called *Weierstrass form*. Much study in this century has been devoted to elliptic curves over the fields C , R , Q , Z [Cas66] [Cho65] [Ful69]. Also of interest are elliptic curves over finite fields and their algebraic closures, F_q and \overline{F}_q . [BMP75] [Sch85] [Tat74].

An elliptic curve may also be defined in projective coordinates, as

$$E : y^2 z = x^3 + A x z^2 + B z^3, \quad (11)$$

where $A, B \in K$ and the discriminant

$$\Delta = 4A^3 + 27B^2 \neq 0 \quad (12)$$

in K .

The point $(x, y, 1)$ in projective coordinates corresponds to the point (x, y) in affine coordinates, where 1 is the unit of the field K . The point $(0, 1, 0)$ corresponds to a *point at infinity* on the elliptic curve in affine coordinates.

Let $E(K)$ denote the set of solutions of the curve E over the field K , together with the point at infinity, denoted 0. A well-known result is that $E(K)$ is an abelian group under a composition operation called “tangents and chords.” The description of this operation is easiest for $E(R)$. Any line in R^2 intersects the curve E in either zero or three points. (A point of tangency is counted twice, the third point of intersection for a vertical line is considered 0.) The composition of points P and Q , written $P + Q$, is the reflection of the third point colinear with them. Thus 0 is the identity. Figure 1 illustrates this operation.

Most of the group axioms are easily verified; to show $E(R)$ is associative requires certain theorems of algebraic geometry [Ful69]. Since the composition operation can be expressed as a rational polynomial function, it can be generalized from R to any field. We will assume for analysis of algorithms that we can compose points on an elliptic curve $E(F_p)$ in time $O(n^2)$, where $n = \log p$.

3.2 Group structure

Lemma 4 $E(\mathbb{F}_p)$ has rank 2.

Proof. This follows from a morphism with \mathbf{C}/L , where \mathbf{C} is the complex plane and L is a lattice. Since two generators are sufficient for the lattice, two are sufficient for the elliptic curve. ■

Lemma 5 If $E(\mathbb{F}_p) \cong \mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2}$, where N_2 divides N_1 , then N_2 divides $p - 1$.

Proof. This is a fairly deep result. See, for example, Cassel's survey [Cas66]. ■

Lemma 6 The only points of order 2 on an elliptic curve are 0 and those with y -coordinate 0.

Proof. By definition of composition, if $P = (x, y)$ then $-P = (x, -y)$, since they lie on the same vertical line. Points of order 2 are self-inverse, and thus $P = 0$ and $P = (x, 0)$ are the only solutions. ■

3.3 Simple case

Definition 7 The *simple case* of elliptic curves consists of those curves over a finite field \mathbb{F}_p ,

$$E : y^2 = x^3 + B \tag{13}$$

for which $p \equiv 2$ modulo 3 and $B \neq 0$.

We show several useful properties.

Lemma 8 Then to every y -coordinate in \mathbb{F}_p there corresponds exactly one point on an elliptic curve $E(\mathbb{F}_p)$ in the simple case.

Proof. For any $y \in \mathbb{F}_p$, the point $(\sqrt[3]{y^2 - B}, y)$ is on the curve. Since $p \equiv 2 \pmod{3}$, cube roots are unique and therefore there is exactly one point for each y . ■

Corollary. $N_p = p + 1$.

Lemma 9 Let $E(\mathbb{F}_p)$ be an elliptic curve in the simple case. Then $E(\mathbb{F}_p)$ is cyclic.

Proof. By lemmas 4 and 5, $E(\mathbb{F}_p) \cong \mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2}$, where $N_1 N_2 = p + 1$ and N_2 divides $p - 1$. Thus N_2 must be either 1 or 2. But $E(\mathbb{F}_p)$ has only two points of order 2, $(\sqrt[3]{-B}, 0)$ and 0, so N_2 must be 1 and the group is cyclic. ■

4 Pseudo-random number generators

Recent research in computational complexity has led to the notion of a *cryptographically strong pseudo-random bit generator*. Yao formalized this notion in terms in information theory [Yao82], and Blum and Micali gave sufficient conditions for constructing a generator, together with a concrete example using discrete logarithms [BM84]. Later, direct constructions were obtained for generators based on the RSA cryptosystem [ACGS] and the quadratic residuosity problem [BBS83]. Levin made research more formal with his study of weaker sufficient conditions and necessary conditions [Lev85].

A pseudo-random bit generator is interesting in at least two ways. First, it provides a source of randomness indistinguishable in polynomial time from a truly random source, and therefore it can be used reliably in probabilistic algorithms. In fact, Yao shows that the existence of such a generator implies that any randomized polynomial-time algorithm can be simulated by a deterministic sub-exponential-time algorithm [Yao82]. Ajtai and Wigderson generalize these results to probabilistic constant-depth circuits [AW85]. Second, a generator can be used both in public- and private-key cryptosystems; in the latter case, it is the polynomial-time equivalent of the “one-time pad,” an ideal, provably secure cryptosystem.

4.1 Sources

We borrow the following definitions, slightly modified, from Yao's paper. We assume that probability distributions are uniform, and therefore refer to sources simply as sets of strings.

Definition 10 A source S is a set of strings of equal length. A source ensemble \mathcal{S} is a sequence of sources S_1, S_2, \dots . If $\xi(n)$ is the length of the strings in S_n and $\xi(n) < \xi(n+1)$, then the source ensemble is called uniform.

Definition 11 The truly random source ensemble $\mathcal{R} = R_1, R_2, \dots$, where each R_n is the set of all strings of length n .

4.2 Statistical tests

Definition 12 A statistical test is a probabilistic algorithm that takes as input a string and outputs either 0 or 1.

Let T be a statistical test, and let $S = S_1, S_2, \dots$ be a source ensemble. We say that S passes the statistical test T if for all polynomials $Q(n)$, for n sufficiently large,

$$|\Pr[T(x) = 1 | x \in S_n] - \Pr[T(x) = 1 | x \in R_n]| < \frac{1}{Q(n)}. \quad (14)$$

We say that S passes all polynomial-time statistical tests if it passes all such T that run in time polynomial in the lengths of their input.

4.3 Approximation

Definition 13 Let I be a set of strings, let $b: I \rightarrow \{0, 1\}$ be a predicate, and let $\epsilon: \mathbb{N} \rightarrow [0, 1/2]$ be a function. We say an algorithm $b_\epsilon: I \rightarrow \{0, 1\}$ ϵ -approximates b if for each n ,

$$\Pr[b(x) = b_\epsilon(x)] \geq 1/2 + \epsilon(n), \quad (15)$$

where the probability is taken over coin flips in b_ϵ and x of length n in I . Such an algorithm is called an ϵ -approximator for b .

Let $T_Q(n)$ be the running time of any algorithm that $1/Q(n)$ -approximates b on inputs of length n . We say a b is unapproximable if for all polynomials $Q(n)$, $T_Q(n)$ grows faster than any polynomial in n .

4.4 Sufficient conditions

Definition 14 Let I be a set of strings, and let $b: I \rightarrow \{0, 1\}$ and $f: I \rightarrow I$ be functions. Consider the source ensemble \mathcal{S} where S_n is the set of strings

$$b(f^i(s)) \circ b(f^{i-1}(s)) \circ \dots \circ b(f(s)), \quad (16)$$

where $s \in I$ of length n is chosen uniformly at random. If \mathcal{S} is uniform, polynomial, and passes all polynomial-time statistical tests, then then $\langle I, f, b \rangle$ is called a Blum-Micali pseudo-random bit generator.

Theorem 15 Let I be a set of strings, and let $b: I \rightarrow \{0, 1\}$ and $f: I \rightarrow I$ be functions. Also let I_n be the subset of I containing strings of length n . Then $\langle I, f, b \rangle$ is a Blum-Micali pseudo-random bit generator, if

1. (friendship) $b(f(x))$ is computable in time polynomial in $|x|$.
2. (stability) f is a permutation, $|f(x)| = |x|$, and $f(x)$ is computable in time polynomial in $|x|$.
3. (accessibility) There is an algorithm that, given an integer n , selects elements $x \in I_n$ uniformly (if any exist) in time polynomial in $|x|$.
4. (unapproximability) b is unapproximable.

Remark. Yao and Levin propose conditions less strict than these. In particular, Yao replaces I with a sequence of probability distributions. Levin shows that f need not be a permutation.

One condition which we will make less strict is accessibility, by allowing the algorithm to have negligible error. That is, the algorithm may output $x \notin I_n$ with probability asymptotically less than any inverse polynomial in n .

5 A new tool for finding generators

Finding a generating set for an Abelian group G is important both in the definition of pseudo-random bit generators, and in applications such as the Diffie-Hellman key exchange protocol. There are several parts to the problem:

1. finding the cyclic decomposition of G ;
2. computing the orders of elements; and
3. testing that elements are “linearly independent.”

We present in this section a new tool for solving the second part of the problem. Let G be an Abelian group, let N be its order, and let $n = \log N$. The main result is an algorithm that uses polynomially-many (in n) group operations in G to compute the order of an element $x \in G$, with negligible probability of error. For this method to be applied to a group, the group must have an efficient (i.e., polynomial-time) algorithm to perform group operations, and its order (number of elements) must be known. Our order-computing algorithm utilizes Lenstra’s new factorization algorithm [Bac85] [Len85].

The outline of this section is as follows. An intermediate result, a partial factorization algorithm using Lenstra’s algorithm, appears in section 5.1. We then apply this algorithm in section 5.2 to show how to approximate the order of an element $x \in G$. Finally, we present an example of a generating algorithm that uses the results in section 5.3.

5.1 Partial factorization

We now present a polynomial-time algorithm that extracts all “small” prime factors of an integer N , using Lenstra’s factorization algorithm. Throughout the discussion, assume that N is fixed, and let $n = \log N$. Recall that Lenstra’s algorithm yields a factor p of N in expected time $O(L(p)^{\sqrt{2}+O(1)}n^3)$, time, where

$$L(p) \stackrel{\text{def}}{=} e^{\sqrt{\ln p \ln \ln p}}. \quad (17)$$

We will make much use of the key feature of Lenstra’s algorithm, which is that its running time depends on the smallest prime factor. Pomerance observes that this feature can “usually” be applied to determine whether a number is smooth with respect to some bound k , i.e., whether all its prime factors are less than or equal to k [Pom85]. Using this technique, it is also possible to find all prime factors less than or equal to k .

It is not known whether Lenstra’s algorithm is able to find all prime factors. This depends on a conjecture concerning the distribution of smooth numbers. If this conjecture is not true, then there may be certain primes that Lenstra’s algorithm never finds. In this case, the probability of error is no longer negligible. We assume the conjecture is true for the course of discussion. Indeed, Lenstra and others call this assumption “reasonable” [Len85] [Bac85].

This application of Lenstra’s algorithm is significant, because to find small factors or to test smoothness using previous methods would require time proportional to k or to \sqrt{k} . With the new technique, it is possible to find small factors in polynomial time for much larger k , asymptotically greater than any polynomial in n . Algorithm *small-factors*, in figure 2, shows one way to do this, and the following theorem formalizes the observation.

Theorem 16 Let N be an integer and let $n = \log N$. Algorithm *small-factors* finds all factors of N less than or equal to $n^{\ln n / \ln \ln n}$ in time polynomial in n , with negligible error.

Proof. We prove the theorem in three steps. First, we expand $L(k)$ to determine the expected time to find a single small factor of N . Second, we show how to make the probability of error negligible for a single small factor. Finally, we analyze the expected time to find all small factors using algorithm *small-factors*.

Expanding $L(k)$ for $k = n^{\ln n / \ln \ln n}$ gives

$$L(k) = e^{\ln n \sqrt{(2 \ln \ln n - \ln \ln \ln n) / \ln \ln n}}. \quad (18)$$

Thus, for sufficiently large n ,

$$L(k) \leq e^{\sqrt{2} \ln n} = n^{\sqrt{2}}, \quad (19)$$

and $L(k)^{\sqrt{2}} = O(n^2)$. Therefore, the expected time to find a factor smaller than $n^{\ln n / \ln \ln n}$, if one exists, is $O(n^{5+O(1)})$.

If M is sufficiently large in step 2 of the algorithm, then with high probability we find a factor smaller than $n^{\ln n / \ln \ln n}$, if one exists. Consider the process of finding such a factor as a Bernoulli process, since each trial in Lenstra's algorithm has the same probability of success, and these probabilities are independent. Let l_1 be a random variable representing the time between successes; then

$$E[l_1] = O(n^{5+O(1)}). \quad (20)$$

Using a Poisson approximation [Dra67] we have,

$$\Pr[l_1 > nE[l_1]] = e^{-n}. \quad (21)$$

If we set the M as $O(n^{6+O(1)})$, then the probability of error is negligible.

We conclude the analysis as follows. Since N has at most n factors counting multiplicity, we run step 2 at most n times. Each takes time $O(n^{6+O(1)})$, so the total running time is $O(n^{7+O(1)})$, which is polynomial. The probability of error is e^{-n} is negligible. ■

5.2 Approximating the order of an element

We now present a probabilistic algorithm that determines the order of an element of an Abelian group in polynomial expected time with negligible probability of error. First we discuss an algorithm to determine the order of an element x , given the complete factorization of N , the order of G ; then we analyze a similar algorithm in the case when only partial factorization is known—the small factors determined in section 5.1.

Without loss of generality, we assume that G is cyclic. If it is not cyclic, then the analysis and results would apply to the subgroup of G generated by x , where N is the order of the subgroup.

5.2.1 Determining order with complete factorization

Algorithm *with-complete-factorization*, in figure 3, shows how to determine the order of $x \in G$ given all factors of N . We prove two lemmas about this algorithm: first, that it is correct; and second, that it runs in polynomial time.

Lemma 17 Algorithm *with-complete-factorization* is correct.

Proof. By a corollary of the Chinese Remainder Theorem, each $x \in G$ corresponds to a tuple in $\mathbb{Z}_{p_1} \times \dots \times \mathbb{Z}_{p_r}$, i.e.

$$x \leftrightarrow (x_1, \dots, x_r), x_i \in \mathbb{Z}_{p_i^{\alpha_i}}. \quad (22)$$

For each i , computing $(N/p_i^{\alpha_i})x$ has the bijection

$$\frac{N}{p_i^{\alpha_i}}x \leftrightarrow (0, \dots, x_i, \dots, 0). \quad (23)$$

Finding the smallest $p_i^{\beta_i}$ for which $(Np_i^{\beta_i}/p_i^{\alpha_i})x = 0$ is the same as finding the order of x in $\mathbb{Z}_{p_i^{\alpha_i}}$. Since the p_i are pairwise relatively prime, the order of x is the product of the orders of x_i in $\mathbb{Z}_{p_i^{\alpha_i}}$, and the algorithm is correct. ■

Lemma 18 Algorithm *with-complete-factorization* uses polynomially-many (in n) group operations.

Proof. Since N has no more than $\log N$ prime factors, counting multiplicity, $\alpha_1 + \dots + \alpha_k \leq n$. Thus the number of computations of the form $(Np_i^{\beta_i}/p_i^{\alpha_i})x$ is at most n . Each computation requires $O(n)$ group operations, using successive doubling. Thus, $O(n^2)$ group operations are needed, in all. ■

5.2.2 Determining order with partial factorization

Suppose only partial information is known about the prime factors of N . Consider the modified version of algorithm *with-complete-factorization*, called *with-partial-factorization* (figure 4). Again, we prove two lemmas about the algorithm: first, that its probability of error is negligible; and second, that it runs in polynomial time.

For the first lemma, assume that $m = p_{k+1}^{\alpha_1} \cdots p_l^{\alpha_l}$, p_i prime, $p_i < p_{i+1}$, and without loss of generality assume $p_k < p_{k+1}$. One may ask, what is the probability that the algorithm errs, i.e., outputs o , where $o \neq \text{order}_G(x)$?

Lemma 19 Algorithm *with-partial-factorization* errs with probability at most n/p_{k+1} , where p_{k+1} is the smallest prime dividing m .

Proof. Suppose $\text{order}_G(x) = p_1^{\beta_1} \cdots p_l^{\beta_l}$, $0 \leq \beta_i \leq \alpha_i$. How does the algorithm behave? There are three cases.

1. $p_{k+1}^{\beta_{k+1}} \cdots p_l^{\beta_l} = m$; correct answer, $o = p_1^{\beta_1} \cdots p_k^{\beta_k} m = \text{order}_G(x)$.
2. $p_{k+1}^{\beta_{k+1}} \cdots p_l^{\beta_l} = 1$; correct answer, $o = p_1^{\beta_1} \cdots p_k^{\beta_k} = \text{order}_G(x)$.
3. $1 < p_{k+1}^{\beta_{k+1}} \cdots p_l^{\beta_l} < m$; incorrect answer, $o = p_1^{\beta_1} \cdots p_k^{\beta_k} m \neq \text{order}_G(x)$.

To find the probability of the third case, consider the subgroup S of elements whose orders divide N/m , and the quotient group G/S . Since G is cyclic, the order of G/S is m , and $G/S \cong \mathbb{Z}_m$. The number of elements of order neither m nor 1 in G/S is $m - \varphi(m) - 1$. The probability of an incorrect answer is exactly the probability that an element of G/S has order neither m nor 1, or $1 - \varphi(m)/m - 1/m$.

An upper bound for the probability can be determined by expanding $\varphi(m)$ and observing that $l-k \leq n$, since each index represents a distinct prime factor.² Expanding $\varphi(m)$ gives

$$1 - \frac{\varphi(m)}{m} - \frac{1}{m} = 1 - \prod_{k+1 \leq i \leq l} \frac{p_i - 1}{p_i} - \frac{1}{m}. \quad (24)$$

We can compute an upper bound on this value, as follows:

$$1 - \prod_{k+1 \leq i \leq l} \frac{p_i - 1}{p_i} - \frac{1}{m} \leq 1 - \left(\frac{p_{k+1} - 1}{p_{k+1}} \right)^{l-k} \leq \frac{l-k}{p_{k+1}} \leq \frac{n}{p_{k+1}}. \quad (25)$$

This is the probability of error stated in the lemma, so we are done. ■

Lemma 20 Algorithm *with-partial-factorization* uses polynomially-many (in n) group operations.

Proof. The proof is similar to that for algorithm *with-complete-factorization*. The running time is no greater, since the partial factorization of N has no more factors than the complete factorization. Therefore, $O(n^3)$ group operations are sufficient. ■

The two lemmas lead to our main result, that

Theorem 21 Let G be an Abelian group with an efficient composition operation, let x be an element of G , and let $n = \log N$ where N is the order of G . Using Lenstra's factorization algorithm, given x and N , it is possible to compute the order of x in G in time polynomial in n with negligible error.

Proof. Let $p_{k+1} = n^{1/\ln n / \ln \ln n}$; then algorithm *small-factors* can compute a partial factorization of the form needed for algorithm *with-partial-factorization* in time polynomial in n , with negligible error. Assuming N is in this partially-factored form, the probability of error in computing the order is $n^{1/\ln n / \ln \ln n}$, which is negligible. By lemma 20, the algorithm runs in time polynomial in n , if G has an efficient composition operation. ■

By lemma 2, the subset of G consisting of those elements of maximum order is non-negligible. Thus we have a corollary to the theorem.

Corollary. Let G , x , n and N be as in theorem 21. Using Lenstra's factorization algorithm, it is possible to test whether an element x of G has maximum order in time polynomial in n with negligible error.

²Knuth and Trabb-Pardo observe that an integer N has at most $n/\log n$ distinct factors [KT76]; this is a stronger bound for $l-k$. One may also show that $l-k \leq n/\log p_{k+1}$. However, the weak upper bound is adequate for the proof.

5.3 An example

We now present an example of a generating algorithm. The algorithm, *cyclic-generating-test* (see figure 5, operates on the family of multiplicative groups modulo p , where p is a prime, and tests whether an element is a generator. The test is a straightforward application of algorithm *with-partial-factorization*. If G has N elements, then we can apply *with-partial-factorization* to answer, with negligible error, the question,

"Is $\text{order}_G(x) = N$?"

If the answer is "yes," then with high probability, x generates G . This is a result of the corollary to theorem 21. (It is interesting to note that if the answer is "no," we cannot make the claim of high probability, because the subset of G consisting of those elements not of maximum order may be negligible.)

6 A new oracle proof technique

This section presents a new method of proving the equivalence of predicting the value of a single bit of a discrete logarithm, and computing the logarithm itself. If computing the logarithm is a "hard problem," then the method is strong enough to prove that $O(\log n)$ bits are simultaneously unapproximable, where n is the length of the logarithm. The method requires only that the group is Abelian, and that its cyclic decomposition is known.

The novel idea in the oracle proof technique is a notion of "correlation" that provides a measure of closeness to the correct logarithm. This measure can be used to obtain information about the range of values an index may take; by refining this information, the index can be found.

Previous methods for proving such theorems used properties of a group which are not always available. In particular, the Blum-Micali proof method [BM84] (and others based on it, e.g.[LW83]) required that the order of the group be even and used "quadratic residuosity" and square root operations. Although there is a square root operation for elliptic curves (since composition is defined by rational polynomials), the order of the group may be odd, and therefore previous methods are not applicable. Furthermore, elliptic curve groups may be non-cyclic, a condition not encountered in previous proof methods.

6.1 Preliminaries

Definition 22 Let G be an additive, Abelian, cyclic group with N elements, and let g be a generator of G . (Thus $G \cong \mathbf{Z}_N$.) The *half bit* of the index $b_{G,g,0}: G \rightarrow \mathbf{Z}_2$ is defined as

$$b_{G,g,0}(x) = \begin{cases} 1, & \text{if } \text{index}_{G,g}(x) \geq N/2; \\ 0, & \text{if } \text{index}_{G,g}(x) < N/2. \end{cases} \quad (26)$$

The half bit can also be called the *most significant bit* of the index. Other "most significant bits" can be defined recursively; the i -th most significant bit $b_{G,g,i}: G \rightarrow \mathbf{Z}_2$ is defined as

$$b_{G,g,i}(x) = b_{G,g,i-1}(2x). \quad (27)$$

In the discussion following, the group G and the generator g are assumed fixed, and the shorter notation $b_i(x)$ is used. The main theorem of this section is

Theorem 23 Let G be a cyclic group with an efficient composition operation, let g be a generator of G , and let $n = \log N$ where N is the order of G . Let $\epsilon > 0$, $0 \leq i < n$. The following problems are probabilistically reducible to each other in time polynomial in n , ϵ^{-1} and 2^i :

1. Compute $\text{index}(x)$.
2. Compute $b_i(x)$ correctly for $1/2 + \epsilon$ of $x \in G$.

The reduction of the second problem to the first is obvious. The following sections discuss the reduction in the other direction: the "oracle proof technique." The reduction is so named because such theorems traditionally have been proved by constructing an algorithm to compute $\text{index}(x)$ using an oracle that "guesses" one bit of the index with probability $1/2 + \epsilon$.

Computing $b_i(x)$ correctly for a fraction of $x \in G$ can be described more formally, in terms of approximation (see section 4). In particular, let $b_{i,\epsilon}$ be an ϵ -approximator to b_i . Recall that G and g are fixed. We use the notion of approximation to define *correlation*.

Definition 24 Let x_1 and x_2 be elements of G . The i -th bit correlation of x_1 and x_2 , $\rho_{i,\epsilon}: G \times G \rightarrow [-1/2, 1/2]$, is defined as

$$\rho_{i,\epsilon}(x_1, x_2) \stackrel{\text{def}}{=} \Pr[b_i(x_1 + rg) = b_{i,\epsilon}(x_2 + rg)] - \frac{1}{2}, \quad (28)$$

where r is a uniformly-distributed random variable taking integer values between 0 and $N - 1$.

Lemma 25 The following are true for all integers k and elements x_1 and x_2 of G :

$$\begin{aligned} & (\text{approximation}) \quad \rho_{i,\epsilon}(x_1, x_2) \geq \epsilon \\ & (\text{periodicity}) \quad |\rho_{i,\epsilon}(x_1, x_2) - \rho_{i,\epsilon}(x_1 + \lfloor k \frac{N}{2^i} \rfloor g, x_2)| \leq \frac{2^{i+1}}{N} \\ & (\text{symmetry}) \quad |\rho_{i,\epsilon}(x_1, x_2) + \rho_{i,\epsilon}(x_1 + \lfloor k \frac{N}{2^i} + \frac{N}{2^{i+1}} \rfloor g, x_2)| \leq \frac{2^{i+1}}{N}, \quad 0 \leq k < 2^i \\ & (\text{locality}) \quad |\rho_{i,\epsilon}(x_1 + g, x_2) - \rho_{i,\epsilon}(x_1, x_2)| \leq \frac{2^{i+1}}{N}. \end{aligned} \quad (29)$$

Proof. (*Approximation*) Observe that by the definitions of correlation and approximation,

$$\rho_{i,\epsilon}(x, x) = \Pr[b_i(x + rg) = b_{i,\epsilon}(x + rg)] - \frac{1}{2} \geq \epsilon. \quad (30)$$

(*Periodicity, symmetry and locality*) View the correlation as a piece-wise constant function on a circle, taking values 0 and 1. (See figure 6.)

Comparing $\rho_{i,\epsilon}(x_1, x_2)$ to $\rho_{i,\epsilon}(x_1 + \Delta x_1 g, x_2)$ amounts to rotating the circle by Δx_1 and comparing it to itself. However, since Δx_1 is an integer, the rotation may cause regions to overlap slightly. In other words, one comparison in each contiguous region of 0's or 1's may yield an “erroneous” value. Since there are at most 2^{i+1} such regions, we arrive at the stated limits. ■

6.2 Algorithms and analysis

We now present three algorithms to show the proof: algorithms *estimate-correlation* in figure 7, *decide-square-roots* in figure 8, and *compute-index* in figure 9. We also analyze their running times and probabilities of error.

6.2.1 Estimating correlation

By choosing random values for r , it is possible to estimate the correlation very well. First, we present an algorithm that does this estimation; then we analyze its running time and probability of error. The algorithm, *estimate-correlation*, appears in figure 7.

Lemma 26 Let δ be a positive real number, and let $\xi_{i,\epsilon}(cg, x)$, a random variable, be the estimate produced by algorithm *estimate-correlation* using M samples. If $M > \epsilon^{-1}\delta^{-1/2}$, then for all c and x ,

$$\Pr[|\xi_{i,\epsilon}(cg, x) - \rho_{i,\epsilon}(cg, x)| > \frac{\epsilon}{2}] < \delta. \quad (31)$$

Proof. Let c and x be fixed, and write $\xi = \xi_{i,\epsilon}(cg, x)$. Then, since the estimate is the average value of a Bernoulli process, we have

$$E[\xi] = \rho_{i,\epsilon}(cg, x); \quad (32)$$

$$\sigma_\xi^2 = \frac{1}{M} E[\xi](1 - E[\xi]) \leq \frac{1}{4M}. \quad (33)$$

By the weak law of large numbers,

$$\Pr[|\xi - E[\xi]| > \frac{\epsilon}{2}] \leq \frac{\sigma_\xi^2}{M(\epsilon/2)^2} \leq \frac{1}{M^2\epsilon^2} < \delta, \quad (34)$$

as stated in the lemma. ■

It is of significance to note that the upper bound on the probability of error is independent of the values of c and x , and of any other estimations made. This is because the estimation randomizes over r , and δ is an upper bound regardless of the value being estimated.

As an application of this lemma, we show that it is possible to estimate the correlation very well in polynomial time.

Lemma 27 Using polynomially-many (in n , ϵ^{-1} and δ^{-1}) coin flips, group operations and oracle invocations, algorithm *estimate-correlation* errs by $\epsilon/2$ or more with probability at most δ .

Proof. Selecting $M = \epsilon^{-1}\delta^{-1/2}$, we use $O(\epsilon^{-1}\delta^{-1/2}n)$ coin flips and group operations, and $O(\epsilon^{-1}\delta^{-1/2})$ oracle invocations, as follows. The first step of the algorithm computes M n -bit numbers, requiring $O(Mn)$ coin flips. The second step uses the oracle once per trial, or M times, and each computation of $r_j g$ requires $O(n)$ group operations, using successive doubling. The time to compute $b_i(cg + r_j g)$ (since c and r_j are known) and $x + r_j g$ is comparatively small. ■

6.2.2 Deciding square roots

We now present an algorithm to “decide square roots,” a key method in determining the logarithm. Let $[a, b]$ denote the interval between a and b , inclusive, where a and b are real numbers. When a and b are sufficiently close, it is possible answer the question,

“Given that $\text{index}(2^{i+1}x) \in [2^{i+1}a, 2^{i+1}b]$, is $\text{index}(2^i x)$ in $[2^i a, 2^i b]$ or in $[2^i a + N/2, 2^i b + N/2]?$ ”

Lemma 28 Let $[a, b]$ be an interval with

$$0 \leq a \leq b \leq a + \frac{N\epsilon}{2^{i+1}} - 3 < \frac{N}{2^{i+1}}, \quad (35)$$

and let c be the integer closest to $(a + b)/2$. Let δ be any positive real number. If $\text{index}(2^{i+1}x) \in [2^{i+1}a, 2^{i+1}b]$ and $\text{index}(2^i x) \in [2^i a, 2^i b]$, then for $M > \epsilon^{-1}\delta^{-1/2}$,

$$\Pr[\xi_{i,\epsilon}(cg, x) < 0] < \delta. \quad (36)$$

Proof. The hypothesis $\text{index}(2^i x) \in [2^i a, 2^i b]$ implies

$$\text{index}(x) \in \bigcup_{0 \leq k < 2^i} [a + k \frac{N}{2^i}, b + k \frac{N}{2^i}]. \quad (37)$$

Then for some value of k , it is true that

$$|c + \lfloor k \frac{N}{2^i} \rfloor - \text{index}(x)| \leq \frac{b - a + 1}{2}. \quad (38)$$

Furthermore, because $\rho_{i,\epsilon}$ is periodic,

$$|\rho_{i,\epsilon}(cg + k \frac{N}{2^i} g, x) - \rho_{i,\epsilon}(cg, x)| \leq \frac{2^{i+1}}{N} \quad (39)$$

for all k . By lemma 25,

$$|\rho_{i,\epsilon}(cg, x) - \rho_{i,\epsilon}(\text{index}(x)g, x)| \leq \frac{2^{i+1}}{N} \cdot \frac{b - a + 1}{2} + \frac{2^{i+1}}{N} \leq \frac{\epsilon}{2}. \quad (40)$$

Consequently, we have a lower bound on the correlation,

$$\rho_{i,\epsilon}(cg, x) > \frac{\epsilon}{2}. \quad (41)$$

Using the estimation algorithm with $M > \epsilon^{-1}\delta^{-1/2}$, we arrive at

$$\Pr[\xi_{i,\epsilon}(cg, x) < 0] < \delta, \quad (42)$$

proving the lemma. ■

Using these observations, we construct an algorithm *decide-square-roots* (see figure 8) to answer the initial question.

Lemma 29 Given x , and assuming that $\text{index}(2^{i+1}x) \in [2^i a, 2^{i+1} b]$, algorithm *decide-square-roots* is correct with probability $> 1 - \delta$, where δ is the error bound in estimating the correlation.

Proof. The probability of error is less than δ , whether $\text{index}(x) \in [2^i a, 2^i b]$ or $\text{index}(x) \in [2^i a + N/2, 2^i b + N/2]$. Since these are the only intervals in which the index may lie, the algorithm is correct with probability $> 1 - \delta$. Furthermore, by lemma 27, the algorithm runs in time polynomial in n, ϵ^{-1} and $\delta^{-1/2}$. ■

6.2.3 Computing the index

Algorithm *compute-index* (figure 9) computes the index of x by successively restricting the range of indexes in which $\text{index}(2^{j+i}x)$ may lie. To show that this algorithm runs in polynomial time, we determine the probability that the restricted range for $\text{index}(x)$ is incorrect, assuming an initial range for $\text{index}(2^{n+i}x)$ is correct. By showing the probability is sufficiently small, we determine that in few iterations of the main part of the algorithm, we can find the index.

Lemma 30 Algorithm *compute-index* finds $\text{index}(x)$ using polynomially-many (in n, ϵ^{-1} and 2^i) expected coin flips, group operations and oracle invocations.

Proof. Suppose the interval $[a, b]$ selected in step 1 is correct. By lemma 29, the probability of choosing the next $[a, b]$ correctly in step 2 is at least $1 - \delta$. Thus, the probability of success in n consecutive choices is

$$(1 - \delta)^n \geq 1 - n\delta, \quad (43)$$

since probabilities are independent by the definition of the estimation algorithm.

If $[a, b]$ contains $\text{index}(2^i x)$ upon entering step 3, then $[a, b]$ contains exactly one integer, because step 2 reduces $b - a$ by a factor of 2 each time. Thus

$$b - a \leq \frac{1}{2^n} \left(\frac{N\epsilon}{2^{i+1}} - 3 \right) < \frac{N\epsilon}{2^{n+i+1}} < 1 \quad (44)$$

upon entering step 3. Step 4 then recovers the index.

To determine the running time, observe that there are about $2^{i+1}\epsilon^{-1}$ intervals, so the expected number to search is $2^i\epsilon^{-1}$. Further, if $\delta = 1/2n$, then the probability of success in equation 43 is at least $1/2$, and thus two trials of step 2 are expected given the correct interval. In all, $2^{i+1}\epsilon^{-1}$ trials of steps 1-3 are expected. Since algorithm *decide-square-roots* and all other operations are polynomial in n, ϵ^{-1} and $\delta^{-1/2}$, algorithm *compute-index* is polynomial in n, ϵ^{-1} and 2^i . ■

6.3 Proof of main theorem

Finally, we complete the proof of the main theorem. Lemma 30 shows that an ϵ -approximator for b can be used to compute $\text{index}(x)$ in polynomially-many coin flips, group operations and oracle invocations. If the group has an efficient composition operation, we have a probabilistic polynomial-time reduction from index computation to ϵ -approximation, and the theorem is proved. ■

6.4 Extensions

The oracle proof technique is easily modified to handle non-cyclic groups and to show the simultaneous security of $O(\log n)$ bits.

7 Simple case of elliptic curves

In a restricted class of elliptic curves, it is possible to construct a pseudo-random bit generator in a relatively straightforward manner. This restricted class has certain properties that make the construction and proofs easy. The intent of discussing this class is to provide a simple application of the sufficient conditions for pseudo-random bit generation, which will set a better foundation for the general case in section 8.

7.1 Statement of theorem

Definition 31 Let $E(\mathbb{F}_p)$ be an elliptic curve in the simple case (section 3.3), and let G be a generator. Then $\langle E(\mathbb{F}_p), G \rangle$ is a *curve-generator pair*.

We say an algorithm *solves the elliptic logarithm problem* for a curve-generator pair $\langle E(\mathbb{F}_p), G \rangle$ if for every $P \in E(\mathbb{F}_p)$ it computes $\text{index}_{E(\mathbb{F}_p), G}(P)$ correctly. Let $Q(n)$ be a polynomial, and let $T_Q(n)$ be the running time of any algorithm that solves the elliptic logarithm problem for at least a fraction $1/Q(n)$ of all curve-generator pairs, where n is the length of p . By Miller's arguments [Mil85a], we have

Conjecture 32 (Simple elliptic logarithm intractability assumption) $T_Q(n)$ grows faster than any polynomial in n .

This leads to our main theorem. Let an *instance of the simple case* be a tuple

$$\langle E(\mathbb{F}_p), G, P \rangle, \quad (45)$$

where $E(\mathbb{F}_p)$ is an elliptic curve in the simple case, G generates $E(\mathbb{F}_p)$, and P is a point on the curve. The set of all instances where p is an n -bit prime is denoted I_n .³

Theorem 33 Let I be the set of instances in the simple case. Let f and b be defined for an instance $s = \langle E(\mathbb{F}_p), G, P \rangle$ as

$$f(s) = \langle E(\mathbb{F}_p), G, \psi(P) \rangle, \quad (46)$$

where

$$\psi(P) = \phi(P)G \text{ and } \phi(P) = \begin{cases} y, & \text{if } P = (x, y); \\ p, & \text{otherwise,} \end{cases} \quad (47)$$

and

$$b(s) = b_{E(\mathbb{F}_p), G, 0}(P). \quad (48)$$

Under conjecture 32, $\langle I, f, b \rangle$ is a Blum-Micali pseudo-random bit generator.

Proof. We prove the theorem by showing that the four sufficient conditions for a pseudo-random bit generator (section 4.4) are satisfied. Specifically, we show that b and f are friendship functions, f is a stable, b is accessible, and b is unapproximable. The proofs for each of these parts follow.

7.2 Friendship

Friendship is the easiest of the four parts to show. In particular, we simply prove

Lemma 34 There is an algorithm that computes bf in the simple case in polynomial time.

Proof. Let $s = \langle E(\mathbb{F}_p), G, P \rangle$ be an instance. Using the definitions of f and b , we can write bf as

$$bf(s) = \begin{cases} 1, & \text{if } \text{index}_{E(\mathbb{F}_p), G}(\psi(P)) \geq (p+1)/2; \\ 0, & \text{otherwise.} \end{cases} \quad (49)$$

Since $\text{index}_{E(\mathbb{F}_p), G}(\psi(P)) = \phi(P)$, substituting the definition of ϕ leads to

$$bf(s) = \begin{cases} 1, & \text{if } P = 0; \\ 1, & \text{if } P = (x, y) \text{ and } y \geq (p+1)/2; \\ 0, & \text{otherwise.} \end{cases} \quad (50)$$

This is easily computed in time polynomial in n . ■

³A data structure representing an instance as would probably contain n , p , B , and the x - and y -coordinates of G and P (or the special symbol “0”). We write $\langle E(\mathbb{F}_p), G, P \rangle$ for convenience.

7.3 Stability

We base this on two intermediate results.

Lemma 35 ϕ is a bijection of $E(\mathbf{F}_p)$ and $\{0, \dots, p\}$ in the simple case.

Proof. By lemma 8 each point has a unique y -coordinate. Therefore points are mapped to unique elements of $\{0, \dots, p\}$. ■

Lemma 36 ψ is a permutation of $E(\mathbf{F}_p)$ in the simple case.

Proof. Since G generates $E(\mathbf{F}_p)$, the set

$$\{aG \mid 0 \leq a < p+1\} \quad (51)$$

contains all points on the curve. Thus ψ on input P generates $E(\mathbf{F}_p)$ based on $\phi(P)$. Since ϕ is a bijection into $\{0, \dots, p\}$, the curve is completely generated, and ψ is a permutation. ■

Lemma 37 (Stability) In the simple case, for each n , f is a permutation on I_n . Further, there is an algorithm that computes f in polynomial time.

Proof. The first part is true by definition of f , because ψ permutes $E(\mathbf{F}_p)$. The second part is true because computing f requires $O(\log \phi(P)) = O(n)$ group operations, which is polynomial in n . ■

7.4 Accessibility

To show that the predicate b is accessible, we construct an algorithm and analyze it. The algorithm is probabilistic and produces elements of I_n with uniform probability in polynomial expected time. The algorithm, called *simple-accessibility*, appears in figure 10. We prove two lemmas about this algorithm, thereby showing accessibility.

We first recall the result of Bach [Bac83] also used in the Blum-Micali accessibility proof. (There it was used to produce $p-1$ in factored form.)

Lemma 38 (Bach, 1983) There is a probabilistic polynomial-time algorithm that takes as input n and outputs an integer of length n , together with its factorization. The integers are uniformly distributed.

Lemma 39 Algorithm *simple-accessibility* produces elements x of I_n in the simple case with uniform probability.

Proof. Since failure at steps 2–5 leads to “go to 1,” it is sufficient to show that each step selects p , E , G , or P with uniform probability among the acceptable values for that component. Using Bach’s algorithm all $p+1$ are equally likely; thus, all primes $p \equiv 2 \pmod{3}$ are as well. Since all legal B are equally likely for a given p , all curves E are also. Algorithm *cyclic-generating-test* accepts all generators of $E(\mathbf{F}_p)$, because the factorization of $p+1$ is known. Thus, since ϕ is a bijection (lemma 35), generators G and points P are selected with equal likelihood. ■

Lemma 40 Algorithm *simple-accessibility* outputs an element $x \in I_n$ in the simple case in polynomial (in n) expected time.

Proof. The expected number of trials is $O(n \log n)$, as follows. Step 1 always succeeds. By the prime number theorem, and assuming that half of all primes are congruent to 2 mod 3, the probability of success in step 2 is $1/O(n)$. In step 3, finding a B has probability at least $1/2$. In step 4, finding an a has probability at least $1/2$; a generator, $1/O(\log n)$, by lemma 3. In step 5, finding an a has probability at least $1/2$. The probability of success in every step is $1/O(n \log n)$, leading to the expected value given above.

The running time for one trial is $O(n^6)$, as follows. Bach’s algorithm runs in time $O(n^8)$. Testing primality is $O(n^4)$, using the technique of Solovay and Strassen [SS77]. Checking that G is a generator, with the factorization of $p+1$ known, takes $O(n^2)$ group operations, or time $O(n^4)$, by lemma 18. Computing ϕ^{-1} involves taking cube roots modulo p , which requires time $O(n^3)$.

This leads to an expected running time of $O(n^7 \log n)$, which is polynomial in n . ■

7.5 Unapproximability

Let A be some algorithm that computes b correctly with probability at least $1/2 + 1/Q(n)$ on elements of I_n , and let $T(n)$ be its running time.

Lemma 41 There is a fraction $1/2Q(n)$ of all curve-generator pairs $(E(\mathbb{F}_p), G)$, where p is an n -bit prime, such that algorithm $A(\langle E(\mathbb{F}_p), G, \cdot \rangle)$ $1/2Q(n)$ -approximates $b_{E(\mathbb{F}_p), G, 0}(\cdot)$.

Proof. This follows from a counting argument. The main idea is that the probability is minimized when A is entirely correct on some curve-generator pairs, and nearly a $1/2Q(n)$ -approximator on the rest. For every $E(\mathbb{F}_p)$ where p is an n -bit prime, we have

$$2^{n-1} + 1 \leq N_p \leq 2^n. \quad (52)$$

If the curves in the curve-generator pairs on which A is correct contain the maximum number of elements, and the rest contain the minimum number, then we arrive at the stated probability. ■

Using algorithm A as the ϵ -approximator, we can compute $\text{index}(x)$ with algorithm *simple-index* (figure 11). See figure 9 for algorithm *compute-index*.

Lemma 42 (Unapproximability) The predicate b in the simple case is unapproximable.

Proof. By theorem 23, algorithm *simple-index* computes the index correctly for the fraction of curve-generator pairs for which A is an $1/2Q(n)$ -approximator in lemma 41. Furthermore, it does so in time polynomial in n , $Q(n)$ and $T(n)$. Suppose b is not unapproximable, and $T(n)$ is a polynomial in n . Then the algorithm solves the elliptic logarithm problem in time polynomial in n for the $1/2Q(n)$ fraction. This contradicts conjecture 32, and therefore b is unapproximable. ■

8 General case of elliptic curves

The pseudo-random bit generator for the general case is like that for the simple case, but we use a pair of elliptic curves to define the friendship function. The general case is described in detail in the author's dissertation. The following are the major differences between the general case and the simple case.

- An instance of the general case is a tuple containing two elliptic curves, two generators for each, and a point which may be on either curve. The curves are related by a transformation called *twisting*. This construction is essential to the definition of the friendship function f .
- The proof of accessibility requires a much more complicated algorithm than in the simple case. Two particular problems are that the group $E(\mathbb{F}_p)$ is not necessarily cyclic in the general case, and that the complete factorization of its order N_p is not known. The problems are solved by using the *Weil pairing* (for which Miller has recently developed a polynomial-time algorithm [Mil85b]), and algorithm *with-partial-factorization* (section 5.2).
- Since two elliptic curves are involved, the counting arguments involved in proving unapproximability are more difficult than in the simple case, though not unreasonable.

9 Conclusion

We propose several extensions to our research.

9.1 Reduction from discrete logarithm

The discrete logarithm problem stands on its own as a hard mathematical problem. Unlike quadratic residuosity and inverting RSA, it does not reduce easily to another hard problem, such as factoring. Whether it reduces to the elliptic logarithm problem is an interesting open problem.

9.2 Elliptic curves with factored order

The results in the general case (section 8) depend on an algorithm to compute the partial factorization of the order of the group. While this is sufficient for the generation of pseudo-random bits, it would be more elegant to use a completely-factored order, as Blum and Micali do [BM84]. This would require an algorithm like Bach's [Bac83] to generate an elliptic curves together with the factorization of its order.

- Is there a polynomial-time algorithm which, on input n , outputs an elliptic curve $E(\mathbb{F}_p)$ together with the factorization of its order, where $E(\mathbb{F}_p)$ is selected with uniform probability among all curves where p is an n -bit prime.

9.3 Subexponential elliptic logarithm algorithm

Adleman's algorithm for computing discrete logarithms [Adl79] was a breakthrough in 1979. Although Miller argues strongly for the ineffectiveness of techniques similar to Adleman's for computing elliptic logarithms [Mil85a], it may be possible to devise an alternative method. Ideally, this method would run in subexponential time.

- Is there an algorithm that computes elliptic logarithms in subexponential time?

References

- [ACGS] Werner Alexi, Benny Chor, Oded Goldreich, and Claus P. Schnorr. RSA and Rabin functions: Certain parts are as hard as the whole. To appear, SIAM Journal of Computing.
- [Adl79] Leonard Adleman. A subexponential algorithm for the discrete logarithm problem with applications to cryptography. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, pages 55–60, IEEE Computer Society, 1979.
- [AW85] Miklos Ajtai and Avi Wigderson. Deterministic simulation of probabilistic constant depth circuits. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society, 1985.
- [Bac83] Eric Bach. How to generate factored random numbers. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pages 184–188. Association for Computing Machinery, 1983.
- [Bac85] Eric Bach. Lenstra's algorithm for factoring with elliptic curves, an exposé. 1985. Unpublished manuscript.
- [BBS83] Lenore Blum, Manuel Blum, and Michael Shub. Comparison of two pseudo-random number generators. In *Proceedings of Crypto'82*, pages 61–78, Plenum Press, 1983.
- [BM84] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal of Computing*, 13(4):850–864, 1984.
- [BMP75] I. Borosh, C.J. Moreno, and H. Porta. Elliptic curves over finite fields: II. *Mathematics of Computation*. 29(131):951–964. July 1975.
- [Cas66] J.W.S. Cassels. Diophantine equations with special reference to elliptic curves. *Journal of the London Mathematical Society*, 41:193–291. 1966.
- [Cho65] S. Chowla. *The Riemann Hypothesis and Hilbert's Tenth Problem*. Gordon and Breach, 1965.
- [DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions in Information Theory*, IT-22(6):644–654, November 1976.
- [Dra67] Alvin W. Drake. *Fundamentals of Applied Probability Theory*. McGraw-Hill, Inc., 1967.
- [Ful69] William Fulton. *Algebraic Curves*. W.A. Benjamin, 1969.

- [GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. In *Proceedings of the 25th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society, 1984.
- [GK86] Shafi Goldwasser and Joseph Kilian. A provably correct, probably fast primality testing algorithm. 1986. To appear, *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*.
- [Jol73] Jean-René Joly. Equations et variétés algébriques sur un corps fini. *L'Enseignement Mathématique*, 59:74–79, 1973.
- [Kob84] Neal Koblitz. *Introduction to Elliptic Curves and Modular Forms*. Volume 97 of *Graduate Texts in Mathematics*, Springer-Verlag, 1984.
- [KT76] Donald E. Knuth and Luis Trabb-Pardo. Analysis of a simple factorization algorithm. *Theoretical Computer Science*, 3:321–348, 1976.
- [Len85] H.W. Lenstra. Elliptic curve factorization. Memorandum, 1985.
- [Lev85] Leonid Levin. One-way functions and pseudorandom generators. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, pages 363–365, Association for Computing Machinery, 1985.
- [LW83] Douglas L. Long and Avi Wigderson. How discreet is the discrete log? In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pages 413–420, Association for Computing Machinery, 1983.
- [Mil85a] Victor Miller. Elliptic curves and cryptography. 1985. To appear, *Proceedings of Crypto'85*.
- [Mil85b] Victor Miller. Short programs for functions on curves. 1985. Unpublished manuscript.
- [Pom85] Carl Pomerance. How to factor a number. Seminar, MIT Laboratory for Computer Science, 1985.
- [RS62] J. Rosser and L. Schoenfeld. Approximate formulas for some functions of prime numbers. *Illinois Journal of Mathematics*, 6:64–94, 1962.
- [Sch85] René Schoof. Elliptic curves over finite fields and the computation of square roots mod p . *Mathematics of Computation*, 44:483–494, April 1985.
- [SS77] R. Solovay and V. Strassen. A fast Monte-Carlo test for primality. *SIAM Journal of Computing*, 6:84–85, 1977.
- [Tat74] John T. Tate. The arithmetic of elliptic curves. *Inventiones Mathematicae*, 23:179–206, 1974.
- [Yao82] Andrew C. Yao. Theory and applications of trapdoor functions. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, IEEE Computer Society, 1982.

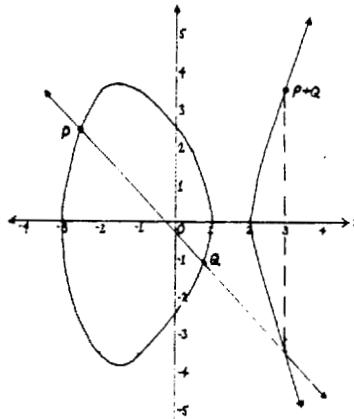


Figure 1: An elliptic curve, showing tangents and chords operation.

Assume M is given. Algorithm computes small prime factors of an integer N .

1. Initialize $S \leftarrow \{\}$.
2. Run Lenstra's algorithm for time M to find a factor p . If no factor is found in this time, go to 4.
3. Set $S \leftarrow S \cup \{p\}$, $N \leftarrow N/p$; go to 2.
4. Return S .

Figure 2: Algorithm *small-factors*.

Suppose $N = p_1^{\alpha_1} \cdots p_k^{\alpha_k} m$, p_i prime, $p_i < p_{i+1}$, $\alpha_i > 0$. Algorithm finds order of x in Abelian group G using complete factorization of N .

1. For each i , do step 2.
2. Let β_i be the smallest integer such that $(N p_i^{\beta_i} / p_i^{\alpha_i})x = 0$.
3. Output $p_1^{\beta_1} \cdots p_k^{\beta_k}$.

Figure 3: Algorithm *with-complete-factorization*.

Suppose $N = p_1^{\alpha_1} \cdots p_k^{\alpha_k} m$, p_i prime, $p_i < p_{i+1}$, $\alpha_i > 0$, m not necessarily prime. Algorithm finds order of x in Abelian group G using partial factorization of N .

1. For each i , do step 2.
2. Let β_i be the smallest integer such that $(N p_i^{\beta_i} / p_i^{\alpha_i})x = 0$.
3. If $(N/m)x = 0$, then output $p_1^{\beta_1} \cdots p_k^{\beta_k}$; else output $p_1^{\beta_1} \cdots p_k^{\beta_k} m$.

Figure 4: Algorithm *with-partial-factorization*.

Suppose N is the order of G . Algorithm determines whether x generates Abelian group G .

1. Use algorithm *small-factors* to compute a partial factorization of N .
2. Apply algorithm *with-partial-factorization* to compute an approximation o to $\text{order}_G(x)$.
3. If $o = N$, output "yes"; else output "no."

Figure 5: Algorithm *cyclic-generating-test*.

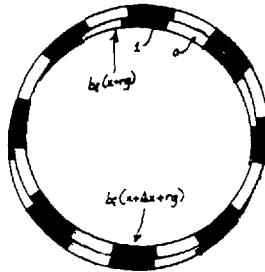


Figure 6: Correlation circle.

Estimates $\rho_{i,\epsilon}(cg, x)$ for an integer c .

1. Choose r_j at random, $0 \leq r_j < N$, $0 \leq j < M$.
2. For each r_j , compare $b_i(cg + r_j g)$ to $b_{i,\epsilon}(x + r_j g)$. Set *correct* to the number of equal results.
3. Return *correct*/ M .

Figure 7: Algorithm *estimate-correlation*.

Given x , and assuming that $\text{index}(2^{i+1}x) \in [2^{i+1}a, 2^{i+1}b]$, determines in which interval $\text{index}(2^i x)$ lies.

1. Let c the integer closest to $(a + b)/2$, and compute $\xi_{i,\epsilon}(cg, x)$ using algorithm *estimate-correlation*.
2. If the estimate is positive, then return $[2^i a, 2^i b]$; else return $[2^i a + N/2, 2^i b + N/2]$.

Figure 8: Algorithm *decide-square-roots*.

1. Choose an interval $[a, b]$ with

$$0 \leq a \leq b \leq a + \frac{N\epsilon}{2^{i+1}} - 3 < \frac{N}{2^{i+1}}.$$

2. For j from $n-1$ to 0, set $y = 2^j x$ and use algorithm *decide-square-roots* to determine whether $\text{index}(2^i y)$ is in $[2^i a, 2^i b]$ or $[2^i a + N/2, 2^i b + N/2]$, given $\text{index}(2^{i+1}y) \in [2^{i+1}a, 2^{i+1}b]$. In the former case, set $[a, b] \leftarrow [a/2, b/2]$; in the latter, set $[a, b] \leftarrow [a/2 + N/2^{i+1}, b/2 + N/2^{i+1}]$.
3. Let c be the integer in $[2^{i+1}a, 2^{i+1}b]$. If there is no integer, or if $cg \neq 2^i x$, then go to 2.
4. Given that $\text{index}(2^i x) = c$, test up to 2^i integer values of the form

$$\frac{c}{2^i} + k \frac{2^i}{N},$$

$0 \leq k < 2^i$, to find $\text{index}(x)$.

Figure 9: Algorithm *compute-index*.

Algorithm selects $x \in I_n$ with uniform probability in polynomial expected time.

1. Apply Bach's algorithm to produce $p+1$ in factored form. If $p+1 = 2^{n-1}$, then let $p = 2^n - 1$.
2. Test that p is prime, and $p \equiv 2 \pmod{3}$. If not, go to 1.
3. Guess an n -bit number B . If $B = 0$ or $B \geq p$, go to 1. The elliptic curve is $E : y^2 = x^3 + B$.
4. Guess an n -bit number a . If $a > p$, go to 1. Else let $G = \phi^{-1}(a)$. Check that G generates $E(\mathbb{F}_p)$ using algorithm *cyclic-generating-test* (section 5.3). If not, go to 1.
5. Guess an n -bit number a . If $a > p$, go to 1. Else let $P = \phi^{-1}(a)$
6. Output $(E(\mathbb{F}_p), G, P)$.

Figure 10: Algorithm *simple-accessibility*.

Computes $\text{index}_{E(\mathbf{F}_p), G}(P)$ for some $(E(\mathbf{F}_p), G)$.

1. Run algorithm *compute-index* using algorithm *A* with fixed $(E(\mathbf{F}_p), G)$ as the approximator, to produce c .
2. Output c .

Figure 11: Algorithm *simple-index*.

Two Remarks Concerning the Goldwasser-Micali-Rivest Signature Scheme

Oded Goldreich

Computer Science Department
Technion, Haifa 32000, Israel

ABSTRACT

The focus of this note is the Goldwasser-Micali-Rivest Signature Scheme (presented in the 25th FOCS, 1984). The GMR scheme has the salient property that, unless factoring is easy, it is infeasible to forge any signature even through an adaptive chosen message attack. We present two technical contributions with respect to the GMR scheme:

- 1) *The GMR scheme can be made totally “memoryless”:* That is, the signature generated by the signer on message M does not depend on the previous signed messages. (In the original scheme, the signature to a message depends on the number of messages signed before.)
- 2) *The GMR scheme can be implemented almost as efficiently as the RSA:* The original implementation of the GMR scheme based on factoring, can be speeded-up by a factor of $|N|$. Thus, both signing and verifying take time $O(|N|^3 \log^2 |N|)$. (Here N is the moduli.)

1. Introduction

In 1984, Goldwasser, Micali and Rivest presented a signature scheme robust against adaptive chosen message attack [GMR]: *no adversary who first receives signatures to messages of his choice, can later create a signature of even a single additional message.* The scheme uses two pairs of trapdoor permutations, and is proven to be secure (in the above sense) if these pairs have a certain *clawfree* property (i.e. it is infeasible to find x and y such that $f_0(x) = f_1(y)$). It was shown that the intracability assumption of factoring implies the clawfree condition, and thus a concrete factoring-based implementation was presented.

Work done while author was in the Laboratory for Computer Science, MIT.

Partially supported by a Weizmann Postdoctoral Fellowship, an IBM Postdoctoral Fellowship, and NSF Grant DCR-8509905.

The original GMR scheme suffers from two aesthetic drawbacks:

- 1) The signature scheme is not completely “memoryless”. That is, the signature generated by the signer slightly depends on the previous signed messages.
- 2) The signing process in the suggested factoring-based implementation is too slow. Let n be the security parameter (i.e. the length of the moduli), then signing takes more than n^4 steps (as opposed to n^3 steps in the RSA).

In this note we suggest a modification of the abstract GMR scheme, and a speed-up in its factoring-based implementation. These suggestions eliminate both drawbacks listed above, while maintaining the security of the original scheme. That is, the modified scheme is totally “memoryless”, and its implementation using factoring is almost as fast as the RSA (while being much more secure than RSA!).

The rest of this note is organized as follows. In section 2 we briefly review the GMR signature scheme. The reader is encouraged to consult the original paper [GMR] for a more complete exposition of the GMR scheme. (This reference is also an excellent source for a critical review and a historical account of the problem of obtaining digital signatures.) In section 3, we present a modification which makes the GMR scheme “memoryless”. In section 4, we present a speed-up in the factoring-based implementation of the GMR scheme. Our conclusions are presented in Section 5.

2. Overview of the GMR Scheme

The GMR scheme is basically a two-stage authentication process. First the signer's entry in the public file is used to authenticate a random *point of reference* (hereafter called *REF*). Next this REF is used to authenticate the message in a bit-by-bit manner. The same REF is never used to authenticate two different messages. The scheme utilizes two pairs of trapdoor clawfree permutations, denoted (f_0, f_1) and (g_0, g_1) . The f -pair is used for the first stage (authentication of the REF), while the g -pair is used for the second stage (authentication of the message).

The REFs are generated from the public file by using a tree structure (hereafter called *the REF tree*). The same REF tree is used to sign all messages. The root of the REF tree contains part of the signer's (entry in the) public file. Each internal node in the REF tree has a constant number of children (say three). Leaves in the REF tree are used to authenticate messages (in the second stage of the signing process). It was originally proposed [GMR] that only the third node of each internal node be a leaf, while the other children be potential internal nodes. As we will see in the sequel, this particular tree structure (for the REF tree) is not essential.

Authentication of the nodes in the REF tree is done successively: each node is authenticated by its father in the tree. A crucial detail in the GMR scheme is the manner

in which one REF authenticates its children. Each REF is partitioned into five parts: its "name" (denoted x), the names of its children (denoted y_1, y_2, y_3), and a "tag" (denoted t) binding them all together. The tag is computed using the first pair of (trapdoor) claw-free permutations f_0 and f_1 , through what is called *an authentication step*.

Let $F^{-1}(\sigma, x) = f_\sigma^{-1}(x)$, for every $\sigma \in \{0,1\}$. Let $F^{-1}(\alpha\sigma, x) = F^{-1}(\alpha, f_\sigma^{-1}(x))$, for every $\sigma \in \{0,1\}$ and $\alpha \in \{0,1\}^*$.

Then the five parts of the REF satisfy $t = F^{-1}(y_1 y_2 y_3, x)$.

(The REF tree should not be confused with the tree-like nature of the authentication step.)

The authentication of the message, in the second stage, consists of a single authentication step that uses the g -pair. This authentication step binds the message m (to be signed) to a leaf in the REF tree. The tag t binding the leaf named x with the message m satisfies $t = G^{-1}(m, x)$, where G^{-1} is defined analogously to F^{-1} based on the permutations g_0 and g_1 .

To sign a new message, m , the signer identifies a leaf that was not used so far (named z). If no such leaf exists, the signer identifies an unused internal node named x (i.e. a node which is a potential internal node but has no children yet), randomly selects names for its children (y_1, y_2, y_3), computes a tag binding the children to their father (i.e. uses the trapdoor information to compute $t = F^{-1}(y_1 y_2 y_3, x)$), stores the names of the children, and sets $z = y_3$. The signer retrieves (or recomputes) the tags for all the authentication steps leading from the root of the REF tree to the leaf named z . (It is crucial that, in all signatures produced, the same names are used for the same nodes.) This completes the first stage of the signing process. Next the signer uses the trapdoor information (for the pair g_0 and g_1) to compute the tag $G^{-1}(m, z)$. The signature consists of the list of all authentication steps mentioned above (corresponding to a path from the root to a leaf and an extra step authenticating the message by the leaf).

To verify the validity of a signature, the verifier checks that the list corresponds to a path from the root to a leaf, and that all tags along this path are valid. To validate $t = F^{-1}(\alpha, x)$, the verifier computes $F(\alpha, t)$ and compares it to x .

3. Making the GMR Scheme Memoryless

As hinted in the outline of the GMR scheme, the particular way of using the nodes in the REF tree is not essential to the scheme. In the original scheme, the nodes were used in a "greedy" manner so to minimize the length of the first signatures. The drawback in that suggestion is that it was required to remember the identity of the last node used for message authentication (i.e. to store the number of messages signed so far). Our aim is to eliminate the dependency of the next signature on the past. Thus, we suggest to use the REFs in the tree differently than in the original scheme.

Let n be the security parameter, and let $k = (\log_2 n)^2$. For message authentication (second stage), we will use only the REFs in the k th level of the REF tree. In other words, the REF tree will be a full binary tree of depth k . The key idea is to use a randomly chosen (k -level) leaf in order to authenticate a new message. With very high probability, we will never use the same leaf twice (in the second stage). This is the case since 2^k was chosen to be much larger than the number of messages to be ever signed.

The last detail to be mentioned is that the names of the nodes in the REF tree should be generated using a pseudorandom function (applied to the location of the node), rather than randomly. This way, we guarantee that the same names are always used for the same nodes, without having to store these names. (The idea to use pseudorandom functions to eliminate this part of the storage requirement in the GMR scheme was suggested by Leonid Levin.) Assuming the existence of one-way permutations, pseudorandom functions can be efficiently constructed [GGM].

Proving that the modified signature scheme is as secure as the original scheme requires a two-step argument. First, one should consider a hybrid scheme where the names of the nodes in the REF tree are generated randomly as in the original scheme. The proof of security for this scheme is conceptionally identical to the proof presented in [GMR], and is only a little more involved in details. Next, one uses the polynomially indistinguishability of the pseudorandom functions (from random functions), to show that the proposed scheme is as secure as the hybrid scheme.

Remark: the identity of the (k -level) leaf to be used can be computed by applying a pseudorandom function to the message to be signed. Thus, no coin tosses are required during the signing process. One can easily show that the security feature is preserved.

4. Speeding-up the Singing Process in the Factoring Based Implementation

The computational bottleneck in the signing/verifying process are, respectively, the computation of $F^{-1}(\cdot, \cdot)$ given the trapdoor and the computation of $F(\cdot, \cdot)$. In the factoring-based implementation presented in [GMR], $F^{-1}(\cdot, \cdot)$ was computed in $|N|^4$ steps, while $F(\cdot, \cdot)$ was computed in $|N|^3$ steps, where N is the modulus in use. In this section we show how to compute F^{-1} in $|N|^3$ steps.

In the implementation based on factoring, $N = pq$ is the product of two primes satisfying $p \equiv q \equiv 3 \pmod{4}$ and $p \not\equiv q \pmod{8}$. This way, each quadratic residue mod N has a unique square root which is a quadratic residue itself, and neither $+2$ nor -2 is a quadratic residue mod N . f_0 and f_1 are defined to be permutations over the set of quadratic residues mod N :

$$f_0(x) = x^2 \pmod{N}$$

and

$$f_1(x) = 4 \cdot x^2 \pmod{N}.$$

Let \sqrt{x} denote the square root of x which is a quadratic residue mod N . Thus, $f_0^{-1}(x) = \sqrt{x}$ and $f_1^{-1}(x) = \sqrt{x/4}$. Note that $\sqrt{4} \neq 2 \pmod{N}$. This observation is the key for proving that, unless factoring is easy, the permutations f_0 and f_1 defined above constitute a pair of (trapdoor) clawfree permutations. For more details see [GMR].

Recall the definition of $F^{-1}(\alpha, x)$

$$F^{-1}(\sigma_1 \sigma_2 \cdots \sigma_n, x) = f_{\sigma_1}^{-1}(f_{\sigma_2}^{-1}(\cdots f_{\sigma_n}^{-1}(x) \cdots))$$

The obvious way to compute $F^{-1}(\alpha, x)$ is by successively taking square roots. This results in $\Theta(|\alpha|)$ exponentiations. We present an alternative and faster way for computing $F^{-1}(\alpha, x)$. This way requires only a constant number of exponentiations, and is based on the following observation:

Let m denote the length of α , and $i(\alpha)$ denote the integer naturally encoded by the string α . Let $R_N(2^m, z)$ denote the 2^m th root of z modulo N (i.e. the quadratic residue obtained from z by repeating the $\sqrt{\cdot}$ operator m times). Then

$$F^{-1}(\alpha, x) = \frac{R_N(2^m, x)}{(R_N(2^m, 4))^{i(\alpha)}} \pmod{N}$$

(See example in the Appendix.)

Computing $R_N(2^m, z)$ reduces to computing it modulo each of the factors (i.e. computing $R_p(2^m, z)$ and $R_q(2^m, z)$) and applying the Chinese Remainder Theorem. Rather than computing $R_p(2^m, z)$ by successive applications of $\sqrt{\cdot}$, we compute it by one exponentiation:

(Pre)-compute, $\text{inv}_p(2) = (p+1)/4$, the “inverse” of 2 modulo $\phi(p) = p-1$.

(Note that $R_p(2, z) = z^{\text{inv}_p(2)}$.)

(Pre)-compute, $\text{inv}_p(2^m) = (\text{inv}_p(2))^m \pmod{p-1}$, the “inverse” of 2^m modulo $\phi(p)$.

Compute $r_p = z^{\text{inv}_p(2^m)} \pmod{p}$.

(Clearly, $r_p = R_p(2^m, z)$.)

Thus, computing F^{-1} reduces to a constant number of modular exponentiations.

5. Conclusions

Incorporating the two modifications into the GMR signature scheme, we get a scheme which is as secure as the original one, and is both “memoryless” and “practical”. (It is important to note that assuming the intractability of factoring, pseudorandom functions f can be implemented, and that evaluating $f(\beta)$ can be done in $n^3 \cdot |\beta|$ steps. Also

note that the length of the argument to f is $k = (\log_2 n)^2$.)

The “dependency on the past” in the original GMR scheme has nothing to do with the resolution of the “Paradox” mentioned in [GMR]. The paradox is resolved by observing that the adversary is uniform, while the real signer is “non-uniform”. For further discussion see [GMR].

Throughout this note, we have implicitly assumed that the length of the message to be signed is linear in the security parameter (n). However, the GMR scheme works also if this is not the case, while the running time (naturally) increases. In case we are using the factoring-based implementation and the message has length $m \gg n$, the signing time increases by an additive term of $O(m \cdot n)$ and the verification time increases by an additive term of $O(m \cdot n^2)$. A different approach suggested recently by Damgård is to first hash the message using *collision free hash functions* and then to sign the hashed value [D]. Interestingly, Damgård also shows that such hash functions can be constructed based on the existence of any pair of clawfree permutations.

ACKNOWLEDGEMENTS

Section 4 is joint work with Shafi Goldwasser. I would like to thank her for the collaboration, and for the permission to present the result here. I would also like to thank Louis Guillou for suggesting a simplification in the presentation of this result.

I am grateful to Shafi Goldwasser, Silvio Micali and Ron Rivest for many illuminating discussions concerning their signature scheme.

REFERENCES

- [D] Damgård, I.B., “Collision Free Hash Functions and Public Key Signature Schemes”, manuscript, 1986.
- [DH] Diffie, W., and Hellman, M.E., “New Directions in Cryptography”, *IEEE Trans. on Inform. Theory*, Vol. IT-22, No. 6, November 1976, pp. 644-654.
- [GGM] Goldreich, O., S. Goldwasser, and S. Micali, “How to Construct Random Functions”, *Proc. of 25th Symp. on Foundation of Computer Science*, 1984, pp. 464-479. To appear in *Jour. of ACM*.
- [GMR] Goldwasser, S., S. Micali, and R.L. Rivest, “A Paradoxical Solution to the Signature Problem”, *Proc. of 25th Symp. on Foundation of Computer Science*, 1984, pp. 441-448. A better version is available from the authors.
- [RSA] Rivest, R.L., Shamir, A., and Adleman, L., “A Method for Obtaining Digital Signatures and Public Key Cryptosystems”, *Comm. of the ACM*, Vol. 21, February 1978, pp. 120-126.

APPENDIX

Recall the definition of $F^{-1}(\alpha, x)$

$$F^{-1}(\sigma_1 \sigma_2 \cdots \sigma_n, x) = f_{\sigma_1}^{-1}(f_{\sigma_2}^{-1}(\cdots f_{\sigma_n}^{-1}(x) \cdots))$$

and

$$f_0^{-1}(x) = \sqrt{x} \text{ and } f_1^{-1}(x) = \sqrt{x/4},$$

where \sqrt{z} is the square root of z which is a quadratic residue modulo N .

Let us consider the case where the length of α is 2. We get,

$$F^{-1}(00, x) = f_0^{-1}(f_0^{-1}(x)) = \sqrt{\sqrt{x}} = \frac{R_N(2^2, x)}{(R_N(2^2, 4))^0} \pmod{N}$$

$$F^{-1}(01, x) = f_0^{-1}(f_1^{-1}(x)) = \sqrt{\sqrt{x/4}} = \frac{R_N(2^2, x)}{(R_N(2^2, 4))^1} \pmod{N}$$

$$F^{-1}(10, x) = f_1^{-1}(f_0^{-1}(x)) = \sqrt{\sqrt{x}/4} = \frac{R_N(2^2, x)}{(R_N(2^2, 4))^2} \pmod{N}$$

$$F^{-1}(11, x) = f_1^{-1}(f_1^{-1}(x)) = \sqrt{\sqrt{x/4}/4} = \frac{R_N(2^2, x)}{(R_N(2^2, 4))^3} \pmod{N}$$

PUBLIC-KEY SYSTEMS BASED ON THE DIFFICULTY OF TAMPERING (Is there a difference between DES and RSA?)

Yvo Desmedt * and Jean-Jacques Quisquater **

(*) Katholieke Universiteit Leuven, ESAT, Belgium ¹.

Current address: Université de Montréal, Dépt. IRO, Case postale 6128, succursale A, Montréal (Québec), H3C 3J7 Canada.

(**) Philips Research Laboratory Brussels, Avenue Van Becelaere, 2, B-1170 Brussels, Belgium.

Abstract

This paper proposes several public key systems which security is based on the tamperfreeness of a device instead of the computational complexity of a trapdoor one-way function. The first identity-based cryptosystem to protect privacy is presented.

EXTENDED ABSTRACT

1 Introduction

We first give three main motives for this paper and overview the presented ideas.

Since the invention of public-key systems by Diffie and Hellman almost all public-key systems proposed were based on some computational hard problems (e.g. factoring). It was however shown that it is not easy to design a secure public-key system based on computational hard problems. Examples of failures are the Lu-Lee system, the Merkle-Hellman knapsack scheme (and others) and the Matsumoto-Imai scheme. If we remark that the McEliece scheme is not enough analysed to be used, there do not exist fast public-key systems (the speed of RSA is today less than 64 kbit/sec.). This is one of the main reasons to come up with other public-key systems.

Bennett and Brassard remarked that it is not necessary to use computational complexity to design a public-key system. As an example they started from the uncertainty principle, which claims that some physical problems are very hard to solve (impossible to measure). Bennett and Brassard mentioned that their system would remain secure if $NP=P$ and if factoring would be easy. However the cryptosystems they proposed are today impractical. One can conclude that a second reason for this paper is to design cryptosystems which are not based on the assumption that trapdoor one-way functions exist.

The authenticity of the public key is a major problem in the set-up of a secure cryptosystem, certainly in the case of a large network. A nice solution was proposed by Shamir in 1984 called "identity-based cryptosystem". Instead of using the public key of the receiver (to encrypt in order to protect the privacy of a message), the name of the receiver is used as public key. The secret key of each user was calculated by an authority at the start-up of the system. (It is not excluded that the authority destroys itself after the start-up of the system.) Public-key systems, identity-based cryptosystems and their key generation are systematically explained in Fig. 1.

¹This research was done while the author was aangesteld navorser NFWO at the Katholieke Universiteit Leuven

- public-key system

"RSA"



- identity-based system

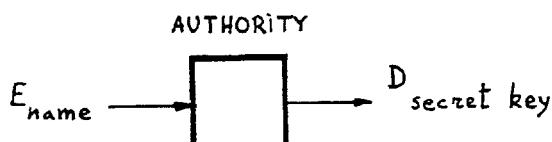


Figure 1: Key generation for public-key and identity-based systems

encryption device (E)

decryption device (D)

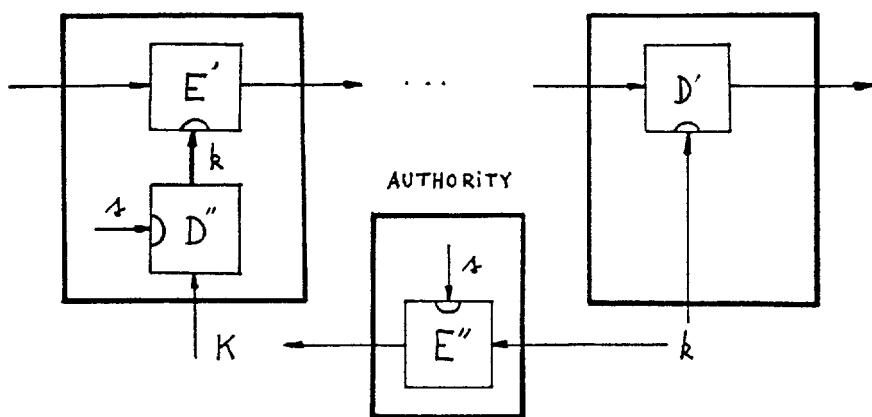


Figure 2: A first implementation of a public-key system

In our paper we start from the assumptions that hard conventional systems exist and that it is possible to make tamperfree devices. Remark that the first assumption is based on the complexity of algorithms, but seems acceptable, certainly if one takes into consideration that it is much harder to build trapdoor one-way public-key systems than conventional ones. Without the second assumption a lot of modern uses of cryptography would become unsecure. Indeed a secure system must be tamperfree otherwise an opponent can simply steal the secret key used in the system. Several practical systems start from this second assumption. E.g., a software copyright protection system proposed by NPL becomes completely insecure if tamperfree devices can not be build. Remark too that each identification method is at least partially based on some tamperfree system or card (see also Section 5).

Given two conventional cryptosystems and the existence of tamperfree implementations we propose in our full paper several public-key systems, and the first identity-based cryptosystem to protect privacy.

2 Public keys

2.1 The basic idea

Let us give an example of such a system. From now on we call E' , D' , E'' and D'' the encryption and decryption of respectively the first and second conventional cryptosystems. Special cases use the algorithm DES in encryption mode for E' and E'' or decryption mode for D' and D'' . To obtain a public-key system three devices are used: an encryption device (corresponding to the operation E), a decryption device (corresponding to the operation D) and a system which generates the public key starting from the secret key (corresponding to the operation G). Each user of the system generates a secret key k . He obtains his corresponding public key K by applying G on k , or $K = G(k)$. The device G is nothing but E'' with a supersecret key s (which in the best case nobody knows). The device G is tamperfree so that it is hard to find the key s . In this example the supersecret key s is used in all devices G .

2.2 Two implementations of such a public-key system

We now discuss two implementations to obtain such a public-key system (see also Fig. 2 and Fig. 3).

In the first example (see also Fig. 2) the decryption device (D) uses the secret key. In fact here D is equal to D' . The encryption device (E) uses as a black box the public key K . The system E is build up using E' and D'' . The box E is tamperfree. In the box E first D'' is used to find k , or $k = D''(K)$ using the supersecret key s . This last calculation is done inside E , and no trace of this calculation and its result can leak out to the outside world. In other words because the device E is tamperfree it is hard to find k . The encryption of messages is done by E' using the key k .

The described scheme can be used to protect, as a public-key system, the privacy and authenticity of messages as well to sign. To protect privacy the sender uses E with the public key of the receiver (although the receiver uses D with his secret key). Remark again that nevertheless the sender uses in fact the secret key of the sender, he cannot access it. To sign the sender uses D with his secret key (evidently redundancy is introduced in the

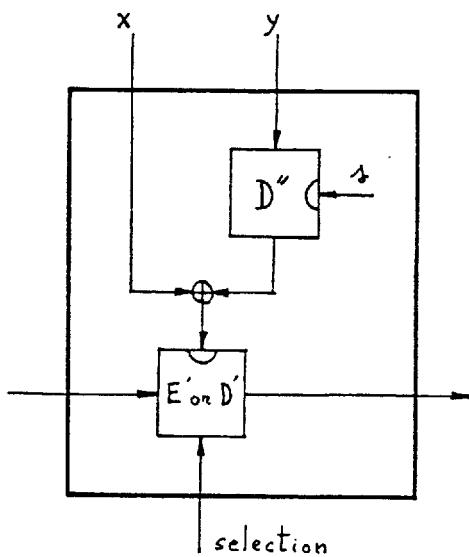


Figure 3: A second implementation of a public-key system

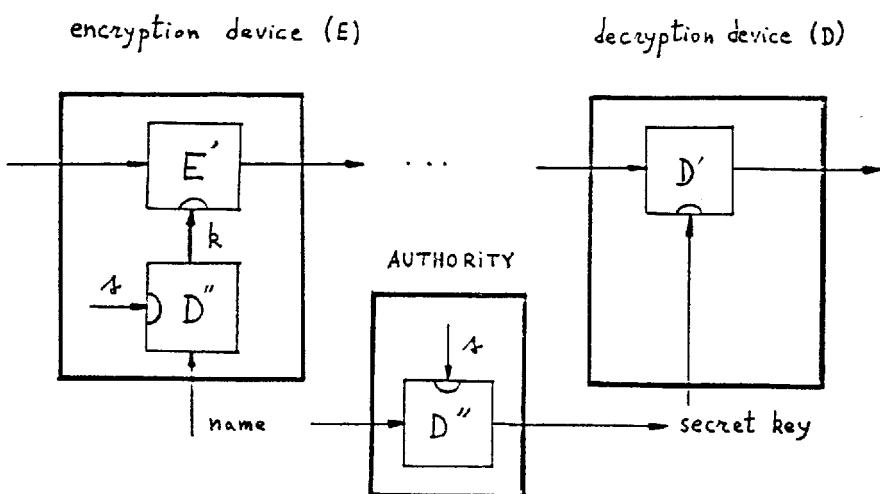


Figure 4: The first identity-based system to protect privacy

message). The receiver can check the signature (using the mentioned redundancy). The sender is the only one who could generate that signature.

The second implementation has the advantage that each user in the system has the same tamperfree device for encryption as well as for decryption. Let us describe such a system in some words. For this paragraph, we refer to Fig. 3. Let (T) be the tamperfree device used in the system. As for the first system, each user i generates a secret key k_i and a corresponding public key K_i . For that he uses the device G as already discussed. The device (T) contains E' , D' , D'' and the supersecret key s as described in Fig. 3. To send an encrypted message to a user B , a user A uses the device (T) in mode encryption and applies his secret key k_A to the input x and the public key K_B of the user B to the input y . To decrypt this message the user B uses the device (T) in mode decryption and applies his secret key k_B to the input z and the public key K_A of the user A to the input y . In these two phases, the effective key in use is the same but is unknown to the two parties. There are many variants to this scheme with the possibility of a session key, a.s.o. Let us remark that using a symmetric cryptosystem (sometimes called conventional system) together with such a symmetric implementation (the devices are the same for the encryption and the decryption) leads to an asymmetric cryptosystem (sometimes called public-key system).

3 Identity-based cryptosystem

By modifying a little bit previous examples it is no longer necessary to use public keys (or the public key of somebody is equal to his name or identification). The key generation machine G now is modified. The system G now uses D'' (with the supersecret key s) and the input of G is the name (or a sufficient identification of the person to be unique), the output is the secret key of the user (see also Fig. 4). In order to avoid frauds the uses of G are controlled by an authority. Each user can use G only once, and is only allowed to give as input something that corresponds with his identification (birth day, name of his father, name of company, ...). This is a first advantage because it avoids in large networks the authentication of the public key. This technique gives a first solution to a problem open by Adi Shamir, to propose an identity-based cryptosystem to protect privacy.

4 Security

In this section only necessary conditions in order to obtain a secure implementation are discussed. Sufficient conditions are still under research.

The system E'' has to be a secure cryptosystem such that all attacks fail in finding s by cryptanalytic methods. Therefore it is necessary that E'' is secure e.g. against an adaptive chosen text attack. The reader could wonder how an adaptive chosen text attack could be set up, certainly if an authority limits the use of the device G (as in the case of identity-based cryptosystem). The answer is that the adaptive aspect can be obtained if several users (which have e.g. special names) collaborate.

Evidently the cryptosystems E' , D' and D'' have also to be secure cryptosystems.

Another necessary condition is that the system may not have (or use) weak keys (a term introduced by Davies related to weak keys in DES) or similar weaknesses. Using a weak key there is no difference between an encryption and a decryption operation.

Indeed an asymmetry is required to obtain public-key systems. If not, this implies that everybody can generate signatures of an opponent using his public key, because E' will in fact internally use the secret key of the opponent and for weak keys this E' operation is the same as the D' operation. In general in order to protect signatures (with the described scheme) it must be hard to generate outputs of D' starting from outputs of E' . So semi-weak keys are also dangerous. The same remark holds for the protection of privacy. Otherwise everybody could decrypt message send to Bob, using Bob's public key for a similar reason.

5 Advantages, disadvantages and other aspects

A major advantage of the discussed systems is the speed. Using DES (and dropping weak keys) much faster public-key systems can be made. An important disadvantage of the system is that everybody who knows s can attack all users! However in some cases such a property is desired (by the authority), as in the case of communications between persons of a same company (e.g. a bank). In this context we remark that the key distribution problem in some large companies (when a normal conventional system is used), can be hard to solve.

Remark also that in previous discussions one can e.g. replace the supersecret key s , by some secret function. In the discussed example E' , D' , E'' and D'' are public known conventional algorithms. It is trivial to understand that the same holds if E' , D' , E'' and D'' are secret. In other words if some organization promotes secret algorithms, key distribution centers can be avoided and one can use the described public-key method. Indeed in order to maintain the secrecy of the used secret algorithms, the devices must be at least tamperfree.

Finally one can question that the described system is really a public-key system. To solve this problem one can use the well known Turing test. Suppose DES and RSA are used (to be mathematically correct n DESEs are used with n different keys), is it then possible to find in polynomial time (as function of n) if DES or RSA is used? It is well known that the answer is yes, using the Jacobi symbol in a known plaintext attack. In a secure implementation of RSA and DES it must be hard to make a difference between real random and the ciphertext in polynomial time. As a consequence if DES (in such public-key system) and RSA are used in a secure implementation, no difference can be observed in polynomial time.

Remark that in a part of our paper on the importance of good key scheduling schemes (1985, CRYPTO '85), we did not obtain a real public-key system as we do here, moreover, some of our assumptions there are the opposite of some assumptions here.

It is not too hard to find better schemes which satisfy some desired properties, some of these other schemes are still under research. For instance, in the context of tamperfree devices, it is possible to design claw-free functions with conventional cryptoalgorithms and thus to have very fast algorithms to sign documents (Rivest, Goldwasser, Micali, Goldreich).

Another advantage is that the above idea of identity-based cryptosystem can be used in a protocol in order to protect passports. Let us again start from the assumption that tamperfree devices and that conventional cryptosystems exist, where the decryption operation can not be obtained by applying polynomially the encryption operation. Remark that the assumption of tamperfree devices is also necessary in Shamir's protocol (presented

at the same conference). Indeed if an owner of a passport is able to find his corresponding secret (the square roots in Shamir's protocol), there is no protection against cloning. For very busy businessman or consultants or researchers it can be an important advantage to clone themselves, in order that the cloned one handles the public relation and other aspects, for which the original persons are too busy. If a difference has to be made between the identity of the person and his cloned version, the person himself is not allowed to know the secret corresponding with his secret. So tamperfree devices are necessary.

Our identification protocol is very similar to the one of Shamir, except that a different type of algorithm is used and that the country that is visited generates the random. Again we use the identity-based cryptosystem to protect signatures. Each country (e.g. Israel) distributes to other countries the E devices, containing their supersecret s . During use, a visitor (e.g. Alice) tells the officials her nationality (e.g. Israeli) and her identity. The country which she visits (e.g. Belgium) then uses the tamperfree device obtained from Israel and the name (identity) of Alice is used as key by that country (e.g. Belgium). Belgium generates then some random t and gives $E(t)$ to Alice. If Alice knows her secret key (obtained from her country: Israel), she is able to decrypt it and obtain t , which she gives to Belgium. If both t 's match Belgium accepts Alice identity. The disadvantage of this system is that 200 different kinds of machines are necessary (each for each country). The advantage is that each country relies on their own technology to avoid false passports made by other countries. A proof for the security of the discussed protocol is still under research.

6 Open Problems

A main open problem is to find an identity-based cryptosystem which protects privacy and which security is not based on the assumption of the existence of tamperfree devices.

Another open problem is to overcome the problem of the supersecret key s , mentioned in Section 5. Does there exist an identity-based cryptosystem to protect privacy which security is based on tamperfree devices and computational complexity and which use different supersecret s for different users. In other words that system would remain secure if the computational problem is solved, but the tamperfreeness is still valid, or if the reverse situation happened.

The authors have the impression that both mentioned open problems are strongly related.

Remarks

Other works, more or less related to this one, were made by M. E. Smid, R. E. Lennon, S. M. Matyas and C. H. Meyer, H. Beker and M. Walker.

Acknowledgement

The authors are grateful to Adi Shamir for the discussions related to Section 6.

A SECURE AND PRIVACY-PROTECTING PROTOCOL FOR TRANSMITTING PERSONAL INFORMATION BETWEEN ORGANIZATIONS

David Chaum & Jan-Hendrik Evertse

Centre for Mathematics and Computer Science
Kruislaan 413 1098 SJ Amsterdam The Netherlands

Abstract: A multi-party cryptographic protocol and a proof of its security are presented. The protocol is based on RSA using a one-way-function. Its participants are individuals and organizations, which are not assumed to trust each other. The protocol implements a "credential mechanism", which is used to transfer personal information about individuals from one organization to another, while allowing individuals to retain substantial control over such transfers.

It is proved that the privacy of individuals is protected in a way that is optimal against cooperation of all organizations, even if the organizations have infinite computational resources. We introduce a "formal credential mechanism", based on an "ideal RSA cryptosystem". It allows individuals a chance of successful cheating that is proved to be exponentially small in the amount of computation required. The new proof techniques used are based on probability theory and number theory and may be of more general applicability.

1. INTRODUCTION

The aim of this paper is to present in a formal way, and to prove the desired properties of, a multi-party cryptographic protocol called a "credential mechanism" that was introduced in [Ch 85]. In this section, the protocol is re-introduced and then an overview of the paper is given.

1.1. Credential mechanisms

A credential mechanism is a cryptographic protocol that provides for transfers of information about individuals between organizations. The information about individuals transferred will consist of *credentials* belonging to some fixed set. If individuals identify themselves to each organization

This research was supported in part by the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

tion essentially uniquely, such as by their name, date of birth, or some universal identification number, then credentials about an individual can be transferred between organizations without control by that individual. To give individuals control over such transfers, the credential mechanism allows individuals to use different *pseudonyms* with different organizations. Different individuals use different pseudonyms. Organizations have no more identifying information about individuals than these pseudonyms, and thus, from the point of view of the organizations, credentials can be linked to pseudonyms rather than to individuals.

When information about an individual is to be sent from one organization to another, the first organization issues a certificate, called a “credential on a pseudonym”, to the individual, showing that a particular credential applies to his pseudonym used with that organization; then the individual transforms this certificate into “the same credential on a (second) pseudonym” used with the second organization; and finally the individual shows this credential on the second pseudonym to the second organization.

The following is a precise description of the properties of the pseudonyms and credentials of the credential mechanism:

Property 1. The set of pseudonyms can be partitioned in two ways: into I-sets each containing the pseudonyms used by an individual and into O-sets each containing the pseudonyms known to an organization.

Property 2. Each I-set and each O-set have at most one pseudonym in common.

Property 3. For any individual, it is easy to compute a credential on a pseudonym if some organization has issued the same credential on a pseudonym belonging to the same I-set; otherwise computing that credential on that pseudonym is infeasible for that individual (unforgeability).

Property 4. The credential mechanism does not reveal any information to even cooperating organizations about how the pseudonyms are partitioned into I-sets (unlinkability).

By these properties, a credential mechanism guarantees each individual that different organizations (possibly by cooperation with other organizations and some other individuals) can never link the information they have about him. For an organization can only link the information about that individual to his pseudonym used with that organization; and by property 4 the credential mechanism does not reveal to the organizations which pseudonyms belong to which individual. A credential mechanism also protects each organization against individuals trying to convince it that certain credentials apply to them while this is in fact not true. This is so since by property 3, no individual is able to compute a credential on one of his pseudonyms if he did not previously get that credential on one of his other pseudonyms. Property 2 also protects organizations by, for example, preventing a credential issued by one organization on some pseudonym from being transformed into a credential on more than one pseudonym used with any particular organization.

To achieve properties 1-3, both pseudonyms and credentials on pseudonyms must be constructed in a special way. The credential mechanism must ensure that individuals construct their pseudonyms in this way. But, because of property 4, individuals cannot be required to show

organizations how they have constructed their pseudonyms. Therefore, credential mechanisms include a *validating part* which is a protocol by which individuals convince the organizations that they have constructed their pseudonyms correctly without revealing how they have done so.

1.2. Overview of the paper

In §2 we introduce some formalism about protocols and attacks on protocols (these are ways by which some of the participants of the protocol, possibly by cooperating, violate the rules of a protocol) which will serve as a mathematical framework in which properties of the credential mechanism can be stated and proved.

In §3 we describe a credential mechanism based on RSA with a single composite modulus N . The validating part is based on a “one-way” function, and all credentials on pseudonyms are RSA-signatures. Since only one modulus is used, and since it is not assumed that all organizations trust each other, a special organization participates in the credential mechanism, called a “signature authority”, which is the only organization that has to know the factorization of N used in making signatures. The signature authority is trusted by the other organizations—but not by the individuals—and is willing to provide suitable signatures requested by organizations. The signature authority also participates in the validating part. In §§3.1-3.4 an overview of the credential mechanism is given which can be read independently of §2. In §3.5 the credential mechanism is described by means of the formalism introduced in §2.

In §4 we prove that property 4 (the unlinkability of the pseudonyms) holds for the credential mechanism as described in §3.5 in the following respect: all information revealed by the credential mechanism about how the pseudonyms are partitioned into I-sets is already revealed by the *moments* that pseudonyms, credentials, etc. are issued by or shown to an organization. It is argued that this kind of information is revealed by any credential mechanism, so that our credential mechanism offers optimal unlinkability.

In §5 we introduce the “formal credential mechanism”. This mechanism is equivalent to the actual credential mechanism, except that it is based on an “ideal” RSA cryptosystem and an “ideal” one-way function. It is possible to establish a correspondence between messages in “ideal RSA” and messages in “real RSA” by means of a multiplicative homomorphism. Our formal credential mechanism is endowed with a computational model which precisely describes which “computations” each participant of the credential mechanism can perform. Thus our model of a formal credential mechanism can be compared with that used for RSA based ping-pong protocols in [EGS 85].

In §6 we state the main theorem about the formal credential mechanism: that in each possible attack on the formal credential mechanism, the probability that individuals will agree with the organizations about the use of pseudonyms which do not have properties 1, 2 and 3, has an upper bound which is an exponentially decreasing function of the number of computations done in the validating part. In §6 we also give an example of an attack by which individuals could try to agree with organizations about the use of pseudonyms in the formal credential mechanism

which do not satisfy properties 1, 2 and 3 mentioned in §1.1. With this attack we show that the upper bound given in the theorem cannot essentially be improved.

In §7, we prove the theorem mentioned in §6. Unlike the ping-pong protocols of [EGS 85], our formal credential mechanism models a protocol in which RSA is used with only a single modulus, but with different encryption and decryption exponents. Therefore our method of proof is entirely different from theirs. §§5-7 can be read independently of §4.

The reason that we prove properties 1, 2 and 3 for the formal credential mechanism instead of the actual credential mechanism, is that the following seems likely: for a proper choice of the composite modulus and the one-way function, it is computationally infeasible for an individual to agree with an organization about the use of a pseudonym, if that individual is not able to agree with the organization about the use of the corresponding pseudonym in the formal credential mechanism. An investigation of the correctness of this statement is beyond the scope of this paper.

In §8 we mention a few extensions of the credential mechanism.

2. PROTOCOLS AND ATTACKS

For the analysis of the credential mechanism provided in this paper, it is necessary to make clear what is meant by notions like “protocols” or “attacks on protocols”. In this section we give definitions of these notions which are modifications of those of DeMillo, Lynch and Merritt [DLM 82]. As noted before, this section need not be read before §§3.1-3.4 in which the credential mechanism is introduced.

2.1. Some probability theory

In the sequel we need some discrete probability theory which is introduced here.

We fix an enumerable set $\Omega = \{\omega_1, \omega_2, \dots\}$. To each ω_i in Ω we attach a real number $Pr[\omega_i]$ in the closed interval $[0, 1]$ such that

$$\sum_{i=1}^{\infty} Pr[\omega_i] = 1.$$

Subsets of Ω are called *events*. The *probability* of event \mathcal{Q} , denoted by $Pr[\mathcal{Q}]$, is given by

$$\sum_{\omega \in \mathcal{Q}} Pr[\omega].$$

The *conditional probability* of event \mathcal{Q} , given event \mathcal{B} , is defined by

$$Pr[\mathcal{Q} | \mathcal{B}] = \frac{Pr[\mathcal{Q} \cap \mathcal{B}]}{Pr[\mathcal{B}]}$$

if $\Pr[\emptyset] \neq 0$ and is not defined otherwise. When stating results involving conditional probabilities we always assume that these are defined, without explicitly mentioning this. If $\mathcal{E}_1, \dots, \mathcal{E}_r$ are events with $\Pr[\mathcal{E}_2 \cap \dots \cap \mathcal{E}_r] \neq 0$, then we have the elementary equality:

$$\Pr[\mathcal{E}_1 \cap \dots \cap \mathcal{E}_{r-1} | \mathcal{E}_r] = \prod_{i=1}^{r-1} \Pr[\mathcal{E}_i | \mathcal{E}_{i+1} \cap \dots \cap \mathcal{E}_r]. \quad (1)$$

A *stochastic variable* can be any function from Ω to any arbitrary set. Obviously, a stochastic variable can assume only finitely or enumerable many values. For each value x of the stochastic variable X , we put $\Pr[X=x] = \Pr[X^{-1}(x)]$, where $X^{-1}(x) = \{\omega \in \Omega : X(\omega) = x\}$. More generally, if X_1, \dots, X_t are stochastic variables with values x_1, \dots, x_t , respectively, we put

$$\Pr[X_1=x_1, \dots, X_t=x_t] = \Pr[X_1^{-1}(x_1) \cap \dots \cap X_t^{-1}(x_t)].$$

If confusion is not likely to arise, we shall abbreviate $\Pr[X_1=x_1, \dots, X_t=x_t]$ by $\Pr[x_1, \dots, x_t]$. We say that a stochastic variable X is *uniformly distributed* over a finite set Γ if

$\Pr[X=\gamma] = (\#\Gamma)^{-1}$ for each γ in Γ , where as usual, $\#\Gamma$ denotes the cardinality of Γ . A stochastic variable X is said to be *independent* of the stochastic variables Y_1, \dots, Y_r , if for all values x, y_1, \dots, y_r of X, Y_1, \dots, Y_r , respectively, we have $\Pr[x, y_1, \dots, y_r] = \Pr[x]\Pr[y_1, \dots, y_r]$.

Let $X: \Omega \rightarrow \mathcal{N}_X$ and $Y: \Omega \rightarrow \mathcal{N}_Y$ be stochastic variables and let $F: \mathcal{N}_X \rightarrow \mathcal{N}_Y$ be a function. We write $Y=F(X)$ if $\Pr[\{\omega \in \Omega : Y(\omega) \neq F(X(\omega))\}] = 0$. Obviously, if $\Pr[X=x] \neq 0$ then $\Pr[Y=y | X=x] = 1$ if $y = F(x)$ and 0 otherwise.

We denote the set-theoretic difference of the sets A and B by $A \setminus B$. For any set A , we denote by $F(A)$ the collection of *finite* subsets of A and by $F^+(A)$ the collection of finite ordered tuples with entries in A . Both the empty set and the empty tuple are denoted by \emptyset .

2.2. Protocols

Informally speaking, a protocol is a description of a *stochastic process* in which *participants*, belonging to a finite set of participants P , transmit messages between each other which belong to a finite or enumerable *message space* M . The time in this stochastic process will be an enumerable set of *moments*, $T = \{0, 1, 2, \dots\}$.

The elements of the set $M \times P \times P$ are named *steps*. We shall often denote steps (m, α, β) by $\alpha \rightarrow \beta : m$ (" α sends m to β ") if $\alpha \neq \beta$, and $\alpha : m$ (" α generates m ") if $\alpha = \beta$.

Let $X = M \times P \times P \times T$. For any subset y of X (including X itself) and subsets A, B of P and U of T , we put

$$y(A, B, U) = y \cap (M \times A \times B \times U).$$

Thus $y(A, B, U)$ describes a set of steps in which a participant of A sends a message to a participant in B (or a participant in A generates a message if $A \cap B \neq \emptyset$) during U . For convenience

we shall often abbreviate $\{\alpha\}$ by α and $P \setminus \{\alpha\}$ by P_α for $\alpha \in P$, while the subsets $\{t\}$, $\{0, \dots, t-1\}$, $\{0, \dots, t\}$, and $\{1, 2, \dots\}$ of T are for convenience written as t , $< t$, $\leq t$, and $t > 0$, respectively.

A stochastic subset of X will be a mapping from Ω to the collection of subsets of X . If Y is such a stochastic subset, then we define $Y(A, B, U)$ by $Y(A, B, U)(\omega) = Y(\omega)(A, B, U)$ for $A, B \subseteq P$ and $U \subseteq T$. Thus if y is a value of Y , then $y(A, B, U)$ is the corresponding value of $Y(A, B, U)$.

Definition 1. A protocol \mathcal{P} is a tuple $(P, M, p_\alpha : \alpha \in P)$, where P is the (finite) set of participants of \mathcal{P} , M is the (finite or enumerable) message space of \mathcal{P} and p_α is the choice for α .

A choice for α is a collection of functions $\{p_{\alpha,t} : t > 0\}$ such that

$$\begin{aligned} p_{\alpha,t} : X(\alpha, P, < t) \times X(P_\alpha, \alpha, < t) \times F(X(\alpha, P, t)) &\rightarrow [0, 1], \\ \sum_{s \in F(X(\alpha, P, t))} p_{\alpha,t}(x, y, s) &= 1 \text{ for all } x \in X(\alpha, P, < t), y \in X(P_\alpha, \alpha, < t). \end{aligned} \quad (2)$$

Thus a protocol can be considered as a collection of rules according to which communication between participants takes place. The actual communication is described in the *execution process*:

Definition 2. The execution process of the protocol $\mathcal{P} = (P, M, p_\alpha : \alpha \in P)$ is a stochastic subset $S = S_{\mathcal{P}}$ of $X = M \times P \times P \times T$ such that

- (i) for every $t \in T$, the values of $S(P, P, t)$ are finite sets;
- (ii) $Pr[S(P, P, 0) = \emptyset] = 1$;
- (iii) for each value s of S and $\alpha \in P$, $t \in T$ we have

$$\begin{aligned} Pr[s(\alpha, P, t) | s(\alpha, P, < t), s(P_\alpha, P, \leq t)] \\ = Pr[s(\alpha, P, t) | s(\alpha, P, < t), s(P_\alpha, \alpha, < t)] = p_{\alpha,t}(s(\alpha, P, < t), s(P_\alpha, \alpha, < t), s(\alpha, P, t)), \end{aligned} \quad (3)$$

where $p_\alpha = \{p_{\alpha,t} : t > 0\}$ is the choice for α .

Values of the execution process are called *executions*. If s is an execution and $(m, \alpha, \beta, t) \in s(\alpha, \beta, t)$ then we say that during execution s , (m, α, β) is executed by α at moment t , and that m is sent by α and received by β at moment t if $\alpha \neq \beta$ and generated by α at moment t if $\alpha = \beta$.

Each choice $p_\alpha = \{p_{\alpha,t} : t > 0\}$ for α can be considered as a stochastic system (for instance, a mathematical object like a probabilistic Turing machine or a physical object like a computer network) which outputs s at moment t with probability $p_{\alpha,t}(x, y, s)$ after it has been given input y and after it has given output x before moment t . The execution process describes the communication between these systems.

(i) states that at each moment, a participant executes only a finite number of steps. (ii) states that at moment 0, “nothing” has happened. (iii) states that whatever a participant does at moment t may depend on all messages which it sent or received before moment t , but is not influenced by the messages which it did not send or receive.

It might be possible that for some $\alpha \in P$ and $t \in T$, $S(\alpha, P, t)$ assumes the empty set. In that case α does “nothing” at moment t . Protocols with a finite running time, t_0 , say, can be

considered as protocols for which $\Pr[S(P, P, \{t \in T : t > t_0\}) = \emptyset] = 1$.

Our model differs from that of DeMillo, Lynch and Merritt [DLM 82] in that it satisfies the following assumptions:

- each message arrives at the same moment that it is sent, and at the same receiver to which the sender wanted to send its message;
- participant γ can find out no more about the communication between α and β than what he learns about this from his communication with α and β ; in other words, the communication channel between α and β does not “leak”;
- the sender and receiver know each others identity.

However, situations in which these assumptions are not valid, can be considered in our model by adding new participants. For instance, weaknesses in a computer network, causing messages to arrive too late or even at the wrong place, or leaking communication channels can be described in our model by considering the computer network or the communication channels as participants of the protocol. (Partial) sender- or recipient-anonymity can be dealt with in our model by giving each participant a number of *representatives*. The representatives communicate with each other and know each other's identities, and each participant communicates with its representatives. Apart from its own representatives, no participant has a priori knowledge about which representatives belong to which participant, and he might find out something about the relationship between the participants and representatives only from the messages which he receives during an execution of the protocol.

From any protocol $\mathcal{P} = (P, M, p_\alpha : \alpha \in P)$ it is possible to construct a new one, by dividing the participants into pairwise disjoint sets, and considering these sets as participants. Let Q be a partition of P , i.e. a collection of pairwise disjoint sets of which the union equals P . Using (1) and (3) it is possible to show that for each A in Q and each execution s of \mathcal{P} ,

$$\begin{aligned} \Pr[s(A, P \setminus A, t) | s(A, P \setminus A, < t), s(P \setminus A, P, \leq t)] \\ = \Pr[s(A, P \setminus A, t) | s(A, P \setminus A, < t), s(P \setminus A, A, < t)] \\ =: p_{A,t}(s(A, P \setminus A, < t), s(P \setminus A, A, < t), s(A, P \setminus A, t)). \end{aligned} \quad (4)$$

Thus $\mathcal{P}' = \{Q, M, p_A : A \in Q\}$ can be considered as a protocol in which $p_A = \{p_{A,t} : t > 0\}$ is the choice for A .

2.3. Attacks

In this subsection we consider *attacks* on protocols. If $p_\alpha = \{p_{\alpha,t} : t > 0\}$ and $p'_\alpha = \{p'_{\alpha,t} : t > 0\}$ are two choices for α then $p_\alpha \neq p'_\alpha$ means that for at least one t , the functions $p_{\alpha,t}$ and $p'_{\alpha,t}$ are different.

Definition 3. Let $\mathcal{P} = (P, M, p_\alpha : \alpha \in P)$ be a protocol and J a (possibly empty) subset of P . An attack by J on \mathcal{P} is a protocol $\mathcal{P}' = (P, M, p'_\alpha : \alpha \in P)$ such that

$$p_\alpha \neq p'_\alpha \text{ for } \alpha \in J, \quad p_\alpha = p'_\alpha \text{ for } \alpha \in P \setminus J.$$

We say that the participants in J are *cheating*.

Thus an attack can be interpreted as a violation of the rules of a protocol by some of the participants.

By considering computer networks or communication channels as participants, it is possible to describe attacks such as passive or active eavesdropping, or redirection of messages. By using representatives, as introduced in the previous subsection, our model allows attacks to be described in which some participant pretends to be somebody else.

In the security analysis of protocols, it is important to know whether non-cheating participants are able to find out if other participants are cheating. Below, a precise definition of *detection* of an attack is given.

Definition 4. Let $\mathcal{P} = (P, M, p_\alpha : \alpha \in P)$ be a protocol, J a subset of P and \mathcal{P}' an attack by J on \mathcal{P} . Denote by $S_{\mathcal{P}}, S_{\mathcal{P}'}$ the execution processes of \mathcal{P} and \mathcal{P}' , respectively, and let s be an execution of \mathcal{P}' . We say that $\alpha \in P \setminus J$ can *detect* \mathcal{P}' during s if

$$\Pr[S_{\mathcal{P}}(P_\alpha, \alpha, T) = s(P_\alpha, \alpha, T)] = 0,$$

whereas

$$\Pr[S_{\mathcal{P}'}(P_\alpha, \alpha, T) = s(P_\alpha, \alpha, T)] > 0.$$

One possible way by which α may detect an attack on the protocol \mathcal{P} is when at some moment t he receives messages from a participant β which are not *allowed* for \mathcal{P} . By this we mean that, given the communication between α and β before moment t , α received messages from β at moment t which he could not have received with positive probability during an execution of \mathcal{P} . (This need not imply that β is cheating). We now express this by means of the terminology introduced above. Let $S_{\mathcal{P}}$ denote the execution process of \mathcal{P} , and let s be an execution of (an attack on) \mathcal{P} . Thus $s(\alpha, \beta, < t)$ and $s(\beta, \alpha, < t)$ describe the communication between α and β before moment t , during s . Then the messages sent from β to α at moment t during execution s are *allowed* for \mathcal{P} if

$$\Pr[S_{\mathcal{P}}(\beta, \alpha, t) = s(\beta, \alpha, t) | S_{\mathcal{P}}(\beta, \alpha, < t) = s(\beta, \alpha, < t), S_{\mathcal{P}}(\alpha, \beta, < t) = s(\alpha, \beta, < t)] > 0. \quad (5)$$

In the situation that we are dealing with cryptographic protocols, participants often have limited computational abilities and therefore limited possibilities to cheat. To incorporate this in our model we assume that each participant α of some protocol \mathcal{P} has a collection of choices \mathcal{C}_α , each element of which satisfies (2). Then each attack $\mathcal{P}' = (P, M, p'_\alpha : \alpha \in P)$ on \mathcal{P} must satisfy

$$p'_\alpha \in \mathcal{C}_\alpha \text{ for } \alpha \in P. \quad (6)$$

3. DESCRIPTION OF THE CREDENTIAL MECHANISM

In §3.1 we explain the main idea behind the credential mechanism. §3.2 contains a more detailed overview of the credential mechanism. In §3.3 we give a concise description of the credential mechanism by means of a convenient protocol language. This description of the credential mechanism will be referred to throughout the paper. §3.4 contains additional comments on the credential mechanism. §§3.1-3.4 can be read independently of §2. Finally, in §3.5 we describe a mathematical model for the credential mechanism by means of the formalism introduced in §2. There, all the notions introduced in §§3.2-3.4 will be given a precise mathematical meaning.

3.1. Main idea behind the credential mechanism

Our credential mechanism is a cryptographic protocol based on RSA used with a single composite modulus N . The participants of the credential mechanism are individuals and organizations. N is public, i.e. known to all participants of the credential mechanism; only one special organization in the credential mechanism, the “signature authority” Z , knows how to factor N . The messages transmitted in the credential mechanism belong to \mathbb{Z}_N^* , which is the multiplicative group of all residue classes modulo N containing integers coprime with N . The order of \mathbb{Z}_N^* is as usual denoted by $\phi(N)$. Only Z has the ability to compute RSA-signatures on these messages. An RSA-signature on message m is a message $m^{\bar{c}} \bmod N$, where c is a public integer coprime with $\phi(N)$, and \bar{c} is an integer with $\bar{c} \equiv 1 \pmod{\phi(N)}$, which is known only to Z . The credentials will be public positive integers coprime with $\phi(N)$, belonging to a finite set C . The product of all elements of C is denoted by b .

Suppose i is an individual participating in the credential mechanism. The pseudonyms used by i are formed as follows: first i gets a number u from Z which i uses as a pseudonym with Z ; then i generates, for each organization A participating in the credential mechanism, a random number r_A from \mathbb{Z}_N^* . Then i uses as pseudonym with A the number $u_A \equiv ur_A^b \pmod{N}$, where b is the product of all credentials. For the organizations, these pseudonyms just look like random numbers in \mathbb{Z}_N^* ; this prevents different organizations from linking the pseudonyms used by the same individual.

A credential $c \in C$, applying to individual i , can be sent from organization A to organization B as follows:

- A asks Z to compute $d_A \equiv u_A^{\bar{c}} \pmod{N}$ for him. After A receives this he sends d_A to i . i checks if A sent him the correct message by verifying that $d_A^c \equiv u_A \pmod{N}$.
- i computes $d_B \equiv u_B^{\bar{c}}$ by first dividing d_A by $r_A^{b/c}$ and then multiplying with $r_B^{b/c}$. (Note that the exponent b/c is the product of all credentials except c so that i can compute it).
- i sends d_B to B and B verifies that $d_B^c \equiv u_B \pmod{N}$.

Individuals should never be able to show a credential to some organization if they did not

get this credential before from another organization. Individuals might be able to compute credentials themselves, without having gotten them from some organization, if they have the freedom to construct their pseudonyms u_A in another way than described above. If for instance i can use $u_B \equiv r_B^b \pmod{N}$ as a pseudonym with B instead of $ur_B^b \pmod{N}$, then he can compute each credential $u_B^c \pmod{N}$ by himself. Moreover, if two individuals i and i' use pseudonyms u_B and $u_{B'} \equiv u_B r^{b_B} \pmod{N}$ with B , where r is chosen by both individuals, then maybe i needs to get a credential from some other organization before he can compute one for B ; but once i is able to show a credential to B , i' is able to show the same credential to B , without having gotten it from some other organization.

To avoid the problems just mentioned, we extended our credential mechanism with a validating part, which forces individuals to form their pseudonyms u_A in the way described above, but does not require individuals to reveal more about how they have actually constructed their pseudonyms. In the validating part for pseudonym u_A , i sends messages to Z , which are constructed in a special way, by means of a one-way function. These messages are *candidates* for building blocks of a *validator*, to be issued by Z to i later on. Then Z selects at random half of these candidates, and asks i to show how he actually constructed these. If i constructed these properly, then Z computes the validator from the candidates of which i did not reveal the construction, and submits this validator to i . Because there is an RSA-signature in the validator, i could not have computed the validator by himself. Later, i transforms this validator into another validator which is shown to A together with pseudonym u_A . There must be a special relationship between u_A and this validator which is checked by A . If this relationship holds, A accepts u_A as a pseudonym. Z also checks this relationship, to make sure that later he does not issue credentials on improperly formed pseudonyms.

3.2. Overview of the credential mechanism

In the actual credential mechanism, it does not make a difference whether some individual communicates with some organization, thereby identifying himself with a pseudonym, or some *representative* of this individual communicates with that organization and identifies himself with that pseudonym. In our description of the credential mechanism, we shall assume that communication takes place between organizations and representatives. Thus the participants in the credential mechanism will be the signature authority Z , the organizations A_1, \dots, A_L , the individuals i_1, \dots, i_R and the individuals' representatives, where no representative represents more than one individual and each individual has different representatives for the communication with different organizations.

Initialization. The notation introduced here will be used throughout the remainder of this paper and will not be re-introduced later. Before the actual credential mechanism starts, Z chooses two large primes P and Q and keeps these secret. Then Z makes the modulus $N = PQ$ public. After that, Z makes public:

a set $C = \{c_1, \dots, c_K\}$ of positive integers, to be used as credentials;

pairs of primes $(p_1, q_1), \dots, (p_L, q_L)$ used to make validators for A_1, \dots, A_L , respectively; an even integer $n > 4$, the *security parameter*, which determines the amount of work done in the validating part; a positive integer a , elements m_1, \dots, m_n of \mathbb{Z}_N^* and a “one-way function” $f: \mathbb{Z}_N^* \rightarrow \mathbb{Z}_N^*$, which are all used in the validating part.

It is assumed that the numbers $\phi(N)$, $a, p_1, \dots, p_L, q_1, \dots, q_L, c_1, \dots, c_K$ are pairwise coprime, and that $p_1, \dots, p_L, q_1, \dots, q_L$ are larger than $\frac{1}{2}n$. (This last condition is just a technical one, needed later in some of our arguments). We put $b_j = p_j^2 c_1 \cdots c_K$ for $j = 1, \dots, L$. We shall not specify f . We assume that f is “close to random” and has at least the properties that anybody can easily compute it and that it is not a homomorphism and not invertible by any participant of the credential mechanism (possibly apart from Z). (One could take

$f(x) = x^{\pi_1} + x^{\pi_2} + 1 \pmod{N}$ for certain prime numbers π_1 and π_2 or another polynomial which is not a power of a linear polynomial. But we do not know if such a choice for f is good enough). All computations, relations etc. in the credential mechanism will be modulo N , and therefore the suffix “mod N ” will be omitted. If b is a public number, used for exponentiation in RSA then the corresponding secret exponent, known only to Z , is denoted by \bar{b} . Thus $\bar{b}\bar{b} \equiv 1 \pmod{\phi(N)}$. In general, all expressions in exponents without a bar will be integers computable by all participants of the credential mechanism, while only Z is able to compute exponents with a bar.

Below we give an overview of the whole credential mechanism. It is built up from *subprotocols* (i.e. collections of steps to be executed in the credential mechanism) which are indicated by roman digits and the individual or representative, organization and credential involved. We describe that part of the credential mechanism in which i_k is involved. The representative of i_k communicating with A_j is denoted by g_j .

In most of the subprotocols, some participant checks if the messages which it received satisfy some special relationship. The check results in ‘true’ if this relationship holds, and ‘false’ otherwise. No steps in a subprotocol are executed after one of its checks has resulted in ‘false’.

O(i_k): i_k gets a pseudonym from Z .

- i_k asks Z for a pseudonym.
- Z checks if a pseudonym can be issued to i_k , and if this is the case, chooses u_k from \mathbb{Z}_N^* and sends this as a pseudonym to i_k .

I(i_k, A_j): i_k gets a validator from Z which will be shown to A_j in a modified form. Put

$$p = p_j, q = q_j, b = b_j.$$

- i_k asks Z for a validator for A_j and Z decides if this can be issued.
- i_k chooses numbers r_l, s_l ($l = 1, \dots, n$) at random from \mathbb{Z}_N^* . Then he computes $\tilde{r}_l := m_l r_l^a$ and $a_{kl} := f(u_k \tilde{r}_l) s_l^{pq}$ for $l = 1, \dots, n$ and sends all a_{kl} to Z . (A uniform choice from \mathbb{Z}_N^* can be obtained by choosing an integer uniformly from $\{1, \dots, N\}$, by doing another choice in the unlikely event that this integer has a factor in common with N and so on, until an integer coprime with N is chosen).

The numbers a_{kl} are the candidates for the building blocks of the validator which will be issued

by Z later on. These candidates are constructed in a special way but by the uniform choice of the numbers r_l, s_l they look just like random elements of \mathbb{Z}_N^* . This special way of construction should give the organizations sufficient security; while the random choice of the numbers r_l, s_l is meant for giving individuals the desired privacy.

- Z selects at random a subset S of $\{1, \dots, n\}$ of cardinality $\frac{1}{2}n$ and asks i_k to show, for each l in S , numbers r_l and s_l such that $a_{kl} = f(u_k(m_l r_l^q)^b) s_l^{pq}$. Then Z checks if a_{kl} and the numbers r_l, s_l sent to it by i_k satisfy these relationships for $l \in S$.
- If these relationships are satisfied, then Z computes the validator

$$v_{kj} := u_k^{\bar{p}^2} \left\{ \prod_{l \notin S} a_{kl} \right\} \bar{p}^q$$

and sends this to i_k . (\bar{p}^2 denotes a secret exponent corresponding to p^2). Then i_k checks if this validator has the proper form by verifying that

$$v_{kj}^{p^2 q} = u_k^q \left\{ \prod_{l \notin S} a_{kl} \right\} p^q .$$

$\Pi(g_j, A_j)$. i_k forms a pseudonym u_{g_j} and a validator \tilde{v}_{g_j} from u_k and v_{kj} , and lets his representative g_j show these to A_j . Let again $p = p_j, q = q_j, b = b_j$, and put $t_1 = 1$.

- Let σ be a permutation of $\{1, \dots, n\}$ with $\sigma(\{1, \dots, n\}) = S$, where S is the set chosen by Z in $I(i_k, A_j)$. Let $r'_l = \tilde{r}_{\sigma(l)}, s'_l = s_{\sigma(l)}$ for $l = 1, \dots, \frac{1}{2}n$.

Put $r_{g_j} := r'_{1j}$. i_k computes $u_{g_j} := u_{g_j}^b$, which will be the pseudonym used with A_j , and $w_{g_j} = \prod_{l=2}^{\frac{1}{2}n} u_k r_l^b$. Then i_k sends u_{g_j} and w_{g_j} to g_j , g_j sends these to A_j and finally, A_j sends these numbers to Z .

- A_j and Z check if $u_{g_j} \neq u_{g'_j}$ and $u_{g_j} w_{g_j} \neq u_{g'_j} w_{g'_j}$ for each pair $u_{g'_j}, w_{g'_j}$ sent to A_j by some other representative g'_j .

Each individual i_k forms a validator to be shown to A_j from the validator issued by Z in $I(i_k, A_j)$. The last check prevents different individuals from computing their validators for A_j from the same validator issued by Z .

- i_k computes $t_l := r'_{l1} r_{g_j}^{-1}$ for $l = 2, \dots, \frac{1}{2}n$, and

$$\tilde{v}_{g_j} = r_{g_j}^{b/p^2} (s'_{1j} \cdots s'_{\frac{1}{2}n})^{-1} \times v_{kj}$$

and sends these numbers to g_j . Then g_j shows these numbers to A_j and A_j sends them to Z .

A straightforward computation shows that

$$\tilde{v}_{g_j} = (u_{g_j}^b)^{\bar{p}^2} \left\{ \prod_{l=1}^{\frac{1}{2}n} f(u_{g_j}^b t_l^b) s_l^{pq} \right\} \bar{p}^q (s'_{1j} \cdots s'_{\frac{1}{2}n})^{-1} = u_{g_j}^{\bar{p}^2} \left\{ \prod_{l=1}^{\frac{1}{2}n} f(u_{g_j}^b t_l^b) \right\} \bar{p}^q .$$

- Both A_j and Z check if

$$w_{g_j} = \prod_{l=2}^{i_n} u_{g_j} t_l^{p^2 q}, \text{ and } \tilde{v}_{g_j}^{p^2 q} = u_{g_j}^q \left\{ \prod_{l=1}^{i_n} f(u_{g_j} t_l^b) \right\}^p.$$

If these conditions hold then both A_j and Z accept u_{g_j} .

III(g_j, A_j, c): A_j issues credential c on the pseudonym of representative g_j .

- g_j asks A_j for credential c on his pseudonym at the request of i_k . Then A_j checks if this can be issued and, after having positively decided on this, asks Z to compute credential c on the pseudonym u_{g_j} of g_j .
- Z checks if u_{g_j} has been shown with a proper validator, and if so, computes $d_{g_j} := u_{g_j}^{\bar{c}}$ and sends this credential to A_j . A_j issues this credential to g_j and g_j gives it to i_k .
- i_k checks if $d_{g_j}^c = u_{g_j}$, and computes $d_k := d_{g_j} r_{g_j}^{-b_j/c} = u_k^{\bar{c}}$.

IV(g_h, A_h, c): g_h shows credential c on his pseudonym with A_h .

- i_k computes $d_{g_h} = d_k r_{g_h}^{b_k/c} = u_{g_h}^{\bar{c}}$ and sends this to g_h . g_h shows this credential to A_h , and A_h checks if $d_{g_h}^c = u_{g_h}$.

3.3. Concise description of the credential mechanism.

We shall use the notation of §3.2. Further notation introduced here will be used later without re-introduction. For reference purposes we describe the credential mechanism by means of a “protocol language” introduced below.

Protocol language.

$\alpha \rightarrow \beta : m$

α sends message m to β

$\alpha \rightarrow \beta \rightarrow \gamma \rightarrow \dots \rightarrow \kappa : m$

α sends m to β , β sends m to γ ... to κ

$\alpha : \text{chooses } \gamma \text{ from } \Gamma$

α chooses an element γ from the set Γ in some unspecified way

$\alpha : \text{chooses } \gamma \text{ uniformly from } \Gamma$

α chooses γ uniformly from the set Γ and independently of all other steps executed at the same moment or before

$\alpha : \text{computes } m := (\text{expression})$

α computes the expression and assigns the value to m

$\alpha \rightarrow \beta \rightarrow \dots \rightarrow \kappa : \text{checks } P$

α computes the value of the predicate P , which is either ‘true’ or ‘false’. Then this value is sent from α to β , from β to ... to κ . If this value is ‘true’ the subprotocol continues; otherwise the subprotocol stops immediately after the check.

Description of the credential mechanism. We shall describe that part of the credential mechanism in which individual i_k is involved. The representative of i_k communicating with A_j is

denoted by g_j , for $j = 1, \dots, L$. First we describe the initialization, before the actual credential mechanism starts.

Initialization

Z : chooses large primes P, Q

computes $N := PQ$

chooses even integer $n > 4$ and one-way function f

chooses m_1, \dots, m_n from \mathbb{Z}_N^*

chooses positive integers $c_1, \dots, c_K, a, p_1, \dots, p_L, q_1, \dots, q_L$,

such that all these numbers are pairwise coprime and coprime with $\phi(N)$

and $p_1, \dots, p_L, q_1, \dots, q_L$ are primes larger than $\frac{1}{2}n$

computes $\bar{c}_1, \dots, \bar{c}_K, \bar{a}, \bar{p}_1, \dots, \bar{p}_L, \bar{q}_1, \dots, \bar{q}_L$

Z makes public: $N, n, f, m_1, \dots, m_n, c_1, \dots, c_K, a, p_1, \dots, p_L, q_1, \dots, q_L$

O(i_k). Z gives pseudonym to i_k .

1. $i_k \rightarrow Z$: asks for pseudonym
2. $Z \rightarrow i_k$: checks if pseudonym can be given
3. Z : chooses u_k from \mathbb{Z}_N^*
4. $Z \rightarrow i_k$: u_k

I(i_k, A_j). Z gives i_k validator for A_j .

Put $p = p_j, q = q_j, b = b_j$.

1. $i_k \rightarrow Z$: asks for validator for A_j
2. $Z \rightarrow i_k$: checks if a validator for A_j can be issued
3. i_k : chooses $(r_l, s_l) : l = 1, \dots, n$ uniformly from $(\mathbb{Z}_N^*)^{2n}$
4. i_k : computes $\tilde{r}_l := m_l r_l^q, a_{kl} := f(u_k r_l^b) s_l^{pq} \quad (l = 1, \dots, n)$
5. $i_k \rightarrow Z$: $a_{kl} \quad (l = 1, \dots, n)$
6. Z : chooses S uniformly from $\{\mathcal{S} \subset \{1, \dots, n\} : |\mathcal{S}| = \frac{1}{2}n\}$
7. $Z \rightarrow i_k$: S
8. $i_k \rightarrow Z$: $r_l, s_l \quad (l \in S)$
9. $Z \rightarrow i_k$: checks $a_{kl} = f(u_k(m_l r_l^q)^b) s_l^{pq}$ for $l \in S$
10. Z : computes $v_{kj} := u_k^p \left(\prod_{l \notin S} a_{kl} \right)^{pq}$
11. $Z \rightarrow i_k$: v_{kj}
12. $i_k \rightarrow Z$: checks $v_{kj}^{1/q} = u_k^q \left(\prod_{l \notin S} a_{kl} \right)^p$

II(g_j, A_j). g_j shows pseudonym with validator to A_j .

Let r_l, s_l be the numbers chosen in step 3 of I(i_k, A_j), let S the set chosen by Z in step 6, and let σ be the permutation of $(1, \dots, n)$ defined by $\sigma(1) < \sigma(2) < \dots < \sigma(\frac{1}{2}n), \sigma(\frac{1}{2}n + 1) < \dots < \sigma(n)$ and $\sigma(\{\frac{1}{2}n + 1, \dots, n\}) = S$. Put $p = p_j, q = q_j, b = b_j$ and $t_1 = 1$.

1. i_k : computes $r_{g_j} := \tilde{r}_{\sigma(1)}, u_{g_j} := u_k r_{g_j}^b, w_{g_j} := \prod_{l=2}^{\frac{1}{2}n} u_k \tilde{r}_{\sigma(l)}^b$

2. $i_k \rightarrow g_j \rightarrow A_j \rightarrow Z : u_{g_j}, w_{g_j}$
3. $Z \rightarrow A_j \rightarrow g_j \rightarrow i_k : \text{checks } u_{g_j} \neq u_{g'_j} \text{ and } u_{g_j} w_{g_j} \neq u_{g'_j} w_{g'_j} \text{ for any pair } u_{g'_j}, w_{g'_j} \text{ submitted to } A_j \text{ by another representative } g'_j.$
4. $i_k : \text{computes } t_l := \frac{\tilde{r}_{\sigma(l)}}{r_{g_j}} \quad (l = 2, \dots, \frac{1}{2}n),$
 $\tilde{v}_{g_j} := r_{g_j}^{b/p^2} \left(\prod_{l=1}^{\frac{1}{2}n} s_{\sigma(l)} \right)^{-1} v_{kj} \quad (= u_{g_j}^{\tilde{p}^2} \left(\prod_{l=1}^{\frac{1}{2}n} f(u_{g_j} t_l^b) \right)^{\tilde{p}q})$
5. $i_k \rightarrow g_j \rightarrow A_j \rightarrow Z : t_l \quad (l = 2, \dots, \frac{1}{2}n), \tilde{v}_{g_j}$
6. $Z \rightarrow A_j \rightarrow g_j \rightarrow i_k : \text{checks } \tilde{v}_{g_j}^{p^2q} = u_{g_j}^q \left(\prod_{l=1}^{\frac{1}{2}n} f(u_{g_j} t_l^b) \right)^p, w_{g_j} = \prod_{l=2}^{\frac{1}{2}n} u_{g_j} t_l^b$

III(g_j, A_j, c). A_j issues credential c to g_j.

Let r_{g_j}, u_{g_j} have the same meaning as in step 1 of *II(g_j, A_j)*.

1. $i_k \rightarrow g_j \rightarrow A_j : \text{asks for credential } c$
2. $A_j \rightarrow g_j \rightarrow i_k : \text{checks if } c \text{ can be issued}$
3. $A_j \rightarrow Z : u_{g_j}$
4. $Z \rightarrow A_j : \text{checks if it has received proper validator for } u_{g_j}$
5. $Z : \text{computes } d_{g_j} := u_{g_j}^{\bar{c}}$
6. $Z \rightarrow A_j \rightarrow g_j \rightarrow i_k : d_{g_j}$
7. $i_k \rightarrow g_j \rightarrow A_j \rightarrow Z : \text{checks } d_{g_j}^c = u_{g_j}$
8. $i_k : \text{computes } d_k := r_{g_j}^{-b_j/c} d_{g_j} \quad (= u_k^{\bar{c}})$

IV(g_h, A_h, c). g_h shows credential c to A_h.

d_k will be the number computed in step 8 of *III(g_j, A_j, c)* and u_{g_h}, r_{g_h} will be the numbers computed in step 1 of *II(g_h, A_h)*.

1. $i_k : \text{computes } d_{g_h} := d_k r_{g_h}^{b_h/c} \quad (= u_{g_h}^{\bar{c}})$
2. $i_k \rightarrow g_h \rightarrow A_h : d_{g_h}$
3. $A_h \rightarrow g_h \rightarrow i_k : \text{checks } d_{g_h}^c = u_{g_h}$

3.4. Comments

This subsection contains some remarks and assumptions on the credential mechanism.

Outside world. Participants of the credential mechanism may not only communicate with each other, mostly over some computer network, but also with the “outside world”. For instance, events happening in the outside world may influence an organization’s decision to issue a credential, or an individual’s decision to ask for or show a credential. For that reason, the outside world should be considered as another participant of the credential mechanism.

Time order of the steps. The credential mechanism is built up from the subprotocols in the set Π consisting of

$$O(i_k), I(i_k, A_j), II(g_j, A_j), III(g_j, A_j, c), IV(g_j, A_j, c)$$

for all individuals i_k , representatives g_j and organizations A_j . In each subprotocol \mathcal{P} of Π , certain relationships between messages are checked, resulting in the value ‘true’ if the relationship holds and ‘false’ otherwise. We agreed that no steps in \mathcal{P} are executed after one of the checks in \mathcal{P} resulted in ‘false’. We say that a subprotocol in Π has been *properly executed* if it has been executed such that none of its checks resulted in ‘false’. We require that subprotocols are executed without any interruption. We allow that different subprotocols in Π run in parallel or in overlapping time intervals, however with the following obvious

Consistency restriction:

- steps in $I(i_k, A_j)$ are executed only after $O(i_k)$ has been properly executed;
- if g_j is the representative of i_k communicating with A_j , then steps in $II(g_j, A_j)$ are executed only after $I(i_k, A_j)$ has been properly executed;
- steps in $III(g_j, A_j, c)$ are executed only after $II(g_j, A_j)$ has been properly executed;
- steps in $IV(g_h, A_h, c)$ are executed only after $III(g_j, A_j, c)$ has been properly executed for some j in $\{1, \dots, L\}$ and some g_j representing the same individual as g_h .

There might be more restrictions on the time order in which the steps in the credential mechanism are executed, for instance:

- pseudonyms, validators or credentials must be issued before some “deadline”;
- the time passing between the moment that an individual or its representative gets a pseudonym, validator or credential from an organization, and the moment that another representative of this individual shows this to another organization, depends statistically on events in the outside world;
- the decision of an organization about issuing a particular credential on a pseudonym depends on whether other credentials have been shown on that pseudonym, on the number of times that that credential has been issued before on other pseudonyms, or on messages received from the outside world.

Simple credential mechanism. A possible way to state properties of the credential mechanism, is to compare it with the following simple credential mechanism.

When some organization agrees to give a credential c to an individual, that organization just gives the individual’s representative the number c , without any cryptographic protection. Later on, another representative of that individual shows this number c to another organization. The validating part of this credential mechanism runs as follows: individual i_k asks Z for permission to communicate with organization A_j . If Z gives this permission then he sends a special validator v_j to i_k which is independent of i_k . Later, i_k initiates the conversation with A_j by letting his representative show v_j to A_j .

Obviously, this simple credential mechanism does not give the organizations any security;

individuals are always able to show a validator or credential to some organization by means of their representatives without having got this from another organization. This simple credential mechanism would work well if none of the individuals would ever cheat; compared with other possible credential mechanisms, it gives individuals maximal possibilities of getting validators or credentials and maximal freedom in choosing the moments at which they show these to some organization.

Main condition: except for cheating, our credential mechanism should give individuals as many abilities as the simple credential mechanism described above, more precisely:

the credential mechanism must offer each participant the same freedom in communicating with the outside world, and each non-cheating individual the same possibilities of getting validators or credentials and the same freedom in deciding when to ask for or show these to some organization, as the simple credential mechanism described above.

Checks. The checks done in the credential mechanism are divided in two parts: the *decision checks* in which organizations check if they can issue a pseudonym, validator or credential to some individual or representative; and the other, so-called *security checks* by which participants may detect attacks. In any execution of the credential mechanism, no step in a subprotocol \mathcal{P} in II is executed after some check in \mathcal{P} has resulted in ‘false’; we allow however that the execution of a subprotocol is repeated after a security check by an individual has resulted in ‘false’. If no participant cheats, then all security checks will give the value ‘true’; only the checks in steps 3 of the sets $II(g_j, A_j)$ may give the value ‘false’ with very small probability.

We now briefly discuss the checks in steps 3 of the sets $II(g_j, A_j)$. These checks differ from the other security checks in that they compare messages which were sent by different representatives to an organization. Without these checks, two individuals, i_1 and i_2 , say, can successfully conspire in the following way against A_j : i_1 follows the validating part and lets his representative g_{1j} show u_1, w_1 to A_j in step 2, and $v_1, t_{12}, \dots, t_{1,\frac{1}{2}n}$ in step 5 of $II(g_{1j}, A_j)$, where

$$w_1 = \prod_{l=2}^{\frac{1}{2}n} u_1 t_{1l}^b, \quad v_1 = u_1^{\frac{1}{2}} \left\{ f(u_1) \prod_{l=2}^{\frac{1}{2}n} f(u_1 t_{1l}^b) \right\}^{pq}.$$

($b = b_j$, $p = p_j$ and $q = q_j$ have the same meaning as in §3.3). i_2 chooses $u_2 = u_1 t_m^b$ for some m in $\{1, \dots, \frac{1}{2}n\}$, where $t_{11} := 1$, and computes $w_2 := u_1 w_1 / u_2$, $v_2 := t_{1m}^b / p^2 v_1$ and $t_{2l} := t_{1,\tau(l)} / t_{1m}$ for $l = 2, \dots, \frac{1}{2}n$, where τ is a permutation of $\{1, \dots, \frac{1}{2}n\}$ with $\tau(m) = 1$. Then i_2 lets his representative g_{2j} show u_2, w_2, v_2 and t_{2l} ($l = 2, \dots, \frac{1}{2}n$) to A_j in $II(g_{2j}, A_j)$. All security checks by A_j in $II(g_{2j}, A_j)$ are satisfied, except that $u_1 w_1 = u_2 w_2$, hence without the check in step 3, A_j would have accepted both u_1 and u_2 . It is easy to check that credentials issued on u_1 can be easily transformed into credentials on u_2 and vice-versa.

Suppose that A_j received u_1 and w_1 from representative g_{1j} in step 2 of $II(g_{1j}, A_j)$. Problems might arise when there is a dispute in which A_j claims that it received numbers u_2 and w_2 from another representative g_{2j} such that $u_1 = u_2$ or $u_1 w_1 = u_2 w_2$ and refuses to accept pseudonym u_1 , and that g_{1j} does not accept this refusal. Below we describe a method to deal with

such a dispute with the help of a mutually trusted referee, in such a way that g_{1j} does not have to reveal which individual it represents. We assume that $\frac{1}{2}n$, where n is the security parameter, is coprime with all the primes p_j, q_j and credentials c introduced in §3.2.

For $i = 1, 2$, let u_i and w_i be the numbers sent to A_j in step 2, $v_i = \tilde{v}_{g_{ij}}$ the validator computed in step 4 and τ_i the tuple $(t_{i1}, \dots, t_i, v_n)$ sent to A_j in step 5, where all steps belong to $II(g_{ij}, A_j)$. In the case of a dispute as described above, each g_{ij} sends u_i, w_i, v_i and τ_i to the referee. First the referee computes the tuples $\sigma_i = (u_i, u_i t_{il}^b : l = 2, \dots, \frac{1}{2}n)$ for $i = 1, 2$. Note that none of these tuples needs contain distinct entries. Then the referee checks, if indeed $u_1 = u_2$ or $u_1 w_1 = u_2 w_2$ and

$$u_i^{p^2q} = u_i^q \left[\prod_{l=1}^{\frac{1}{2}n} f(u_i t_{il}^b) \right]^p, \quad w_i = \prod_{l=2}^{\frac{1}{2}n} u_i t_{il}^b \quad \text{for } i = 1, 2. \quad (7)$$

If (7) holds and the two tuples σ_1 and σ_2 can be made equal by reordering, then the referee concludes that the individual represented by g_{1j} conspired with somebody else and decides that A_j does not have to accept u_1 as a pseudonym. His motivation for this conclusion is the following: since for any integer d with $\gcd(d, \phi(N)) = 1$ the mapping $x \mapsto x^d$ is bijective, the tuple σ_1 can be considered as a random tuple in $(\mathbb{Z}_N^*)^{\frac{1}{2}n}$. The number of tuples σ_1 in $(\mathbb{Z}_N^*)^{\frac{1}{2}n}$ which contain u_1 and whose other $\frac{1}{2}n - 1$ entries have product w_1 is equal to $\phi(N)^{\frac{1}{2}n-2}$. If g_{1j} , or the individual which it represents, did not reveal the set σ_1 before showing it to the referee, then somebody else could have generated u_2 and τ_2 such that σ_1 equals σ_2 after reordering, only by correctly guessing a tuple which is apart from its order equal to σ_1 , while knowing no more than u_1 and w_1 . But the chance of such a correct guess is at most $(\frac{1}{2}n)! \times \phi(N)^{2 - \frac{1}{2}n}$. In practical situations when N has about 200 digits, this probability can be neglected.

If σ_1 can not be made equal to σ_2 by reordering, then the referee accuses signature authority Z of cheating. From $u_1 w_1 = u_2 w_2$ it follows that by cooperation, g_{1j} and g_{2j} can compute $(u_1 u_2^{-1})^b$. (Of course, g_{1j} and g_{2j} can compute this also if $u_1 = u_2$). The referee assumes that participants in the credential mechanism other than Z have only a negligibly small chance of learning at the same time $u_1, r \in \mathbb{Z}_N^*$ and $u_2 := u_1 r^b$, tuples $\tau_i = (t_{i1}, \dots, t_i, v_n)$ of \mathbb{Z}_N^* and validators v_i satisfying (7) for $i = 1, 2$, such that no reordering of σ_1 is equal to σ_2 . Theorem 3 in §7.2 can be considered as a motivation for this.

3.5. A mathematical model for the credential mechanism

In this subsection we shall describe the credential mechanism by means of the terminology introduced in §2. To this end, we must interpret each step described in §3.3 as a step in an execution of a protocol in the sense of §2, and give the notions introduced in §3.4 a precise meaning.

After having introduced some necessary notation, we consider the checks, introduce the “shadow”, which is an extraction of the credential mechanism that contains in essence the same information as the simple credential mechanism of §3.4, consider the steps executed by the parti-

cipants of the credential mechanism in more detail, give a proper formulation of the “main condition” by using the shadow, and finally consider the time order in which the steps are executed.

In our model for the credential mechanism we assume that all communication channels are secure against passive and active eavesdropping, and that messages are received at the same moment that they are sent and at the right place. The only essential assumptions in our analysis of the credential mechanism are that the communication channels between individuals and their representatives, and between the organizations and Z are secure. The analysis of the credential mechanism in this paper still holds true if the other assumptions are removed, however this would require uninteresting technical complications in our arguments.

Notation. We shall use the same notation as in §2 and §3.1-3.4. In particular, $T = \{0, 1, 2, \dots\}$ denotes the time, and N the composite modulus of the underlying RSA-system. We suppose that the set consisting of $N, n, f, m_1, \dots, m_n, c_1, \dots, c_K, a, p_1, \dots, p_L, q_1, \dots, q_L$ is fixed and known to each participant before the credential mechanism starts. Again we assume that $\phi(N), c_1, \dots, c_K, a, p_1, \dots, q_L$ are pairwise coprime and that p_1, \dots, q_L are primes larger than $\frac{1}{2}n$. Let Π be the set of subprotocols introduced in §3.4 and $\mathbb{N} = \{1, 2, \dots\}$. Put (cf. §2.1)

$$Y = F^+(\mathbf{Z}_N^*) \cup F(\{1, \dots, n\}) \cup \{\text{true}, \text{false}\}.$$

The set of participants P of the credential mechanism consists of the outside world E , the signature authority Z , the organizations A_j ($j = 1, \dots, L$), the individuals i_k ($k = 1, \dots, R$), a set of LR representatives, and the *allocation center* C which is responsible for allocating the representatives to the individuals. We shall discuss later in more detail how this allocation takes place. The message space M is equal to $M' \cup M''$ where $M' = \Pi \times \mathbb{N} \times Y$ and M'' is an unspecified set, containing the messages which E and C may send or receive.

Let $\mathcal{P} \in \Pi$. For convenience we denote the set $\{\mathcal{P}\} \times \mathbb{N} \times Y \times P_E \times P_E \times T$ by \mathcal{P} . Thus Π defines a partition of $M' \times P_E \times P_E \times T$ in subprotocols. A step of the form $((\mathcal{P}, r, y), \alpha, \beta)$ corresponds to the step of \mathcal{P} in the description of §3.3 which has number r at the left margin, and in which y is sent from α to β (or generated by α if $\alpha = \beta$). Messages in which an individual or representative asks for a pseudonym, validator or credential, will be indicated by triples $(\mathcal{P}, r, \emptyset)$, for appropriate \mathcal{P} and r . Any step of the form $((\mathcal{P}, r, y), \alpha, \beta)$ is indicated as “step r of \mathcal{P} ”.

Checks. Apart from the security and decision checks described in §3.4, the participants must do some other checks. We assume that at each moment that an individual, Z or an organization receives messages from another participant, it checks if these messages are allowed for the credential mechanism (cf. §2.3, (5)). (For instance, Z checks that the tuple which it receives in step 4 of $(I(i_k, A_j)$ has exactly $2n$ entries, and each individual or organization checks if the time order in which he receives certain messages from some participant is not in conflict with the consistency restriction). These obvious additional checks are also called security checks. Messages which are allowed for the credential mechanism will satisfy all security checks with only the fol-

lowing exception: that two individuals, with representatives g_{1j} and g_{2j} for A_j , respectively, do not cheat and by accident generate their numbers such that the checks in step 3 of $I(g_{1j}, A_j)$ and $I(g_{2j}, A_j)$, result in ‘false’.

As before, we assume that a step in \mathcal{P} is executed only if no previously executed step in \mathcal{P} contained a check which resulted in ‘false’ but allow that the execution of \mathcal{P} is repeated after a security check of the individual involved in \mathcal{P} resulted in ‘false’.

Shadow. The mapping σ on the message space M is defined as follows:

$$\begin{aligned}\sigma(m) = m & \left\{ \begin{array}{l} \text{if } m \in M' \\ \text{or } m = (\mathcal{P}, r, y) \in M' \text{ with } y \in \{\text{true, false}\} \end{array} \right., \\ \sigma(m) = (\mathcal{P}, r) & \text{if } m = (\mathcal{P}, r, y) \in M' \text{ with } y \notin \{\text{true, false}\}.\end{aligned}$$

σ is extended to $X = M \times P \times P \times T$ by putting

$$\sigma(m, \alpha, \beta, t) = (\sigma(m), \alpha, \beta, t).$$

For $a \subseteq X$ we put $\sigma(a) = \{\sigma(s) : s \in a\}$. We denote $\sigma(X)$ by Ξ , and for each subset η of Ξ we write $\eta(A, B, U) = \eta \cap (\sigma(M \times A \times B \times U))$ for $A, B \subseteq P$ and $U \subseteq T$. If S is the execution of (an attack on) the credential mechanism then we put $\Sigma = \sigma(S)$ and for any subsets A and B of P and U of T we abbreviate $\sigma(S(A, B, U))$ by $\Sigma(A, B, U)$. Σ is called the *shadow* of the (attack on) the credential mechanism.

The shadow is essentially equal to the simple credential mechanism of §3.4, except that it contains values of security checks. But if no participant cheats then these security checks will all result in ‘true’ with very high probability.

Individuals, organizations and Z . If a value of Σ is given. (i.e. the moments at which the participants execute their messages), then the steps executed by individuals and organizations are completely determined, except for the choices of the pseudonyms u_k in step 1 of $O(i_k)$ (which are not specified), and the uniform choices of the tuples $(r_l, s_l : l = 1, \dots, n)$ in step 3 of $I(i_k, A_j)$ and the sets S generated in step 6 of $I(i_k, A_j)$ for $1 \leq k \leq R$ and $1 \leq j \leq L$. When saying that at moment t , α makes a uniform choice from a finite set Γ , we implicitly assume that this choice is independent of the other steps executed at or before moment t in which α generated, sent or received a message. We now explain this with the terminology of §2.

Let Γ be a finite subset of Y . By “ α chooses γ uniformly from Γ at moment t in step r of \mathcal{P} ” the following is meant:

let $\Gamma(\mathcal{P}, r, \alpha, t)$ denote the choice of α at moment t in step r of \mathcal{P} , and let $W(\mathcal{P}, r, \alpha, t)$ denote the collection of other steps in which α generated, sent or received a message at or before moment t , i.e.

$$W(\mathcal{P}, r, \alpha, t) = S(\alpha, P, \leq t) \cup S(P_\alpha, \alpha, \leq t) \setminus ((\{\mathcal{P}, r\}) \times \Gamma(\mathcal{P}, r, \alpha, t) \times \{(\alpha, \alpha, t)\}).$$

Then for each γ in Γ , and each value w of $W(\mathcal{P}, r, \alpha, t)$ we have

$$\Pr[\Gamma(\mathcal{P}, r, \alpha, t) = \gamma \mid (\mathcal{P}, r, \alpha, \alpha, t) \in \Sigma(\alpha, \alpha, t), W(\mathcal{P}, r, \alpha, t) = w] = \frac{1}{\#\Gamma}. \quad (8)$$

The following two cases are of interest to us:

- $\Gamma = (\mathbb{Z}_N^*)^{2n}$, $\#\Gamma = \phi(N)^{2n}$, $\alpha = i_k$, $\mathcal{P} = I(i_k, A_j)$, $r = 3$;
- $\Gamma = \{\mathcal{S} \subset \{1, \dots, n\} : \#\mathcal{S} = \lfloor \frac{n}{2} \rfloor\}$, $\#\Gamma = \binom{n}{\lfloor \frac{n}{2} \rfloor}$, $\alpha = Z$, $\mathcal{P} = I(i_k, A_j)$, $r = 6$.

For the sake of completeness we mention that (8) also holds in case of an attack on the credential mechanism, in which α does not cheat but may receive messages from cheating participants.

Representatives, allocation center and outside world. We assume that none of the representatives, the allocation center C or the outside world E will ever cheat. In no execution of (an attack on) the credential mechanism E sends messages to or receives messages from C or the representatives.

During executions of (attacks on) the credential mechanism, E and C send only messages from M'' and “neglect” messages outside M'' , which they may have received during an execution of some attack on the credential mechanism, i.e. the messages they generate or send, are statistically independent of received messages which do not belong to M'' . Moreover, the allocation center sends messages only to individuals and representatives and neglects messages received from other participants than those.

The representatives belong to a fixed set of cardinality LR . We explain how the allocation of representatives to individuals takes place. At moment 1, C allocates a representative to each pair (i_k, A_j) , in such a way that different representatives are allocated to different pairs. It is assumed that each allocation has the same probability $(LR!)^{-1}$. At moment 2, C informs each individual, which representatives are allocated to him for communication with the organizations A_1, \dots, A_L , respectively, and informs each representative to which individual it has been allocated and for communication with which organization.

Let g_j be the representative of individual i_k communicating with A_j . After g_j has been informed that it has been allocated to i_k for communication with A_j , its activities during any execution of (an attack on) the credential mechanism consist only of the following: if g_j receives message m at moment t from a participant $\neq i_k$ then it sends m to i_k at moment $t + 1$; whereas if g_j receives m from i_k at moment t then it sends m to A_j at moment $t + 1$.

Representatives and allocation center are merely artificial constructions, meant to make the description of the mathematical model somewhat easier, and explain how the credential mechanism looks like from the point of view of the organizations. In general, they will not be used in practical implementations of the credential mechanism.

Main condition. $\Sigma(\alpha, P, t)$ describes the communication of participant α with the outside world or the allocation center, at moment t , the probability with which α may show pseudonyms, validators or credentials at moment t if α is an individual, or the probability with which α may issue a validator or credential at moment t if α is an organization. The conditional probability of

$\Sigma(\alpha, P, t)$ given $S(\alpha, P, < t)$ and $S(P_\alpha, \alpha, < t)$ describes the freedom of participants to communicate with the outside world, the freedom of individuals to decide whether to ask for or show a pseudonym, validator or credential, and the freedom for Z or the organizations to issue such a pseudonym, validator or credential, at moment t . This freedom should be as large as in the simple credential mechanism and hence any restrictions on this freedom should be expressible in the shadow. Thus the main condition can be stated as follows:

for each α in P , t in T and execution s of the credential mechanism we have

$$\begin{aligned} & \Pr[\sigma(\alpha, P, t) | s(\alpha, P, < t), s(P_\alpha, \alpha, < t), \text{sec}(\alpha, t)] \\ &= \Pr[\sigma(\alpha, P, t) | \sigma(\alpha, P, < t), \sigma(P_\alpha, \alpha, < t), \text{sec}(\alpha, t)], \end{aligned} \tag{9}$$

where $\sigma = \sigma(s)$, $\text{sec}(E, t) = \emptyset$ and $\text{sec}(\alpha, t)$ is the set of values of security checks by α at moment t on messages received from participants other than E if $\alpha \neq E$. If we assume that all security checks result in ‘true’ (which is extremely likely during executions of the credential mechanism if no participant cheats), then (9) implies that Σ is an execution process of a protocol in the sense of §2.

We remark that (9) holds true also for an attack on the credential mechanism in which α does not cheat.

Time order of steps. The shadow of the credential mechanism describes the order of the moments at which the steps in the credential mechanism can be executed. We require that steps from the same subprotocol are executed in the same order as described in §3.3, and at consecutive moments. The time order at which steps from different subprotocols \mathcal{P} are executed is subject to the consistency restriction given in §3.4. Other restrictions on the time order of the steps (e.g. those given in §3.4), must imply the main condition (9).

4. UNLINKABILITY

An equivalent statement of property 4 mentioned in §1.1 says that the credential mechanism does not reveal any information about which representatives represent which individuals. This property can not be proved in this strict sense. Suppose for instance, that first signature authority Z gives a validator to individual i_k and that later, a representative g_j shows a validator to A_j , at a moment at which no other validators have been issued or shown. Then Z and A_j will find out by cooperation, that g_j represents i_k . Another situation where information is revealed about the linking between representatives and individuals is the following: suppose that credential c is issued only once, on a pseudonym of representative g_j , say, and shown once on a pseudonym of representative g_h . Then by cooperation, the issuing and receiving organization will find out that g_j and g_h represent the same individual. We notice that information of the type mentioned above will also be revealed if instead of the credential mechanism of §3.5, the simple credential mechanism described in §3.4 would have been used. Using the model of §3.5 for the credential mechanism, we shall prove that the credential mechanism is optimal in the following sense: all information revealed by the credential mechanism about the relationship between individuals and

representatives is already revealed by the *shadow* of the credential mechanism. As mentioned in §3.5, this shadow is essentially equal to the simple credential mechanism considered in §3.4.

4.1. Statement of the result

We shall use the same notation and make the same assumptions as in §§2,3. Thus P is a set consisting of signature authority Z , the organizations A_1, \dots, A_L , the individuals i_1, \dots, i_R , LR representatives, the allocation center C and the outside world E . Let J be a subset of P , consisting of Z, A_1, \dots, A_L and some of the individuals and let $J_0 = \{i_1, \dots, i_R\} \setminus J$ be a set of non-cheating individuals. We consider attacks by subsets of J on the credential mechanism. An attack on the credential mechanism is called *safe* for J if it has the following properties (cf. §§2.3,3.5):

- if the messages received by an organization (or Z) before moment t from a representative (or individual) were allowed for the credential mechanism, then at moment t that organization (Z) sends back messages to that representative (individual) which are also allowed for the credential mechanism;
- no individual sends messages to other individuals or to other individuals' representatives; however, individuals may communicate over the outside world.

Loosely speaking, in safe attacks, cheating individuals, organizations and Z try to hide their cheating from individuals of which they believe that they do not cheat or from representatives of which they believe that they represent a non-cheating individual. An organization can only be sure that some representative represents a cheating individual if he receives messages from that representative which are not allowed for the credential mechanism. The only property of safe attacks which we shall use is, that the messages received by the non-cheating individuals in J_0 from participants other than E will satisfy all security checks by these individuals. This is true since in particular the messages received by these individuals' representatives from the organizations are allowed for the credential mechanism. We assume that Z has infinite computational resources, i.e. we make no further restrictions on the choices of Z .

Before stating Theorem 1, we recall that the allocation center is denoted by C . Thus $\Sigma(C, J_0, 2)$ denotes the allocation of representatives at moment 2 to the individuals in J_0 (cf. §3.5). We shall abbreviate this by $\Sigma(C, J_0)$. Values $\sigma(C, J_0, 2)$ of $\Sigma(C, J_0, 2)$ will correspondingly be abbreviated by $\sigma(C, J_0)$. We define Θ_t as the union of J_0 and the set of representatives of J_0 which have communicated with some organization, up to moment t . Then for each $t > 1$ there is a function θ_t such that $\Theta_t = \theta_t(S(P, J, \leq t))$, since Θ_t contains exactly those representatives not allocated in $\Sigma(C, J, 2)$. S will denote the execution process of (an attack on) the credential mechanism, $\Sigma = \sigma(S)$ (cf. shadow in §3.4) and $\Sigma(A, B, U) = \sigma(S(A, B, U))$ for $A, B \subseteq P$ and $U \subseteq T$.

THEOREM 1. *Let J be a set consisting of Z, A_1, \dots, A_L and some of the individuals, and $J_0 = \{i_1, \dots, i_R\} \setminus J$. Then for each attack on the credential mechanism which is safe for J and in which the individuals in J_0 do not cheat, each execution s of this attack, and each $t > 1$ we have*

$$\begin{aligned} & \Pr[\sigma(C, J_0) | s(J, P, \leq t), s(P, J, \leq t)] \\ &= \Pr[\sigma(C, J_0) | \Theta_t = \theta_t, \sigma(J, \theta_t \cup E, \leq t), \sigma(\theta_t \cup E, J, \leq t)], \end{aligned}$$

where $\sigma = \sigma(s)$ and $\theta_t = \theta_t(s(P, J, \leq t))$.

As mentioned above, from $s(J, P, \leq t)$ and $s(P, J, \leq t)$ it is possible to find out which representatives are allocated to individuals in J_0 . Theorem 1 tells us that all additional information revealed to J about which representative represents precisely which individual in J_0 is already revealed by the shadow of the credential mechanism.

Remark 1. In the case, that organizations do not know which individuals are cheating, they can try to find this out by statistically analyzing their set of received messages. We can not state an analogue of Theorem 1 when the set of individuals in J is not fixed, since the collection of attacks on the credential mechanism is not endowed with a probability measure.

Remark 2. Suppose that in Theorem 1, $J = \{Z, A_1, \dots, A_L\}$ and J_0 consists of all individuals, and that up to moment t the following happened: for $j = 1, \dots, L$, all individuals got their validators for A_j at the same moment, and all representatives showed their validator to A_j at the same moment; and moreover, no credential was issued or shown. Then $\Theta_t = \theta_t$, where θ_t consists of all individuals and representatives, and the sets $\Sigma(J, \theta_t, \leq t)$ and $\Sigma(\theta_t, J, \leq t)$ are independent of $\Sigma(C, J_0)$. Hence in this situation, all information revealed to J about $\Sigma(C, J_0)$ is coming from the sets $\Sigma(J, E, \leq t)$ and $\Sigma(E, J, \leq t)$, i.e. from J 's communication with the outside world.

Remark 3. In the statement of Theorem 1, it is essential to assume that the credentials c_1, \dots, c_K and the exponent a and the primes p_1, \dots, q_L used in the validating part are all coprime with $\phi(N)$. The individuals have the certainty that this requirement is satisfied if for instance all these numbers are primes larger than $\frac{1}{2}N$. Below we describe a protocol, based on an injective one-way function h , in which any individual can convince himself with probability at least $2/3$ that some odd exponent d made public by Z is coprime with $\phi(N)$. That individual can reduce Z 's chance of successful cheating by repeating this protocol as many times as he wants. Let i_k be an individual. In step 1, i_k chooses a number x uniformly from \mathbb{Z}_N^* , and sends $y := x^d$ to Z . In step 2, Z computes x' with $x'^d = y$ and sends $h_0 := h(x')$ to i_k . In step 3, i_k checks if $h(x) = h_0$.

If d is coprime with $\phi(N)$, then the value x' computed by Z is always equal to x , and hence h_0 is equal to $h(x)$. Suppose that d is not coprime with $\phi(N)$, and let d_P, d_Q denote the numbers of solutions of $x^d \equiv 1 \pmod{P}$, $x^d \equiv 1 \pmod{Q}$, respectively, where P and Q are the prime factors of N . Then there are exactly $d_P d_Q$ different x' with $x'^d \equiv y \pmod{N}$. Z knows that i_k must have chosen x in step 1 as one of the d -th roots of y but he has no information about which root was precisely chosen by i_k . Hence in step 3 Z can do no better than guessing which root was chosen by i_k and the chance that he guesses wrong is $1 - (d_P d_Q)^{-1}$ which is at least $2/3$. By the injectivity of h , the chance that i_k will receive a value h_0 different from $h(x)$ in step 3 is at least $2/3$. i_k might try to cheat by sending Z a value y in step 1 of which he does not know the d -th root. However, if the one-way function h used in step 3 is “good enough”, then i_k will not be able to compute the d -th root of y from h_0 .

4.2. Lemmas

The idea behind the proof of Theorem 1 is to construct, from the messages sent or received by Z and the organizations and from a given allocation of representatives to individuals, a set of tuples $(r_l, s_l: l=1, \dots, n)$, chosen by the individuals in step 3 of each subprotocol $I(i_k, A_j)$, and argue that the sets of tuples constructed from different allocations have equal probability. In this section we state and prove two lemmas needed in the proof.

Let J consist of Z, A_1, \dots, A_L and some of the individuals, and let $J_0 = \{i_1, \dots, i_R\} \setminus J$. We fix an attack on the credential mechanism which is safe for J and in which the individuals in J_0 do not cheat, and denote the execution process of this attack by S . Further we put $\Sigma = \sigma(S)$. Up to now, by an execution we just meant an arbitrary value of the execution process of the attack we are considering, which can be any subset of $X = M \times P \times P \times T$. In the sequel we shall restrict ourselves to *proper* executions, i.e. executions s with the following properties:

- the set of elements of s belonging to the subprotocols in which individuals in J_0 or their representatives are involved satisfies the description of these subprotocols in §3.3; in particular, in each step 3 of $I(i_k, A_j)$ with $i_k \in J_0$ a tuple $(r_l, s_l: l=1, \dots, n) \in (\mathbb{Z}_N^*)^{2n}$ is chosen, and in each of the subprotocols involving individuals in J_0 or their representatives, the messages are computed as prescribed in §3.3, and the steps are executed in the order corresponding with the description of §3.3 and at consecutive moments;
- s satisfies the conditions imposed on the messages generated, sent or received by E , C and the representatives as described in §3.5;
- the time order in which the steps in s , involving an individual in J_0 or one of his representatives, are executed, is subject to the consistency restriction of §3.4.

In attacks which are safe for J and in which the individuals in J_0 do not cheat, executions which are not proper have always probability 0. There may be proper executions with probability 0.

In Lemma 1 below we compute the probability of a proper execution. We put $J'_0 = J_0 \cup \{E\}$. For any proper execution s with $\sigma = \sigma(s)$, and any subset U of T , we denote by $\kappa(\sigma(J_0, P, U))$ the number of all steps 3 of $I(i_k, A_j)$ executed during U , for $i_k \in J_0$ and $1 \leq j \leq L$. It is clear that this number depends indeed only on $\sigma(J_0, P, U)$.

Lemma 1. *For every $t > 0$ there are functions A_t and B_t such that for each proper execution s ,*

$$\begin{aligned} \Pr[S(P, P, \leq t)] &= \phi(N)^{-2n\kappa(\sigma(J_0, P, \leq t))} \\ &\quad \times A_t(\sigma(J'_0, P, \leq t), \sigma(C, J_0), \theta_t, \sigma(J, \theta_t \cup E, \leq t)) \\ &\quad \times B_t(s(J, P, \leq t), s(P, J, \leq t)), \end{aligned}$$

where $\sigma = \sigma(s)$ and $\theta_t = \theta_t(s(J, P, \leq t))$ is the value for Θ_t .

Proof. In the proof of this lemma only, undefined conditional probabilities will be given the value 1. We shall prove Lemma 1 by induction on t . We start with $t = 0$.

The statement of Lemma 1 trivially holds true for $t = 0$ by taking $\kappa(\sigma(J_0, P, 0)) = 0$ and letting A_0 and B_0 be functions identically equal to 1. Suppose that Lemma 1 has been proved for moment $t - 1$ where $t > 0$. (induction hypothesis). We proceed to prove Lemma 1 for moment t .

Let s be a proper execution. For convenience we put

$$Pr(t) = Pr[s(P, P, \leq t)], \quad Pr(t-1) = Pr[s(P, P, < t)].$$

Let R denote the set of representatives. By (1) (cf. §2.1) we have

$$\begin{aligned} Pr(t) &= Pr[s(J'_0, P, \leq t), s(C, P, \leq t), s(R, P, \leq t), s(J, P, \leq t)] \\ &= P_1 P_2 P_3 P_4 Pr(t-1), \end{aligned} \tag{10}$$

where

$$\begin{aligned} P_1 &= Pr[s(J'_0, P, t) | s(J'_0, P, < t), s(C, P, \leq t), s(R, P, \leq t), s(J, P, \leq t)], \\ P_2 &= Pr[s(C, P, t) | s(J'_0, P, < t), s(C, P, \leq t), s(R, P, \leq t), s(J, P, \leq t)], \\ P_3 &= Pr[s(R, P, t) | s(J'_0, P, < t), s(C, P, \leq t), s(R, P, \leq t), s(J, P, \leq t)], \\ P_4 &= Pr[s(J, P, t) | s(J'_0, P, < t), s(C, P, \leq t), s(R, P, \leq t), s(J, P, \leq t)]. \end{aligned}$$

Note that (10) also holds true if some of the conditional probabilities on the right-hand side are not defined. Moreover, if one of the conditional probabilities in (10) is not defined then one of the other factors in the right-hand side of (10) is 0. Therefore, (10) remains true if we replace an undefined conditional probability by any value we like. To the defined conditional probabilities we may apply (4) (cf. §2.2) with the partition $J'_0, \{C\}, R$ and J of P . Thus we obtain

$$Pr(t) = P'_1 P'_2 P'_3 P'_4 Pr(t-1), \tag{11}$$

where

$$\begin{aligned} P'_1 &= Pr[s(J'_0, P, t) | s(J'_0, P, < t), s(P, J'_0, < t)], \\ P'_2 &= Pr[s(C, P, t) | s(C, P, < t), s(P, C, < t)], \\ P'_3 &= Pr[s(R, P, t) | s(R, P, < t), s(P, R, < t)], \\ P'_4 &= Pr[s(J, P, t) | s(J, P, < t), s(P, J, < t)]. \end{aligned}$$

Because of the relationships between messages sent and received by the representatives, which hold also for our proper execution s , we have $P'_3 = 1$. Moreover, if $t = 1$ we have $P'_2 = (LR!)^{-1}$, since each allocation is equally likely, if $t = 2$ then $P'_2 = 1$ since the messages sent by the allocation center to the individuals and representatives are determined by the allocation at moment 1, and if $t > 2$ then also $P'_2 = 1$, since $s(C, P, t) = \emptyset$ with probability 1. (Here we used that s satisfies the conditions on the messages generated and sent by C). By combining these facts we obtain that there is a function C_t such that

$$P'_2 P'_3 P'_4 = C_t(s(J, P, \leq t), s(P, J, \leq t)). \tag{12}$$

We now consider P'_1 . Suppose for the moment that P'_1 is defined. All elements of

$s(J'_0, P, t)$ are of the form $x = (m, \alpha, \beta, t)$, where $\alpha \in J'_0$. If $m \in M''$ or $m = (\varphi, r, y) \in M'$ and $y \in \{\text{true, false}\}$, then $x \in \sigma(J'_0, P, t)$. Otherwise we have $m = (\varphi, r, y)$, where $(\varphi, r, \alpha, \beta)$ belongs to $\sigma(J'_0, P, t)$ and y is either a tuple $(r_l, s_l : l = 1, \dots, n)$ chosen in step 3 of some $I(i_k, A_j)$, or can be derived from a tuple previously chosen in step 3 of some $I(i_k, A_j)$ and from messages previously received by J_0 , by using the computations in the credential mechanism described in §3.3. It follows that $S(J'_0, P, t)$ is completely determined by $\Sigma(J'_0, P, t)$, $B(t)$, $S(J'_0, P, < t)$ and $S(P, J'_0, < t)$, where $B(t)$ is the set of tuples chosen at moment t in step 3 of some $I(i_k, A_j)$, for $i_k \in J_0$. Let $b(t)$ be the value of $B(t)$ corresponding to $s(J'_0, P, t)$. Then

$$P'_{11} = \Pr[b(t), \sigma(J'_0, P, t) | s(J'_0, P, < t), s(P, J'_0, < t)] = P'_{11} P'_{12}, \quad (13)$$

where

$$\begin{aligned} P'_{11} &= \Pr[b(t) | \sigma(J'_0, P, t), s(J'_0, P, < t), s(P, J'_0, < t)], \\ P'_{12} &= \Pr[\sigma(J'_0, P, t) | s(J'_0, P, < t), s(P, J'_0, < t)]. \end{aligned}$$

$b(t)$ contains exactly $\kappa(\sigma(J'_0, P, t))$ tuples $(r_l, s_l : l = 1, \dots, n)$. By assumption, the distributions of these tuples are uniform on $(\mathbb{Z}_N^*)^{2n}$ and independent of each other and of $S(J'_0, P, < t)$ and $S(P, J'_0, < t)$. By repeatedly applying (8) to these tuples and using (1) we obtain

$$P'_{11} = \phi(N)^{-2n\kappa(\sigma(J_0, P, t))}, \quad (14)$$

provided that P'_{11} is defined. If $P'_{11} = 0$, hence (13) still holds true if we replace P'_{11} by the right-hand side of (14). Since we are considering a safe attack, the security checks by individuals in J_0 on messages received from participants other than E will be satisfied with probability 1. Therefore we may apply main condition (9) for each α in J'_0 without the stochastic variable $\sec(\alpha, t)$ on both sides of the equality. By combining (9) for each α in J'_0 with (1), we obtain

$$P'_{12} = \Pr[\sigma(J'_0, P, t) | \sigma(J'_0, P, < t), s(P, J'_0, < t)]. \quad (15)$$

We note that E receives messages only from J_0, E and J , that $\sigma(J'_0, E, < t)$ is contained in $\sigma(J'_0, P, < t)$, while $\sigma(J, E, < t)$ is contained in $\sigma(J, \theta_t \cup E, < t)$. J_0 receives messages only from C, E, Z and the representatives in θ_t . $\sigma(E, J_0, < t)$ and $\sigma(Z, J_0, < t)$ are contained in $\sigma(J'_0, P, < t)$ and $\sigma(J, \theta_t \cup E, < t)$, respectively. Moreover, from the relationship between messages sent, and messages received by representatives (which holds for proper executions), it follows that $\sigma(\theta_t, J_0, < t)$ is uniquely determined by $\sigma(C, J_0)$ and $\sigma(J, \theta_t, < t)$. By combining these facts we conclude that $\sigma(P, J'_0, < t)$ can be expressed as a function in $\sigma(J'_0, P, < t)$, $\sigma(C, J_0)$, θ_t and $\sigma(J, \theta_t \cup E, < t)$. A combination of this with (13), (14) and (15) yields that there is a function D_t with

$$P'_{11} = \phi(N)^{-2n\kappa(\sigma(J_0, P, t))} D_t(\sigma(J'_0, P, < t), \sigma(C, J_0), \theta_t, \sigma(J, \theta_t \cup E, < t)).$$

By combining this with (12) and (11) we obtain

$$\Pr(t) = \phi(N)^{-2n\kappa(\sigma(J_0, P, t))} D_t(\sigma(J'_0, P, \leq t) C_t(s(J, P, \leq t), s(P, J, \leq t)) \times \Pr(t-1).$$

Note that this is true also if P'_1 is not defined. Together with the induction hypothesis this proves Lemma 1 for moment t . \square

Let F_1, \dots, F_r be functions of S . We say that the values s_1, \dots, s_r of $F_1(S), \dots, F_r(S)$, respectively, come from the same proper execution if there is a proper execution s such that $s_1 = F_1(s), \dots, s_r = F_r(s)$. Lemma 2 gives the number of possibilities for the messages generated, sent or received by the individuals in J_0 , given only the moments at which the individuals in J_0 generate, send or receive their messages, the allocation of representatives to J_0 , and the steps involving the members of J .

Lemma 2. *There is a function λ with the following property: for each $t > 0$, and all values σ_0 of $\Sigma(J'_0, P, \leq t)$, σ_{C0} of $\Sigma(C, J_0)$, θ_t of Θ_t , s_{JP} of $S(J, P, \leq t)$, s_{PJ} of $S(P, J, \leq t)$, σ_{J^*} of $\Sigma(J, \Theta_t \cup E, \leq t)$, and σ_{*J} of $\Sigma(\Theta_t \cup E, J, \leq t)$, such that*

$$\Pr[s_{JP}, s_{PJ}, \theta_t, \sigma_{J^*}, \sigma_{*J}] > 0$$

and

$\sigma_0, \sigma_{C0}, \theta_t, \sigma_{J^*}, \sigma_{*J}$ come from the same proper execution,
there are exactly $\phi(N)^{2n\kappa(\sigma_0) - \lambda(n, \theta_t, \sigma_{*J})}$ values s_0 of $S(J'_0, P, \leq t)$ such that $s_0, \sigma_0, \sigma_{C0}, \theta_t, s_{JP}, s_{PJ}, \sigma_{J^*}$ and σ_{*J} come from the same proper execution.

Proof. In this proof we shall often refer to the description of the credential mechanism in §3.3, and the reader is advised to consult this. The values s_0 of $S(J'_0, P, \leq t)$ we are looking for, consist of tuples $x = (m, \alpha, \beta, u)$ with $m \in M$, $\alpha \in J'_0$, $\beta \in P$ and $u \leq t$. If $m \in M''$ or $m = (\mathcal{P}, r, y) \in M'$ with $y \in \{\text{true, false}\}$ then $x \in \sigma_0$. For the remaining tuples x we have $\alpha \neq E$, $\beta \neq E$ and $m = (\mathcal{P}, r, y) \in M'$, where each $\sigma(x) = (\mathcal{P}, r, \alpha, \beta, u)$ is contained in σ_0 and the set of values y must satisfy the constraints imposed by the description of the credential mechanism in §3.3, the given allocation of representatives, and the steps in which the members in J generated, sent or received their messages before moment t . Our purpose is to count the number of possibilities for the set of values y . From the description of the credential mechanism in §3.3 it follows that each y is either equal to one of the tuples $(r_l, s_l : l = 1, \dots, n)$ generated in the steps 3 of $I(i_k, A_j)$ up to moment t , or can be derived from these tuples and the messages received by J_0 up to moment t from Z or their representatives, by using the computations prescribed in the credential mechanism. But in proper executions, the messages received by J_0 can be determined from the allocation σ_{C0} and s_{JP} . Hence the number of possibilities for the set of values y is equal to the number of possibilities for the set of tuples $(r_l, s_l : l = 1, \dots, n)$ generated in each step 3 of $I(i_k, A_j)$ up to moment t for $i_k \in J_0$ and $1 \leq j \leq L$. We shall prove that this number is equal to $\phi(N)^{2n\kappa(\sigma_0) - \lambda(n, \theta_t, \sigma_{*J})}$ where

$$\lambda(n, \theta_t, \sigma_{*J}) = n\lambda_1 + \frac{1}{2}n\lambda_2 + 2\lambda_3 + (\frac{1}{2}n - 2)\lambda_4, \quad (16)$$

and λ_1 is the number of steps 5 of $I(i_k, A_j)$, λ_2 the number of steps 8 of $I(i_k, A_j)$, λ_3 the number

of steps 2 of $II(g_j, A_j)$, and λ_4 the number of steps 5 of $II(g_j, A_j)$, for all $i_k \in J_0$, $g_j \in \theta_t$, and organizations A_j , which are executed up to moment t . Note that all these steps are contained in σ_{*j} .

Let $i_k \in J_0$ and g_j the representative allocated to i_k for communication with A_j . Suppose that at or before moment t , step 3 of $I(i_k, A_j)$ has been executed (this can be derived from σ_0), and let $\rho = (r_l, s_l : l = 1, \dots, n)$ be the tuple chosen in this step. We have to count the number of values for ρ such that $\rho, \sigma_0, \sigma_{C0}, \dots, \sigma_{*j}$ come from the same proper execution. Each step v , executed up to moment t , in which Z or an organization receives a message from an individual in J_0 or one of its representatives (which is contained in s_{pj}), imposes certain constraints on ρ , which will reduce the number of possibilities for ρ . Without any of these constraints, there are $\phi(N)^{2n}$ possibilities for ρ .

If v is step 5 of $I(i_k, A_j)$, then Z receives numbers a_{kl} ($l = 1, \dots, n$), which should be equal to $f(u_k(m_l r_l^a)^{b_l}) s_l^{p_l q_l}$. These relationships reduce the number of possibilities for ρ by a factor $\phi(N)$. Indeed, since $p_l q_l$ is coprime with $\phi(N)$, each r_l determines a unique s_l such that these relationships are satisfied.

If v is step 8 of $I(i_k, A_j)$ then Z receives r_l, s_l with $l \in \bar{S}$, where \bar{S} is the set sent by Z to i_k in step 7. As remarked before, each s_l is determined by a_{kl} and r_l and does not impose additional conditions on ρ . Obviously, the released values r_l reduce the number of possibilities for ρ by a factor $\phi(N)^{\frac{1}{2}n}$.

If v is step 2 of $II(g_j, A_j)$, then A_j receives u_{g_j} , which must be equal to $u_k \tilde{r}_{\sigma(1)}^{b_j}$, and w_{g_j} , which must be equal to $u_k^{\frac{1}{2}n-1} (\prod_{l=2}^{\frac{1}{2}n} \tilde{r}_{\sigma(l)})^{b_j}$, where σ is the permutation defined in the description of $II(g_j, A_j)$ in §3.3, and $\tilde{r}_{\sigma(l)} = m_{\sigma(l)} r_{\sigma(l)}^a$ for $l = 1, \dots, \frac{1}{2}n$. Since both a and b_j are coprime with $\phi(N)$, $r_{\sigma(1)}$ and $\prod_{l=2}^{\frac{1}{2}n} r_{\sigma(l)}$ are uniquely determined by u_{g_j} and w_{g_j} . This reduces the number of possibilities for ρ by a factor $\phi(N)^2$.

Finally, if v is step 5 of $II(g_j, A_j)$ then A_j receives \tilde{v}_{g_j} , and also numbers t_l ($l = 2, \dots, \frac{1}{2}n$) which should be equal to $\tilde{r}_{\sigma(l)} / \tilde{r}_{\sigma(1)}$, respectively. From this it is possible to determine uniquely a set of values r_l , not shown to Z in step 8 of $I(i_k, A_j)$. Since each of these r_l determines a unique s_l , this leaves us only one possibility for ρ . In other words, the number of possibilities for ρ is reduced by a factor $\phi(N)^{\frac{1}{2}n-2}$.

Other steps v in which Z or the organizations received messages up to moment t do not further reduce the number of possibilities for ρ . Thus for each tuple ρ generated in step 3 of some $I(i_k, A_j)$ we have a specified number of possibilities. We obtain (16) by taking the product of all these numbers of possibilities, over all steps 3 of all $I(i_k, A_j)$ with $i_k \in J_0$, executed at or before moment t . \square

4.3. Proof of Theorem 1

We shall prove that for each $t > 1$ there are functions E_t and F_t such that for all values σ_{C0} of $\Sigma(C, J_0)$, θ_t of Θ_t , s_{JP} of $S(J, P, \leq t)$, s_{PJ} of $S(P, J, \leq t)$, σ_{J^*} of $\Sigma(J, \Theta_t \cup E, \leq t)$ and σ_{*J} of $\Sigma(\Theta_t \cup E, J, \leq t)$, with $Pr[\theta_t, s_{JP}, s_{PJ}, \sigma_{J^*}, \sigma_{*J}] = Pr[s_{JP}, s_{PJ}] > 0$ we have

$$Pr[\sigma_{C0}, s_{JP}, s_{PJ}] = \phi(N)^{-\lambda(n, \theta_t, \sigma_{*J})} E_t(\sigma_{C0}, \theta_t, \sigma_{J^*}, \sigma_{*J}) F_t(s_{JP}, s_{PJ}). \quad (17)$$

This suffices. For by summing over all σ_{C0} we obtain that there is a function G_t with

$$Pr[s_{JP}, s_{PJ}] = \phi(N)^{-\lambda(n, \theta_t, \sigma_{*J})} G_t(\theta_t, \sigma_{J^*}, \sigma_{*J}) F_t(s_{JP}, s_{PJ}).$$

Hence $G_t(\theta_t, \sigma_{J^*}, \sigma_{*J}) \neq 0$ and $F_t(s_{JP}, s_{PJ}) \neq 0$. This gives

$$Pr[\sigma_{C0} | s_{JP}, s_{PJ}] = \frac{E_t(\sigma_{C0}, \theta_t, \sigma_{J^*}, \sigma_{*J})}{G_t(\theta_t, \sigma_{J^*}, \sigma_{*J})} = Pr[\sigma_{C0} | \sigma_{J^*}, \sigma_{*J}, \theta_t].$$

We now prove (17). We fix $\sigma_{C0}, \theta_t, s_{JP}, s_{PJ}, \sigma_{J^*}, \sigma_{*J}$ with $Pr[\theta_t, s_{JP}, s_{PJ}, \sigma_{J^*}, \sigma_{*J}] \neq 0$ and assume that $Pr[\sigma_{C0}, \theta_t, \sigma_{J^*}, \sigma_{*J}] \neq 0$ which is no restriction. We have

$$Pr[\sigma_{C0}, s_{JP}, s_{PJ}] = \sum_{\sigma_0} \left(\sum_{s_0} Pr[s_0, s_{JP}, s_{PJ}, \sigma_{C0}] \right), \quad (18)$$

where the outer sum is taken over all values σ_0 of $\Sigma(J'_0, P, \leq t)$ such that $\sigma_0, \sigma_{C0}, \theta_t, \sigma_{J^*}, \sigma_{*J}$ come from the same proper execution, and the inner sum over all values s_0 of $S(J'_0, P, \leq t)$ such that $s_0, \sigma_0, \sigma_{C0}, \theta_t, s_{JP}, s_{PJ}, \sigma_{J^*}$, and σ_{*J} come from the same proper execution. From s_0, σ_{C0}, s_{JP} , and s_{PJ} it is possible to derive all steps executed by C and the representatives. Hence if $s_0, \sigma_0, \sigma_{C0}, \theta_t, s_{JP}, s_{PJ}, \sigma_{J^*}$, and σ_{*J} come from the same proper execution s , then $Pr[s_0, s_{JP}, s_{PJ}, \sigma_{C0}] = Pr[s(P, P, \leq t)]$. By Lemma 1 this implies that

$$Pr[s_0, s_{JP}, s_{PJ}, \sigma_{C0}] = \phi(N)^{-2n\kappa(\sigma_0)} A_t(\sigma_0, \sigma_{C0}, \theta_t, \sigma_{J^*}) B_t(s_{JP}, s_{PJ}).$$

Together with Lemma 2 this gives

$$\sum_{s_0} Pr[s_0, s_{JP}, s_{PJ}, \sigma_{C0}] = \phi(N)^{-\lambda(n, \theta_t, \sigma_{*J})} A_t(\sigma_0, \sigma_{C0}, \theta_t, \sigma_{J^*}) B_t(s_{JP}, s_{PJ}).$$

By taking the sum over all σ_0 for which $\sigma_0, \sigma_{C0}, \theta_t, \sigma_{J^*}$ and σ_{*J} come from the same proper execution, we obtain

$$\sum_{\sigma_0} \left(\sum_{s_0} Pr[s_0, s_{JP}, s_{PJ}, \sigma_{C0}] \right) = \phi(N)^{-\lambda(n, \theta_t, \sigma_{*J})} E_t(\sigma_{C0}, \theta_t, \sigma_{J^*}, \sigma_{*J}) B_t(s_{JP}, s_{PJ})$$

for some function E_t . Together with (18) this implies (17). This completes the proof of Theorem 1. \square

5. THE FORMAL CREDENTIAL MECHANISM

In this section we describe a formal credential mechanism which is similar to that of §3 except that it is based on an “ideal RSA-cryptosystem” of which the underlying message space is a free multiplicative module over the rational numbers with denominator coprime with $\phi(N)$ and the one-way function maps this message space on multiplicatively independent elements of this space. In §§6,7 we shall analyze this formal credential mechanism for protection of organizations against cheating individuals.

5.1. Description of the underlying message space

In order to give a proper description of the underlying cryptosystem we have to introduce some notions about free multiplicative modules.

For any set \mathcal{V} and any integral domain \mathcal{R} , we denote by $\mathcal{R}[\mathcal{V}]$ the set of all finite formal products

$$V_1^{\xi_1} \cdots V_t^{\xi_t}$$

where $\xi_1, \dots, \xi_t \in \mathcal{R}$ and V_1, \dots, V_t are different elements of \mathcal{V} . The empty product is denoted by 1. We shall identify two formal products $V_1^{\xi_1} \cdots V_t^{\xi_t}$ and $W_1^{\eta_1} \cdots W_s^{\eta_s}$ if and only if there is an r with $r \leq t$ and $r \leq s$ such that after reordering the terms of both products, $\xi_i = 0$ for $r < i \leq t, \eta_i = 0$ for $r < i \leq s$ and $V_i = W_i$ and $\xi_i = \eta_i$ for $1 \leq i \leq r$. $\mathcal{R}[\mathcal{V}]$ is a free \mathcal{R} -module, of which the addition is the multiplication of formal products, defined by adding the exponents, and scalar multiplication is raising a formal product to a power in \mathcal{R} , i.e. multiplying the exponents of that product with that power.

We shall recursively construct a free \mathcal{R} -module which contains \mathcal{V} and which is closed under application of some “formal one-way function”. Put

$$\mathcal{R}_1 = \mathcal{R}[\mathcal{V}], \quad \mathcal{F}_1 = \{F_X : X \in \mathcal{R}_1\};$$

$$\mathcal{R}_i = \mathcal{R}[\mathcal{R}_{i-1} \cup \mathcal{F}_{i-1}], \quad \mathcal{F}_i = \{F_X : X \in \mathcal{R}_i \setminus \mathcal{R}_{i-1}\} \text{ for } i = 2, 3, 4, \dots$$

and assume that $F_X \neq F_Y$ if $X \neq Y$ and $\mathcal{F}_i \cap \mathcal{R}_i = \emptyset$ for $i = 1, 2, 3, \dots$. Put $\mathcal{F} = \bigcup_{i=1}^{\infty} \mathcal{F}_i$ and define the function $F : \mathcal{R}[\mathcal{V} \cup \mathcal{F}] \rightarrow \mathcal{F}$ by $F(X) = F_X$. The module $\mathcal{R}[\mathcal{V} \cup \mathcal{F}]$ endowed with the function F just defined is denoted by $\mathcal{R}[F, \mathcal{V}]$. Different choices of the sets \mathcal{F}_i will lead to isomorphic modules $\mathcal{R}[F, \mathcal{V}]$. If \mathcal{Q} is a subset of $\mathcal{R}[F, \mathcal{V}]$ we denote by $\mathcal{R}[F, \mathcal{Q}]$ the smallest \mathcal{R} -submodule of $\mathcal{R}[F, \mathcal{V}]$ which contains \mathcal{Q} and is closed under application of F .

To the numbers u_1, \dots, u_R issued to the individuals i_1, \dots, i_R by Z in the credential mechanism, and to m_1, \dots, m_n , chosen by Z during the initialization of the credential mechanism, we associate formal variables $U_1, \dots, U_R, M_1, \dots, M_n$, respectively. Also other formal variables H_1, \dots, H_T are introduced, which correspond to numbers chosen by the individuals

themselves. Let

$$\begin{aligned}\tilde{\mathbb{Q}} &= \left\{ \frac{a}{b} : a, b \in \mathbb{Z}, \gcd(b, \phi(N)) = 1 \right\}, \\ \mathcal{K} &= \{U_1, \dots, U_R, M_1, \dots, M_n, H_1, \dots, H_T\}, \\ \mathcal{L} &= \tilde{\mathbb{Q}}[F, \mathcal{K}],\end{aligned}$$

where F is a formal one-way function as described above.

F-homomorphisms as defined below establish a relationship between the pairs (\mathcal{L}, F) and (\mathbb{Z}_N^*, f) . A mapping $\psi : \mathcal{L} \rightarrow \mathbb{Z}_N^*$ is called an *F-homomorphism* if it has the following properties:

- ψ is a homomorphism with respect to multiplication;
- $\psi(F(W)) = f(\psi(W))$ for $W \in \mathcal{L}$;
- $\psi(W^{\frac{a}{b}}) = [\psi(W)^a]^b$ for $W \in \mathcal{L}, a \in \mathbb{Z}, b \in \mathbb{Z}$ with $\gcd(b, \phi(N)) = 1$;
- $\psi(U_k) = u_k$ for $k = 1, \dots, R$, $\psi(M_l) = m_l$ for $l = 1, \dots, n$.

Thus an *F-homomorphism* is uniquely determined by its images in H_1, \dots, H_T .

5.2. Computations on \mathcal{L}

\mathcal{L} is closed under the following operations: multiplications, multiplicative inversions, applications of F , and *taking roots*, i.e. raising to powers a^{-1} where a is an integer, coprime with $\phi(N)$. We shall endow \mathcal{L} with a computational model, based on these operations, which is used in protocols of which (part of) the transmitted messages are elements of \mathcal{L} .

A *computation* on \mathcal{L} is a repeated application of multiplications, multiplicative inversions, and F , (but *not* taking roots), to elements of \mathcal{L} . If $\mathfrak{D} \subseteq \mathcal{L}$ then $X \in \mathcal{L}$ is said to be *computable* from \mathfrak{D} if it can be obtained by applying a computation to elements of \mathfrak{D} . Thus the set of elements of \mathcal{L} computable from \mathfrak{D} is equal to $\mathbb{Z}[F, \mathfrak{D}]$. By assumption, the elements of $\mathcal{K} = \{U_1, \dots, U_R, M_1, \dots, M_n, H_1, \dots, H_T\}$ are computable.

An *extended computation* on \mathcal{L} is a repeated application of multiplications, multiplicative inversions, F , and also taking roots, to elements of \mathcal{L} . Each element of \mathcal{L} can be obtained from \mathcal{K} by extended computations.

Consider a protocol in which \mathcal{L} is (part of) the message space, such that each participant may apply all possible computations to the elements of \mathcal{K} and to the messages which it received during an execution of the protocol, but only a few distinguished participants are allowed to do extended computations. Let α be a participant which is not allowed to do extended computations, and let $\mathfrak{D}(< t)$ be the stochastic set of elements of \mathcal{L} , received by α before t , which are not computable from \mathcal{K} . Then at moment t , α can compute each element of the set $\mathbb{Z}[F, \mathcal{K} \cup \mathfrak{D}(< t)]$. Obviously, this set is stochastic and might grow for increasing t .

Let ψ be an *F-homomorphism*, and let \mathfrak{D} be a subset of \mathcal{L} , containing \mathcal{K} . Suppose that

some participant α has received all messages in $\psi(\mathcal{D})$ during the execution of some protocol, of which \mathbb{Z}_N^* is (part of) the message space. If α can break RSA for the modulus N , then he can, in principle, compute $\psi(A)$ for each A in \mathcal{Z} . If α can not break this RSA-system then in principle he can still compute $\psi(A)$ for each A in \mathcal{Z} which is computable from \mathcal{D} , by applying multiplications and multiplicative inversions in \mathbb{Z}_N^* and f to the elements of $\psi(\mathcal{D})$. As far as we know, the following question —which should be stated more precisely— is still open:

are there a modulus N and a one-way function $f: \mathbb{Z}_N^* \rightarrow \mathbb{Z}_N^*$, such that for each F-homomorphism ψ and each A in \mathcal{Z} not computable from \mathcal{D} , computing $\psi(A)$ from $\psi(\mathcal{D})$ is as difficult as breaking RSA for the modulus N ?

5.3. The formal credential mechanism

We shall use the same notation as in the previous sections. The formal credential mechanism will have the same set of participants P as the credential mechanism of §3, consisting of the signature authority Z , the organizations A_1, \dots, A_L , the individuals i_1, \dots, i_R , the individuals' representatives, the outside world E and the allocation center C . As before, the numbers $\phi(N)$, $c_1, \dots, c_K, a, p_1, \dots, p_L, q_1, \dots, q_L$ are pairwise coprime and p_1, \dots, q_L are primes larger than $\frac{1}{2}n$, where $n > 4$ is the security parameter in the validating part. The message space M will be defined in the same way as in §3.5, except that \mathbb{Z}_N^* is replaced by \mathcal{Z} . Thus $M = M' \cup M''$, where $M' = \Pi \times \mathbb{N} \times Y$ with $Y = F^+(\mathcal{Z}) \cup F(\{1, \dots, n\}) \cup \{\text{true}, \text{false}\}$ and M'' is the set of messages sent or received by E and C . $A \in \mathcal{Z}$ is said to be contained in a message m if $m = (\mathcal{P}, r, y) \in M'$ and A is an entry of y . We say that participant α generates (sends, receives) $A \in \mathcal{Z}$ in step r of \mathcal{P} if α generates (sends, receives) the message (\mathcal{P}, r, A) . Again the time will be modelled as a set of discrete moments, $T = \{0, 1, 2, \dots\}$. Again we put $X = M \times P \times P \times T$. For each $x = (m, \alpha, \beta, t) \in X$, we let $C(x)$ denote the set of elements of \mathcal{Z} contained in m . In particular, $C(x) = \emptyset$ if $m \in M''$. If a is a subset of X then we put $C(a) = \bigcup_{x \in a} C(x)$.

We postulate that all participants of the credential mechanism can apply all computations to elements of \mathcal{K} and elements of \mathcal{Z} which they received during an execution of (an attack on) the formal credential mechanism; only signature authority Z can do extended computations. The computational abilities of each participant α of the formal credential mechanism can be expressed in terms of its collection of choices \mathcal{C}_α (cf. §2.3). The outside world, allocation center and representatives will all have a unique choice, satisfying the same conditions as in §3.5. \mathcal{C}_Z contains all choices satisfying property (2) with $\alpha = Z$ (cf. §2.2). If $\alpha \in \{i_1, \dots, i_R, A_1, \dots, A_L\}$ then \mathcal{C}_α contains all choices $p_\alpha = \{p_{\alpha,t} : t > 0\}$ which have, apart from property (2) in §2.2, the following restriction: if $y \in X(\alpha, P, < t)$, $x \in X(P_\alpha, \alpha, < t)$ and $s \in X(\alpha, P, t)$ then $p_{\alpha,t}(y, x, s) = 0$ if $C(s)$ contains elements from \mathcal{Z} which are not computable from the elements in $C(y) \cup C(x)$.

The formal credential mechanism can be described in a similar way as the “real” credential mechanism of §3; only all numbers in \mathbb{Z}_N^* appearing in the real credential mechanism are replaced by their inverse images under ψ , where ψ is some F-homomorphism. The formal

credential mechanism will satisfy the conditions of §3.5 with \mathfrak{L} replacing \mathbb{Z}_N^* . The only exception is made for the elements of \mathfrak{L} corresponding to the numbers r_l, s_l chosen in step 3 of some $I(i_k, A_j)$: these will correspond to different elements of the set $\{H_1, \dots, H_T\}$ which need not be chosen by means of a uniform distribution. We notice that in particular, Z chooses the sets S in step 6 of each $I(i_k, A_j)$ uniformly from the subsets of $\{1, \dots, n\}$ of cardinality $\frac{1}{2}n$ and independently of the other steps executed at the same moment or before.

Elements of \mathfrak{L} , chosen or computed in the formal credential mechanism will be denoted in the same way as the corresponding messages in the real credential mechanism of §3.3, except that lower case characters appearing in the bases of the expressions in §3.3 are replaced by corresponding capitals, that f is replaced by F and that exponents \bar{b} are replaced by b^{-1} . Apart from that, steps in the formal credential mechanism will be denoted in the same way as the corresponding steps in §3.3. Subprotocols in the formal credential mechanism will be given the same names as the corresponding subprotocols in the real credential mechanism.

6. UNFORGEABILITY

In this section we formulate analogues of properties 1,2 and 3 mentioned in §1.1 for the formal credential mechanism, give a theorem, stating that the probability that these properties do not hold is bounded above by a number which is exponentially small in the security parameter n appearing in step 3 of $I(i_k, A_j)$, and give an example of an attack, showing that the upper bound in the theorem is optimal.

6.1. Statement of the result

We shall use the same notation, and make the same assumptions, as in the previous sections. In particular, \mathfrak{L} will have the meaning of §5.1, $\phi(N), c_1, \dots, c_K, a, p_1, \dots, p_L, q_1, \dots, q_L$ are pairwise coprime, and p_1, \dots, q_L are primes larger than $\frac{1}{2}n$. We put $b_j = p_j^2 c_1 \cdots c_K$ for $j = 1, \dots, L$. For any $P \in \mathfrak{L}$ and $c \in \mathbb{Z}$ with $\gcd(c, \phi(N)) = 1$ we shall refer to $P^{c^{-1}}$ as “credential c on pseudonym P ”. The same computational model for the formal module \mathfrak{L} as introduced in §5.1 will be used.

Consider an attack \mathcal{Q} on the formal credential mechanism in which Z does not cheat. Let P be a pseudonym, used by some representative g_j during an execution of \mathcal{Q} . We say that P is *properly validated* for organization A_j during this execution if the following happens:

- in step 2 of $II(g_j, A_j)$, g_j sends P to A_j together with some message W_P , and $P \neq Q$ and $PW_P \neq QW_Q$ for each other pseudonym Q which was sent to A_j together with W_Q at the same moment or before in step 2 of some other subprotocol $II(g'_j, A_j)$;
- in step 5 of $II(g_j, A_j)$, g_j sends $V_P, T_{2P}, \dots, T_{\frac{1}{2}n, P}$ to A_j such that

$$W_P = \prod_{l=2}^{\frac{1}{2}n} PT_{lP}^{b_l}, \quad V_P = P^{p_j^{-2}} \left(F(P) \prod_{l=2}^{\frac{1}{2}n} F(PT_{lP}^{b_l}) \right)^{(p_j q_j)^{-1}}.$$

There is nothing which prevents organizations from accepting pseudonyms which are not properly validated. However, organizations accepting such pseudonyms will in the worst case only harm themselves, since Z is only willing to compute credentials on pseudonyms which are properly validated. (cf. $II(g_j, A_j)$ and step 4 of $III(g_j, A_j, c)$ in §3.3).

We say that a set of pseudonyms, properly validated during an execution of \mathcal{Q} , has the *unforgeability property* if it satisfies the following three analogues of properties 1,2 and 3 of §1.1:

- it can be partitioned in two ways:
in I-sets each containing the pseudonyms used by the representatives of a fixed individual, and O-sets each containing the pseudonyms which have been properly validated for a fixed organization;
- each I-set and each O-set have at most one pseudonym in common;
- if P is a pseudonym, properly validated before moment t , and c a credential, then $P^{c^{-1}}$ is computable from \mathcal{K} and the set of elements of \mathcal{Z} received by the individuals before moment t if and only if this set contains $Q^{c^{-1}}$ where Q is a pseudonym belonging to the same I-set as P .

The set of pseudonyms for the attack \mathcal{Q} is defined as the stochastic variable of which each value is the set of pseudonyms properly validated during an execution of \mathcal{Q} . Henceforth we shall implicitly assume, when speaking of an attack on the formal credential mechanism, that the choice of any participant α in this attack belongs to the collection \mathcal{C}_α described in §5.3. The following theorem is proved in §7.

THEOREM 2. *Let \mathcal{Q} be any attack on the formal credential mechanism in which Z does not cheat (possibly the credential mechanism itself). Then the probability that the set of pseudonyms for \mathcal{Q} does not have the unforgeability property, is at most $LR \times (\frac{n}{2n})^{-1}$.*

Remark 1. By Stirling's formula, the upper bound mentioned in Theorem 2 is approximately equal to $LR \times (\frac{n}{2n})^{\frac{n}{2}} 2^{-n}$. In the next subsection we shall describe an attack, showing that the upper bound for the probability in Theorem 2 can not be essentially improved.

Remark 2. Each attack on the formal credential mechanism can be "translated" into an attack on the real credential mechanism of §3 by means of an F-homomorphism ψ . In fact, each attack on the real credential mechanism can be considered as such a translation, if values of \mathbf{Z}_N^* , obtained during an execution of such an attack by other means than multiplications, multiplicative inversions or applications of f , are assigned to $\psi(H_1), \dots, \psi(H_T)$.

As a consequence of the remark made at the end of §5.2 it is still unknown whether there is an attack on the real credential mechanism which gives individuals a chance considerably larger than $LR \times (\frac{n}{2n})^{-1}$ of getting validators for a set of pseudonyms not having the properties 1,2 or 3 mentioned in §1.1.

6.2. Attacks on the formal credential mechanism

It will be proved in §7, that a set of pseudonyms, constructed as prescribed in the formal credential mechanism, will have the unforgeability property. Therefore, individuals might only obtain a set of pseudonyms without the unforgeability property by means of some attack on the formal credential mechanism. We assume that Z is trusted by the organizations, whence that Z does not cheat in such an attack. The choices of the individuals in such an attack will have the constraints described in §5.3.

Cheating individuals will only be able to compute a set of validators for a set of pseudonyms without the unforgeability property if they succeed, by means of some attack, in getting elements of \mathcal{Z} from which these validators can be computed. There are two ways by which cheating individuals could get such elements. The first way is a kind of cooperation, in which each cheating individual sends all his received elements of \mathcal{Z} to the other cheating individuals. Such a cooperation can even take place with organizations. The second method by which a cheating individual i_k may obtain appropriate elements of \mathcal{Z} is by sending, in step 5 of some $I(i_k, A_j)$, elements A_{kl} ($l = 1, \dots, n$) to Z which are not all computed in the way described in step 4. Since the set S , generated by Z in step 6, will be chosen uniformly from all subsets of $\{1, \dots, n\}$ of cardinality $\frac{1}{2}n$, and independently of all what happened previously in the attack, i_k may have a considerable chance of not being able to show all R_l, S_l to Z in step 8. Moreover, even if i_k is able to show all these R_l, S_l , the probability that he will get a validator V_{kj} in step 11 which is useful for his purposes, is in general quite small.

An example of an attack. We shall describe an attack on the formal credential mechanism in which a single individual tries to obtain a validator for a pseudonym on which he can compute each credential he likes just by himself. In this attack he need not cooperate with other participants of the credential mechanism.

Let i_k, A_j be an individual and an organization and let g_j be i_k 's representative communicating with A_j . Let $a, p_j, q_j, c_1, \dots, c_K$ have the same meaning as in §3.3, and put $p = p_j, q = q_j, c = c_1 \cdots c_K, b = p^2c$. Since $\gcd(p, c) = 1$ there are integers α, β such that $\alpha p^2 + \beta c = 1$, which i_k can compute easily by means of Euclid's algorithm. All steps we refer to will be in $I(i_k, A_j)$. In step 3 i_k chooses $2n$ elements R_l, S_l ($l = 1, \dots, n$) of \mathcal{Z} , as in the formal credential mechanism. In step 4 he computes

$$A_{kl} := F(U_k^{1-\alpha p^2} (M_l R_l^a)^b) S_l^{pq} \text{ for } l = 1, \dots, \frac{1}{2}n,$$

$$A_{kl} := F(U_k (M_l R_l^a)^b) S_l^{pq} \text{ for } l = \frac{1}{2}n + 1, \dots, n.$$

i_k is able to show R_l, S_l ($l \in S$) to Z in step 8 which for all l in S satisfy the condition of step 9, if and only if Z chooses $S = \{\frac{1}{2}n + 1, \dots, n\}$ in step 6. The probability that Z chooses this set is $(\frac{n}{2n})^{-1}$. Provided that the check in step 9 gives the value 'true', Z sends to i_k the validator

$$V_{kj} := U_k^{p^{-2}} \left(\prod_{l=1}^{\frac{1}{2}n} A_{kl} \right)^{(pq)^{-1}}.$$

By multiplying V_{kj} with $U_k^{-\alpha}(M_1 R_1^a)^b \prod_{l=1}^{\frac{n}{2}} S_l^{-1}$, and using that $1-\alpha p^2 = \beta c$, i_k obtains

$$\tilde{V}_{g_j} := U_{g_j}^{p^{-2}} \left\{ \prod_{l=1}^{\frac{n}{2}} F(U_{g_j} T_l^b) \right\}^{(pq)^{-1}},$$

where

$$U_{g_j} = U_k^{1-\alpha p^2} (M_1 R_1^a)^b = U_k^{\beta c} (M_1 R_1^a)^b,$$

$$T_1 = 1, \text{ and } T_l = M_l R_l^a (M_1 R_1^a)^{-1} \text{ for } l = 1, \dots, \frac{n}{2}.$$

When g_j sends U_{g_j} , $\tilde{W}_{g_j} := \prod_{l=2}^{\frac{n}{2}} U_{g_j} T_l^b$, \tilde{V}_{g_j} , and T_l ($l = 2, \dots, \frac{n}{2}$) to A_j during $II(g_j, A_j)$, then A_j will accept U_{g_j} as a pseudonym. i_k is now able to compute $U_{g_j}^{c_i^{-1}}$ for each credential c_i and show these to A_j even when no other organization has issued this credential to one of i_k 's representatives. Hence no set of pseudonyms containing U_{g_j} can have the unforgeability property. We repeat that the probability of success for i_k is at most $(\frac{n}{2})^{-1}$. If all R individuals would try this attack for all L organizations, and if LR is small compared with $(\frac{n}{2})^{-1}$, then the chance that at least one of the individuals is successful is approximately $LR \times (\frac{n}{2})^{-1}$. This shows that Theorem 2 is optimal.

7. PROOF OF THEOREM 2

In §7.1 we introduce some notation, needed in the proof of Theorem 2. In §7.2 we prove that a set of pseudonyms, constructed in the way prescribed in the formal credential mechanism, has the unforgeability property, and in §§7.3-7.5 we shall prove that whatever attack they try, individuals will have only a very small chance of being able to validate properly a pseudonym which is not of the form described in the formal credential mechanism.

7.1. Notation

For any integral domain \mathcal{R} with unity, we denote by \mathcal{R}^t ($t = 1, 2, \dots$) the \mathcal{R} -module of t -tuples (x_1, \dots, x_t) over \mathcal{R} , and by \mathcal{R}^∞ the \mathcal{R} -module of infinite tuples (x_1, x_2, \dots) over \mathcal{R} of which at most finitely entries are non-zero. The tuple of which all entries are 0, is in each module \mathcal{R}^t ($t = \infty, 1, 2, \dots$) denoted by $\mathbf{0}$.

Let $\mathbb{N}, \mathbb{Z}, \mathbb{Q}$ denote the set of positive integers, the set of all integers, and the set of rational numbers, respectively. For any $c \in \mathbb{N}$, put

$$\tilde{\mathbb{Q}}_c = \left\{ \frac{a}{b} : a, b \in \mathbb{Z}, \gcd(b, c\phi(N)) = 1 \right\}.$$

In particular, $\tilde{\mathbb{Q}} = \tilde{\mathbb{Q}}_1$. If $\alpha, \beta \in \tilde{\mathbb{Q}}$ we write $\alpha \equiv \beta \pmod{c}$ if there is a $\gamma \in \tilde{\mathbb{Q}}_c$ with $\alpha - \beta = \gamma c$. If $\mathbf{a} = (a_1, a_2, \dots)$, $\mathbf{b} = (b_1, b_2, \dots) \in \tilde{\mathbb{Q}}^t$ ($t \in \{\infty, 1, 2, \dots\}$) then we write $\mathbf{a} \equiv \mathbf{b} \pmod{c}$ if $a_i \equiv b_i \pmod{c}$ for all i .

$\bmod c$ for $i = 1, 2, \dots$.

We shall use the same notation as in the previous sections. In particular, we have $\mathcal{Z} = \tilde{\mathbf{Q}}[F, \mathcal{K}] = \tilde{\mathbf{Q}}[\mathcal{K} \cup \mathcal{F}]$, where, similar to §5.1, $\mathcal{K} = \{U_1, \dots, U_R, M_1, \dots, M_n, H_1, \dots, H_T\}$ and \mathcal{F} is the set of images of F . Since \mathcal{F} is enumerable we may write $\mathcal{F} = \{F_1, F_2, F_3, \dots\}$. Thus each element of \mathcal{Z} can be written as

$$\prod_{i=1}^R U_i^{x_i} \prod_{i=1}^{\infty} F_i^{y_i} \prod_{i=1}^n M_i^{w_i} \prod_{i=1}^T H_i^{z_i} \quad (19)$$

or, more compactly, as

$$U^{\mathbf{x}} F^{\mathbf{y}} M^{\mathbf{w}} H^{\mathbf{z}},$$

where

$$\begin{aligned} \mathbf{x} &= (x_1, \dots, x_R) \in \tilde{\mathbf{Q}}^R, \quad \mathbf{y} = (y_1, y_2, \dots) \in \tilde{\mathbf{Q}}^\infty, \\ \mathbf{w} &= (w_1, \dots, w_n) \in \tilde{\mathbf{Q}}^n, \quad \mathbf{z} = (z_1, \dots, z_T) \in \tilde{\mathbf{Q}}^T, \end{aligned}$$

and $U^{\mathbf{x}}$, $F^{\mathbf{y}}$, $M^{\mathbf{w}}$ and $H^{\mathbf{z}}$ are abbreviations of

$$\prod_{i=1}^R U_i^{x_i}, \prod_{i=1}^{\infty} F_i^{y_i}, \prod_{i=1}^n M_i^{w_i}, \text{ and } \prod_{i=1}^T H_i^{z_i},$$

respectively.

Similar to the previous sections, Z denotes the signature authority, A_1, \dots, A_L the organizations, and i_1, \dots, i_R the individuals participating in the formal credential mechanism. As before, $c_1, \dots, c_K, a, p_1, \dots, p_L, q_1, \dots, q_L$ are positive integers which are pairwise coprime, and coprime with $\phi(N)$, and p_1, \dots, q_L are primes larger than $\frac{1}{2}n$. From now on we assume, when referring to the formal credential mechanism, that it is modified in the following way: whenever Z or some representative must send an element of \mathcal{Z} to some individual, then it sends it to all individuals. Thus all individuals will have exactly the same computational abilities in \mathcal{Z} . It is obviously sufficient to prove Theorem 2 for this modified formal credential mechanism.

We say that $X \in \mathcal{Z}$ is *computable by the individuals*, or that the individuals can compute X , at moment t if X is computable from \mathcal{K} and the elements of \mathcal{Z} received by the individuals before moment t during an execution of an attack on the formal credential mechanism. By saying that the individuals can compute X before moment ∞ we mean that there is a moment at which the individuals can compute X . We shall need the following obvious but important fact: if $\mathcal{W} = \{W_1, \dots, W_S\}$ is a subset of \mathcal{Z} containing \mathcal{K} , then each element of \mathcal{Z} which is computable from \mathcal{W} can be written in the form

$$\prod_{i=1}^S W_i^{n_i} \prod_{i=1}^{\infty} F_i^{y_i} \quad (20)$$

where the n_i and y_i are all integers such that at most finitely many of the y_i are non-zero, and

$y_i = 0$ if $F_i = F(X)$ and X is not computable from \mathcal{W} .

7.2. Proof of unforgeability if pseudonyms are properly formed

If P is a pseudonym, used by some representative g_j , which has been properly validated for some organization A_j , then t_P denotes the moment at which the validation of this pseudonym was completed, that is the moment at which step 5 of $H(g_j, A_j)$ was executed.

Consider an execution of some attack in the formal credential mechanism and suppose that the set of pseudonyms properly validated during this attack, \mathcal{E} say, has the following properties:

- for each $P \in \mathcal{E}$, there are a pair (k, j) with $1 \leq k \leq R, 1 \leq j \leq L$, and $R_{kj} \in \mathbb{Z}$ such that

$$P = U_k R_{kj}^b \text{ and the individuals can compute } R_{kj} \text{ at moment } t_P; \quad (21)$$

- for each pair (k, j) with $1 \leq k \leq R, 1 \leq j \leq L$ there is at most one pseudonym P in \mathcal{E} satisfying (21).

Then we have

Lemma 3. \mathcal{E} has the unforgeability property.

Proof. Define the I-sets I_1, \dots, I_R and O-sets O_1, \dots, O_L such that the pseudonym in \mathcal{E} which satisfies (21) belongs to I_k and O_j . Then each I-set and each O-set have at most one pseudonym in common. Let c be a credential, P a pseudonym in I_1 which has been validated before moment t , and suppose that the set consisting of \mathcal{K} and the elements of \mathbb{Z} received by the individuals before moment t , does not contain any $Q^{c^{-1}}$ with $Q \in I_1$. We shall prove that the individuals can not compute $P^{c^{-1}}$ at moment t . By repeating the same argument for the other I-sets, one proves that \mathcal{E} has the unforgeability property.

Let

$$\mathcal{G} = \{U^x F^y M^w H^z : x = (x_1, \dots, x_R) \in \tilde{\mathbb{Q}}^R, x_1 \in \tilde{\mathbb{Q}}_c, y \in \tilde{\mathbb{Q}}^\infty, w \in \tilde{\mathbb{Q}}^n, z \in \tilde{\mathbb{Q}}^T\}$$

and $\mathcal{D}(u)$ the set of elements of \mathbb{Z} computable by the individuals at moment u for each moment u . We shall prove that $\mathcal{D}(t) \subset \mathcal{G}$. Obviously, $\mathcal{D}(0) = \mathbb{Z}[F, \mathcal{K}]$ is contained in \mathcal{G} . For each u with $1 \leq u \leq t$ we show that $\mathcal{D}(u-1) \subset \mathcal{G}$ implies $\mathcal{D}(u) \subset \mathcal{G}$. Then it follows by induction on u that $\mathcal{D}(t) \subset \mathcal{G}$.

Fix u with $1 \leq u \leq t$. To establish our induction step, we have to consider the set of elements of \mathbb{Z} received by the individuals at moment $u-1$. This set consists of either credentials or validators. Suppose that some individual receives a validator at moment $u-1$. This validator is a $p_j^2 q_j$ -th root on some element of $\mathcal{D}(u-2)$. Since \mathcal{G} is closed under exponentiation with numbers in $\tilde{\mathbb{Q}}_c$, this shows that this validator belongs to \mathcal{G} . By a similar argument it follows that credentials $\neq c$ on pseudonyms received by some individual at moment u must belong to \mathcal{G} . Suppose that at moment $u-1$ some credential $Q^{c^{-1}}$ on a pseudonym Q has been received by some individual. Then by assumption, Q must belong to one of the sets I_k with

$k > 1$. Moreover, Q must have been properly validated before moment $u - 1$. Hence by (21), Q can be written as $U_k R_{kj}^{b_j}$ for some $k > 1$ and j and some R_{kj} in $\mathfrak{D}(u - 2)$. Since c divides b_j , this shows that $Q^{c^{-1}} \in \mathcal{G}$. We infer that all elements of \mathfrak{L} , received by the individuals at moment $u - 1$ belong to \mathcal{G} . But since \mathcal{G} is closed under computations, this proves that $\mathfrak{D}(u)$ is contained in \mathcal{G} .

We shall now show that $P^{c^{-1}} \notin \mathcal{G}$. Assume the contrary. By (21) and the fact that P has been properly validated before moment t , there are j and $R_{1j} \in \mathfrak{D}(t)$ such that $P = U_1 R_{1j}^{b_j}$. Since c divides b_j this implies that $U_1^{c^{-1}}$ belongs to $\mathfrak{D}(t)$, whence to \mathcal{G} , which is false. Hence our assumption must have been wrong. This completes the proof of Lemma 3. \square

Roughly speaking, Lemma 3 proves that any set of pseudonyms having the form prescribed in the formal credential mechanism, must have the unforgeability property. In order to prove Theorem 2, it is therefore sufficient to show that the probability that individuals are able to validate properly pseudonyms which are not of the required form is very small. We shall state this more precisely.

By an A_j -validator for a pseudonym P we shall mean a tuple

$$V_j(P) = (V_P, W_P, T_{2P}, \dots, T_{\frac{v_n}{2}n, P}),$$

of elements in \mathfrak{L} such that

$$V_P = P^{p_j^{-2}} \left[F(P) \prod_{l=2}^{\frac{v_n}{2}n} F(PT_{lP}^{b_l}) \right]^{(p_j q_j)^{-1}}, \quad W_P = \prod_{l=2}^{\frac{v_n}{2}n} PT_{lP}^{b_l}, \quad (22)$$

where p_j, q_j, b_j have the same meaning as before. If P has been properly validated for A_j , then A_j has received an A_j -validator for P . Two A_j -validators $V_j(P_1) = (V_{P_1}, W_{P_1}, T_{2,P_1}, \dots, T_{\frac{v_n}{2}n, P_1})$ and $V_j(P_2) = (V_{P_2}, W_{P_2}, T_{2,P_2}, \dots, T_{\frac{v_n}{2}n, P_2})$ for pseudonyms P_1 and P_2 , respectively, are called *equivalent* if, after reordering, the tuples $(P_1, P_1 T_{2,P_1}^{b_1}, \dots, P_1 T_{\frac{v_n}{2}n, P_1}^{b_j})$ and $(P_2, P_2 T_{2,P_2}^{b_1}, \dots, P_2 T_{\frac{v_n}{2}n, P_2}^{b_j})$ are equal. (Loosely speaking this means that the products of the F -values appearing in V_{P_1} and V_{P_2} , respectively, are equal). Note that in none of these tuples, the entries need be distinct. If the validators $V_j(P_1)$ and $V_j(P_2)$ are equivalent, then $P_1 W_{P_1} = P_2 W_{P_2}$. Hence any two pseudonyms which have been properly validated for A_j must have been shown to A_j together with inequivalent validators.

For each attack on the formal credential mechanism, and each j in $\{1, \dots, L\}$, the following events are defined:

E_{1j} : in some execution, there is a pseudonym P in \mathfrak{L} , properly validated for A_j , such that none of the roots $(PU_k^{-1})^{b_j}$ can be computed by the individuals at moment t_P , for $k = 1, \dots, R$.

E_{2j} : in some execution, there is a moment t at which the individuals can compute: two pseudonyms P_1 and P_2 ; the b_j -th root of their quotient, $(P_1 P_2^{-1})^{b_j}$; and two *inequivalent* A_j -validators $V_j(P_1)$ and $V_j(P_2)$ for P_1 and P_2 respectively.

Informally, E_{1j} is the event, that a pseudonym, properly validated for A_j , is not of the form

prescribed in the credential mechanism, and E_{2j} is the event that two “similar” pseudonyms have been properly validated with “non-similar” validators.

Suppose that in some execution of an attack on the formal credential mechanism, none of the events E_{1j}, E_{2j} ($1 \leq j \leq L$) takes place. Let \mathcal{E} be the set of properly validated pseudonyms in this execution. Then each pseudonym in \mathcal{E} must satisfy (21) for some pair (k, j) , since none of the events E_{1j} takes place. Moreover, no two pseudonyms in \mathcal{E} can satisfy (21) for the same pair (k, j) . For if two pseudonyms, P_1, P_2 say, in \mathcal{E} , would have satisfied (21) for the same pair (k, j) , then at some moment t the individuals would have been able to compute P_1, P_2 and $(P_1 P_2^{-1})^{b_j}$. But since none of the events E_{2j} was supposed to take place, the validators for P_1 and P_2 must be equivalent, which contradicts the fact that these pseudonyms have been properly validated.

Together with Lemma 3 this implies the following: the probability that the set of pseudonyms for (an attack on) the formal credential mechanism does not have the unforgeability property is at most equal to the probability of event $E_{11} \cup E_{12} \cup \dots \cup E_{1L} \cup E_{2L}$. Theorem 3 states, in a more precise form, that this event has probability at most $LR \times (\frac{n}{\lambda n})^{-1}$. This implies Theorem 2 at once.

THEOREM 3. *For each attack on the formal credential mechanism in which Z does not cheat, we have*

$$\Pr[E_{1j}] \leq R \times (\frac{n}{\lambda n})^{-1} \text{ and } \Pr[E_{2j}] = 0 \text{ for } j = 1, \dots, L.$$

7.3. Preliminaries to the proof of Theorem 3

We shall use the same notation as in the previous sections. In particular, i_1, \dots, i_R will be the individuals participating in the credential mechanism. We fix j in $\{1, \dots, L\}$ and put $p = p_j$, $q = q_j$ and $b = b_j$. As mentioned before, each element of \mathcal{Z} can be expressed as a finite product of powers of which the bases belong to $\mathcal{K} \cup \mathcal{F}$ and the exponents to $\tilde{\mathbb{Q}}$. We shall show, that any condition that the individuals can ever compute certain elements of \mathcal{Z} can be expressed as the solvability of some system of linear equations modulo p^2q of which some of the coefficients are stochastic variables and the unknowns belong to $\tilde{\mathbb{Q}}$.

Consider an attack on the formal credential mechanism, and denote the changed version of $I(i_k, A_j)$ in this attack also by $I(i_k, A_j)$. In step 5 of $I(i_k, A_j)$, i_k sends elements A_{kl} ($l = 1, \dots, n$) to Z , which might have been computed in an other way than described in the formal credential mechanism, and might have been chosen by means of a probability distribution depending on all steps previously executed in which some individual was involved. In step 6, Z chooses a set \mathcal{S}_k (which is now indexed in order to distinguish sets \mathcal{S} generated in different $I(i_k, A_j)$), by means of a probability distribution which is uniform on the collection of subsets of $\{1, \dots, n\}$ of cardinality $\frac{1}{2}n$, and independent of all other steps executed at the same moment or before in the credential mechanism. In step 8, i_k has to send R_l, S_l to Z for each l in \mathcal{S}_k such that

$A_{kl} = F(U_k(M_l R_l^q)^b) S_l^{pq}$. If Z concludes in step 9 that indeed i_k constructed each A_{kl} with $l \in \mathbb{S}_k$ in the proper way, then he sends in step 11,

$$V_{kj} = U_k^{p^{-2}} \left\{ \prod_{l \notin \mathbb{S}_k} A_{kl} \right\}^{(pq)^{-1}}$$

to i_k .

From now on, we exclude attacks useless to the individuals in which some security check by Z in $I(i_k, A_j)$ other than that in step 9 is ‘false’, or that i_k , while knowing the correct R_l, S_l for some l in \mathbb{S}_k in step 8, sends false R_l or S_l to Z .

For each k in $\{1, \dots, R\}$ we introduce the following notation:

$$Y_k = \prod_{l=1}^n A_{kl}, \quad V_k = \left[U_k^{p^{-2}} (Y_k \left\{ \prod_{l \in \mathbb{S}_k} F(U_k(M_l R_l^q)^b) \right\}^{-1})^{(pq)^{-1}} \right]^{\xi_k},$$

where $\xi_k = 1$ if the check in step 9 of $I(i_k, A_j)$ gave the value ‘true’ and $\xi_k = 0$ otherwise. Hence if the check in step 9 did not fail, V_k is equal to $V_{kj} \prod_{l \in \mathbb{S}_k} S_l$, whereas $V_{kj} = 1$ if this check failed. We

notice, that Y_k is controlled completely by i_k ; it is independent of \mathbb{S}_k . We assume from now on that i_k receives V_k from Z instead of V_{kj} in step 11, for $k = 1, \dots, R$. This does not change the computational abilities of the individuals.

Let t_k be the moment at which step 11 of $I(i_k, A_j)$ is executed, that is the moment at which i_k (and so the other individuals) receives V_k from Z . We shall derive the upper bounds in Theorem 3 subject to the condition that

$$t_1 \leq t_2 \leq \dots \leq t_R. \tag{23}$$

By repeating the argument for each other possible order of the t_k ’s, one can prove Theorem 3.

Henceforth we shall assume (23). Each i_k sends A_{k1}, \dots, A_{kn} to Z before moment t_k . Hence for $k = 1, \dots, R$, the individuals can compute Y_k before they receive V_1, \dots, V_R . In view of (20), the Y_k ’s have the form

$$\begin{aligned} Y_1 &= U^{x_1} F^{y_1} M^{w_1} H^{z_1}, \\ Y_k &= V_1^{\delta_{1k}} \cdots V_{k-1}^{\delta_{k-1,k}} U^{x_k} F^{y_k} M^{w_k} H^{z_k} \text{ for } k = 2, \dots, R, \end{aligned} \tag{24}$$

where $\delta_{1k}, \dots, \delta_{k-1,k} \in \tilde{\mathbb{Q}}_{pq}^R$, $x_k \in \tilde{\mathbb{Q}}_{pq}^R$, $y_k \in \tilde{\mathbb{Q}}_{pq}^\infty$, $w_k \in \tilde{\mathbb{Q}}_{pq}^n$, $z_k \in \tilde{\mathbb{Q}}_{pq}^T$. Apart from V_1, \dots, V_R , individuals may receive validators for other organizations, or credentials on pseudonyms, which are all d -th roots on messages previously computable by the individuals, where d is a positive integer coprime with pq . We took this into consideration by allowing that the coefficients in (24) belong to $\tilde{\mathbb{Q}}_{pq}$ rather than \mathbb{Z} .

For $k = 1, \dots, R$, let \mathcal{T}_k be the set of positive integers j such that $F_j = F(U_k(M_l R_l^q)^b)$ for some R_l sent by i_k to Z in step 8 of $I(i_k, A_j)$, and define $e(\mathcal{T}_k) \in \tilde{\mathbb{Q}}^\infty$ by

$$F^{\mathbf{e}(\mathfrak{T}_k)} = \prod_{l \in \mathbb{S}_k} F(U_k(M_l R_l^q)^b).$$

Moreover, $\mathbf{d}_k \in \tilde{\mathbb{Q}}_R$ denotes the vector of which the k -th coordinate is 1, and the other coordinates are 0. With this notation we have

$$V_k = \left(U^{p^{-2}\mathbf{d}_k} \left[Y_k F^{-\mathbf{e}(\mathfrak{T}_k)} \right]^{(pq)^{-1}} \right)^{\xi_k} \text{ for } k = 2, \dots, R. \quad (25)$$

The following lemma is crucial in the proof of Theorem 3.

Lemma 4. Consider an execution of some attack on the formal credential mechanism in which Y_1, \dots, Y_R satisfy (24) and V_1, \dots, V_R satisfy (25). Suppose that at a moment t_h+1 (where $h \in \{1, \dots, R\}$ and $t_{R+1} := \infty$), the individuals can compute $U^{\mathbf{u}} F^{\mathbf{f}} M^{\mathbf{m}} H^{\mathbf{h}} \in \mathcal{L}$, where $\mathbf{u} \in \tilde{\mathbb{Q}}^R$, $\mathbf{f} \in \tilde{\mathbb{Q}}^\infty$, $\mathbf{m} \in \tilde{\mathbb{Q}}^n$ and $\mathbf{h} \in \tilde{\mathbb{Q}}^T$. Then there are $m_1, \dots, m_h \in \tilde{\mathbb{Q}}$ such that

$$\sum_{k=1}^h m_k \xi_k (\mathbf{d}_k + p q^{-1} \mathbf{x}_k) \equiv p^2 \mathbf{u} \pmod{p^2}, \quad (26)$$

$$\sum_{k=1}^h m_k \xi_k (\mathbf{y}_k - \mathbf{e}(\mathfrak{T}_k)) \equiv p q \mathbf{f} \pmod{p}, \quad (27)$$

$$\sum_{k=1}^h m_k \xi_k (\mathbf{y}_k - \mathbf{e}(\mathfrak{T}_k)) \equiv p q \mathbf{f} \pmod{q}. \quad (28)$$

Moreover, if $p^2 \mathbf{u} \in \tilde{\mathbb{Q}}_p^R$, then $m_1 \xi_1, \dots, m_h \xi_h$ belong to $\tilde{\mathbb{Q}}_p$.

Proof. By (21) there are $n_1, \dots, n_h \in \tilde{\mathbb{Q}}_{pq}$, $\mathbf{a} \in \tilde{\mathbb{Q}}_{pq}^R$, $\mathbf{b} \in \tilde{\mathbb{Q}}_{pq}^\infty$, $\mathbf{c} \in \tilde{\mathbb{Q}}_{pq}^n$ and $\mathbf{d} \in \tilde{\mathbb{Q}}_{pq}^T$ such that

$$U^{\mathbf{u}} F^{\mathbf{f}} M^{\mathbf{m}} H^{\mathbf{h}} = V_1^{n_1} \cdots V_h^{n_h} U^{\mathbf{a}} F^{\mathbf{b}} M^{\mathbf{c}} H^{\mathbf{d}}. \quad (29)$$

For $k = 1, \dots, R$, put

$$W_k = U_k^{p^{-2}} \left[U^{\mathbf{x}_k} F^{\mathbf{y}_k - \mathbf{e}(\mathfrak{T}_k)} M^{\mathbf{w}_k} H^{\mathbf{z}_k} \right]^{(pq)^{-1}}. \quad (30)$$

Then by (24) and (25),

$$W_k^{\xi_k} = V_k \left(\prod_{j=1}^{k-1} V_j^{-\delta_{jk} \xi_k} \right)^{(pq)^{-1}}.$$

Using these relationships, it is possible to express each V_k as a product of powers of $W_1^{\xi_1}, \dots, W_h^{\xi_h}$, in which the exponents belong to $\tilde{\mathbb{Q}}$ and may have denominators divisible by p or q . Together with (29) this shows that there are m_1, \dots, m_h in $\tilde{\mathbb{Q}}$ such that

$$U^{\mathbf{u}} F^{\mathbf{f}} M^{\mathbf{m}} H^{\mathbf{h}} = W_1^{m_1 \xi_1} \cdots W_h^{m_h \xi_h} U^{\mathbf{a}} F^{\mathbf{b}} M^{\mathbf{c}} H^{\mathbf{d}}.$$

By combining this with (30) and equating the exponents of U and F , we obtain

$$\sum_{k=1}^h m_k \xi_k (p^{-2} \mathbf{d}_k + (pq)^{-1} \mathbf{x}_k) + \mathbf{a} = \mathbf{u},$$

$$\sum_{k=1}^h m_k \xi_k (pq)^{-1} (\mathbf{y}_k - \mathbf{e}(\mathcal{T}_k)) + \mathbf{b} = \mathbf{f}.$$

(We shall not need the relationships coming from the exponents of M and H). Now we obtain (26) by multiplying the first equation with p^2 and reducing it modulo p^2 , and (27), (28) by multiplying the second equation with pq and reducing it modulo p and q , respectively.

Suppose that $p^2 \mathbf{u} \in \tilde{\mathbb{Q}}_p^R$, and that not all numbers $\xi_k m_k$ with $1 \leq k \leq h$ belong to $\tilde{\mathbb{Q}}_p$. Then there is an integer d , divisible by p , such that all numbers $dm_k \xi_k$ are integers and at least one of them is not divisible by p . But by multiplying (26) with d and reducing it modulo p , we obtain

$$\sum_{k=1}^h dm_k \xi_k \mathbf{d}_k \equiv 0 \pmod{p},$$

whence $dm_k \xi_k \equiv 0 \pmod{p}$ for $k = 1, \dots, h$. This contradiction shows that all $m_k \xi_k$ must belong to $\tilde{\mathbb{Q}}_p$ if $p^2 \mathbf{u} \in \tilde{\mathbb{Q}}_p^R$. This completes the proof of Lemma 4. \square

The following consequence of Lemma 4 will be useful.

Lemma 5. *Let $\mathbf{f} \in \mathbb{Z}^\infty$ and suppose that all coordinates of \mathbf{f} have absolute values smaller than p . If there is a moment at which the individuals can compute $F^{(pq)^{-1}} \mathbf{f}$, then $\mathbf{f} = \mathbf{0}$.*

Proof. Suppose that at some moment, the individuals can compute $F^{(pq)^{-1}} \mathbf{f}$. Then by Lemma 4, eq. (26), there are m_1, \dots, m_R , with $m_1 \xi_1, \dots, m_R \xi_R \in \tilde{\mathbb{Q}}_p$, such that

$$\sum_{k=1}^R m_k \xi_k (\mathbf{d}_k + pq^{-1} \mathbf{x}_k) \equiv \mathbf{0} \pmod{p^2}.$$

By reducing this equation modulo p , and using that q is coprime with p , we obtain $m_k \xi_k \equiv 0 \pmod{p}$ for $k = 1, \dots, R$. A substitution of this into (27) yields that $\mathbf{f} \equiv \mathbf{0} \pmod{p}$. But since by assumption, the absolute values of the coordinates of \mathbf{f} are smaller than p this proves Lemma 5. \square

7.4. Proof of $Pr[E_{2j}] = 0$

Consider an execution of some attack on the formal credential mechanism in which Z does not cheat. Suppose that in this execution there is a moment t at which the individuals can compute pseudonyms P_1, P_2 , the b -th root of their quotient $(P_1 P_2^{-1})^{b^{-1}}$ and A_j -validators $V_j(P_1), V_j(P_2)$ for P_1 and P_2 respectively, where

$$V_j(P_i) = (V_{P_i}, W_{P_i}, T_{2,P_i}, \dots, T_{\frac{1}{2}n, P_i}) \text{ for } i = 1, 2.$$

We have to prove that $V_j(P_1)$ and $V_j(P_2)$ are equivalent.

Put $T_{1,P_1} = T_{1,P_2} = 1$ and let $\mathbf{f}(i) \in \tilde{\mathbb{Q}}^\infty$ be defined by

$$F^{\mathbf{f}(i)} = \prod_{l=1}^{\frac{1}{2}n} F(P_l T_{l,P_i}^b)$$

for $i = 1, 2$. Then

$$V_{P_i} = P_i^{p^{-2}} F^{(pq)^{-1}\mathbf{f}(i)}$$

for $i = 1, 2$. At moment t the individuals can compute

$$(P_2 P_1^{-1})^{b^{-1}} V_{P_1} V_{P_2}^{-1} = F^{(pq)^{-1}(\mathbf{f}(1) - \mathbf{f}(2))}.$$

For $i = 1, 2$ the coordinates of $\mathbf{f}(i)$ are non-negative integers of which the sum is equal to $\frac{1}{2}n$. Moreover, we assumed that $p > \frac{1}{2}n$. Hence the coordinates of $\mathbf{f}(1) - \mathbf{f}(2)$ have absolute values less than p . Together with Lemma 5 this shows that $\mathbf{f}(1) = \mathbf{f}(2)$. But this means exactly that $V_j(P_1)$ and $V_j(P_2)$ are equivalent. \square

7.5. Proof of $\Pr[E_{1j}] \leq R \times (\frac{n}{\frac{1}{2}n})^{-1}$

Consider an attack \mathcal{Q} on the credential mechanism in which Z does not cheat, and suppose that during some execution of \mathcal{Q} a pseudonym P is properly validated at a moment t_P , at which none of the b -th roots $(PU_k^{-1})^{b^{-1}}$ ($k = 1, \dots, R$) is computable by the individuals. We have to prove that this can happen with probability at most $R \times (\frac{n}{\frac{1}{2}n})^{-1}$. In the proof of this fact we need some further notation which is introduced below.

We recall that t_k is the moment at which step 11 of $I(i_k, A_j)$ is executed, and that A_{k1}, \dots, A_{kn} are the numbers which i_k sent to Z in step 5 of $I(i_k, A_j)$. We define s_k as the moment at which step 8 of $I(i_k, A_j)$ is executed, that is the moment at which i_k shows R_l and S_l to Z for $l \in \mathbb{S}_k$. The stochastic partial function $f_k : \{1, \dots, n\} \rightarrow \mathbb{N}$ is defined as follows: if at moment s_k the individuals can compute R_l, S_l such that $A_{kl} = F(U_k(M_l R_l^a)^b) S_l^{pq}$ then we put $f_k(l) = j$, where j is the positive integer determined by $F_j = F(U_k(M_l R_l^a)^b)$. If the individuals can not compute such R_l and S_l then we do not define $f_k(l)$. f_k is well-defined, that is at moment s_k the individuals can compute at most one pair (R_l, S_l) for each A_{kl} . For suppose that at moment s_k the individuals can compute R_{1l}, S_{1l} and R_{2l}, S_{2l} with $A_{kl} = F_1 S_{1l}^{pq} = F_2 S_{2l}^{pq}$, where $F_i = F(U_k(M_l R_{il}^a)^b)$ for $i = 1, 2$. Then at moment s_k the individuals can compute $(F_1 F_2^{-1})^{(pq)^{-1}} = S_{1l} S_{2l}^{-1}$. By applying Lemma 5 we obtain $F_1 = F_2$, whence $R_{1l} = R_{2l}$, $S_{1l} = S_{2l}$, as required.

In the first step of the proof we show that f_k is injective. Suppose that f_k is defined in both l and m , where l and m are distinct integers in $\{1, \dots, n\}$, and that $f_k(l) = f_k(m)$. Then at moment s_k the individuals can compute R_l and R_m such that $U_k(M_l R_l^a)^b = U_k(M_m R_m^a)^b$. But this implies that $(M_l M_m^{-1})^{a^{-1}} = R_l R_m^{-1}$. Hence the individuals can compute $(M_l M_m^{-1})^{a^{-1}}$ at moment s_k . But this is impossible. For since all elements of \mathcal{Z} received by the individuals are of the form $D^{e^{-1}}$, where D was computable by the individuals before this message was received and e is an integer coprime with a , all elements of \mathcal{Z} ever computable by the individuals must be of the form

$U^u F^f M^m H^h$, where all coordinates of u , f , m and h have denominators coprime with a .

In the second step of the proof we show that for $k = 1, \dots, R$, \mathbb{S}_k is statistically independent of x_j, y_j and f_j for $1 \leq j \leq k$ and \mathbb{S}_j for $1 \leq j < k$. The steps in a subprotocol are executed at consecutive moments. In particular, Z chooses \mathbb{S}_k at moment $s_k - 2$, uniformly from the collection of subsets of $\{1, \dots, n\}$ of cardinality $\frac{1}{2}n$, and independently of the steps executed at or before moment $s_k - 2$. Together with (23), this proves that \mathbb{S}_k is independent of x_j and y_j for $1 \leq j \leq k$ and \mathbb{S}_j for $1 \leq j < k$. By definition, the set of elements of \mathcal{Z} which is computable by the individuals at moment s_k is equal to the set of elements of \mathcal{Z} which is computable from \mathcal{H} and the elements of \mathcal{Z} received by the individuals before moment s_k . The elements of \mathcal{Z} sent to the individuals at moment $s_k - 1$ are all roots of elements of \mathcal{Z} which were computable by the individuals at moment $s_k - 2$. Hence the partial function f_k is completely determined by the set of messages executed at or before moment $s_k - 2$. This proves that \mathbb{S}_k is also independent of f_1, \dots, f_k .

For $k = 1, \dots, R$ we put $\mathcal{U}_k = f_k(\{1, \dots, n\})$, and $\mathcal{W}_k = \mathcal{U}_1 \cup \dots \cup \mathcal{U}_k$. It is easy to check that $\mathcal{T}_k = f_k(\mathbb{S}_k)$, where \mathcal{T}_k is the set defined in (25). From the injectivity of f_k it follows that $\xi_k = 1$ (the check in step 9 of $I(i_k, A_j)$ does not fail) if and only if $\#\mathcal{T}_k = \frac{1}{2}n$ and that the vector $\mathbf{e}(\mathcal{T}_k)$, defined in (25), is equal to (e_1, e_2, \dots) , where $e_j = 1$ if $j \in \mathcal{T}_k$ and $e_j = 0$ otherwise. If \mathcal{R} is some integral domain, \mathcal{U} a finite subset of \mathbb{N} , and $\mathbf{y} = (y_1, y_2, \dots) \in \mathcal{R}^\infty$, then we denote by $\mathbf{y}(\mathcal{U})$ the tuple $(\tilde{y}_1, \tilde{y}_2, \dots)$ with $\tilde{y}_i = y_i$ if $i \in \mathcal{U}$ and $\tilde{y}_i = 0$ otherwise.

In the third step of the proof we shall show that there exist an integer T with $1 \leq T \leq R$ and integers m_k for $1 \leq k \leq T$ such that $m_k \xi_k$ is not divisible by q for at least one k in $\{1, \dots, T\}$ and

$$\sum_{k=1}^T m_k \xi_k (\mathbf{y}_k(\mathcal{W}_T) - \mathbf{e}(\mathcal{T}_k)) \equiv 0 \pmod{q}. \quad (31)$$

Let $V_j(P) = (V_P, W_P, T_{2P}, \dots, T_{\frac{1}{2}n, P})$ be an A_j -validator for P , let \mathcal{T} be the set of integers j such that $F_j = F(PT_{lP}^b)$ for some l in $\{1, \dots, \frac{1}{2}n\}$, where $T_{1P} = 1$. Define the vector $\mathbf{f}(\mathcal{T}) \in \tilde{\mathbb{Q}}^\infty$ by

$$F^{\mathbf{f}(\mathcal{T})} = \prod_{l=1}^{\frac{1}{2}n} F(PT_{lP}^b).$$

Since $\gcd(p, q) = 1$, there are rational integers α, β with $\alpha p + \beta q = 1$. It follows (cf (22)) that

$$\tilde{V} := F^{q^{-1}\mathbf{f}(\mathcal{T})} = (V_P^p P^{-1})^\alpha F^{\beta \mathbf{e}(\mathcal{T})}$$

can be computed by the individuals at moment t_P . By Lemma 4, eq. (28), with $h = R$, there are $\tilde{m}_1, \dots, \tilde{m}_R \in \tilde{\mathbb{Q}}$ such that

$$\sum_{k=1}^R \tilde{m}_k \xi_k (\mathbf{y}_k - \mathbf{e}(\mathcal{T}_k)) \equiv p \mathbf{f}(\mathcal{T}) \pmod{q}. \quad (32)$$

Since p and q are distinct primes larger than $\frac{1}{2}n$ and the coordinates of $\mathbf{f}(\mathcal{T})$ are non-negative and have sum $\frac{1}{2}n$, not all $\tilde{m}_k \xi_k$ are integers divisible by q . Let T be the smallest integer with $1 \leq T \leq R$ such that (32) is solvable with $\tilde{m}_k \xi_k \equiv 0 \pmod{q}$ for $k > T$. Then

$$\sum_{k=1}^T \tilde{m}_k \xi_k (y_k - e(\mathcal{T}_k)) \equiv p f(\mathcal{T}) \pmod{q}, \quad (33)$$

and $\tilde{m}_T \xi_T$ is not divisible by q . By Lemma 4 with $h = T$, \tilde{V} is not computable by the individuals before moment t_T , so definitely not at moment s_T . But since the individuals can compute \tilde{V} at moment t_P , we have

$$s_T < t_P. \quad (34)$$

If the sets \mathfrak{W}_T and \mathcal{T} would have an element in common, then the individuals were able to compute R_1 and R_2 at moment s_T such that $U_k R_1^b = P R_2^b$ for some $k \leq T$. Together with (34) this would imply that before moment t_P the individuals can compute $(P U_k^{-1})^{b^{-1}}$, which is against our assumption. Hence $\mathfrak{W}_T \cap \mathcal{T} = \emptyset$. By applying this to (33), using that the coordinates of $f(\mathcal{T})$ are 0 on the places outside \mathcal{T} and considering only the coordinates of both sides of (33) on the places in the set \mathfrak{W}_T , and multiplying each \tilde{m}_k with d , where d is the smallest positive integer d such that all $d\tilde{m}_k$ are integers for $k = 1, \dots, T$, we obtain that (31) is satisfied with $m_k := d\tilde{m}_k$ for $1 \leq k \leq T$, and that $m_k \xi_k$ is not divisible by q for at least one k in $\{1, \dots, T\}$.

In the fourth step we shall prove that,

$$\mathfrak{U}_i \cap \mathfrak{U}_h = \emptyset \text{ for } 1 \leq i < h \leq R. \quad (35)$$

Assume that (35) is false and let i and h be integers with $1 \leq i < h \leq R$ such that \mathfrak{U}_i and \mathfrak{U}_h have an element in common. Then at moment s_h the individuals can compute R_1 and R_2 such that $U_i R_1^b = U_h R_2^b$. But this implies that at moment s_h , so definitely before moment t_h , they can compute $U^{p^{-2}(\mathbf{d}_i - \mathbf{d}_h)}$. Together with Lemma 4 this shows that there are m_1, \dots, m_{h-1} in $\tilde{\mathbb{Q}}_p$ with

$$\sum_{k=1}^{h-1} m_k \xi_k (\mathbf{d}_k + p q^{-1} \mathbf{x}_k) \equiv \mathbf{d}_i - \mathbf{d}_h \pmod{p^2}.$$

By comparing the h -th coordinates on both sides of this equation and reducing them modulo p , we obtain that $0 \equiv -1 \pmod{p}$, which is impossible. Hence our assumption that (35) is false was wrong.

Our assertion that $Pr[E_{1j}] \leq R \times \left(\frac{n}{\sqrt{n}}\right)^{-1}$ follows at once by combining the results of the previous four steps with Lemma 6 below with $K = GF(q)$.

Lemma 6. *Let K be a field and let $y_1, \dots, y_R, f_1, \dots, f_R, \mathcal{S}_1, \dots, \mathcal{S}_R$ be stochastic variables, defined on the same probability space, such that*

y_1, \dots, y_R assume there values in K^∞ ;

f_1, \dots, f_R are injective partial functions : $\{1, \dots, n\} \rightarrow \mathbb{N}$ such that the sets $\mathfrak{U}_k := f_k(\{1, \dots, n\})$ are pairwise disjoint;

$\mathcal{S}_1, \dots, \mathcal{S}_R$ are subsets of $\{1, \dots, n\}$;

for each k , the distribution of \mathcal{S}_k is uniform on the collection of subsets of $\{1, \dots, n\}$ of cardinality $\frac{1}{2}n$ and independent of y_k, f_k and y_j, f_j, \mathcal{S}_j for $1 \leq j < k$.

For $k = 1, \dots, R$, let $\mathfrak{W}_k = \mathfrak{U}_1 \cup \dots \cup \mathfrak{U}_k$, $\mathcal{T}_k = f_k(\mathcal{S}_k)$,

$\xi_k = 1$ if $\#(\mathcal{T}_k) = \frac{1}{2}n$ and $\xi_k = 0$ otherwise,

and $\mathbf{e}(\mathcal{T}_k) \in K^\infty$ the vector of which the coordinates are 1 on the places in \mathcal{T}_k and 0 on the places outside \mathcal{T}_k .

For $T = 1, \dots, R$, let X_T be the event that there are $m_1, \dots, m_T \in K$ such that $m_k \xi_k \neq 0$ for at least one k with $1 \leq k \leq T$ and

$$\sum_{k=1}^T m_k \xi_k (\mathbf{y}_k(\mathcal{W}_T) - \mathbf{e}(\mathcal{T}_k)) = 0. \quad (36)$$

Then

$$Pr[X_1 \cup \dots \cup X_R] \leq R \times \left(\frac{n}{\frac{1}{2}n}\right)^{-1}.$$

Proof. For $T = 1, \dots, R$ let X_T^c the event that X_T does not take place. It suffices to prove that

$$Pr[X_1] \leq \left(\frac{n}{\frac{1}{2}n}\right)^{-1} \text{ and } Pr[X_T \cap X_{T-1}^c] \leq \left(\frac{n}{\frac{1}{2}n}\right)^{-1} \text{ for } T = 2, \dots, R, \quad (37)$$

since

$$Pr[X_1 \cup \dots \cup X_R] \leq Pr[X_1] + \sum_{T=2}^R Pr[X_T \cap X_{T-1}^c].$$

We shall prove (37) only for $T > 1$; the argument for $T = 1$ is even easier and is therefore omitted. Fix T in $\{2, \dots, R\}$ and denote by W the stochastic tuple consisting of $\mathbf{y}_j, f_j (j \leq T)$ and $\mathcal{S}_j (j < T)$. Let \mathcal{Q} be the set of values for W for which (36) has a non-trivial solution (i.e. a solution with $m_k \xi_k \neq 0$ for at least one k in $\{1, \dots, T\}$) but the system

$$\sum_{k=1}^{T-1} m_k \xi_k (\mathbf{y}_k(\mathcal{W}_{T-1}) - \mathbf{e}(\mathcal{T}_k)) = 0 \text{ in } m_1, \dots, m_{T-1} \in K \quad (38)$$

has only solutions with $m_k \xi_k = 0$ for $k = 1, \dots, T-1$. Fix w in \mathcal{Q} and denote the entries of w by $y_j, f_j (1 \leq j \leq T)$ and $\mathcal{S}_j (1 \leq j < T)$. Let m_1, \dots, m_T be a non-trivial solution of (36). By (38), $m_T \xi_T$ is non-zero. Hence \mathcal{W}_T has cardinality at least $\frac{1}{2}n$. Moreover, there are $m'_1, \dots, m'_{T-1} \in K$ such that

$$\sum_{k=1}^{T-1} m'_k \xi_k (\mathbf{y}_k(\mathcal{W}_T) - \mathbf{e}(\mathcal{T}_k)) = \mathbf{y}_T(\mathcal{W}_T) - \mathbf{e}(\mathcal{T}_T). \quad (39)$$

By combining this with the fact that the sets \mathcal{W}_k are pairwise disjoint, and considering only the coordinates on the places in \mathcal{W}_{T-1} , we obtain

$$\sum_{k=1}^{T-1} m'_k \xi_k (\mathbf{y}_k(\mathcal{W}_{T-1}) - \mathbf{e}(\mathcal{T}_k)) = \mathbf{y}_T(\mathcal{W}_{T-1}). \quad (40)$$

If (40) could be satisfied by two different tuples $(m'_k \xi_k : k = 1, \dots, T-1)$, then a subtraction of these tuples would yield a non-trivial solution of (38) which does not exist by assumption. Hence $m'_1 \xi_1, \dots, m'_{T-1} \xi_{T-1}$ are uniquely determined by (40). But this implies that the set \mathcal{T}_T is

uniquely determined by (39). By using that f_k is injective, we obtain that S_T is uniquely determined by (39). But S_T is uniformly distributed on the collection of $\binom{n}{\lfloor \frac{1}{2}n \rfloor}$ subsets of $\{1, \dots, n\}$ of cardinality $\lfloor \frac{1}{2}n \rfloor$, and independently of W . Hence $Pr[X_T | W=w] \leq \binom{n}{\lfloor \frac{1}{2}n \rfloor}^{-1}$ for each $w \in \mathcal{Q}$.

Therefore

$$Pr[X_T \cap X_{T-1}^c] = \sum_{w \in \mathcal{Q}} Pr[X_T, W=w] = \sum_{w \in \mathcal{Q}} Pr[X_T | W=w]Pr[W=w] \leq \binom{n}{\lfloor \frac{1}{2}n \rfloor}^{-1}.$$

This proves (37). \square

8. POSSIBLE EXTENSIONS

In this section we briefly mention some extensions of the credential mechanism presented in this paper.

An advantage of this credential mechanism is its flexibility. It has only one major restriction: the set of credentials must be public and fixed before validators are issued, and the amount of computation required also depends linearly on the cardinality of this set. A credential mechanism presented in [Ch 84], which is a variant of that considered here, solves these problems in a natural way. For that mechanism, it is possible to prove an analogue of Theorem 2. A result as strong as Theorem 1 does not hold, but instead one can prove that in essence almost no information about the relationship between pseudonyms used with different organizations is revealed.

We also considered a variation on the credential mechanism that differs mainly in that the validators shown to the organizations are just RSA-signatures on products of the values of the one-way function. For this variation we were able to prove an analogue of Theorem 2 which is not quite as tight as the result in this paper. This variation has the advantage that it can be easily extended—without loss of security for the organizations or privacy of the individuals—to a credential mechanism in which each I-set and O-set can have a restricted number of pseudonyms in common which may be larger than 1. Such an extension may be useful in practice.

In [Ch 84], several ways were presented to build more elaborate and potentially useful structures from these basic credential mechanisms, but the security of such constructions is as yet unproved.

Acknowledgement

The authors thank Bert den Boer and Jeroen van de Graaf for their comments and for their suggestions to improve the presentation.

References

- [Ch 84] D. Chaum, *Showing credentials without identification: transferring signatures between unconditionally unlinkable pseudonyms.* Preprint, available from the author.
- [Ch 85] D. Chaum, *Security without identification: transaction systems to make big brother obsolete.* Communications of the ACM , 28(10), Oct. 1985
- [DLM 82] R.A. DeMillo, N.A. Lynch, M.J. Merritt, *Cryptographic protocols.* In Proc. 14th ACM Symposium on Theory of computing, pp. 383-400. ACM, 1982.
- [EGS 85] S. Even, O. Goldreich, A. Shamir, *On the security of ping-pong protocols when implemented using the RSA.* Presented at Crypto 85, Santa Barbara, August 1985.

How to Prove All NP Statements in Zero-Knowledge and a Methodology of Cryptographic Protocol Design

(Extended Abstract)

Oded Goldreich

Dept. of Computer Sc.
Technion
Haifa, Israel

Silvio Micali

Lab. for Computer Sc.
MIT
Cambridge, MA 02139

Avi Wigderson

Inst. of Math. and CS
Hebrew University
Jerusalem, Israel

ABSTRACT

Under the assumption that encryption functions exist, we show that
all languages in NP possess zero-knowledge proofs.

That is, it is possible to demonstrate that a CNF formula is satisfiable without revealing any other property of the formula. In particular, without yielding neither a satisfying assignment nor weaker properties such as whether there is a satisfying assignment in which $x_1=TRUE$, or whether there is a satisfying assignment in which $x_1=x_3$ etc.

The above result allows us to prove two fundamental theorems in the field of (two-party and multi-party) cryptographic protocols. These theorems yield automatic and efficient transformations that, given a protocol that is correct with respect to an extremely weak adversary, output a protocol correct in the most adversarial scenario. Thus, these theorems imply powerful methodologies for developing two-party and multi-party cryptographic protocols.

1. INTRODUCTION

A fundamental measure proposed by Goldwasser, Micali and Rackoff [GMR] is that of the amount of knowledge released during an interactive proof. Informally, an interactive proof is a two-party protocol through which one party (*the prover*) can convince his counterparts (*the verifier*) in the validity of some statement concerning a common input. (The prover should be able to do so if and only if the statement is indeed valid.) Loosely speaking, an interactive proof system is called zero-knowledge if whatever the verifier

Work done while the first author was in the Laboratory for Computer Science, MIT, partially supported by an IBM Postdoctoral Fellowship, and NSF Grant DCR-8509905. The second author was supported by NSF Grant DCR-8413577 and an IBM Faculty Development Award. Work done while the third author was in Mathematical Sciences Research Institute, Berkeley.

could generate in polynomial-time after interacting with the prover, he could also generate in polynomial-time when just told by a trusted oracle that the assertion is indeed valid. In other words, zero-knowledge proofs have the remarkable property of being both convincing and yielding nothing except that the assertion is indeed valid.

Despite their importance, very few examples of (non-trivial¹⁾) zero-knowledge proofs have been known until recently. Furthermore, all previously known proofs were of languages in $NP \cap Co-NP$ and heavily relied on special “symmetric” properties of “Number Theoretic” languages. The much more general potential offered by the notion of zero-knowledge proofs has remained immaterialized.

In this extended abstract, we first show how to construct zero-knowledge interactive proofs for every language in NP. This yields an extremely powerful cryptographic tool: the *ability to prove any NP statement in a zero-knowledge manner*. In particular, the generality of this tool allows an untrusted party to prove that he is behaving according to a predetermined protocol, without yielding any of his secrets. We example the power of this tool by three concrete applications. However, the general effect of this result is demonstrated in its generic application as part of a compiler which translates protocols operating in a weak adversary model to protocols which achieve the same goals in the most adversarial environment.

1.1 What is an interactive proof system

It is traditional to view NP as the class of languages whose elements posses short proofs of membership. A “proof that $x \in L$ ” is a witness w_x such that $P_L(x, w_x) = 1$, where P_L is a polynomial-time computable Boolean predicate associated to the language L such that $P_L(x, y) = 0$ for all y if x is not in L . The witness must have length polynomial in the length of the input x , but needs not be computable from x in polynomial time. A slightly different point of view is to consider NP as the class of languages L for which a powerful prover may prove membership in L to polynomial-time deterministic verifiers. The interaction between the prover and the verifier, in this case, is trivial: the prover sends a witness (“proof”) and the verifier computes for polynomial time to verify that it is indeed a proof.

This formalism was recently generalized by allowing more complex interaction between the prover and the verifier and by allowing the verifier to toss coins and to be convinced by overwhelming statistical evidence [GMR, B]. The motivation of Goldwasser, Micali and Rackoff for this generalization was to *consider the most general manner in which one party can prove theorems to another party, and to study the “amount of knowledge revealed in such interactions”* [GMR]. This generalization is crucial for establishing the non-triviality of the notion of zero-knowledge proofs (see Remarks 4 and 5).

1) All languages in BPP have trivial zero-knowledge proofs, in which the prover tells the verifier nothing; the verifier can test membership in BPP languages by himself.

An interactive proof system for a language L is a protocol (i.e. a pair of local programs) for two probabilistic interactive machines called the *prover* and the *verifier*. We denote these predetermined programs by P and V , respectively. Initially both machine have access to a common input tape. The two machines send messages to one another through two communication tapes. Each machine only sees its own tapes, the common input tape and the communication tapes. In particular, it follows that one machine cannot monitor the internal computation of the other machine nor read the other's coin tosses, current state, program etc. The verifier is bounded to a number of steps which is polynomial in the length of the common input, after which he stops either in an *accept* state or in a *reject* state. At this point we put no restrictions on the local computation conducted by the prover.

We require that, whenever the verifier is following his predetermined program V , the following two conditions hold:

- 1) *Completeness of the interactive proof system:* If the common input x is in L and the prover runs his predetermined program P , then the verifier accepts x with probability $\geq 1 - |x|^{-c}$, for every constant $c > 0$. In other words, the prover can convince the verifier that $x \in L$.
- 2) *Validity of the interactive proof system:* If the common input x is NOT in L , then for every program P' run by the prover the verifier rejects x with probability $\geq 1 - |x|^{-c}$, for every constant $c > 0$. In other words, the prover cannot fool the verifier.

Remark 1: Note that it does not suffice to require that the verifier cannot be fooled by the predetermined prover P (such a mild condition would have presupposed that the "prover" is trusted by the verifier). We require that no matter how the prover plays, he will fail to "prove" incorrect statements.

Remark 2: As is the case with NP, the conditions imposed on acceptance and rejection are not symmetric. Therefore the existence of an interactive proof for the language L does not imply its existence for the complement of L .

Remark 3: The above "definition" follows the one of Goldwasser, Micali and Rackoff [GMR]. A different definition due to Babai [B], restricts the verifier's actions to generating random strings, sending them to the prover, and evaluating a deterministic polynomial-time predicate at the end of the interaction. In other words, in Babai's framework the coin tosses are public, while in the more general definition of [GMR] the verifier may use a private coin (the output of which may not be revealed to the prover). Designing proof systems seems to be much simpler in the [GMR] model, but making statements about them seems easier if one restricts oneself to Babai's model. Surprisingly, the two models are equivalent as far as language recognition is concerned [GS].

Remark 4: The ability to toss coins is crucial to the non-triviality of the notion of an interactive proof system. Suppose that a language L has an interactive proof system in which the verifier does not toss coins. Then, without loss of generality, this proof system is a trivial one: The prover just guesses the legal conversation, sends it to the verifier which just verifies its validity in deterministic polynomial-time. (It follows that $L \in NP$.)

1.2 What is a zero-knowledge proof

Intuitively, a zero-knowledge proof is a proof which yields nothing but its validity. This means that for all practical purposes, “whatever” can be done after interacting with a zero-knowledge prover, can be done when just believing that the assertion he claims is indeed valid. (In “whatever” we mean not only the computation of functions but also the generation of probabilistic distributions.) Thus, zero-knowledge is a property of the predetermined prover: its robustness against attempts of an arbitrary (polynomial-time) verifier to extract knowledge from him via the interaction. This is captured by the formulation appearing in [GMR] and sketched below.

Denote by $V^*(x)$ the probability distribution generated by a machine V^* which interacts with (the prover) P on the common input $x \in L$. We say that the proof system is *zero-knowledge* if for all probabilistic polynomial-time machines V^* , there exists a probabilistic polynomial-time algorithm M_{V^*} that on input x can produce a probability distribution $M_{V^*}(x)$ that is polynomially-indistinguishable from the distribution $V^*(x)$. (For every algorithm A , let $p_A(x)$ denote the probability that A outputs 1 on input x and an element chosen according to the probability distribution $D(x)$. Similarly, $p_A'(x)$ is defined with respect to the probability distribution $D'(x)$. $D(\cdot)$ and $D'(\cdot)$ are *polynomially-indistinguishable* if for every probabilistic polynomial-time algorithm A , $|p_A(x) - p_A'(x)| \leq |x|^{-c}$, for every constant $c > 0$ and for all sufficiently long x . This notion originates from [GM] and in [Y].)

Remark 5: It is easy to see that if a language L has a zero-knowledge proof system in which only one message is sent, then $L \in BPP$. Thus, the non-triviality of the interaction is a necessary condition for the non-triviality of the notion of zero-knowledge.

1.3 Previous results concerning interactive proofs

In section 1.1, we implicitly discussed the classes of languages having k -move interactive proof systems (i.e. k message exchanges). Let $IP(k)$ denote the class of languages membership in which can be proved through a general interaction consisting of k messages, and let $RIP(k)$ denote languages proven through the restricted type interaction in which the verifier tosses “public coins”. Babai [B] showed that for every constant k , $RIP(k) = RIP(2) \subseteq NP^B$ for almost all oracles B . This means that his restricted hierarchy collapses. Goldwasser and Sipser [GS] showed that, surprisingly, for every k , $IP(k) \subseteq RIP(k+3)$. Both the above results say nothing about preservation of zero-knowledge by the transformations.

Several Number Theoretic languages, not known to be in BPP , have been previously shown to have zero-knowledge proof systems. The first language for which such a proof system has been demonstrated is Quadratic Non-Residuosity [GMR]. Other zero-knowledge proof systems were presented in [GMR], [GHY], and [G]. All these languages are known to lie in $NP \cap Co-NP$.

1.4 Our Results

In this extended abstract, we present only our results which are directly related to cryptography. For these results, we assume the existence of an *arbitrary* secure encryption function. We first show how to prove any NP statement in zero-knowledge. Next we use this ability to develop a methodology of cryptographic protocol design.

We have omitted from this extended abstract our results which are not related to cryptography. These result, which *do not rely on any assumptions*, consists of zero-knowledge interactive proof systems for Graph Isomorphism and Graph Non-Isomorphism. The mere existence of an interactive proof system for Graph Non-Isomorphism is interesting, since Graph Non-Isomorphism is not known to be in *NP*. For details see our paper [GMW].

1.5 Related Work

Using the intractability assumption of quadratic residuosity, Brassard and Crepeau have discovered independently (but subsequently) zero-knowledge proof systems to all languages in NP [BC1]. These proof systems heavily rely on *particular properties of quadratic residues* and do not seem to extend to arbitrary encryption functions. Recently, Brassard and Crepeau showed that *if factoring is intractable* then every NP language has a “zero-information” interactive proof system [BC2]. It should be stressed that the protocol they proposed constitutes an interactive proof provided that factoring is intractable. In other words, the validity of the interactive proofs depends on an intractability assumption; while in this paper and in [BC1] the proofs do not rely on such an assumption. On the positive side, the protocol presented in [BC2] is “zero-information” in the following strong sense: for every verifier program V^* there is an algorithm M_{V^*} , such that the probability distribution generated by M_{V^*} (on input $x \in L$) is *identical* to the probability distribution generated by V^* (when interacting with the prover on the input x).

Independently, Chaum [Cha] discovered a protocol which is very similar to the one in [BC2]. Chaum also proposed an interesting application of such “zero-information proofs”. His application is to a setting in which the verifier may have infinite computing power while the prover is restricted to polynomial-time computations (see also [CEGP]). In such a setting it makes no sense to have the prover demonstrate properties (as membership in a language) to the verifier. However, the prover may wish to demonstrate to the verifier that he “knows” something without revealing what he “knows”. More specifically, given a SAT formulae, the prover wishes to convince the verifier that he “knows” a satisfying assignment in a manner that would yield no information which of the satisfying assignments he knows. A definition of the notion of “a program knowing a satisfying assignment” can be derived from [GMR].

1.6 Organization of the Paper

In Section 2 we state our assumptions, and introduce some conventions. In Section 3 we show how to use any one-way permutation in order to construct a zero-knowledge

interactive proof for any language in NP. In Section 4 we example the cryptographic applications of the above result. In Section 5 we outline the fundamental theorems for two-party and multi-party cryptographic protocols.

2. Preliminaries

Throughout this paper we assume the existence of an *arbitrary* secure encryption schemes in the sense of Goldwasser and Micali [GM]. Such schemes exist if unapproximable predicates exist [GM]. The existence of unapproximable predicates has been shown by Yao to be a weaker assumption than the existence of one-way permutations [Y]. In particular, the infeasibility assumption of factoring (equivalently: the assumption that squaring modulo a composite integer is a one-way permutation) implies that the least significant bit of the modular square root is an unapproximable predicate [ACGS]. Note that the existence of one-way permutation is the basis for most of the works and results in "modern cryptography". See for example [DH, RSA, R1, GM, GM Riv, BM].

An encryption scheme secure as in [GM] is a probabilistic polynomial-time algorithm f that on input x and a random string r , outputs an encryption $f(x, r)$. Decryption is unique, that is $f(x, r) = f(y, s)$ implies $x = y$.

Notations: Let A be a set.

- 1) $Sym(A)$ denote the set of permutations over A .
- 2) When writing $a \in_R A$, we mean an element chosen at random with uniform probability distribution from the set A .

3. Zero-Knowledge Proofs for All Languages in NP

We begin by presenting a zero-knowledge interactive proof for graph 3-colourability. Using this interactive proof, we present zero-knowledge proofs for every language in NP.

3.1 A Zero-Knowledge Proof for Graph 3-Colourability

The common input to the following protocol is a graph $G(V, E)$. In the following protocol, the prover needs only to be a probabilistic polynomial-time machine which gets a proper 3-colouring of G as an auxiliary input. Let us denote this colouring by $\phi: V \rightarrow \{1, 2, 3\}$. Let $n = |V|$, $m = |E|$. For simplicity, let $V = \{1, 2, \dots, n\}$.

The following four steps are executed m^2 times, each time using independent coin tosses.

- 1) The prover chooses a random permutation of the 3-colouring, encrypts it, and sends it to the verifier. More specifically, the prover chooses a permutation $\pi \in_R Sym(\{1, 2, 3\})$, and random r_v 's, computes $R_v = f(\pi(\phi(v)), r_v)$ (for every $v \in V$), and sends the sequence R_1, R_2, \dots, R_n to the verifier.
- 2) The verifier chooses at random an edge $e \in_R E$ and sends it to the prover.
- 3) If $e = (u, v) \in E$ then the prover reveals the colouring of u and v and "proves" that they correspond to their encryptions. More specifically, the prover sends

$(\pi(\phi(u)), r_u)$ and $(\pi(\phi(v)), r_v)$ to the verifier. If $e \notin E$ then the prover stops.

- 4) (The verifier checks the “proof” provided in step (3).)

The verifier checks whether $R_u = f(\pi(\phi(u)), r_u)$, $R_v = f(\pi(\phi(v)), r_v)$, $\pi(\phi(u)) \neq \pi(\phi(v))$, and $\pi(\phi(u)), \pi(\phi(v)) \in \{1, 2, 3\}$. If either condition is violated the verifier rejects and stops. Otherwise the verifier continues to the next iteration.

If the verifier has completed all m^2 iterations then it accepts.

The reader can easily verify the following facts: When the graph is 3-colourable and both prover and verifier follow the protocol then the verifier always accepts. When the graph is not 3-colourable and the verifier follows the protocol then no matter how the prover plays, the verifier will reject with probability at least $(1-m^{-1})^{m^2} = \exp(-m)$. Thus, the above protocol constitutes an interactive proof system for 3-colourability.

Proposition: If $f(\cdot, \cdot)$ is a secure probabilistic encryption, then the above protocol constitutes a zero-knowledge interactive proof system for 3-colourability.

proof’s sketch: It is clear that the above prover conveys no knowledge to the SPECIFIED verifier. We need however to show that our prover conveys no knowledge to all possible verifiers, including cheating ones that deviate arbitrarily from the protocol.

Let V^* be an arbitrary fixed program of a probabilistic polynomial-time machine interacting with the prover P , specified by the protocol. We will present a probabilistic polynomial-time machine M_{V^*} that generates a probability distribution which is polynomially indistinguishable from the probability distribution induced on V^* ’s tapes during its interaction with the prover P . In fact it suffices to generate the distribution on the random tape and the communication tape of V^* .

Our demonstration of the existence of such M_{V^*} is constructive: given an interactive program V^* , we use it in order to construct the machine M_{V^*} . The way we use V^* in this construction does not correspond to the traditional notion of (a subroutine) reduction [K, C], but rather to a more general notion of reduction suggested in [AHU, pp. 373-374]. The machine M_{V^*} monitors the execution of V^* . In particular, M_{V^*} chooses the random tape of V^* , reads messages from V^* ’s communication tape, and writes messages to V^* ’s communication tape. Typically, M_{V^*} tries to guess which edge the machine V^* will ask to check. M_{V^*} encrypts an illegal colouring of G such that it can answer V^* in case it (M_{V^*}) is lucky. The cases in which M_{V^*} fails will be ignored: M_{V^*} will just rewind V^* to the last success, and try its luck again. It is crucial that from the point of view of V^* the case which leads to M_{V^*} success and the case which leads to M_{V^*} failure are polynomially indistinguishable.

The machine M_{V^*} monitoring V^* , starts by choosing a random tape r for V^* . M_{V^*} places r on its record tape and proceeds in m^2 rounds as follows.

- 1) M_{V^*} picks an edge $(u, v) \in_R E$ and a pair of integers $(a, b) \in_R \{(i, j) : 1 \leq i \neq j \leq 3\}$ at random. M_{V^*} chooses random r_i ’s and

computes $R_i = f(c_i, r_i)$, where $c_i = 0$ for $i \in V - \{u, v\}$, $c_u = a$ and $c_v = b$. M_{V^*} places the sequence of R_i 's on the communication tape of V^* .

- 2) M_{V^*} reads e from the communication tape of V^* . If $e \notin E$ (V^* cheats) then M_{V^*} appends the R_i 's and e to its record tape, outputs the record tape, and stops. If $e \neq (u, v)$ (unlucky for M_{V^*}) then M_{V^*} rewinds V^* to the configuration at the beginning of the current round, and repeats the current round with new random choices. If $e = (u, v)$ (lucky for M_{V^*}) then M_{V^*} proceeds as follows: First, it places (a, r_u) and (b, r_v) on the communication tape of V^* . Second, it appends the R_i 's, e , (a, r_u) and (b, r_v) to its record tape; and finally, it proceeds to the next round.

If all rounds are completed then M_{V^*} outputs its record and halts. A technical lemma (to be stated and proved in the final paper) guarantees that the three possible “answers” of the verifier (i.e. $e \notin E$, $e \in E - \{(u, v)\}$ and $e = (u, v)$) occur with essentially the same probability as in the interaction of V^* and the real prover. Thus, the probability that the simulation of a particular round requires more than $k \cdot m$ rewinds is smaller than 2^{-k} , and M_{V^*} terminates in polynomial time. The only difference between the probability distribution of the true interactions and the distribution generated by M_{V^*} is that the first contain probabilistic encryptions of colourings while the second contains probabilistic encryptions of mostly 0's. However, a second technical lemma (postponed to the final paper) asserts that this difference is indistinguishable in probabilistic polynomial-time.

Remark 6: The above protocol needs m^2 rounds. In the final version of our paper we will present two alternative ways of modifying the above protocol so to get a four-round zero-knowledge protocol for graph 3-colorability. In both modifications the idea is to have the verifier send the prover “encryptions” of all his questions (i.e. which edge he wants to check for each copy of the coloured graph) before the prover sends to the verifier the corresponding coloured graphs. By this, the verifier commits himself to a test before seeing the encrypted colourings (equivalently, the tests are only a function of the common input and the random coin tosses of the verifier). *How can the verifier encrypt his questions?* This is the point in which the two modifications differ.

- 1) Assuming the intractability of integer factorization, the verifier encrypts as follows. The prover first randomly chooses a Blum integer N (i.e. a composite integer which is the product of two large primes each congruent to 3 modulo 4). To encrypt the bit $\sigma \in \{0, 1\}$, the verifier randomly chooses a residue $r \in Z_N^*$ with Jacobi Symbol $(-1)^\sigma$, computes $s = r^2 \bmod N$, sends s to the prover and proves (in zero-knowledge) that he “knows” a square root of s (consult [FMRW, GMR]). Note that even with infinite computing power, the prover can not know σ . To reveal σ , the verifier presents r . Assuming the intractability of factoring, the verifier can not “change his mind” about σ .
- 2) A trivial solution follows by modifying the definition of an interactive proof such that the prover is also restricted to polynomial-time computation, and his

"computational advantage" over the verifier is merely in having an auxiliary input. Note that this is the natural cryptographic scenario.

Remark 7: The above protocol can be easily modified to yield a zero-information protocol that constitutes a proof system if factoring is intractable. The modification consists of having the prover encrypt the colouring by using a Blum integers selected by the verifier (analogue to (1) in Remark 6). More details will be given the final version of our paper.

3.2 Zero-Knowledge proofs for all Languages in NP

Incorporating the standard reductions into the protocol of Section 3.1, we get

Theorem 1: If $f(\cdot, \cdot)$ is a secure probabilistic encryption, then every NP language has a zero-knowledge interactive proof system.

Proof: For every language $L \in NP$ the protocol incorporates a fixed reduction of L to 3-colourability. Each party computes the 3-colourability instance from the common input, and then the prover proves to the verifier that this instance is 3-colourable (using the protocol of section 3.1). **QED**

Slightly less obvious is the proof of the following Theorem 2 that adapts Theorem 1 to the cryptographic scenario, in which all players are bounded to efficient computation.

Theorem 2: If $f(\cdot, \cdot)$ is a secure probabilistic encryption, every language in NP has a zero-knowledge interactive proof system whose prover is a probabilistic polynomial-time machine which gets a NP proof as an auxiliary input.

Proof: We would like the parties to proceed as in the proof of Theorem 1. The problem is whether the prover is powerful enough to execute his role in that protocol. Note that if the prover is given a colouring of the reduced 3-colourability instance then he can follow the instructions of the protocol in Section 3.1. However, the prover is only given a NP proof for the membership in an arbitrary language in NP . The difficulty is resolved by noticing that the standard reductions used in the protocol *efficiently* transform also the witnesses to the corresponding instances (see details below).

Most known Karp-reductions have the property that, given a NP-proof to the original instance, one can easily obtain a NP-proof for the reduced instance. Let L_1 be a language which is Karp-reducible to the language L_2 by the polynomial-time function t . Let $L \in NP$ and $x \in L$, then we denote by $w(x)$ a witness for x (i.e. $P_L(x, w(x))=1$, where P_L is the polynomial-time predicate associated to L). If there exist a polynomial-time computable function g such that for every instance $x_1 \in L_1$ we have $w(t(x_1)) = g(w(x_1))$ then we say that L_1 is *Levin-reducible* to L_2 . (This is "half the condition" in the definition of polynomial reducibility as appeared in Levin's paper

“Universal Search Problems” [L].) Thus, it suffices to verify that the generic reduction to SAT, and the “popular” reductions of SAT to 3SAT and of 3SAT to 3C, are all in fact Levin-reductions. **QED**

Remark 8: Theorem 1 can be generalized to show that not only NP is in zero-knowledge, but also “probabilistic NP” is. In other words, if $f(\cdot, \cdot)$ is a secure probabilistic encryption, then for every fixed k every language in $IP(k)$ has zero-knowledge proof systems. The same holds for Theorem 2 and all languages in $RIP(k)$ (note that Goldwasser and Sipser’s transformation of $IP(k)$ -protocols to $RIP(k)$ -protocols requires the prover to conduct approximate counting). For more details see [GMW].

4. Examples of Particular Applications of Theorem 2

Theorem 2 has a dramatic effect on the design of cryptographic protocols. Typically these protocols must cope with the problem of the parties convincing each other that they are sending messages which are computed according to the protocol. Such proofs should be carried out without yielding any secret knowledge. Since it is always possible to give NP proofs that the messages are computed properly, we can now give zero-knowledge proofs of this fact. Let us demonstrate this point, by using Theorem 2 to present simple solutions to three problems, which until recently were considered extremely difficult or even impossible. The more general implications of Theorem 2, are outlined in the preceding chapter.

A central notion in the field of cryptography is that of a secret. By a secret we mean a piece of data that once given can be recognized as the desired one. More formally, a *secret* s is *recognizable through* $g(s)$ if g is a one-way function. For example, the factorization (p, q) of a composite integer $N = p \cdot q$ is a secret recognizable through N . The digital signature $S_U(m)$ of user U to the message m is a secret recognizable through the message m and the public-key of U .

4.1 Oblivious Transfer of Arbitrary Secrets

The notion of Oblivious Transfer, suggested by Rabin [R2], has attracted a lot of attention within the study of cryptographic protocols. An *Oblivious Transfer* is a two-party protocol through which one party (unknowingly) transfers with probability $1/2$ a large amount of knowledge to his counterpart, and yields no knowledge otherwise [R2, EGL]. Initially, the *sender* (S) knows a secret s recognizable through $g(s)$, and the *receiver* (R) knows $g(s)$. If both parties follow the protocol then R gets s with probability $1/2$. If R follows the protocol then for S , the a-posteriori probability that R got s equals the a-priori probability. Rabin required that an attempt by S to reduce the probability that R receives s is detected [R2] with very high probability; while Even, Goldreich and Lempel only required that such an attempt is detected with probability $1/2$ [EGL].

In the following we will assume that factoring is hard. For the case that the secret is the factorization of a given integer, a protocol satisfying Rabin’s conditions was

presented by Fischer, Micali, Rackoff and Wittenberg [FMRW] (modifying [R2]). This protocol easily extends to arbitrary secrets, but in this case detection of “cheating” is only guaranteed with probability $1/2$. It has been conjectured that no protocol can meet Rabin’s condition (i.e. allow to always detect attempts to reduce the probability of a transfer) for arbitrary secrets [EGL].

Using Theorem 2, we show that the conjecture in [EGL] is false. The proposed protocol proceeds as follows: First, the sender encrypts the secret s using a randomly chosen composite N . Next, the sender provides the receiver with a zero-knowledge proof that the encrypted message is indeed the desired secret (note that this is a NP statement). Finally, the sender uses the [FMRW] Oblivious Transfer to send the factorization of N such that it is received with probability $1/2$.

Remark 9: Recently, we presented an Oblivious Transfer protocol based on the security of an arbitrary public-key encryption function.

4.2 Verifiable Secret Sharing

The notion of a verifiable secret sharing was presented by Chor, Goldwasser, Micali, and Awerbuch [CGMA], and constitutes a powerful tool for multi-party protocol design. A *verifiable secret sharing* is a $n+1$ -party protocol through which a *sender* (S) can distribute to the *receivers* (R_i ’s) pieces of a secret s recognizable through $g(s)$. The n pieces satisfying the following three conditions (with respect to $1 \leq l < u \leq n$):

- 1) It is infeasible to obtain any partial information about the secret from any l pieces;
- 2) Given any u messages the entire secret can be easily computed;
- 3) Given a piece it is easy to verify that it belongs to a set satisfying condition (2).

The notion of a verifiable secret sharing differs from Shamir’s secret sharing [Sha], in that the secret is recognizable and that the pieces should be verifiable as authentic (i.e. condition (3)).

We will consider solutions which are polynomial in n and in the security parameter. The first solution, presented in [CGMA], relies on RSA (resp. factoring) and works for $l = O(\log n)$ (resp. $l = O(\log \log n)$, see also [CGG]). Relying on the difficulty of testing quadratic residuosity this solution was improved, independently by [FM] and [AGY], to allow $l = \alpha n$ and $u = (1-\alpha)n$ for every fixed $\alpha < 1/2$. Recently, Feldman [F] presented a solution allowing $u = l + 1 \leq n$, assuming the intractability of the discrete logarithm function. Most of the above solutions are conceptually very complicated.

Combining Theorem 2 with Shamir’s scheme [Sha], we present a conceptually simple solution allowing $u = l + 1 \leq n$, assuming the existence of *arbitrary* one-way permutations. To share a secret $s \in Z_p$ recognizable through $g(s)$, the sender proceeds as follows: First, the sender chooses at random a l -degree polynomial over Z_p^* and evaluates it in n fixed points (these are the pieces in Shamir’s scheme). Next, the sender encrypts the i th piece using the Public-Key of the i th receiver, and sends all encrypted secrets to all receivers. Finally, the sender provides each receiver with a zero-knowledge proof that the encrypted messages correspond to the evaluation of a single polynomial over Z_p^* .

(note that this is a NP statement).

Recently, Benaloh has presented a much more efficient solution based on the intractability of quadratic residuosity [Bena].

4.3 Proving that a String is Pseudorandom

The notion of a pseudorandom bit generator, suggested by Blum and Micali [BM] and Yao [Y], is central to cryptography. A *pseudorandom bit generator* is an efficient deterministic program which stretches a randomly selected n -bit long seed into a longer bit sequence which is polynomially-indistinguishable from a random string [BM, Y]. A *pseudorandom function generator* is an efficient deterministic program that uses a random n -bit seed to construct an oracle which is polynomially-indistinguishable from a random oracle [GGM].

Using Theorem 2, a party which has selected the seed can present zero-knowledge proofs that the sequence/function he is producing/implementing is indeed pseudorandom.

5. Two Theorems for Cryptographic Protocols

In this section, we present an extremely powerful methodology for designing correct cryptographic protocols. The methodology consists of efficient “correctness and privacy preserving” transformations of protocols from a weak adversary model to the most adversarial model. These transformations are informally summarized as follows

Informal Theorem A: There exist an efficient compiler transforming a protocol P designed for $n=2t+1$ honest players, to a cryptographic protocol P' that achieves the same goals even if t of its n players are faulty. Faulty players are allowed to deviate from P' in an arbitrary but polynomial-time way.

In the formal statement of the corresponding Theorem, we avoid talking about “achieving goals”. The “goal of a protocol” is a semantic object that is not well understood. Instead, we make statements about well understood syntactic objects: the probability distribution on the tapes of interactive machines. In the final version of this paper we will define the notions of a “correctness preserving compiler” and a “privacy preserving compiler”. Both notions will be defined as relations between the probability distribution on the tapes of interactive machines during the execution of protocol P (in a weak adversarial environment) and the distribution on these tapes during the execution of P' (in a strong adversarial environment). Loosely speaking, “preserving correctness” means that whatever a party could compute after participating in the original protocol P , he could also compute when following the transformed protocol P' , properly. “Preserving privacy” means that whatever a set of dishonest players can compute after participating in P' , the corresponding players in P can compute when sharing their “knowledge” after participating in P . Similarly we formalize the following

Informal Theorem B: There exist an efficient compiler transforming a two-party protocol P that is correct in a fail-stop model, to a cryptographic two-party-protocol P' that achieves the same goals even if one of the players deviates from

P' in an arbitrary but polynomial-time way.

The proofs of the above Theorems make primary use of Theorem 2 to allow a machine to "prove" to other machines that a message it sent is computed according to the protocol. In addition, these proofs make innovative use of most of the cryptographic techniques developed in recent years. Essential ingredients in the proof of Theorem A are the notions of verifiable secret sharing and simultaneous broadcast proposed and first implemented by Chor, Goldwasser, Micali, and Awerbuch [CGMA]. An essential ingredient in the proof of Theorem B is Blum's "coin flipping into the well" [Blu].

Further Improvements

Theorem A constitutes a procedure for automatically constructing fault-tolerant protocols, the goal of which is to compute a predetermine function of the private inputs scattered among the players. This procedure takes as input a distributed specification of the function (i.e. a protocol for honest players), not the function itself. It is guaranteed that this procedure will output a fault-tolerant protocol for computing this very function (i.e. the "correctness" condition) and that the "privacy" present in the specification will be preserved. Thus, the degree of privacy offered by the output fault-tolerance protocol depends on the specification, and not on the function to be computed. Furthermore, for some functions f it seems to be difficult to write a distributed specification (protocol for honest players) which offers the maximum degree of privacy.

Recently (see forthcoming paper [GMW2]), we found a polynomial-time algorithm which on input a *Turing machine* specification of a n -ary function f , outputs a protocol for n honest players which offers maximum privacy. Namely, at the termination of the protocol, each subset of players can compute from their joint local history only whatever they could have computed from their corresponding local inputs and the value of the function. Thus, we achieve for any n -ary function what Benaloh [Bena] has achieved for the addition and multiplication functions.

Combined with the compiler of Theorem A, our algorithm constitutes an automatic generator of fault-tolerant protocols. This may be viewed as a completeness theorem for fault tolerant distributed computation.

ACKNOWLEDGEMENTS

It is our pleasure to thank Benny Chor and Shafi Goldwasser for many helpful discussions concerning this work.

REFERENCES

- [AHU] Aho, A.V., J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley Publ. Co., 1974.
- [ACGS] Alexi, W., B. Chor, O. Goldreich, and C.P. Schnorr, "RSA and Rabin Functions: Certain Parts Are As Hard As The Whole", to appear in *SIAM Jour. on Computing*. Extended Abstract in *Proc. 25th FOCS*, 1984.
- [AGY] Alon, N., Z. Galil, and M. Yung, "A Fully Polynomial Simultaneous Broadcast in the Presence of Faults", preprint, 1985.
- [B] Babai, L., "Trading Group Theory for Randomness", *Proc. 17th STOC*, 1985, pp. 421-429.
- [Bena] Benaloh, (Cohen) J.D., "Secret Sharing Homomorphisms: Keeping Shares of a Secret Secret", these proceedings.
- [Blu] Blum, M., "Coin Flipping by Phone", *IEEE Spring COMPCOM*, pp. 133-137, February 1982.
- [BM] Blum, M., and Micali, S., "How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits", *SIAM Jour. on Computing*, Vol. 13, 1984, pp. 850-864.
- [BC1] Brassard, G., and C. Crepeau, "Zero-Knowledge Simulation of Boolean Circuits", manuscript 1986.
- [BC2] Brassard, G., and C. Crepeau, "Non-Transitive Transfer of Confidence: A Perfect Zero-Knowledge Interactive Protocol for SAT and Beyond", manuscript, 1986.
- [BD] Broder, A.Z., and D. Dolev, "Flipping Coins in Many Pockets (Byzantine Agreement on Uniformly Random Values)", *Proc. 25th FOCS*, 1984, pp. 157-170.
- [Cha] Chaum, D., "Demonstrating that a Public Predicate can be Satisfied Without Revealing Any Information About How", these proceedings.
- [CEGP] Chaum, D., J.H. Evertse, J. van de Graaf, and R. Peralta, "Demonstrating Possession of a Discrete Logarithm without Revealing It", these proceedings.
- [CGG] Chor, B., O. Goldreich, and S. Goldwasser, "The Bit Security of Modular Squaring given Partial Factorization of the Modulos", *Proc. of Crypto85*, to appear (1986).
- [CGMA] Chor, B., S. Goldwasser, S. Micali, and B. Awerbuch, "Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults", *Proc. 26th FOCS*, 1985, pp. 383-395.
- [C] Cook, S.A., "The Complexity of Theorem Proving Procedures", *Proc. 3rd STOC*, pp. 151-158, 1971.
- [DH] Diffie, W., and M.E. Hellman, "New Directions in Cryptography", *IEEE Trans. on Inform. Theory*, Vol. IT-22, No. 6, November 1976, pp. 644-654.
- [EGL] Even, S., O. Goldreich, and A. Lempel, "A Randomized Protocol for Signing Contracts", *CACM*, Vol. 28, No. 6, 1985, pp. 637-647.
- [F] Feldman, P., "A Practical Scheme for Verifiable Secret Sharing", manuscript, 1986.
- [FM] Feldman, P., and S., Micali, in preparation, 1985.
- [FMRW] Fischer, M., S. Micali, C. Rackoff, and D.K. Wittenberg, "An Oblivious Transfer Protocol Equivalent to Factoring", in preparation. Preliminary versions were presented in *EuroCrypt84* (1984), and in the *NSF Workshop on Mathematical Theory of Security*, Endicott House (1985).
- [GHY] Galil, Z., S. Haber, and M. Yung, "A Private Interactive Test of a Boolean Predicate and Minimum-Knowledge Public-Key Cryptosystems", *Proc. 26th*

- FOCS*, 1985, pp. 360-371.
- [GJ] Garey, M.R., and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, New York, 1979.
- [G] Goldreich, O., "A Zero-Knowledge Proof that a Two-Prime Moduli Is Not a Blum Integer", unpublished manuscript, 1985.
- [GGM] Goldreich, O., S. Goldwasser, and S. Micali, "How to Construct Random Functions", *Proc. of 25th Symp. on Foundation of Computer Science*, 1984, pp. 464-479. To appear in *Jour. of ACM*.
- [GMW] Goldreich, O., S. Micali, and A. Wigderson, "Proofs that Yield Nothing But their Validity", in preparation. An extended abstract will appear in the proceedings of 27th *FOCS*, 1986.
- [GMW2] Goldreich, O., S. Micali, and A. Wigderson, "How to Automatically Generate Correct and Private Fault-Tolerant Protocols", in preparations.
- [GM] Goldwasser, S., and S. Micali, "Probabilistic Encryption", *JCSS*, Vol. 28, No. 2, 1984, pp. 270-299.
- [GMR] Goldwasser, S., S. Micali, and C. Rackoff, "Knowledge Complexity of Interactive Proofs", *Proc. 17th STOC*, 1985, pp. 291-304.
- [GMRiv] Goldwasser, S., S. Micali, and R.L. Rivest, "A Paradoxical Signature Scheme", *Proc. 25th FOCS*, 1984.
- [GS] Goldwasser, S., and M. Sipser, "Arthur Merlin Games versus Interactive Proof Systems", *Proc. 18th STOC*, pp. 59-68, 1986.
- [K] Karp, R.M., "Reducibility among Combinatorial Problems", *Complexity of Computer Computations*, R.E. Miller and J.W. Thatcher (eds.), Plenum Press, pp. 85-103, 1972.
- [L] Levin, L.A., "Universal Search Problems", *Problemy Peredaci Informacii* 9, pp. 115-116, 1973. Translated in *problems of Information Transmission* 9, pp. 265-266.
- [RSA] Rivest, R.L., Shamir, A., and Adleman, L., "A Method for Obtaining Digital Signatures and Public Key Cryptosystems", *Comm. of the ACM*, Vol. 21, February 1978, pp. 120-126.
- [R1] Rabin, M.O., "Digitalized Signatures as Intractable as Factorization", MIT/LCS/TR-212, 1979.
- [R2] Rabin, M.O., "How to Exchange Secrets by Oblivious Transfer", unpublished manuscript, 1981.
- [Sha] Shamir, A., "How to Share a Secret", *CACM*, Vol. 22, 1979, pp. 612-613.
- [Y] Yao, A.C., "Theory and Applications of Trapdoor Functions", *Proc. of the 23rd IEEE Symp. on Foundation of Computer Science*, 1982, pp. 80-91.

How To Prove Yourself: Practical Solutions to Identification and Signature Problems

Amos Fiat and Adi Shamir

Department of Applied Mathematics
The Weizmann Institute of Science
Rehovot 76100, Israel

Abstract.

In this paper we describe simple identification and signature schemes which enable any user to prove his identity and the authenticity of his messages to any other user without shared or public keys. The schemes are provably secure against any known or chosen message attack if factoring is difficult, and typical implementations require only 1% to 4% of the number of modular multiplications required by the RSA scheme. Due to their simplicity, security and speed, these schemes are ideally suited for microprocessor-based devices such as smart cards, personal computers, and remote control systems.

1. Introduction

Creating unforgeable ID cards based on the emerging technology of smart cards is an important problem with numerous commercial and military applications. The problem becomes particularly challenging when the two parties (the prover A and the verifier B) are adversaries, and we want to make it impossible for B to misrepresent himself as A even after he witnesses and verifies arbitrarily many proofs of identity generated by A . Typical applications include passports (which are often inspected and photocopied by hostile governments), credit cards (whose numbers can be copied to blank cards or used over the phone), computer passwords (which are vulnerable to hackers and wire tappers) and military command and control systems (whose terminals may fall into enemy hands). We distinguish between three levels of protection:

1) Authentication schemes: A can prove to B that he is A , but someone else cannot prove to B that he is A .

2) Identification schemes: A can prove to B that he is A , but B cannot prove to someone else that he is A .

3) Signature schemes: A can prove to B that he is A , but B cannot prove even to himself that he is A .

Authentication schemes are useful only against external threats when A and B cooperate. The distinction between identification and signature schemes is subtle, and manifests itself mainly when the proof is interactive and the verifier later wants to prove its existence to a judge: In identification schemes B can create a credible transcript of an imaginary communication by carefully choosing both the questions and the answers in the dialog, while in signature schemes only real communication with A could generate a credible transcript. However, in many commercial and military applications the main problem is to detect forgeries in real time and to deny the service,

access or response that the forger wants. In these cases the transcript and judge are irrelevant, and the two types of schemes can be used interchangeably.

2. Interactive Identification

2.1 Background

The new identification scheme is a combination of zero-knowledge interactive proofs (Goldwasser, Micali and Rackoff [1985]) and identity-based schemes (Shamir [1984]). It is based on the difficulty of extracting modular square roots when the factorization of n is unknown. A related protocol for proving the quadratic residuosity of numbers was presented by Fischer Micali and Rackoff at Eurocrypt 84 (it did not appear in the proceedings), but the new protocol is faster and requires less communication. The main contribution of this paper is to show the relevance of such protocols to practical identification and signature problems.

The scheme assumes the existence of a trusted center (a government, a credit card company, a computer center, a military headquarters, etc.) which issues the smart cards to users after properly checking their physical identity. No further interaction with the center is required either to generate or to verify proofs of identity. An unlimited number of users can join the system without degrading its performance, and it is not even necessary to keep a list of all the valid users. Interaction with the smart cards will not enable verifiers to reproduce them, and even complete knowledge of the secret contents of all the cards issued by the center will not enable adversaries to create new identities or to modify existing identities. Since no information whatsoever is leaked during the interaction, the cards can last a lifetime regardless of how often they are used.

2.2 The Scheme

Before the center starts issuing cards, it chooses and makes public a modulus n and a pseudo random function f which maps arbitrary strings to the range $[0, n)$. The modulus n is the product of two secret primes p and q , but unlike the RSA scheme, only the center knows the factorization of the modulus and thus everyone can use the same n . The function f should be indistinguishable from a truly random function by any polynomially bounded computation. Goldreich Goldwasser and Micali [1984] describe a particular family of functions which is provably strong in this sense, but we believe that in practice one can use simpler and faster functions (e.g., multiple DES) without endangering the security of the scheme.

When an eligible user applies for a smart card, the center prepares a string I which contains all the relevant information about the user (his name, address, ID number, physical description, security clearance etc.) and about the card (expiration date, limitations on validity, etc). Since this is the information verified by the scheme, it is important to make it detailed and to double check its correctness. The center then performs the following steps:

1. Compute the values $v_j = f(I, j)$ for small values of j .
2. Pick k distinct values of j for which v_j is a quadratic residue \pmod{n} and compute the smallest square root s_j of $v_j^{-1} \pmod{n}$.
3. Issue a smart card which contains I , the k s_j values, and their indices.

Remarks:

1. To simplify notation in the rest of this paper, we assume that the first k indices $j = 1, 2, \dots, k$ are used.

2. For non-perfect functions f , it may be advisable to randomize I by concatenating it to a long random string R which is chosen by the center, stored in the card, and revealed along with I .
3. In typical implementations, k is between 1 and 18, but larger values of k can further reduce the time and communication complexities of the scheme.
4. n should be at least 512 bits long. Factoring such moduli seems to be beyond reach with today's computers and algorithms, with adequate margins of safety against foreseeable developments.
5. The center can be eliminated if each user chooses his own n and publishes it in a public key directory. However, this RSA-like variant makes the schemes considerably less convenient.

The verification devices are identical standalone devices which contain a microprocessor, a small memory, and I/O interface. The only information stored in them are the universal modulus n and function f . When a smart card is inserted into a verifier, it proves that it knows s_1, \dots, s_k without giving away any information about their values. The proof is based on the following protocol:

1. A sends I to B .
2. B generates $v_j = f(I, j)$ for $j = 1, \dots, k$.

Repeat steps 3 to 6 for $i = 1, \dots, t$:

3. A picks a random $r_i \in [0, n)$ and sends $x_i = r_i^2 \pmod{n}$ to B .
4. B sends a random binary vector (e_{i1}, \dots, e_{ik}) to A .
5. A sends to B :

$$y_i = r_i \prod_{e_{ij}=1} s_j \pmod{n}.$$

6. B checks that

$$x_i = y_i^2 \prod_{e_{ij}=1} v_j \pmod{n}.$$

Remarks:

1. The verifier B accepts A 's proof of identity only if all the t checks are successful.
2. To decrease the number of communicated bits, A can hash x_i by sending B only the first 128 bits of $f(x_i)$ in step 3. B can check the correctness of this value in step 6 by applying f to the right hand side of the equation and comparing the first 128 bits of the results.
3. A can authenticate a particular message m (e.g., an instruction to a remote control system or a program sent to a remote computer) without having to extract new square roots by sending B the first 128 bits of $f(m, x_i)$ in step 3. If B knows m , he can easily check this value in step 6. A is fully protected against modifications and forgeries of his messages by the pseudo random nature of f , but this is not a real signature scheme: without participating in the interaction, a judge cannot later decide if a message is authentic.

2.3 Security

Lemma 1: If A and B follow the protocol, B always accepts the proof as valid.

Proof: By definition

$$y_i^2 \prod_{e_{ij}=1} v_j = r_i^2 \prod_{e_{ij}=1} (s_j^2 v_j) = r_i^2 = x_i \pmod{n}. \quad \square$$

Lemma 2: Assume that A does not know the s_j and cannot compute in polynomial time the square root of any product of the form $\prod_{j=1}^k v_j^{c_j} \pmod{n}$ ($c_j = -1, 0$ or $+1$, not all of them zero). If B follows the protocol (and A performs arbitrary polynomial time computations), B will accept the proof as valid with probability bounded by 2^{-kt} .

Proof (Sketch): A can cheat by guessing the correct e_{ij} vectors and sending

$$x_i = r_i^2 \prod_{e_{ij}=1} v_j \pmod{n} \quad \text{and} \quad y_i = r_i.$$

However, the probability of this event is only 2^{-k} per iteration and 2^{-kt} for the whole protocol. To increase this probability, A must choose the x_i values in such a way that for a non-negligible fraction of them he can compute the square roots y'_i and y''_i of

$$x_i / \prod_{e_{ij}=1} v_j \pmod{n}$$

for two vectors e'_{ij} and e''_{ij} . The ratio $y'_i/y''_i \pmod{n}$ is of the form $\prod_{j=1}^k s_j^{c_j} \pmod{n}$. This contradicts the assumption, since A himself can simulate B 's random questions and thus compute in expected polynomial time a value we assumed he cannot compute. \square

Lemma 3: For a fixed k and arbitrary t , this is a zero-knowledge proof.

Proof (Sketch): The intuitive (but non-rigorous) reason the proof reveals no information whatsoever about the s_j is that the x_i are random squares, and each y_i contains an independent random variable which masks the values of the s_j . All the messages sent from A to B are thus random numbers with uniform probability distributions, and cheating by B cannot change this fact.

To prove this claim formally, in the full paper we exhibit a probabilistic algorithm which simulates the communication between A and B without knowing the s_j with a probability distribution which is indistinguishable from the real distribution. The expected running time of this algorithm is $t \cdot 2^k$ times the sum of the expected running times of A and B . By assumption, this running time is polynomial. \square

Remarks:

1. Throughout this paper, 2^{kt} is assumed to be much smaller than the time required to factor the modulus n .
2. The quadratic residuosity protocol of Fischer Micali and Rackoff is a special case of this protocol with $k = 1$. The main practical advantage of the new protocol is that for the same security we can use only the square root of the number of iterations, which reduces the time and communication complexities of the protocol and its applications.
3. An adversary who records polynomially many proofs of identity cannot increase his chance of success: If he reuses a recorded x_i , he can playback the recorded answers only if the questions happen to be the same. Since A uses each x_i only once, the probability of

success is still 2^{-kt} .

4. In the parallel version of this protocol, A sends all the x_i , then B sends all the e_{ij} , and finally A sends all the y_i . This version is not zero-knowledge for technical reasons, but its security can be formally proven by the techniques developed in Section 3.

The 2^{-kt} probability of forgery is an absolute constant, and thus there is no need to pick large values of k and t as a safeguard against future technological developments. In most applications, a security level of 2^{-20} suffices to deter cheaters. No one will present a forged passport at an airport, give a forged driver's license to a policeman, use a forged ID badge to enter a restricted area, or use a forged credit card at a department store, if he knows that his probability of success is only one in a million. In all these applications, the forged ID card (rather than the transcript of the communication) can be presented to a judge as evidence in a trial. Even if the only penalty for a failed attempt is the confiscation of the card, and smart cards cost only \$1 to manufacture, each success will cost about one million dollars. For national security applications, we can change the security level to 2^{-30} . Even a patient adversary with an unlimited budget, who tries to misrepresent himself 1000 times each day, is expected to succeed only once every 3000 years.

2.4 Complexity

To attain a 2^{-20} level of security, it suffices to choose $k = 5$, $t = 4$ (for 2^{-30} , increase these values by 1). The average number of modular multiplications required to generate or verify a proof of identity in this case is $t(k + 2)/2 = 14$. The number of bytes exchanged by the parties during the proof is 323, and the secret s_j values can be stored in a 320 byte ROM. Even better performance can be obtained by increasing k to 18 (a 1152 byte ROM). If we use e_{ij} vectors with at most three 1's in them, we have a choice of 988 possible vectors in each iteration. With $t = 2$ iterations, the security level remains about one in a million, but the number of transmitted bytes drops to 165 and the average number of modular multiplications drops to 7.6 (which is two orders of magnitude faster than the 768 multiplications required by the RSA scheme). Note that the $2 \times 18 e_{ij}$ matrix is so sparse that B has to generate at most 6 out of the 18 v_j values to verify the proof.

The time, space, communication and security of the scheme can be traded off in many possible ways, and the optimal choices of k , t and the e_{ij} matrix depends on the relative costs of these resources. Further improvements in speed can be obtained by parallelizing the operations. Unlike the RSA scheme, the two parties can pipeline their operations (with A preparing x_{i+1} and y_{i+1} while B is still checking x_i and y_i), and use parallel multipliers to compute the product of v_j or s_j values in $\log k$ depth. Since the protocol uses only multiplication (and no gcd or division operations which are hard to parallelize), each iteration of the protocol is in NC, and thus the scheme is suitable for very high speed applications.

3. Signatures

3.1 The Scheme

B 's role in the interactive identification scheme is passive but crucial: The random e_{ij} matrix he sends contains no information but its unpredictability prevents cheating by A . To turn this identification scheme into a signature scheme, we replace B 's role by the function f and obtain the following protocol:

To sign a message m :

1. A picks random $r_1, \dots, r_t \in [0, n)$ and computes $x_i = r_i^2 \pmod{n}$.
2. A computes $f(m, x_1, \dots, x_t)$ and uses its first kt bits as e_{ij} values ($1 \leq i \leq t, 1 \leq j \leq k$).
3. A computes

$$y_i = r_i \prod_{\substack{e_{ij}=1 \\ j=1}} s_j \pmod{n} \quad \text{for } i = 1, \dots, t$$

and sends I, m , the e_{ij} matrix and all the y_i to B .

To verify A 's signature on m :

1. B computes $v_j = f(I, j)$ for $j = 1, \dots, k$.
2. B computes

$$z_i = y_i^2 \prod_{\substack{e_{ij}=1 \\ j=1}} v_j \pmod{n} \quad \text{for } i = 1, \dots, t.$$

3. B verifies that the first kt bits of $f(m, z_1, \dots, z_t)$ are e_{ij} .

3.2 Security

The formal proof of security in this extended abstract assumes that n is sufficiently large and that f is a truly random function. Consequently, there can be no generic attack which breaks the scheme for any n and f unless factoring is easy. Practical implementations which use particular moduli n_0 and pseudo-random functions f_0 may still be vulnerable to specialized attacks, but they merely show that n_0 is too small or that f_0 is demonstrably non-random. When n_0 is at least 512 bits long and f_0 is sufficiently strong (e.g., multiple DES with a fixed cleartext and variable key), such attacks are quite unlikely.

Lemma 4: If A and B follow their protocols, B always accepts the signature as valid.

Proof: By definition,

$$z_i = y_i^2 \prod_{\substack{e_{ij}=1 \\ j=1}} v_j = r_i^2 \prod_{\substack{e_{ij}=1 \\ j=1}} (s_j^2 v_j) = r_i^2 = x_i \pmod{n},$$

and thus $f(m, z_1, \dots, z_t) = f(m, x_1, \dots, x_t)$. \square

Lemma 5: A chooses a particular signature among all the possible signatures for the message m with uniform probability distribution.

Proof: Given a signature (e_{ij} matrix and y_i values), it is possible to recreate $r_1^2, \dots, r_k^2 \pmod{n}$ uniquely, and r_1, \dots, r_k in exactly 4^k ways. Since A chooses the r_i at random, the various signatures are chosen with equal probabilities. \square

Lemma 6: Let AL be any polynomial time probabilistic algorithm which accepts n, v_1, \dots, v_k and the signatures of arbitrary messages m_1, m_2, \dots of its choice, and produces a valid signature of another message m_0 of its choice. If the complexity of factoring and 2^{kt} grow non-polynomially with the size of n , AL cannot succeed with non-negligible probability for random functions f .

Proof (Sketch): By contradiction. Using a simple combinatorial argument, we can prove that a polynomial time variant AL' of AL can compute a square root of some product $\prod_{j=1}^k v_j^{e_j}$

$(\text{mod } n)$ ($c_j = -1, 0, \text{ or } +1$, not all of them zero) with a similar probability of success.

To turn AL' into a factoring algorithm for n , pick random s_1, \dots, s_k and define $v_j = s_j^2 \pmod{n}$. Execute AL' with n, v_1, \dots, v_k as input, and use the s_j to supply the signatures of m_1, m_2, \dots requested by AL' . The output of AL' is a square root Q of $\prod_{j=1}^k v_j^{c_j} \pmod{n}$, but another square root S ($= \prod_{j=1}^k s_j^{c_j} \pmod{n}$) is already known. By Lemma 5, AL' cannot find out which one of the four possible roots is S by analysing the given signatures of m_1, m_2, \dots Consequently, $\gcd(Q - S, n)$ is a proper factor of n with probability $1/2$. By repeating this procedure several times, we can make this probability arbitrarily close to 1. \square

It is easy to forge signatures for arbitrary messages m_0 in time T with probability $T \cdot 2^{-kt}$ by guessing the e_{ij} matrix T times. A refinement of Lemma 6 shows that when the complexity of factoring is considerably higher than 2^{kt} , this attack is essentially optimal:

Lemma 7: Let AL be any probabilistic algorithm of the type described in Lemma 6. If AL runs in time T and succeeds with probability $(1 + \epsilon)T2^{-kt}$ for random functions f , then n can be factored with non negligible probability in time $T^2 \cdot 2^{kt}$.

Proof: Will be given in the full paper. \square

Corollary 8: If k and t are chosen so that the ratio between the complexity of factoring and 2^{kt} grows non-polynomially with the size of n , then the $T2^{-kt}$ probability of forgery is tight for polynomial time attacks.

Discussion

The sequential version of the interactive identification scheme is zero-knowledge and thus B cannot deduce any information whatsoever about the s_j from his interaction with A . The parallel identification scheme and the signature scheme, on the other hand, cannot be proven zero-knowledge for very subtle technical reasons. In fact, strong signature schemes cannot be zero-knowledge by definition: If everyone can recognize valid signatures but no one can forge them, B cannot generate by himself A 's messages with the same probability distribution. However, corollary 8 shows that the information about the s_j 's that B gets from signatures generated by A is so implicit that it cannot be used to forge new signatures, and thus the signature scheme is provably secure (if factoring is difficult) even though it is not zero-knowledge.

3.3 Complexity

In the proposed signature scheme, an adversary knows in advance whether his signature will be accepted as valid, and thus by experimenting with 2^{kt} random r_i values, he is likely to find a signature he can send to B . Consequently, the product kt must be increased from 20 to at least 72 when we replace the identification scheme by a signature scheme.

A choice of $k = 9, t = 8$ attains the desired 2^{-72} security level. The private key can be stored in a 576 byte ROM, and each signature requires 521 bytes. The average number of modular multiplications for this choice is $t(k+2)/2 = 44$.

By doubling the key size to 1152 bytes ($k = 18$), we can reduce the size of each signature to 265 bytes ($t = 4$) without changing the 2^{-72} security level. By optimizing the order of the multiplications to compute the t subset products simultaneously, we can reduce their average number to 32. This is only 4% of the number of multiplications required in the RSA signature scheme. Other points along the tradeoff curve for the 2^{-72} security level are summarized in Table 1.

Table 1: Tradeoffs for k and t at the 2^{-72} Security Level

k	t	Secret Key Size (in bytes)	Signature Size (in bytes)	Average # Mult. (Standard)	Average # Mult. (Optimized)	Average # v_i 's B generates
1	72	64	4608 + 9	108	108	1
2	36	128	2304 + 9	72	64	2
3	24	192	1536 + 9	60	49	3
4	18	256	1152 + 9	54	46	4
6	12	384	768 + 9	48	41	6
8	9	512	576 + 9	45	45	8
9	8	576	512 + 9	44	44	9
12	6	768	384 + 9	42	35	12
18	4	1152	256 + 9	40	32	17
24	3	1536	192 + 9	39	28	21
36	2	2304	128 + 9	38	30	24
72	1	4608	64 + 9	37	37	36

4. Extensions

A unique feature of the new identification and signature schemes is that it is possible to change their level of security after the key has been chosen. Consider, for example, an access card with $k = 18$ s_j values: The fast screening procedure at the entrance to the building will be controlled with $t = 1$ (2^{-18} security level), access to the computer room will be controlled by $t = 2$ (2^{-36} security level), while any usage of the computer will leave signed audit trails with $t = 4$ (2^{-72} security level). The only dangerous case is the simultaneous usage of the same s_j values in a parallel identification scheme with a large t and in a signature scheme with a small t (an unlikely combination), which is susceptible to an active playback attack.

Since the verification devices store only small amounts of publicly available information, it is possible to standardize them: One device can store several values of n and f and thus check a variety of personal, financial and occupational ID cards provided by many independent organizations. This possibility is particularly important in department stores which have to recognize many types of credit cards or in check cashing situations which require three ID cards of many possible types.

The proposed schemes can be generalized in a variety of ways. For example, the square roots can be replaced by cubic or higher roots, the e_{ij} matrix can be made non-binary, and the usage of r_i and s_j values can be made more symmetric in the generation of each y_i value. A more radical generalization is suggested by Goldreich, Micali and Wigderson's recent discovery of zero knowledge proofs for NP problems: It is now possible to use any instance of any NP complete problem as the basis for identification and signature schemes. Shamir later improved the time and communication complexities of these proofs, but their practical significance is still unclear.

5. Acknowledgements

We would like to thank Mike Fischer, Oded Goldreich, Shafi Goldwasser, Silvio Micali, Charlie Rackoff, Claus Schnorr and Avi Wigderson for inspiring many of the ideas presented in this paper.

6. Bibliography

1. Fischer, Micali and Rackoff [1984]: *A Secure Protocol for the Oblivious Transfer*, presented at Eurocrypt, April 1984.
2. Goldreich, Goldwasser and Micali [1984]: *How to Construct Random Functions*, 25th Symposium on Foundations of Computer Science, October 1984.
3. Goldreich, Micali and Wigderson [1986]: *Proofs that Yield Nothing But the Validity of the Assertion and the Methodology of Cryptographic Protocol Design*, submitted to 27th Symposium on Foundations of Computer Science, November 1986.
4. Goldwasser, Micali and Rackoff [1985]: *The Knowledge Complexity of Interactive Proof Systems*, 17th ACM Symposium on Theory of Computation, May 1985.
5. Shamir [1984]: *Identity-Based Cryptosystems and Signature Schemes*, Proceedings of Crypto '84, Lecture Notes in Computer Science no. 196, Springer Verlag 1985.

Demonstrating that a Public Predicate can be Satisfied Without Revealing Any Information About How

David Chaum

Centre for Mathematics and Computer Science
Kruislaan 413 1098 SJ Amsterdam the Netherlands

*It's not unlike a technique of probabilistic mathematical proof
in which you allow a receiver to select one of two cases.*

—Norman Shapiro

[responding] *Yes, you're right....
but the residue of doubt is provably, negligibly small.*

—Michael Rabin 1977

Introduction

The problem solved here may be defined in the following way: Both parties y and z agree on a Boolean expression called a predicate; y claims to know a secret value satisfying the predicate; z wants very high certainty that y does have such a value; while y is willing to demonstrate possession of the secret satisfying value, y is unwilling to reveal the secret value to z . The solution requires z to assume that y cannot quickly solve certain problem instances provided by z . But y is sure not to reveal anything about the secret, even if z has unlimited computing power.

Relation to Other Work

The result presented is a dual of those by [Goldreich, et al 86] and [Brassard & Crepeau 86]: their model is an x with infinite computational ability and a z with limited ability; here z may have infinite computational ability and y has only limited ability. Besides being of theoretical interest for this reason, the approach presented here offers several advantages:

- The only possibility for cheating is to solve specific instances of the hard problem (factoring in the example construction) within the time allotted to compute legal responses.
- A variation is secure even if some known fraction of instances of the assumed hard problem

can be solved within the allotted time.

- If there are multiple solutions, no information about which one(s) the prover knows is released by the protocol, even to someone who actually has infinite computing power.
- The model is consistent with previous proposals of the author [Chaum 85b], where an individual may have to demonstrate something to an organization that has potentially unknown resources or abilities. In fact, the result is a special case of a protocol previously presented by the author [Chaum 85a], whose properties are described in [Chaum 85b page 1039]. But the underlying problem assumed hard in that work differs from those relied on here.
- Giving the verifier a chance to cheat of less than 2^s requires only an amount of computation linear in s and the number of gates needed to represent the predicate. For $s=100$ and say 200 digit composites, this requires for each gate only about as much computation as a single RSA decryption.
- The protocol is easily adapted to the dual model.

1. PROTOCOL

In overview, the protocol presented involves y making known to z transformed and encrypted copies of a truth table for each gate of a circuit representation of the predicate, after which z is allowed to “select one of two cases”. The basic idea of getting exponential security by one party first committing by revealing encrypted forms and then allowing the other party to choose between several cases, which is relied on here, was first proposed in the context of cryptographic protocols by Rabin in [77] (which is the subject of the discussion quoted at the beginning of this article).

1.1 Protocol Set-Up

Initially y and z agree on a predicate and its realization by a circuit comprising m gates g_1, \dots, g_m , defined by their respective truth tables T_1, \dots, T_m . The gates are interconnected by n wires w_1, \dots, w_n , with each column of every truth table corresponding to a wire. Thus the predicate may be thought of as a Boolean function on say r secret input bits involving m elementary Boolean operations each (except one) of whose output bits becomes an input for one or more other elementary operations without feedback. This means that the memoryless circuit has r input wires, each of which is an input to one or more gates (elementary operations defined by a corresponding truth table); $n - r - 1$ internal wires, each serving as the output of a single gate and “fanning-out” to serve as input to one or more other gates; and a single output wire of a single gate, which is the output of the whole circuit.

Consider a gate g_k with l inputs and an output defined by a truth table T_k (subsequently denoted without subscript) represented in matrix form as $T = (t_{i,j})$, with $i \in \{1, \dots, 2^l\}$ and $j \in W_k$,

where W_k is the set of wires corresponding to the inputs and outputs of gate g_k and (the cardinality) $\# W_k = l + 1$, which is the total number of inputs and outputs of gate g_k , and $W_k \subseteq \{w_1, \dots, w_n\}$. The entries of T are 0's and 1's, i.e. $t_{i,j} \in \{0,1\}$, in the usual way: the rows (apart from the last column) contain all defined input configurations, and the last entry in each row is the corresponding output.

It is sufficient to consider all the wires as having secret values, except the single output wire for the whole predicate. Since the value of this wire should be 1, the truth table of its gate is modified as follows: all rows with 0 in the output column are removed, and then the output column itself is removed.

First, y chooses an inversion I_j at random for each wire w_j , i.e. $I_j \in \{0,1\}$ for $j \in \{w_1, \dots, w_n\}$, where random choices (as used throughout) are uniform choices that are statistically independent of everything else.

Next, y successively transforms each T , first to a permuted form T' , second to an obscured form T'' , and third to an encrypted form E as follows: (a) Each T is transformed into a matrix $T' = (t'_{i,j})$, by a random row permutation. (b) Each T' is transformed into a table $T'' = (t''_{i,j})$ for which all entries in all columns corresponding to inverted wires are inverted: $t''_{j,i} = t'_{i,j} \oplus I_j$. (c) Each entry of the obscured form T'' is encrypted in a special way to yield $E = (e_{i,j})$: for each entry in T'' a random residue modulo N that is coprime with N , shown as $r_{i,j}$, is chosen with Jacobi symbol $(r_{i,j} / N)$ equal 1 when $t''_{i,j} = 1$ and equal -1 otherwise, and $e_{i,j} \equiv r_{i,j}^2 \pmod{N}$, where N is supplied to y by z .

Then y displays all the matrices E to z and allows z to choose between two cases:

- (1) Display by y of I_j and, for each gate, all the $r_{i,j}$'s used in forming the corresponding E 's. This allows z to recover every T'' from the Jacobi symbols of the $r_{i,j}$'s, to check that the entries of each E are the squares of the corresponding $r_{i,j}$, and to verify that each T'' satisfies $t''_{j,i} = t'_{i,j} \oplus I_j$, for some row permutation T' of T .
- (2) Display by y of one row of $r_{i,j}$'s for each E , which should correspond to the actual row of the truth table that is satisfied by the secret wire values. This allows z to check that the entries of a row of each E are the squares of the corresponding r 's, to recover the corresponding rows of the T'' 's from the Jacobi symbols of the $r_{i,j}$'s, and to verify that all entries $t''_{i,j}$ of the displayed rows with the same j are equal.

2. SECURITY

Theorem: No Shannon-information about the secret wire values is revealed by y following the protocol, assuming N has only two odd prime factors and they are each congruent to 3 modulo 4.

Proof: First note that no information in the Shannon sense is revealed before z chooses a case, since each quadratic residue displayed has exactly the same probability of corresponding to a 1 as to a 0, because it has exactly two distinct roots with each Jacobi symbol. The secret wire values

have no influence on what is revealed in case 1. In case 2, the indices of the displayed rows reveal nothing since the permutation of rows is chosen at random; a bit with index j in a revealed row corresponds with the j th wire, is equal to all other such bits with index j , and is just the exclusive-or of the secret wire value with I_j , which is just the encryption of the secret value under a true one-time pad. \square

Theorem: *The probability that y satisfies z 's verification cannot exceed $\frac{1}{2}$ when y is unable to learn secret wire values satisfying the circuit, assuming y cannot find two square roots of the same residue modulo N that have distinct Jacobi symbols.*

Proof: It is sufficient to show that if y can satisfy z in both cases, then y can learn wire values satisfying the circuit. All T'' are uniquely determined (from the assumption), are known to y , and contain only valid truth table rows when exclusive-ored with the corresponding bits of the I_j 's known to y , as a consequence of y being able to satisfy case 1. From case 2, y knows a way to choose one row from each table T'' such that each wire is assigned the same value in all the chosen rows. Thus, y can form the exclusive-or of the I_j 's known from case 1 with the rows known from case 2, which yields a valid row for each gate (from case 1) with an assignment of bits to wires that satisfies each such row (from case 2). \square

Lemma: *If the above protocol is successfully repeated s times, using moduli each of which can be factored in the allotted time with independent probability p , then the probability of one-half in the previous theorem may be replaced by $(\frac{1}{2} + p / 2)^s$.*

Proof: Follows immediately from elementary probability theory.

3. DISCUSSION

The protocol description used certain well known number theoretic functions (first introduced by Blum [82]) for clarity and concreteness, but the present results should not be interpreted as limited to these specific functions. A natural generalization is to any pair of so called “claw free” (as defined in [Goldwasser et al 85]) one-way bijections with the same image. Other choices of encryption functions switch the protocol to the dual model mentioned in the introduction: any suitable encryption of a single bit (or actually row of bits) with a unique inverse message could be used to encrypt a T'' to form an E .

In the protocol presented above, y must be convinced that N is a “Blum integer,” or better, that it is of the form used in [Goldwasser et al 85]. There are at least two ways to address such a requirement. One is just to complete the protocol and then let y reveal the factorization of N to convince z that no cheating has occurred. When such an after-the-fact check is not acceptable, and where the particular encryption functions used require some such checking based on trap-door information, z could use a protocol of the dual type to convince y that a predicate indicating suitability of the functions is satisfied.

Other claw free functions based on the discrete log problem do not require such checking [Damgård 86].

Acknowledgements

I am pleased to thank Oded Goldreich and Silvio Micali for their excitement about the difference between this work and their own and for encouraging me to publish it. Leoned Levin also expressed enthusiasm and reminded me about the claw free property. Additionally, thanks to Adi Shamir and Johan Hastad for listening to various versions of the protocol and inspiring its simplification; to Jeroen van de Graaf for several discussions; and to Jan-Hendrik Evertse for his comments.

References

- (1) Blum, M., "Coin flipping by telephone," Proceedings of IEEE Compcon, 1982, pp. 133-137.
- (2) Brassard, G. and Crepeau, C., "Zero-knowledge simulation of boolean circuits," preprint of extended abstract, April 1986.
- (3) Chaum, D., "Showing credentials without identification: signatures transferred between unconditionally unlinkable pseudonyms," Presented at Eurocrypt'85, Linz Austria, April 1985a.
- (4) Chaum, D., "Security without identification: transaction systems to make big brother obsolete," *Comm. ACM* 28, 10 (October 1985b), pp. 1030-1044.
- (5) Damgård, I., private communication 1986.
- (6) Goldreich, O., Micali, S., and Wigderson, A., "Proofs that yield nothing but the validity of the assertion and the methodology of cryptographic protocol design," preprint, April 1986.
- (7) Goldwasser, S., Micali, S. and Rivest, R.L., "A 'paradoxical' solution to the signature problem," FOCS 84.
- (8) Rabin, M.O., "Digitalized signatures," in *Foundations of Secure Computation*, Academic Press, NY, 1978.

DEMONSTRATING POSSESSION OF A DISCRETE LOGARITHM WITHOUT REVEALING IT

David Chaum

*Jan-Hendrik Evertse**

Jeroen van de Graaf

*René Peralta***

Centre for Mathematics and Computer Science
Kruislaan 413 1098 SJ Amsterdam The Netherlands

Abstract: Techniques are presented that allow *A* to convince *B* that she knows a solution to the Discrete Log Problem—i.e. that she knows an x such that $\alpha^x \equiv \beta \pmod{N}$ holds—withou revealing anything about x to *B*. Protocols are given both for N prime and for N composite. We prove these protocols secure under a formal model which is of interest in its own right. We also show how *A* can convince *B* that two elements α and β generate the same subgroup in Z_N^* , without revealing how to express either as a power of the other.

1. Introduction

Consider the following problem:

- Alice knows a solution to the Discrete Log Problem (i.e. for a particular α , β and N , she knows the exponent x such that $\alpha^x \equiv \beta \pmod{N}$ holds).
- Alice wants to convince Bob that she knows x .
- Alice is not willing to reveal any information (in the sense defined in the next section)

*Partially supported by the Netherlands Organisation for the Advancement of Pure Research (ZWO).

**Currently at Facultad de Matemáticas, Universidad Católica de Chile, Casilla 6177 Stgo., Santiago, Chile; partially supported by DIUC Grant #211/86.

about the value of x .

- Bob accepts an exponentially small chance that Alice is cheating, i.e. that she pretends to know an x but doesn't. More precisely, the chance that Alice succeeds in cheating without being detected by Bob, will be 2^{-T} , where T is proportional to the time and space required.

In this paper we present a number of protocols which solves this problem, both for the case N a prime, and for the case $N = P_1 P_2$, where P_1 and P_2 are prime and of roughly the same size. Notice that there is no probabilistic polynomial time algorithm known for finding x given α , β and N . But even when Alice is restricted to polynomial computational power (as we will assume), this protocol is of interest, since given α and N she can choose $x \in [1, N-1]$ with $\gcd(x, \phi(N)) = 1$ at random and then compute β simply by exponentiation.

In this paper we define the notion (*almost*) *no information* which is very similar to “zero knowledge”, introduced by Goldwasser, Micali and Rackoff [GMR85] (and which has nothing to do with Shannon-information). The difference is that in the GMR model the prover has unlimited computational power, whereas in our model her power is only polynomial with coin flipping. In section 7 we illustrate the need for such a model by giving an example in which both parties have a symmetrical position, and where it is reasonable to assume that neither has unlimited computational power.

As far as we know, no other protocol with the same functionality has been presented. Very recent results by Goldreich, Micali and Wigderson [GMW86], Brassard and Crepeau [BrCr86], and Chaum [Ch86], however, all imply the following: if Alice has a certificate (or witness) of a particular statement which can be verified in polynomial time, then there exists a polynomial time protocol in which she can convince Bob that she has a certificate, without releasing any knowledge (or information in [Ch86]) about the value of this certificate; consequently, there exists a polynomial time protocol for showing possession of the Discrete Log. Nevertheless, these protocols are not very practical. An important merit of the protocols presented here is their practical feasibility.

The structure of this paper is as follows: The next section describes the model and the notion of information under which we prove our protocols secure. In section 3 and 4 we present the protocols together with their proofs of security in the prime and composite cases, respectively. Section 5 is devoted to a specific variation which surprisingly turns out to be *insecure*. Section 6 gives a protocol to convince another party that two elements generate the same subgroup in Z_N^* . The paper ends with an example and two open problems.

2. The model.

In this paper we will use the model developed in [BKP85], but with some modifications. Below we briefly sketch this model using a modified notation. It should be pointed out that this sketch assumes familiarity with [BKP85] or [GMR85].

We think of a protocol as occurring between two Probabilistic Turing Machines (PTM's) A

and B which operate synchronously. Each PTM has, besides a computation tape and a random tape, a one-way infinite tape for incoming messages. We call this tape the “mailbox” of the machine. The PTM’s communicate by writing into each others mailbox. We call each of the machines in such a system a **CPTM**, for Communicating Probabilistic Turing Machine.

Now consider a **system** of two CPTM’s A and B . The system $[A ; B]$ halts whenever A or B halts; the system accepts only when B halts and accepts. Throughout this paper A will interactively demonstrate possession of a secret to B , so (using the terminology of [GMR85]) we call A the **prover** and B the **verifier**. In [BKP85] this secret is the factorisation of a large composite number. But in general, the solution to an instance of any problem assumed not solvable in random polynomial time may serve as a secret. We define I_A as the pair consisting of the problem instance *and* the secret; I_A is usually created by A , and is considered the input for the system $[A ; B]$. Given I_A , we define I_A^* as the single problem instance, thus *without* the secret; we assume that A sends I_A^* to B before the actual protocol starts. For example, in our Discrete Log protocols $I_A = (\alpha, \beta, N, x)$, and $I_A^* = (\alpha, \beta, N)$.

For simplicity, we explicitly force the time ordering in the messages by requiring that A and B alternately write one symbol in the other’s mailbox. If a party has nothing to communicate it writes the special null symbol, ∇ , not used for any other purpose in the communication. We also assume that both parties do not write superfluous null symbols, so the places where null symbols are written is a function of I_A^* . We define $\Omega := \bigcup_{n=0}^{\infty} \{0, 1, \nabla\}^n$, and the contents of a mailbox as an element of Ω .

The conversation between A and B , defined as the ordered pair containing their respective mailboxes, is considered as the output of the system $[A ; B]$. It depends only on the instance I_A and the bits on the random tapes of A and B . This conversation is denoted as $\text{conv}([A ; B](I_A))$. Then $\Pr(\text{conv}([A ; B](I_A)) = c)$ is defined as the probability that $c \in \Omega^2$ occurs as the conversation between A and B resulting from the initialising instance I_A , under the assumption that the bits on the random tapes of A and B are chosen independently and uniformly.

The following definitions (which are modifications of part (iii) of the definition of an A -simulator preceding theorem 1 in [BKP85]), will serve to make precise the kind of security achieved by our protocols. Informally speaking, they state that the prover A releases no information if there exists a probabilistic polynomial-time simulating machine which, when initialised with I_A^* and for all possible verifiers B' , produces simulated conversations between A and B that have (almost) the same probability distribution as the true conversations between A and B . This simulating machine, denoted S_{A^*} , is a PTM S that contains another machine A^* . This A^* is called as a subroutine and outputs a simulation of A ’s part of the conversation. Input for S_{A^*} is the problem instance without the secret, I_A^* ; the output is denoted as $\text{output}(S_{A^*}(I_A^*))$.

Definition 1 . The prover A releases **no information** if there exists a polynomial-time (simulating) machine S_{A^*} , such that for all CPTM’s B' , all initialising instances I_A , and all possible conversations $c \in \Omega^2$,

$$\Pr(\text{conv}([A; B'](I_A)) = c) = \Pr(\text{output}(S_{A^*}(I_A^*)) = c).$$

The next definition covers the case when the two probability distributions may differ slightly.

Definition 2. The prover A releases **almost no information** if there exist an ϵ , $0 \leq \epsilon < 1$, and a polynomial-time (simulating) machine S_{A^*} such that for all CPTM's B' , all initialising instances I_A of length l

$$\sum_{c \in \Omega^2} \left| \Pr(\text{conv}([A; B'](I_A)) = c) - \Pr(\text{output}(S_{A^*}(I_A^*)) = c) \right| \leq \epsilon^l.$$

For describing a cryptographic protocol in the model presented, we will use the same protocol notation throughout the paper. The meaning of this notation is straightforward; only the next few things might need explanation:

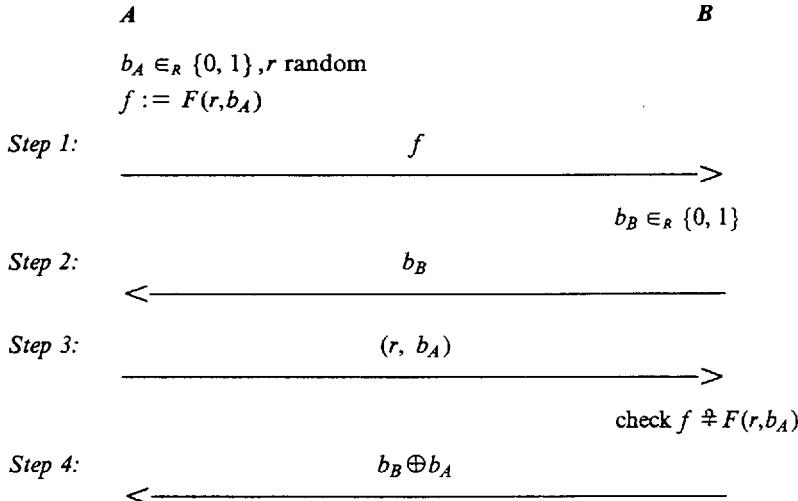
- Expressions shown on the left or right are known to that party only, and are secret from the other party.
- T is the **security parameter**, agreed upon before the protocol starts. Increasing T reduces A 's chance of successfully cheating exponentially, but increases the amount of communication and computation only linearly.
- $e \in_R S$ means that an element e is chosen at random from the set S , where all elements of S have an equal probability of being chosen, independent of all previous events.
- In some steps of the protocol a party checks if a particular equality holds; this is denoted as: check $a \neq b$. If the check fails, cheating is detected and the protocol halts.

The proofs of security for our protocols are considerably simplified by the fact that there is essentially no two-way communication. The nature of the protocols presented here is such that the bits that B reads from his random tape, can also be generated by a **mutually trusted random source**. The correctness of the protocols lies in the randomness of the bits generated, however, there is no reason for B to hide these bits. If a protocol has this property, we say it is **verifier-passive**.

Several **coin flipping** protocols are widely known which allow A and B to generate mutually trusted random bits, see e.g. [Bl82]. Below we briefly describe the general nature of these protocols. Let $b \in \{0,1\}$ be a bit, r be some random padding, and assume that A and B agree on a function F with the following two properties:

- 1) given the function F and the value $F(r, b)$, the bit b cannot be computed by B ;
- 2) given the function F and the value $F(r, b)$, a pair (r', b') for which $F(r, b) = F(r', b')$ and $b \neq b'$ cannot be found by A .

Then A and B can use the following protocol, called Γ , for generating mutually trusted random bits:

Protocol Γ : Coin flipping


Note that in this protocol the role of A and B can be interchanged, but this might depend on the model of computation.

For generating bit strings of length T , this protocol can be extended to a protocol Γ_T in two different ways: a sequential version, where Γ is repeated T times, and a parallel version, where both parties send message tuples of length T . We will use this coin-flipping as a sub-protocol.

Because of the time ordering in the conversation, the meaning of each cell in the mailboxes of A and B is completely determined by Π , the kind of protocol used, the security parameter T and the initialising instance I_A . So in the mailboxes we can distinguish between sequences of cells dedicated to the coin-flipping protocol Γ_T , and sequences of cells dedicated to the top-level protocol Π . More formally, if $\vec{b} = (b_1, \dots, b_T)$ are the bits generated through Γ_T , we define $\pi_A(I_A, \vec{b})$ as those cells written by A (in B 's mailbox) for protocol Π only, with null symbols at all other places; similarly, $\gamma_A(I_A, \vec{b})$ is the output of A with regard to Γ_T only, with null symbols at all other places. B 's part of the conversation is split similarly in π_B and γ_B . For the simulating machines A^* and S we define π_{A^*} , γ_{A^*} , π_S and γ_S on input I_A^* and \vec{b} in the corresponding way.

Theorem 1. Suppose that the protocol Π is verifier-passive, that a coin-flipping protocol Γ_T is used, and that a CPTM A^* exists such that

$$\forall I_A, d \in \Omega, \vec{b} \in \{0, 1\}^T: \Pr(\pi_A(I_A, \vec{b}) = d) = \Pr(\pi_{A^*}(I_A^*, \vec{b}) = d).$$

Then A releases no information through protocol Π .

Proof: We have to show that a polynomial-time simulating machine S_{A^*} can be constructed which simulates the conversation between A and B . This conversation, i.e. the contents of A 's

and B 's mailboxes, can be split up in π_A , π_B , γ_A and γ_B . By assumption, π_A can be simulated by π_{A^*} . And π_B consists of null symbols only, since Π is verifier-passive; therefore π_B is also simulatable. In general, it is easy for S_{A^*} to simulate Γ_T . However, while simulating Π , machine A^* has to guess in advance the bits \vec{b} resulting from Γ_T , otherwise the simulation fails. In the parallel version of Γ_T the probability of A^* guessing correctly all bits is only 2^{-T} . In the sequential version of Γ_T this probability is $\frac{1}{2}$ in each round. But as soon S_{A^*} realizes that the wrong coin is being simulated, the machine is reset to the state it had when that round was entered and tries that round again. Because of this fact, the error probability can be made arbitrary small. Though S_{A^*} 's expected running time is increased by a factor $2T$ when compared to A , the simulation still runs in probabilistic polynomial time. \square

Theorem 2 establishes the analogous result regarding protocols which transfer *almost* no information.

Theorem 2. Suppose the protocol Π is verifier-passive, that a coin-flipping protocol Γ_T is used, and that an ϵ , $0 \leq \epsilon < 1$, and a CPTM A^* exists such that for all CPTM's B' ,

$$\forall I_A, \vec{b} \in \{0, 1\}^T: \sum_{d \in \Omega} \left| \Pr(\pi_A(I_A, \vec{b}) = d) - \Pr(\pi_{A^*}(I_A^*, \vec{b}) = d) \right| \leq \epsilon,$$

where $I := |I_A|$. Then A releases almost no information through the protocol Π .

Proof: The proof is analogous to the proof of Theorem 1. \square

Machine A^* in the statements of Theorems 1 and 2 is called a **prover-simulator** machine or just an **A-simulator** machine.

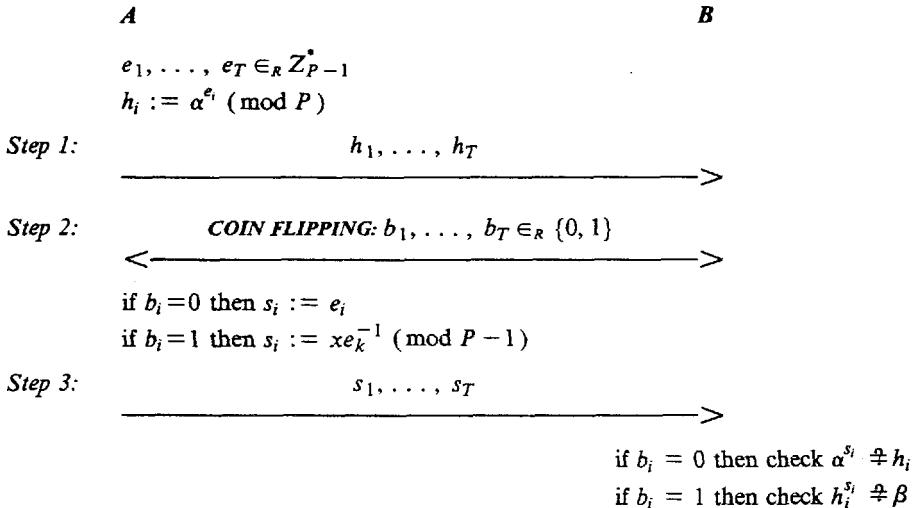
From now on we denote the bits produced by a coin flipping sub-protocol Γ by the word **COIN FLIPPING**, and a two-sided arrow. Furthermore, Z_N is the additive group $(\text{mod } N)$; and Z_N^* is the multiplicative group $(\text{mod } N)$.

3. Protocols for proving possession of the discrete logarithm modulo a prime number.

The problem is the following : A knows a solution to the equation $\alpha^x \equiv \beta \pmod{P}$, where we assume that x is randomly chosen from $[1, P - 1]$. P, α, β are public and B wants to be convinced that A knows x . A wants to convince B , but does not want to release any information about x .

We will give two protocols for this problem. Our first example is an easier protocol; however, it works only if α and β both generate the same sub-group in Z_P^* and A is willing to acknowledge this. If α and β do not generate the same sub-group, protocol 1 releases information about the index of $\langle \beta \rangle$ in $\langle \alpha \rangle$. An intuitive way to think about this protocol is to consider the expression $h_i := \alpha^{e_i}$ made public as lying somewhere between α and β ; upon getting the value of the bits, A shows either how to express h_i as a power of α , or how to express β as a power of h_i .

Protocol 1: $\alpha^x \equiv \beta \pmod{P}$; P, α, β public; x with $\gcd(x, P - 1) = 1$ a secret of A ; α, β generate the same sub-group.



Theorem 3 .

- (a) A can cheat in protocol 1 with probability at most 2^{-T} if she does not know x , and
- (b) there exists a polynomial-time prover simulator A^* .

Proof:

(a) *Correctness:* If A does not know x , then she is not able to compute both possible exponents to be released in step 3. Hence she will get caught with probability at least $\frac{1}{2}$ with each h_i . Thus A will get caught cheating with probability at least $1 - 2^{-T}$.

(b) *Security:* We exhibit a simulator A^* which, for random bits b_1, \dots, b_T , produces random h_1, \dots, h_T in Z_P^* , along with r_i such that $\alpha^{r_i} \equiv h_i$ if $b_i = 0$ and s_i such that $h_i^{s_i} \equiv \beta \pmod{P}$ if $b_i = 1$. We construct A^* as follows:

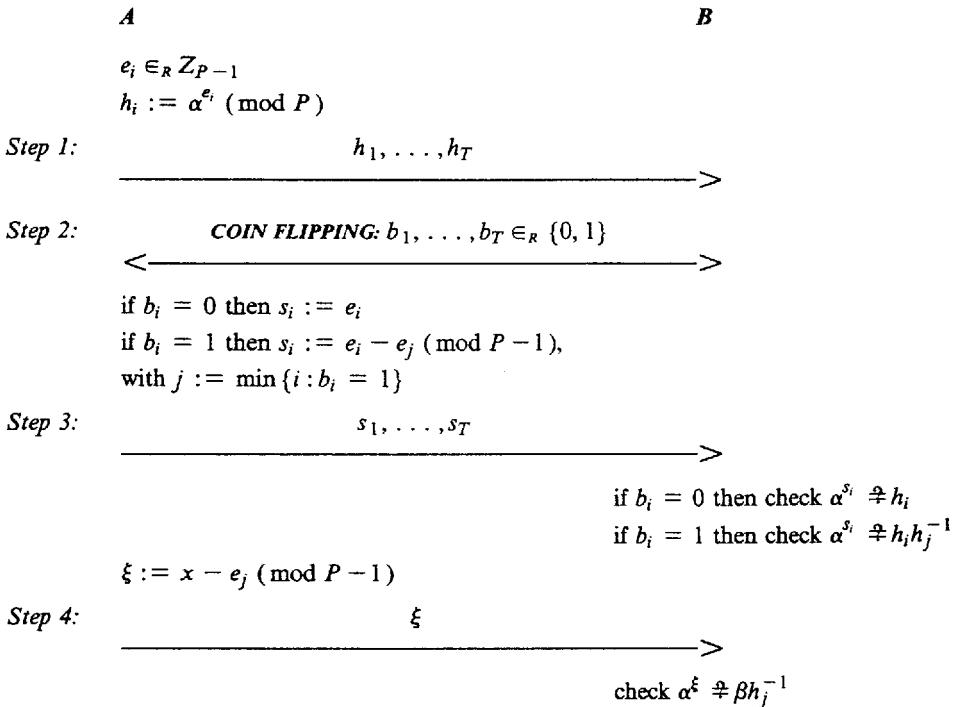
A -Simulator for protocol 1:

- 1: $b_1, \dots, b_T \in_R \{0, 1\}$
- 2: If $b_i = 0$ then $s_i \in_R Z_{P-1}^*$ and $h_i := \alpha^{s_i} \pmod{P}$.
- 3: If $b_i = 1$ then $s_i \in_R Z_{P-1}^*$ and $h_i := \beta^{s_i}$, where $s_i \equiv r_i^{-1} \pmod{P-1}$.
- 4: Output h_i, b_i and s_i for $i \in \{1, \dots, T\}$.

The reader can verify that the b_i 's, h_i 's, r_i 's and s_i 's produced by simulator A^* have the same joint probability distribution as the corresponding numbers produced by A in an execution of protocol 1. Note that the computations in step 3 can be done in polynomial time using Euclid's algorithm. By Theorem 1, protocol 1 reveals no information. \square

Protocol 1 has the advantage that it can be performed sequentially by using $T = 1$ and repeating step 1 to step 3 many times. Protocol 2 below can be proved to be secure only when it is performed in parallel; however, it can also be used if α and β do *not* generate the same group.

Protocol 2: $\alpha^x \equiv \beta \pmod{P}$; P, α, β public; x a secret of A .



Theorem 4 .

- (a) A can cheat in protocol 2 with probability at most 2^{-T} if she does not know x , and
- (b) there exists a polynomial-time prover simulator A^* .

Proof:

(a) *Correctness:* Suppose that A does not know x . Then she will get caught with probability at least $\frac{1}{2}$ for each h_i , for $i \neq j$. This is because A can never answer both possible cases to be sent in step 2. Now, independent of what j is, A 's chance of being caught with h_j is also at least $\frac{1}{2}$, because she cannot know e_j and pass the check after step 4. So the only way A can pass all the checks in step 3 and step 4 is by guessing correctly what the vector \vec{b} will be. This happens with probability $1 - 2^{-T}$.

(b) *Security:* Note that protocol 2 is verifier-passive. We exhibit a simulator machine A^* which produces messages and random bits which have the same joint probability distribution as messages from A and mutually trusted random bits in an execution of the protocol. By Theorem 1 it then follows that protocol 2 releases no information.

In the remainder of the proof we have $K := \{i : b_i = 0\}$, $L := \{i : b_i = 1\}$, $k \in K$ and $l \in L$. For

random bits b_1, \dots, b_T, A^* must produce random h_1, \dots, h_T in Z_P^* , along with e_k such that $\alpha^{e_k} \equiv h_k \pmod{P}$ for each $k \in K$. For each $l \in L$, simulator A^* must produce the difference $s_l \equiv e_l - e_j \pmod{P-1}$ satisfying $\alpha^{s_l} \equiv h_l h_j^{-1} \pmod{P}$, where $j = \min\{i : i \in L\}$. Finally, A^* must produce the difference $\xi \equiv x - e_j \pmod{P-1}$ satisfying $\alpha^\xi \equiv \beta h_j^{-1} \pmod{P}$. We construct machine A^* as follows:

A -Simulator for protocol 2:

- 1: $b_1, \dots, b_T \in_R \{0,1\}$. Let K and L be defined as before.
- 2: For $k \in K$ choose $s_k \in_R Z_{P-1}$, and let $h_k := \alpha^{s_k} \pmod{P}$.
- 3: Choose $\xi \in_R [1, P-1]$. For $l \in L$ choose $s_l \in_R [1, P-1]$.
For $l \in L - \{j\}$ let $h_l := \alpha^{s_l - \xi} \beta$.
- 4: Let $h_j := \alpha^{-\xi} \beta$.
- 5: Output h_i, b_i and s_i for $i = 1, \dots, T$, and ξ .

Observe for step 4 that $h_l \equiv \alpha^{s_l - \xi} \beta \equiv \alpha^{s_l - \xi + x} \equiv \alpha^{(e_l - e_j) - (x - e_j) + x} \equiv \alpha^{e_l}$ and for step 5 that $h_j \equiv \alpha^{-\xi} \beta \equiv \alpha^{-(x - e_j) + x} \equiv \alpha^{e_j}$. Now it follows immediately that the b_i , the e_i , the h_k , the s_l and the ξ produced by A^* have the same joint probability distribution as the ones produced by A in an execution of protocol 2. \square

The crucial difference between A and A^* is, that the simulator A^* does not know the actual values of the e_l (because it does not know x), but only their differences $\pmod{P-1}$. Since the protocol does not reveal the actual values of e_l and x , but only their difference with e_j taken $\pmod{P-1}$, the protocol is secure.

4. A protocol for proving possession of a discrete logarithm modulo a composite public key.

In this section we consider the analogues of protocols 1 and 2 modulo *composite* numbers, where we assume that the proving party A knows the factorization of N (henceforward called N_A).

So the problem is the following: A knows a solution to the equation $\alpha^x \equiv \beta \pmod{N_A}$, where N_A is a composite modulus whose factorisation is known to A only. Again α, β are public and B wants to be convinced that A knows x . And A wants to convince B , but does not want to release any information about either x or the factorization of N_A . Note that the operations on the numbers themselves are carried out modulo N_A , but on the exponents modulo $\phi(N_A)$.

First consider the analogue of protocol 1. As is easily verified, the protocol is feasible, but cannot be proven to be secure. The crucial point here is that when we look at the simulator for protocol 1, this simulator cannot execute step 3 since it does not know $\phi(N_A)$ (namely, this is essentially equivalent to knowing the factorisation of N_A). The simulator must generate pairs (h, s) for which $h^s \equiv \beta \pmod{N_A}$. But since we cannot prove that any simulator can do this in

polynomial time (in fact this does not seem very likely), this modification of protocol 1 cannot be proven secure.

As we will show now, protocol 2 can be used for composite numbers as well, be it at the cost of a very small probability of insecurity.

Protocol 3: $\alpha^x \equiv \beta \pmod{N_A}$; N_A, α, β are public; x and the factorization of N_A a secret of A .

This protocol is exactly the same as protocol 2 except that exponents are chosen modulo $\phi(N_A)$. The sums and differences of exponents are also revealed modulo $\phi(N_A)$.

Theorem 5. (a) A can cheat in protocol 3 with probability 2^{-T} if she does not know x , and (b) there exists a polynomial-time prover simulator A^* that produces a conversation with *almost* the same probability distribution.

Proof:

(a) *Correctness:* the same as for protocol 2.

(b) *Security:* The added complication is that A must not only know x , but, in order to perform the protocol, must also know the value of $\phi(N_A)$. Hence the possibility arises that A may release some information about the factorization of N_A . However, we can use the same simulator machine A^* as in the proof of security for protocol 2, except that A^* chooses exponents uniformly from the set $\{1, \dots, N_A\}$. We can do this since the construction for A^* involves no exponent arithmetic modulo $\phi(N_A)$. Thus the value of $\phi(N_A)$ is not used by A^* . The resulting probability distribution is not identical to the one generated by A , who chooses her exponents in $[1, \dots, \phi(N)]$. However, a straightforward computation shows that the difference of these distributions, as expressed by the sum of absolute differences in definition 2, is negligibly small. Use here that $\frac{N_A - \phi(N_A)}{N_A}$ is exponentially small in the size of N_A (assuming that N_A is the product of two prime numbers of nearly the same size). Thus protocol 3 releases *almost no information*. \square

5. An *insecure* protocol for proving possession of a discrete logarithm modulo a composite number when the factorization of the number is not known.

The problem is the following: A knows a solution to the equation $\alpha^x \equiv \beta \pmod{N}$, where N is a composite modulus. α, β are public and B wants to be convinced that A knows x . A wants to convince B , but does not want to release any information about x . Here A does *not* know the factorization of N .

Protocol 4: $\alpha^x \equiv \beta \pmod{N}$; N, α, β are public; x a secret of A ; A does not know a factor of N .

This protocol is the same as protocol 3 except that exponents are randomly chosen between 1 and N . Sums and differences of exponents are *not* reduced modulo $\phi(N)$ (since A does not know $\phi(N)$) and are instead released as integer sums.

Theorem 6. Protocol 4 releases information with regard to definition 2.

Proof: As before, $K := \{i : b_i = 0\}$, $L := \{i : b_i = 1\}$, $k \in K$ and $l \in L$. Define $s_{\max} := \max \{s_l : l \in L\}$, and s_{\min} similarly. Because we treat the s_l and the e_l as integers, there is no wrap around modulo $\phi(N)$. So $s_{\max} - s_{\min} = (s_{\max} + e_j) - (s_{\min} + e_j) = e_{\max} - e_{\min} =: \lambda$; note that λ is the length of the smallest interval containing all e_l , for $l \in L$. Because $e_{\min} \in [1, N]$ and $e_{\max} = e_{\min} + \lambda \in [1, N]$ we find that $e_{\min} \in [1, N - \lambda]$. This implies that $x = e_{\min} + (x - e_j) - (e_{\min} - e_j) = e_{\min} + \xi - s_{\min} \in [1 + \xi - s_{\min}, N - \lambda + \xi - s_{\min}]$. Now it is immediately clear that A^* , who does not know x , cannot produce a conversation with the same probability distribution as A . This proves that protocol 4 leaks information in the sense of definition 2. \square

We consider the *Shannon* information released by protocol 4. When λ is very close to N , then the number of possible values for x drops from N to $N - \lambda$. It is easy to see that when the number of equally likely possibilities reduces with a factor 2^{-m} , then m bits of Shannon information are revealed. This (or a similar computation using entropy) shows that the amount of information released by this protocol equals $\log_2(N/(N-\lambda)) = \log_2(1-\lambda/N)^{-1}$. Let Λ denote the stochastic variable for the length of the interval, and let $Pr(\Lambda=\lambda)$ be the probability that λ is the length of the interval. Then we define the average release of information as

$$\sum_{\lambda=1}^N Pr(\Lambda=\lambda) \log_2 \frac{1}{(1-\lambda/N)}. A straightforward computation of this sum shows that the average release of information for this protocol is approximately \log_2 |L| \leq \log_2 T, where |L| is the cardinality of L.$$

6. A protocol for proving that two elements generate the same group in Z_P^* or Z_N^* .

Let $\alpha \in Z_P^*$ or Z_N^* , and let $\langle \alpha \rangle$ denote the multiplicative subgroup generated by α . Protocol 1, 2 or 3 can all be used to show that $\langle \alpha \rangle = \langle \beta \rangle$ provided that A knows a relation between α and β . Note that proving that $\langle \alpha \rangle = \langle \beta \rangle$ is a problem not known to be solvable in polynomial time even modulo a prime number.

Protocol 5: $\langle \alpha \rangle = \langle \beta \rangle$ in Z_N^* ; α, β and N are public; x for which $\alpha^x \equiv \beta \pmod{N}$ is a secret of A .

Use protocol 1,2 or 3 in both directions: A shows to B that she knows how to express β as a power of α and how to express α as a power of β .

The correctness of this protocol is easy to understand: $\langle \alpha \rangle = \langle \beta \rangle$ if and only if there exists an x such that $\alpha^x \equiv \beta \pmod{N}$ and a y such that $\beta^y \equiv \alpha \pmod{N}$. A knows x and $\phi(N)$, thus A can compute $y \equiv x^{-1} \pmod{\phi(N)}$ in polynomial time. So A can perform protocol 1, 2 or 3 in both directions, thus showing knowledge of both x and y . The security of sequential use for these protocols lies in the kind of security proved. In the terminology of Berger, Kannan, Peralta [BKP85], the proofs of security of protocols 1-3 show that these protocols are **strongly secure** and

therefore can be executed one after the other without loss of security.

7. Applications and open problems.

We conclude with an application and two open problems:

1. Diffie-Hellman Key Exchange can be made more secure:

In a fair execution of the so-called Diffie-Hellman Key Exchange protocol [DiHe76] both parties choose an exponent x_A and x_B , they send $\beta_A := \alpha^{x_A} \pmod{P}$ and $\beta_B := \alpha^{x_B} \pmod{P}$ to each other, and they use $\beta_B^{x_A} = \beta_A^{x_B}$ as their secret communication key.

Now suppose B has two polynomial time algorithms F_1 and F_2 . On input α, P, β_A algorithm F_1 yields δ , which B uses as his β_B . Using δ^{x_A} as a key, A sends a message to B . This message, together with α, P, β_A , is fed to algorithm F_2 which has x_A as output.

As far as we know it has not been proven that such algorithms F_1, F_2 and such δ do not exist. This would be undesirable, because when B knows x_A he can pretend to be A to a third party. Using protocol 1 or 2 in this paper we can extend the Diffie-Hellman Key Exchange protocol by requiring both parties to show they know the discrete logarithm x . Then for B 's attack to work, he would have to know $\log_\alpha \delta$ besides δ . But as is easily verified, this implies that B has a polynomial algorithm for the Discrete Log Problem (he can himself simulate the message sent by A).

2. Can the same protocols be used with two generators?

Suppose that A wants to prove she knows x_1 and x_2 such that $\alpha_1^{x_1} \alpha_2^{x_2} \equiv \beta \pmod{N}$. Note that the status of this “Relaxed” Discrete Log Problem is not clear.

3. How hard is it to find any root of a given number β modulo a composite N of which the factorization is not known?

This is the problem mentioned at the end of section 3: can one find in polynomial time a pair (h, s) , $s > 1$, which is a solution for $h^s \equiv \beta \pmod{N}$, or is this problem reducible to a hard problem?

Acknowledgements

We are grateful to Oded Goldreich for his critical and helpful comments on an earlier version. Also we would like to thank Bert den Boer for his help and remarks.

References

- [BKP85] R. Berger, S.Kannan, and R. Peralta, “A Framework for the Study of Cryptographic Protocols,” *Proceedings of Crypto 85*, (1985).
- [Bl82] M. Blum, “Coin Flipping by Telephone,” *Proc. IEEE COMPON*, pp. 133-137 (1982).
- [BrCr86] G. Brassard, and C. Crépeau, “Zero-Knowledge Simulation of Boolean Circuits,” *Presented at Crypto 86*, (August 1986).
- [Ch86] D. Chaum, “Demonstrating that a Public Predicate can be Satisfied Without Revealing Any Information About How,” *Presented at Crypto 86*, (August 1986).
- [DiHe76] W. Diffie, and M. Hellman, “New Directions in Cryptography,” *IEEE Transactions on Information Theory* IT 22, pp. 644-654 (1976).
- [GMR85] S. Goldwasser, S.Micali, and C. Rackoff, “The Knowledge Complexity of Interactive Proof Systems,” *17th STOC* (1985).
- [GMW86] O. Goldreich, S. Micali, and A. Wigderson, “How to Prove all NP-statements in Zero-Knowledge, and a Methodology of Cryptographic Protocol Design,” *Presented at Crypto 86*, (August 1986).

Cryptographic Capsules: A Disjunctive Primitive for Interactive Protocols

Josh Cohen Benaloh*

1 Introduction

This paper describes a deceptively (almost embarrassingly) simple technique, that of *cryptographic capsules*, which allows Alice to convince Bob that either X or Y is true without giving Bob any information as to which is the case. Capsules are an instrumental part of the machinery used to compose ballots in the cryptographic election scheme of [CoFi85] (see also [Coh86], [Ben86], and [BeYu86]), but they have far broader applications. Use of capsules substantially simplifies the “zero-knowledge” interactive proof system for quadratic non-residuosity published in [GMR85]. Their use also provides a tremendous simplification of the “result-indistinguishable” interactive proof system published in [GHY85]. Capsules have been incorporated into the zero-knowledge protocol for interactively proving non-isomorphism of graphs described in [GMW86]. Finally, capsules are shown here to provide a mechanism more efficient than that of [GMW86] by which Alice can convince Bob (in a zero-knowledge fashion) of the validity of *any* NP predicate.

Despite their simplicity, it seems that the applications of capsules may go far beyond those mentioned here, and capsules have the potential to become a standard primitive construct for many kinds of interactive protocols.

2 Cryptographic Capsules

A *cryptographic capsule* (or simply *capsule*) is a randomly ordered collection of objects, each of which is of some specified form. The order of the elements of the capsule is randomly permuted to hide which element is of which type; or, alternately, some easily computable ordering function (such as \leq) can be applied to the capsule to obscure the original ordering.

*This work was supported in part by the National Security Agency under Grant MDA904-84-H-0004.

A simple example of a capsule is a pair of integers — one of which is even and one of which is odd, e.g. (4, 13). This capsule, however, is not very interesting because it is readily apparent *which* is the odd integer and which is the even integer.

A somewhat more useful capsule may be an (unordered) pair of integers $\{n_1, n_2\}$ with $n_1 = p_1 q_1$ where p_1 and q_1 are each primes congruent to 1 modulo 4 and $n_2 = p_2 q_2$ where p_2 and q_2 are each primes congruent to 3 modulo 4.

If we assume that distinguishing between these two cases is hard, then this suggests a simple method for flipping a coin over a telephone. Alice prepares such a pair and transmits it to Bob; Bob then selects one element from the pair and transmits his choice to Alice; finally, Alice reveals the factors of both n_1 and n_2 to Bob. We may say that the coin flip is heads if Bob chose the element with factors congruent to 1 modulo 4 and tails otherwise.

This is not an ideal example, since Alice could have simply transmitted a single integer of one of the two preceding classes and waited for Bob to guess which class it was from. The real power of capsules comes from the ability to prove interactively that a capsule is of the required form without the need to later reveal secret information about its contents.

3 Residue Classes and Capsules

Most of the interesting applications of cryptographic capsules so far explored involve their use with residue classes. The feature of residue classes which is important for this application is that two integers can be shown to be of the same residue class without giving any information about the actual residue classes to which the integers belong.

Formally, for any given integers n and y , y is said to be an r^{th} residue modulo n if and only if there exists some integer x such that $y \equiv x^r \pmod{n}$. The following lemma characterizes residue classes.

Lemma 1 *Let $\varphi(n)$ denote the Euler totient function, and choose n and r such that $r|\varphi(n)$ and $r^2 \nmid \varphi(n)$. If y is relatively prime to n and is not an r^{th} residue modulo n , then every w which is relatively prime to n is expressible as $w \equiv x^r y^i \pmod{n}$ for a unique integer i in the range $0 \leq i < r$.*

This i is the *residue class* of w with respect to n , y , and r .

An important (although slightly variant) special case occurs when $r = 2$, and n is the product of two distinct primes. We ignore the choice of y here and denote the set of quadratic residues by class 0 and the set of quadratic non-residues with Jacobi symbol 1 by class 1.

A property of residue classes is apparent from the definition.

Lemma 2 *If x_1 and x_2 are members of residue classes i_1 and i_2 , respectively, then the product x_1x_2 is a member of residue class $i_1 + i_2$.*

Note that for all integers i , residue classes i and $i+r$ are different denotations of the same class. The canonical denotation of a residue class I will be the unique class i with $0 \leq i < r$ such that $i \equiv I \pmod{r}$.

Finally, the following lemma shows how two integers can be shown to be of the same residue class.

Lemma 3 *Two integers x_1 and x_2 which are relatively prime to n are of the same residue class with respect to n , y , and r if and only if there exists some integer v such that $v^r \equiv x_1/x_2 \pmod{n}$.*

Thus, to prove that two integers are of the same residue class, it is necessary only to exhibit an r^{th} root of their quotient.

4 Some Applications

4.1 Elections

In the cryptographic election work of [CoFi85], each voter prepares, as a ballot, a capsule which consists of a random member of residue class 0 (denoting a **no** vote) and a random member of residue class 1 (denoting a **yes** vote). Later, each voter will designate one of the components of his or her capsule as the actual vote. The votes can then be multiplied together, and (by Lemma 2) the resulting product is a member of residue class t , where t is the total number of yes votes. A powerful agent (such as a government) which holds the factorization of the modulus n used can then prove to all participants that the computed product is of residue class t *without* giving any additional information about the residue classes of the factors, thus protecting the privacy of the individual votes.

Where do capsules come in? It is essential that the vote cast by each voter be a member of either class 0 or class 1. If a voter were, for example, able to cast a vote of class 1,000,000, then this one vote would increment the tally by 1,000,000. The voter, however, does not want to reveal to which of class 0 or class 1 his or her vote belongs.

To prove that a chosen capsule C is of the required form, a voter engages in an interactive proof (see [FMR84] and [GMR85]). Each voter prepares a set B of (say) 100 additional capsules — each one, as the original, consisting of a random member of residue class 0 and a random member of residue class 1. Random bits are then generated¹, and used to partition B into sets S and T . The capsules of

¹We assume here that some generally trusted source of randomness can be obtained, perhaps by XORing random bits generated by all (or some trusted subset) of the participants. In the other protocols described, the number of agents is small (usually two), and the challenging agent can generate its own random numbers.

set S are all “opened” to prove that they each consist of a proper **no** vote and a proper **yes** vote. (To open a capsule, a voter “opens” each component w of the capsule by revealing integers x and i , $i \in \{0, 1\}$, such that $w \equiv x^r y^i \pmod{n}$ — see Lemma 1.) Each capsule in T is shown to be “equivalent” to C by showing that it has one component of the same class as the first component of C and one component of the same class as the second component of C . (Recall that by Lemma 3, two integers can be shown to be of the same residue class by showing that there quotient q is an r^{th} residue, and this in turn can be shown by exhibiting an r^{th} root of q .)

Once this process has been completed, it is known that every capsule in S is of the required form (one integer of class 0 and one integer of class 1), and every capsule of T is of the same form as C . Thus, C is of the required form unless *every* capsule in T is improper. Since the partition of B into S and T was chosen randomly *after* the capsules of B were prepared, C could only be improper if the partition were somehow guessed in advance. But the probability of doing this successfully is only 1 in 2^{100} . Hence, there is extremely high confidence that C is a proper capsule, and the voter can then vote by selecting one of the components of C .

Formal proofs that this procedure does not yield any extraneous information are included in [CoFi85].

4.2 Quadratic Residuosity

The work on elections has been previously published (besides [CoFi85], see [Coh86], [Ben86], and [BeYu86] for some extensions), and the above sketch is included only to motivate the use of capsules. In section 4.2, we shall examine how the use of capsules can greatly simplify protocols which have been published in [GMR85] and [GHY85].

4.2.1 Zero-Knowledge Non-residuosity

In [GMR85], a protocol is given whereby Alice convinces Bob that a given y is *not* a quadratic residue modulo a given n . (It is presumed that Alice has the factorization of n and that Bob does not.) Alice convinces Bob that y is not a residue by demonstrating her ability to distinguish members of a set X of randomly chosen residues from members of a set Y consisting of elements formed by multiplying other randomly chosen residues by y . If y were a residue, then all of the elements of X and Y would be random residues (class 0), and Alice would have no hope of distinguishing between them with better than a 50% chance. If, however, y is not a residue, then the elements of Y would be random elements of class 1. With the factorization of n , Alice can distinguish between elements of class 0 and elements of class 1 flawlessly.

In order to avoid acting as a residuosity oracle for Bob, Alice wants to be certain that the numbers she distinguishes between are generated by the protocol, i.e. before telling Bob whether a w which he has produced is a residue or a non-residue, Alice wants to be certain that Bob already *knows* which is the case (under the assumption that y is not a residue). To accomplish this, the authors include a rather cumbersome protocol in which Bob prepares (say) 100 elements of both types, “opens” those designated by Alice, opens additional elements to balance the types remaining, and applies one of four functions to w and each remaining element v according to the classes of w and v .

The simple process of grouping the elements into capsules eliminates the need for the balancing and the four separate functions (as well as the accompanying analyses). The process is essentially the same as a one voter election (Bob is the voter and Alice is the government).

Bob sends Alice a w generated either as a residue or as a product of a residue and y . Bob then prepares and sends to Alice (say) 100 capsules, each of which consists of a randomly chosen residue and the product of y and another random residue. Alice then randomly decides for each capsule whether or not it is to be opened. Those capsules designated by Alice are opened by Bob proving that they are of the stated form. From each remaining capsule, Bob chooses one element, which shall be denoted by x , and shows that x is of the same class as w by revealing a root of the quotient x/w — this demonstrates that if Bob can determine the class of x , he can also determine the class of w since they are the same by Lemma 3. As before, unless Bob already has sufficient information to determine the class of w without Alice’s help, Bob has only 1 chance in 2^{100} of successfully answering Alice’s challenges.

4.2.2 Result-indistinguishable Residuosity

[GHY85] generalizes the result of [GMR85] in such a way that an observer, Carol, watching the protocol between Alice and Bob gains no information from the protocol as to whether Alice convinced Bob that a given z was or was not a quadratic residue.

The key addition to the protocol of [GMR85] is the inclusion of a third set of possibilities. Instead of choosing w from among just two sets X and Y , Bob may select from an additional set Z . Members of X are randomly generated residues (class 0); members of Y are randomly generated non-residues (class 1) — these can be produced by multiplying random residues by a known non-residue y ; finally, members of Z are generated by multiplying random residues by z (all elements of Z are of the same class as z).

To prove to Alice that she is not providing Bob with too much information, Bob must send Alice the (scrambled) members of 4 sets (essentially of the form of X , Y , Z , and \bar{Z} — a complementary set to Z needed to maintain symmetry).

The remainder of the protocol is similar to [GMR85], except that the unrevealed portions of four sets instead of just two have to be simultaneously balanced (necessitating an even more arduous analysis), and a four by three table of functions is needed corresponding to which set w is a member of and which class each unopened element is a member of.

By using three-component capsules, the protocol of [GHY85] can be simplified tremendously. Bob simply prepares a master capsule C , consisting of one member of each of X , Y , and Z , and (say) 100 additional scratch capsules of the same form. Alice designates some subset of the scratch capsules, and Bob opens these. Bob then shows that each remaining scratch capsule is equivalent to C by matching components and showing that their quotients are residues. Alice (now convinced that C was generated as required) tells Bob which capsule component is of a class *different* from the other two — thus transmitting to Bob the class of z .

The chance of Alice being fooled into revealing excessive information to Bob is only 1 in 2^{100} . The chance of Alice fooling Bob in one iteration of this protocol is $1/2$, so by iterating the process, Bob can obtain extremely (exponentially) high confidence that he has not been misled. Finally, it is not hard to show that Carol receives absolutely no information from watching this protocol that she could not have obtained on her own.

The necessary proofs of both [GMR85] and [GHY85] remain unchanged except for some straightforward simplifications and the removal of some analyses which are no longer necessary when the revised protocols are used.

4.3 Graph Non-isomorphism

One example in which capsules are useful *without* the aid of residue classes is seen in a protocol for graph non-isomorphism described in [GMW86]. Their original protocol closely followed the non-residuosity protocol of [GMR85]. Here, a prover designates a graph H given by the verifier as either a permutation of graph G_1 or of graph G_2 only after being convinced that the prover already holds such a permutation. Their protocol now incorporates capsules in a manner similar to that described in Section 4.2.1 (residue classes are replaced by the equivalence classes induced by graph isomorphism, and class equivalence is demonstrated by exhibiting permutations). With this modification, their protocol and its analysis have been simplified.

5 Boolean Circuit Satisfiability

Very recently (also in [GMW86]), Goldreich, Micali, and Wigderson gave a simple and elegant zero-knowledge interactive protocol to prove for any k that a graph is k -colorable *without* revealing any information about a specific coloring (note that it is assumed that the prover possesses a k -coloring of the graph). Because

k -colorability is \mathcal{NP} -complete, this means that *any* positive instance of a problem in \mathcal{NP} for which a prover holds a certificate (e.g. a satisfying assignment for a Boolean formula) can be reduced to graph colorability and shown in a zero-knowledge fashion to be a positive instance. The only assumption made is the existence of a probabilistic cryptosystem which is implied by the existence of a one-way permutation ([GoMi84],[Yao82]).

In this section, we shall examine an alternate approach which gives the same result by a very different method. The method uses capsules to give a zero-knowledge protocol to interactively prove that a given Boolean formula (or arbitrary Boolean circuit with in-degree 2) has a satisfying assignment. Brassard and Crepeau in [BrCr86] independently of both this work and [GMW86] have achieved the same result, and a similar result is given in [Cha86].

The major advantage of this method over the original is efficiency. When a Boolean formula or circuit is reduced to a colorability graph, the number of vertices and edges in the resulting graph is linear in the size of the Boolean formula. Each stage of the interactive proof protocol of Goldreich, Micali, and Wigderson, however, requires a new encryption of the entire graph; and for any fixed confidence level desired, their protocol requires a number of stages which is linear in the number of edges in the graph. Thus, the number of probabilistic encryptions required by this protocol grows quadratically with the size of the graph (or circuit). Because of the local nature of the method presented below, re-encryption is not necessary, and the number of probabilistic encryptions required grows only linearly with the size of the circuit (or graph).

The major disadvantage of this method compared to the original method is that the new procedure requires a (seemingly) stronger cryptographic assumption. Although both methods require a probabilistic encryption function — the best known of which is based on residue classes ([GoMi84]), the method given here requires a probabilistic encryption function for which two encrypted values can be proven (in a zero-knowledge manner) to be encryptions of the same value. Although this property is easily achieved by the residue class based probabilistic encryption (Lemma 3), it is not at all obvious that every probabilistic encryption function has this property. However, by observing that the problem of inverting a probabilistic encryption function is itself in \mathcal{NP} , the original Goldreich, Micali, and Wigderson result can be applied to show that the cryptographic assumption required here is, in fact, no stronger than the assumption of the existence of an arbitrary probabilistic cryptosystem.

5.1 The Satisfiability Scheme

The basic idea of the scheme is again deceptively simple. If Alice wants to prove to Bob that a given formula is satisfiable (and Alice has a satisfying assignment), Alice begins by choosing an n which is the product of two large primes and providing

Bob with n and a y (with Jacobi symbol 1) which is not a quadratic residue modulo n . This is merely the establishment of a probabilistic encryption function. Alice can convince Bob that y is a non-residue by engaging in the non-residuosity protocol of section 4.2.1 or by choosing n of a special form so that (for instance) $y = -1$ is a non-residue.

Alice then draws a circuit to compute the Boolean function (in the obvious way), selects a satisfying assignment and sends Bob an encryption of this assignment (for each variable, Alice sends Bob a residue if that variable is False/0/Off and a non-residue if that variable is True/1/On). Alice then encrypts the output of each gate of the circuit in the same manner and sends Bob these encrypted values as well.

For each gate in the circuit, Alice then interactively proves to Bob that the gate computes the required function. The computation of an AND gate will be shown here, and other Boolean functions will become apparent.

To prove that a given gate computes an AND on its inputs, a full truth table for AND is used. There are, of course, four possibilities: either both inputs and the output are 0; the first input is 0, the second is 1, and the output is 0; the first input is 1, the second is 0, and the output is 0; or both inputs and the output are 1. A four-component capsule can now be prepared such that each of the four components of the capsule is itself an *ordered* triple. To compute AND, the four (unordered) components of the capsule consist of (ordered) triples whose elements are members of residue classes $(0, 0, 0)$, $(0, 1, 0)$, $(1, 0, 0)$, and $(1, 1, 1)$. Once a capsule C is interactively proven to be of this form, Alice selects the component which corresponds to the actual input and output values of the gate and proves that they match by releasing a square root of each quotient.

To prove that a capsule C is of the above form, Alice prepares many (say 100) capsules of this form and Bob selects an arbitrary subset to be opened. Alice then proves that each unopened capsule matches C by matching corresponding components and releasing square roots of the quotients of all three elements of each triple to show that they do, in fact, match.

Finally, Alice interactively proves that the output of the circuit is 1 by proving that this value is a non-residue as in section 4.2.1.²

Remark Some gates may be computed without the need for an interactive proof. For example, an encrypted value may be complemented simply by multiplying it by y , and the XOR of two or more encrypted values is represented by their product (Lemma 2).

A mechanism which could obviate the need for any interactive proofs to verify gate validity is highly desirable. An encryption homomorphism which allows the direct computation of AND or OR together with NOT would of course suffice, and this would allow satisfiability to be proven with a single interactive proof of

²Chaum points out in his work that with a slight modification of this protocol, the need for this final interactive proof can be eliminated.

the value of the output. However, no such probabilistic encryption has yet been found.

6 Conclusions

The method of cryptographic capsules, especially (but not exclusively) when combined with residue classes, seems to be a powerful tool with many applications. This simple tool makes possible several protocols which would be impractical or completely impossible without them. In addition, several previously published protocols can be significantly simplified by the use of capsules.

It is believed that capsules may have many applications which go well beyond those described here, and they may become a standard tool in the design of interactive protocols.

Acknowledgements

The author would like to express many thanks to Oded Goldreich, Shafi Goldwasser, Neil Immerman, Jerry Leichter, Ruben Michel, and David Wittenberg for their help in developing this work and to Mike Fischer who, in addition to giving much guidance and many helpful criticisms, originally suggested the use of the term "capsules".

References

- [Ben86] **Benaloh, J.** "Secret Sharing Homomorphisms: Keeping Shares of a Secret Secret." *Crypto '86*, Santa Barbara, CA (Aug. 1986).
- [BeYu86] **Benaloh, J.** and **Yung, M.** "Distributing the Power of a Government to Enhance the Privacy of Voters." *Proc. 5th ACM Symp. on Principles of Distributed Computing*, Calgary, AB (Aug. 1986), 52–62.
- [BrCr86] **Brassard, G.** and **Crepeau, C.** "Zero-Knowledge Simulation of Boolean Circuits." *Crypto '86*, Santa Barbara, CA (Aug. 1986).
- [Cha86] **Chaum, D.** "Demonstrating that a Public Predicate can be Satisfied Without Revealing Any Information About How." *Crypto '86*, Santa Barbara, CA (Aug. 1986).
- [CoFi85] **Cohen, J.** and **Fischer, M.** "A Robust and Verifiable Cryptographically Secure Election Scheme." *Proc. 26th IEEE Symp. on Foundations of Computer Science*, Portland, OR (Oct. 1985), 372–382.

- [Coh86] **Cohen, J.** “Improving Privacy in Cryptographic Elections.” *TR-454, Yale University, Departement of Computer Science*, New Haven, CT (Feb. 1986).
- [FMR84] **Fischer, M., Micali, S., and Rackoff, C.** “A Secure Protocol for the Oblivious Transfer.” Presented at *Eurocrypt84*, Paris, France (Apr. 1984). (Not in proceedings.)
- [GHY85] **Galil, Z., Haber, S., and Yung, M.** “A Private Interactive Test of a Boolean Predicate and Minimum-Knowledge Public-Key Cryptosystems.” *Proc. 26th IEEE Symp. on Foundations of Computer Science*, Portland, OR (Oct. 1985), 372–382.
- [GoMi84] **Goldwasser, S. and Micali, S.** “Probabilistic Encryption.” *J. Comput. System Sci.* 28, 2 (Apr. 1984), 270–299.
- [GMR85] **Goldwasser, S., Micali, S., and Rackoff, C.** “The Knowledge of Complexity of Interactive Proof-Systems.” *Proc. 17th ACM Symp. on Theory of Computing*, Providence, RI (May 1985), 291–304.
- [GMW86] **Goldreich, O., Micali, S., and Wigderson, A.** “Proofs that Yield Nothing But their Validity and a Methodology of Cryptographic Protocol Design.” *Proc. 27th IEEE Symp. on Foundations of Computer Science*, Toronto, ON (Oct. 1986), 174–187.
- [Yao82] **Yao, A.** “Theory and Applications of Trapdoor Functions.” *Proc. 23rd IEEE Symp. on Foundations of Computer Science*, Chicago, IL (Nov. 1982), 80–91.

Zero-Knowledge Simulation of Boolean Circuits

Gilles BRASSARD [†] and Claude CREPEAU [‡]

Département d'informatique et de R.O.
Université de Montréal
C.P. 6128, Succursale "A"
Montréal (Québec)
Canada H3C 3J7

ABSTRACT

A zero-knowledge interactive proof is a protocol by which Alice can convince a polynomially-bounded Bob of the truth of some theorem without giving him any hint as to how the proof might proceed. Under cryptographic assumptions, we give a general technique for achieving this goal for *every* problem in NP. This extends to a presumably larger class, which combines the powers of non-determinism and randomness. Our protocol is powerful enough to allow Alice to convince Bob of theorems for which she does not even have a proof: it is enough for Alice to convince herself probabilistically of a theorem, perhaps thanks to her knowledge of some trap-door information, in order for her to be able to convince Bob as well, without compromising the trap-door in any way.

1. INTRODUCTION

Assume that Alice holds the proof of some theorem. A *zero-knowledge interactive proof* (ZKIP) is a protocol that allows her to convince a polynomially bounded Bob that she owns such a proof, in a way that he will gain *nothing* else than this conviction: engaging in the protocol with Alice gives Bob no hint on Alice's proof, or at least nothing he can make use of in polynomial time. In particular, it does not enable him to later convince anyone else that Alice has a proof of the theorem or even merely that the theorem is true (much less that he himself has a proof!). This notion was introduced by Goldwasser, Micali and Rackoff [GMR]; the reader is referred to this paper for formal definitions. An intuitive notion of ZKIP suffices to understand this extended abstract.

The early examples of ZKIP's were all number theoretic and restricted to problems in $NP \cap co\text{-}NP$ [GMR, GHY]. It was conjectured by Silvio Micali, and believed by most researchers, that such protocols could not exist for NP-complete problems. Under cryptographic assumptions, we show here that this intuition was wrong by providing a ZKIP for satisfiability. The same result was obtained independently and slightly earlier by [GMW] as they gave a ZKIP for graph 3-colouring. Obviously (because Karp reductions carry NP certificates), it suffices to find a ZKIP for any NP-complete problem in order to get one for every problem in NP. Protocols very similar to ours for satisfiability are also given in [Be, Ch]. Our protocol is more attractive in practice than that of [GMW], but we depend on a specific cryptographic assumption (quadratic residuosity) whereas they merely need to assume the existence of secure encryption schemes in the sense of [GM].

[†] Supported in part by NSERC grant A4107.

[‡] Supported in part by an NSERC postgraduate scholarship; current address: M.I.T.

ZKIP's are conceivable even if Alice does not have a proof to start with. Let us assume that she merely has a *convincing argument* that the theorem is true. In this case, she might wish to convince Bob of the theorem with a level of confidence comparable to her own. This *transfer of confidence* is *zero-knowledge* if it does not provide a polynomially-bounded Bob with any information on the argument itself, except for its existence and Alice's knowledge of it. Our main result is that such protocols exists for a class of problems probably more extensive than NP.

To illustrate the ideas, let us assume that Alice wishes to convince Bob that some integer m (of her choosing) is the product of exactly k distinct primes. Alice is convinced of the truth of her claim because she randomly selected k distinct integers p_1, p_2, \dots, p_k that passed some probabilistic primality test [R, SS] to her satisfaction. Although proofs of primality for these factors exist since PRIMES \in NP [Pr], there is no known feasible algorithm for Alice to get these proofs¹. In other words, Alice knows (with an arbitrary small probability of error) that m is in the proper form, she knows there exists a short proof of this statement, but she cannot find the proof. Using our protocol, she can nonetheless convince Bob without compromising the factorization of m in any way (except for the fact that Bob will know the number of factors).

The above example illustrates the fact that our model does not assume that Alice has more computing power than Bob nor access to some oracle. Although she starts with one piece of additional knowledge (either a formal proof of some theorem or merely a convincing argument), this may be the result of her using trap-door information. The entire protocol itself can be carried out with polynomial time resources.

The general technique allows Alice to guide Bob through the simulation of an arbitrary Boolean circuit without ever having to disclose its inputs or any intermediary results. At the end of the protocol, she can nonetheless convince Bob of the final outcome of the circuit. If this turns out to be 1, Bob will be convinced that the Boolean function computed by the circuit is satisfiable and that Alice holds a satisfying assignment, but he will know nothing else. The bottom line is that, whenever Alice can convince herself probabilistically of a fact or theorem, perhaps thanks to her knowledge of some trap-door information, she can convince Bob as well *without compromising the trap-door*.

2. NUMBER THEORETIC BACKGROUND

Let n be an integer. \mathbb{Z}_n^* denotes the set of integers relatively prime to n between 1 and $n-1$. An integer $z \in \mathbb{Z}_n^*$ is a *quadratic residue* modulo n ($z \in \text{QR}_n$) if there is an $x \in \mathbb{Z}_n^*$ such that $z \equiv x^2 \pmod{n}$. An integer $z \in \mathbb{Z}_n^*$ is a *quadratic non-residue* modulo n ($z \in \text{QNR}_n$) if $z \notin \text{QR}_n$. If p is a prime and if $z \in \mathbb{Z}_p^*$, it is easy to determine whether $z \in \text{QR}_p$ because this is so if and only if $z^{(p-1)/2} \equiv 1 \pmod{p}$. Let $n = pq$ be the product of two distinct odd primes. Given $z \in \mathbb{Z}_n^*$, let z_p and z_q denote $(z \bmod p)$ and $(z \bmod q)$, respectively. Given the factorization of n , it is easy to determine whether $z \in \text{QR}_n$ because this is so if and only if $z_p \in \text{QR}_p$ and $z_q \in \text{QR}_q$. Given the factorization of n and given $z \in \text{QR}_n$, it is also easy (by a Las Vegas algorithm in general [Pe]) to find every $x \in \mathbb{Z}_n^*$ such that $z \equiv x^2 \pmod{n}$. This is however believed to be hard without the factorization of n .

¹ Goldwasser and Kilian's new provably correct and probably fast primality test [GK] allows Alice to "efficiently" (the running time is currently a 12th power polynomial) get short proofs for those primes on which the algorithm turns out to be fast. This might reduce the interest of this particular example, but not the interest of our general protocol.

Given $z \in \mathbb{Z}_n^*$, the Jacobi symbol (z/n) is defined as $+1$ if both z_p and z_q are quadratic residues modulo p and q , respectively, or if both are quadratic non-residues; it is defined as -1 otherwise. It is easy to compute (z/n) even if the factorization of n is unknown [RSA]. Let $\mathbb{Z}_n^*[+1]$ denote the set of $z \in \mathbb{Z}_n^*$ such that $(z/n) = +1$ and define $\mathbb{Z}_n^*[-1]$ similarly. Let $\text{QNR}_n[+1]$ denote $\text{QNR}_n \cap \mathbb{Z}_n^*[+1]$. It is clear that $\mathbb{Z}_n^*[-1] \subset \text{QNR}_n$; moreover, exactly half the members of $\mathbb{Z}_n^*[+1]$ are quadratic residues modulo n and the other half are quadratic non-residues. Both $\mathbb{Z}_n^*[+1]$ and QR_n are closed under multiplication modulo n , the product modulo n of two members of $\text{QNR}_n[+1]$ is a member of QR_n , and the product modulo n of a member of QR_n by a member of $\text{QNR}_n[+1]$ is a member of $\text{QNR}_n[+1]$. A uniformly distributed random element of QR_n can be obtained by randomly choosing some $x \in \mathbb{Z}_n^*$ and squaring it modulo n ; given any fixed $y \in \text{QNR}_n[+1]$, a uniformly distributed random element of $\text{QNR}_n[+1]$ can be obtained by randomly choosing some $x \in \mathbb{Z}_n^*$ and computing $x^2y \bmod n$. Furthermore, everything we have said so far, except for the definition of the Jacobi symbol, remains true if n is of the form $p^i q^j$, where p and q are distinct odd primes and i and j are positive powers of which at least one is odd.

It is believed that no efficient algorithm can distinguish a quadratic residue from a quadratic non-residue, even probabilistically speaking, as long as the latter has Jacobi symbol $+1$ and the factorization of n is unknown. For a more formal statement of this quadratic residuosity assumption (QRA) and for more background on number theory, please refer to [GM].

3. THE ENCRYPTION OF SECRETS

At the beginning of our protocols, Alice randomly chooses two distinct large primes p and q , and she discloses their product $n = pq$ to Bob. Following the QRA, we assume throughout that Bob cannot distinguish a quadratic residue modulo n from a quadratic non-residue, as long as the latter belongs to $\mathbb{Z}_n^*[+1]$. Alice also randomly chooses and discloses to Bob some $y \in \text{QNR}_n[+1]$. (It is proven in [GM] that this cannot help Bob distinguish residues from non-residues.) Using the zero-knowledge interactive protocol of [GHY], Alice convinces Bob that n is of the form $p^i q^j$ for distinct odd primes p and q , and positive integers i and j of which at least one is odd². Using the zero-knowledge protocol of [GMR], Alice convinces Bob that $y \in \text{QNR}_n[+1]$.

At this point, Bob could produce uniformly distributed random members of QR_n and $\text{QNR}_n[+1]$ by choosing a random $x \in \mathbb{Z}_n^*$ and computing either $x^2 \bmod n$ or $x^2y \bmod n$. The fact that only Alice can distinguish between these two occurrences was the basis of Goldwasser and Micali's original probabilistic encryption [GM]. Here, we use this idea *in the reverse direction*: it will always be Alice that produces random members of QR_n and $\text{QNR}_n[+1]$. By convention, members of QR_n are used as encryptions of the bit 0 and members of $\text{QNR}_n[+1]$ are used as encryptions of the bit 1. Whenever Alice shows Bob the encryption z of some bit b , he has no clue as to which bit it encodes (under QRA). It is however possible for Alice to prove to Bob whether $b = 0$ or $b = 1$ by showing him some $x \in \mathbb{Z}_n^*$ such that $z = x^2y^b \bmod n$. This operation will be referred to as *opening the secret* z . Notice that this is a zero-knowledge proof even though a square root of either z or zy^{-1} is given to Bob, because x was randomly chosen by Alice. For this reason, whenever she wishes to

² It would be nicer if Alice could convince Bob directly that n is of the form pq , but we offer in the sequel the first ZKIP capable of achieving this (and therefore we cannot use it yet). This is however of no consequence because Alice could only make herself more vulnerable by choosing $n = p^i q^j$ without $i=j=1$.

open a secret z , there is no need for Alice to use the ZKIP of [GMR] in order to convince Bob of which among zy^{-1} or z belongs to $\text{QNR}_n[+1]$. We give in the last section of this paper a simplified ZKIP for quadratic residuosity when the target is chosen by Bob.

4. CAN BOB COMPUTE ON ENCRYPTED BITS?

Let b_1 and b_2 be two secret bits of Alice, and let z_1 and z_2 be their encryptions as given to Bob. Even though Bob has no knowledge of b_1 or b_2 , he can still compute an encryption of some functions of b_1 and b_2 . For instance, Bob can compute $z_1y \bmod n$, which is an encryption for the negation of b_1 . Similarly, Bob can compute $z_1z_2 \bmod n$, which is an encryption of the exclusive-or of b_1 and b_2 because if $z_1 = x_1^2 y^{b_1} \bmod n$ and $z_2 = x_2^2 y^{b_2} \bmod n$, then

$$z_1z_2 \bmod n = (x_1x_2)^2 y^{b_1+b_2} \bmod n = x^2 y^{(b_1+b_2) \bmod 2} \bmod n,$$

where $x = x_1x_2 y^{(b_1+b_2) \bmod 2} \bmod n$.

Could Bob compute an encryption of the and or the or of b_1 and b_2 given only z_1 and z_2 ? This remains an open question. We will show, however, that it is possible for Bob to do so *with the* (zero-knowledge) *help of Alice*. As a corollary, Bob can compute an encryption of arbitrary Boolean functions of bits for which he only has encryptions. After this computation, Alice can open the result for Bob without ever having had to open the input Boolean variables or any intermediary information. This idea leads to a simple ZKIP for SAT in Section 6.

5. HOW ALICE CAN HELP BOB COMPUTE ON ENCRYPTED BITS

Let $u = b_1b_2 \cdots b_k$ be a k -bit string of Alice. For each i , $1 \leq i \leq k$, let z_i and \hat{z}_i be two encryptions of b_i randomly chosen by Alice. It is easy for Alice to convince Bob that the k -bit strings encrypted by $z_1z_2 \cdots z_k$ and $\hat{z}_1\hat{z}_2 \cdots \hat{z}_k$ are identical without providing Bob with any additional information.

String equality protocol: For each i , $1 \leq i \leq k$, Alice gives Bob some $x_i \in \mathbb{Z}_n^*$ such that $z_i\hat{z}_i \equiv x_i^2 \pmod{n}$. Once again, this is a ZKIP because the encryptions were randomly chosen by Alice and not influenced by Bob. \square

As above, let $u = b_1b_2 \cdots b_k$ and let z_i encrypt b_i for each i , $1 \leq i \leq k$. Now, let $\hat{u} = \hat{b}_1\hat{b}_2 \cdots \hat{b}_k$ be some k -bit string *different* from u and let \hat{z}_i be an encryption of \hat{b}_i for each i , $1 \leq i \leq k$. It is no longer so obvious that Alice can convince Bob that the strings encrypted by $z_1z_2 \cdots z_k$ and $\hat{z}_1\hat{z}_2 \cdots \hat{z}_k$ are different without yielding some additional information (such as a specific i for which $b_i \neq \hat{b}_i$). The fact that this is possible, and the technique that achieves this protocol, illustrate the core of our main result.

String inequality protocol: For each i , $1 \leq i \leq k$, let $v_i = z_i\hat{z}_i \bmod n$. The problem reduces to convincing Bob (by a ZKIP) that the string encrypted by $v_1v_2 \cdots v_k$ is not identically zero. For this, Alice randomly chooses some permutation σ of $\{1, 2, \dots, k\}$ and $x_i \in \mathbb{Z}_n^*$ for $1 \leq i \leq k$. She then computes and discloses to Bob $w_i = x_i^2 v_{\sigma(i)} \bmod n$ for each $1 \leq i \leq k$. At this point, Bob sends either challenge A or challenge B to Alice.

- If Bob sent challenge A, Alice must disclose some i such that w_i encrypts a 1, and open this w_i for Bob by giving him a square root of $w_i y^{-1} \bmod n$.

- If Bob sent challenge B, Alice must disclose the permutation σ and use the string equality protocol to convince Bob that $w_1 w_2 \cdots w_k$ encrypts the same string as $v_{\sigma(1)} v_{\sigma(2)} \cdots v_{\sigma(k)}$.

This process is repeated s times, for some safety parameter s agreed upon between Alice and Bob. In order to convince Bob, Alice must meet every single challenge. \square

Theorem

- (i) The only knowledge obtainable by Bob from this protocol is that $z_1 z_2 \cdots z_k$ and $\hat{z}_1 \hat{z}_2 \cdots \hat{z}_k$ encrypt distinct bit strings, and
- (ii) Alice only has a probability 2^{-s} of convincing Bob of this when in fact the strings are identical.

Proof (sketch).

- (i) Observe that whenever Bob chooses challenge A, he learns that the original bit strings are distinct in at least one place (if Alice was honest), but this gives him no clue as to any single i such that $b_i \neq \hat{b}_i$ because the permutation σ is then kept secret. On the other hand, whenever Bob chooses challenge B, he gains no information whatsoever on the original strings.
- (ii) If in fact $v_1 v_2 \cdots v_k$ encrypts the identically zero string, the only thing Alice can do to hope convincing Bob of the contrary is to guess *exactly* which challenge Bob will choose for each round and to encrypt non-identically zero strings with $w_1 w_2 \cdots w_k$ whenever she expects Bob to use challenge A and identically zero strings otherwise. The results follows from the fact that there are 2^s equally likely sequences of choices for Bob. \square

We are now ready for the main tool used in this paper. Consider any Boolean function $B : \{0, 1\}^t \rightarrow \{0, 1\}$ agreed upon between Alice and Bob, and any bits b_1, b_2, \dots, b_t known to Alice only. For $1 \leq i \leq t$, let z_i be an encryption of b_i known to Bob. Let $b = B(b_1, b_2, \dots, b_t)$. Alice can produce an encryption z for b and *convince Bob that z encrypts the correct bit* without giving him any information on the input bits b_1, b_2, \dots, b_t nor on the result b .

Definition. A *permuted truth table* for the Boolean function B is a binary string of length $(t+1)2^t$ formed of 2^t blocks of $t+1$ bits. The last bit of each block is the value of B on the other t bits of the block, and each assignment of truth values occurs exactly once in the first t bits of some block. For example, here is a permuted truth table for the binary **or**: 011000111101, which should be read as 0 or 1 = 1, 0 or 0 = 0, 1 or 1 = 1 and 1 or 0 = 1.

Boolean computation protocol: Let the situation be as in the paragraph just before the above definition. Alice randomly chooses a permuted truth table for B and she discloses encryptions for each of its bits. At this point, Bob sends either challenge A or challenge B to Alice.

- If Bob sent challenge A, Alice must open the entire encryption of the permuted truth table, so that Bob can check that it is a valid truth table for B .
- If Bob sent challenge B, Alice must point out to the appropriate block in the encryption of the permuted truth table and use the string equality protocol to convince Bob that $z_1 z_2 \cdots z_t z$ encrypts the same bit string as this block.

This process is repeated s times, for some safety parameter s agreed upon between Alice and Bob. In order to convince Bob that z is an encryption for $B(b_1, b_2, \dots, b_t)$, Alice must succeed in meeting every single challenge. \square

A theorem very similar to the one for the string inequality protocol can be stated and the proof is essentially identical. Notice that this protocol is interesting only for small t because it is exponential in t . In the sequel, we will use it *exclusively* with $t \leq 2$. A very similar Boolean computation protocol was discovered independently by Josh Benaloh [Be] as an application of the general tool of “cryptographic capsules” [CF].

6. ZKIP FOR SAT

The zero-knowledge interactive proof for satisfiability should now be obvious. Let $f: \{0,1\}^k \rightarrow \{0,1\}$ be the function computed by some satisfiable Boolean formula for which Alice knows an assignment $b_1, b_2, \dots, b_k \in \{0,1\}$ such that $f(b_1, b_2, \dots, b_k) = 1$. Assume the Boolean formula is given using arbitrary unary and binary Boolean operators. In order to convince Bob that the formula is satisfiable, Alice produces encryptions z_1, z_2, \dots, z_k of b_1, b_2, \dots, b_k , respectively. She then guides Bob through the encrypted evaluation of the formula, one Boolean operator at a time³, using the Boolean computation protocol (with $t \leq 2$). This results in an encryption z for the value of $f(b_1, b_2, \dots, b_k)$. It then only remains for Alice to open z and show Bob that it encrypts a 1.

7. ZKIP FOR THE NUMBER OF PRIME FACTORS

Let us now come back to the problem mentioned in the introduction. Alice has selected k distinct primes p_1, p_2, \dots, p_k and she has formed their product $m = p_1 p_2 \cdots p_k$. She wishes to convince Bob that m is indeed the product of exactly k distinct primes. Let l be the number of bits in m . Each factor will be considered as a length l binary string, with leading zeroes if needed. As a first step, Alice encrypts each of the factors and she discloses these encryptions to Bob. The string inequality protocol is used to convince Bob that the factors are all distinct and that none of them is equal to 1. She then guides Bob through the simulation of a Boolean circuit for iterated multiplication. This produces the encryption of a length kl bit string, which Alice opens to show that it encrypts $(k-1)l$ zeroes followed by the binary representation of m .

At this point, Alice still has to convince Bob that each of these factors is a prime. If she had a proof of this, she could encode it as the input to a proof verification Boolean circuit and guide Bob through its evaluation. Recall, however, that her conviction that each of the p_i is prime comes from her own running of a probabilistic primality test. None of these runs can be considered as convincing by Bob because he cannot trust that Alice was honest in her coin tosses.

This is where our technique is most powerful. Consider a Boolean circuit with two l -bit inputs p and c that outputs 1 if and only if c is a certificate that p is composite (where primes have no certificates and composites have lots [R, SS]). Recall that Bob was given by Alice an encryption of each bit of each p_i . With the help of Alice, he can run as many randomly chosen c 's as he wishes into the circuit for each p_i and ask her to open the circuit outcomes. If he ever gets a 1, he will know for sure that the corresponding p_i is composite and that Alice had been cheating (or perhaps that

³ To save on the number of communications rounds, the various operators can be processed in parallel.

Alice was honest after all, and that she just discovered with him that this p_i is composite!). Otherwise, since he has complete control over the c 's, he can convince himself, with any level of confidence, that m is the product of exactly k distinct primes. This protocol can be adapted if Alice wished instead to convince Bob that there are exactly k distinct primes in the factorization of m , regardless of their multiplicities. A more practical variation allows Alice to convince Bob that the prime factors of n have interesting properties, such as being of the form $2q+1$, where q is also a prime.

8. THE GENERAL PROTOCOL

Recall that **BPP** stands for the class of decision problems that can be solved in probabilistic polynomial time with bounded error probability [G]. It is reasonable to consider **BPP** as the *real* class of tractable problems (rather than **P**) because the error probability can always be decreased below any $\epsilon > 0$ by repeating the algorithm $c \log \epsilon^{-1}$ times and taking the majority answer, where c depends only on the original error probability. It is generally believed that there is no inclusion relation either way between **NP** and **BPP**: non-determinism and randomness seem to be incomparable powers. These powers can be combined in several ways. We believe the most natural to be Babai's class **MA** [Ba], which we would rather call **RNP** as *random NP*. This class is such that $\text{NP} \cup \text{BPP} \subseteq \text{RNP}$, hence **NP** is almost certainly a strict subset of **RNP**. For a discussion as to why we favour **MA** over the seemingly more powerful **AM** or interactive proof systems [GMR], please consult [BC].

Definition. Let Σ stand for $\{0,1\}$. A decision problem $X \subseteq \Sigma^*$ belongs to **RNP** if and only if there exists a predicate $A \subseteq \Sigma^* \times \Sigma^*$ and a polynomial $p(n)$ such that

- (i) $A \in \text{BPP}$, and
 - (ii) $(\forall x \in \Sigma^*)[x \in X \Leftrightarrow (\exists a \in \Sigma^*)[|a| = p(|x|) \text{ and } \langle x, a \rangle \in A]]$
- (such an a is referred to as an *argument* for x). □

Notice that this would correspond to the polynomial hierarchy characterization of **NP** had we insisted that $A \in \text{P}$. The restriction $|a| = p(|x|)$ instead of the usual $|a| \leq p(|x|)$ is there for a technical reason. Notice also that $X \in \text{NP}$ whenever $A \in \text{NP}$.

Intuitively, $X \in \text{RNP}$ means that whenever $x \in X$, there is a (possibly hard to find) short argument for this, and that the validity of this argument can be checked probabilistically in polynomial time. We are about to prove that if $X \in \text{RNP}$, if the proof that $X \in \text{RNP}$ is in the public domain, and if Alice knows an argument a for some $x \in X$, she can convince Bob with a **ZKIP** that $x \in X$. As a warm up, let us first restrict ourselves to one-sided probabilistic algorithms.

Recall that **RP** (sometimes referred to as **R**) is the class of decision problems that can be solved in polynomial time by a *one-sided* bounded error probabilistic algorithm [A]. Here, each time the probabilistic algorithm is run on a yes-instance, it accepts with probability at least $1/2$, whereas it always rejects no-instances. It is well known that $\text{RP} \subseteq \text{NP} \cap \text{BPP}$ and that $\text{co-RP} \subseteq \text{BPP}$, but co-RP and **NP** are probably incomparable. Whenever x is a yes-instance of a **co-RP** problem, one can convince him/herself that this is so (by repeating the algorithm), but there does not have to exist a succinct proof of this.

Theorem (under QRA). Consider a problem $X \in \text{RNP}$ such that the corresponding A (refer to the definition of RNP) belongs to co-RP . Assume that the characterization A for X and a co-RP algorithm for A are in the public domain. Let Alice have an argument a for some $x \in X$. Although she may not have a definite proof that $x \in X$, she convinced herself probabilistically that $\langle x, a \rangle \in A$, hence $x \in X$. It is then possible for Alice to convince Bob in polynomial time that $x \in X$ without disclosing any additional information.

Proof (sketch). Alice and Bob agree on a probabilistic one-sided Boolean circuit for the complement of A . (That is : on any yes-instance of A , using any random choices, the circuit outputs a 0 ; on any no-instance of A , the circuit outputs a 1 for at least 50% of the random choices.) Alice gives Bob an encryption for each bit of x , and she opens them to show that they encrypt x . Alice also gives Bob an encryption for each bit of a , but she keeps a itself secret. She then guides Bob through the evaluation of the Boolean circuit on input $\langle x, a \rangle$, using Bob's coin tosses, until the encrypted outcome is obtained. She then opens this outcome to Bob, who can ascertain that it is indeed a 0. This process is repeated until Bob is convinced that $\langle x, a \rangle \in A$, hence that $x \in X$. Clearly, this gives Bob no information on a (except for its mere existence and Alice's knowledge of it) because the *only* possible outcome for the Boolean circuit is 0, provided Alice was not trying to cheat. Bob does not even learn the length of a because it had to be exactly $p(|x|)$ by definition of RNP . \square

The above protocol does *not* work directly for $X \in \text{RNP}$ in general, because it would not be zero-knowledge. Indeed, Bob would gain information on Alice's argument a from knowledge of which random choices made the circuit accept $\langle x, a \rangle$ and which made it reject, or even merely from knowledge of the number of each of these occurrences. (Recall that if $A \in \text{BPP}$ but $A \notin \text{RP} \cup \text{co-RP}$, the probabilistic Boolean test circuit for A is expected to output sometimes 0 and sometimes 1 on the same input; the most frequent answer being correct with high probability.) Two ideas are needed to solve this difficulty:

- Alice and Bob agree in advance on the number of runs they wish to carry through the test circuit (depending on the error probability they are willing to tolerate). At the end of each run, Alice no longer opens the outcome. After all the runs are completed, Alice guides Bob through the evaluation of a *majority* Boolean circuit, using the previously obtained encrypted outcomes as input. It is only the resulting majority bit that Alice finally opens for Bob.
- Even if Alice is honest, the above idea leaves the door open for Bob to cheat: it could be that the circuit outcome is not what she expected because Bob had deliberately chosen the "random" coin tosses to make this occurrence 50% likely. Assuming Alice's good faith, this could yield up to one bit of information to Bob about the argument a , which is intolerable. Alice would be almost certain that Bob cheated, but it would be too late by then. In order to prevent this possibility, it is essential that all coins be tossed so that neither Alice nor Bob can influence the outcome, and such that Bob does not get to see the outcome (i.e.: coin tossing in a well). Fortunately, such a protocol is very simple: to toss a coin, Alice gives Bob a randomly chosen element of $\mathbb{Z}_n^*[+1]$ and Bob tells her whether to multiply it or not by the standard $y \in \text{QNR}_n[+1]$.

Main Theorem (under QRA). Consider any $X \in \text{RNP}$ and some $x \in X$ for which Alice knows an argument a . Assume the proof that $X \in \text{RNP}$ is in the public domain⁴. Even though Alice may not have a definite proof that $x \in X$, she convinced herself probabilistically that $\langle x, a \rangle \in A$, hence $x \in X$. It is possible for Alice to convince Bob in polynomial time that $x \in X$ and that she knows some argument for this without disclosing any additional information.

Proof (sketch). By the above discussion. □

Let us stress again that this protocol is interesting even when $A \in \text{NP}$, hence $X \in \text{NP}$ (as in section 7 because $\text{PRIMES} \in \text{NP}$), despite the reduction to SAT in these cases. This is so because Alice could know the argument a for x as a result of her choosing a in the first place (as trap-door information) and producing x from it. She might not, however, have an accepting computation for $\langle x, a \rangle$, even though $A \in \text{NP}$. She can nonetheless make use of our protocol. In other words, it does not require Alice to have more computing power than Bob or to have access to some NP-complete oracle. As long as she can convince herself with the help of her own trap-door, she can convince Bob as well without compromising the trap-door.

9. OTHER EXAMPLES OF ZKIP's

Our basic technique can be used in various situations. Let us briefly mention a few of them. It allows Alice to convince Bob of the quadratic residuosity of a member of $\mathbb{Z}_n^*[+1]$ chosen by Bob without yielding additional information, in a way much simpler than those of [GMR, GHY]. It also allows Alice to convince Bob that an encrypted function is a permutation (see below). More generally, all these building blocks can be used directly to obtain *efficient* ZKIP's for a variety of NP-complete problems such as Hamiltonian circuit, clique, knapsack, graph 3-colouring, etc.

Quadratic Residuosity Protocol: Bob shows some $z \in \mathbb{Z}_n^*[+1]$ to Alice and she is willing to convince him of whether it is a quadratic residue or not. Assume initially that $z \in \text{QR}_n$. Alice uses her knowledge of the factors of n to compute some $x \in \mathbb{Z}_n^*$ such that $z = x^2 \pmod n$. Because z was chosen by Bob, it would be far from a ZKIP if Alice revealed x to Bob as proof (it could give Bob a 50% chance of factoring Alice's master secret n). Instead, Alice randomly generates some $u \in \mathbb{Z}_n^*$. She then computes and discloses $w = u^2 \pmod n$. At this point, Bob sends either challenge A or challenge B to Alice.

- If Bob sent challenge A, Alice must disclose u so that Bob can check that $w = u^2 \pmod n$, hence that w is a quadratic residue.
- If Bob sent challenge B, Alice must disclose $ux \pmod n$ so that Bob can check that $(ux)^2 \equiv wz \pmod n$, hence that w has the same quadratic character as z .

This process is repeated s times for some safety parameter s agreed upon between Alice and Bob. The protocol is very similar if $z \notin \text{QR}_n$ but it requires that some standard $y \in \text{QNR}_n[+1]$ has already been proven once and for all. Thus, the protocol of [GMR] must be used *the very first time* in order to make ours effective. □

⁴ i.e.: the predicate A and the BPP algorithm for A are already known to Bob.

A similar protocol is independently given in [Be]; its essence was already in [CF]. Notice also that our protocol would *not* work for quadratic non-residuosity if n had more than two distinct prime factors, whereas the protocol of [GMR] could still be used.

Finally, here is the permutation problem. Let m be some integer agreed upon between Alice and Bob. Let σ be a permutation of $\{1, 2, \dots, m\}$ randomly and secretly chosen by Alice. This permutation can be naturally represented by a table of mk bits, where $k = \lceil \log_2 m \rceil$. Alice discloses to Bob an encryption for each of these bits, so that it will not be possible for her to change her originally chosen permutation. At this point, Bob would like to be convinced that he was given the encryption of a permutation, not just of any function from $\{1, 2, \dots, m\}$ to $\{1, 2, \dots, 2^k\}$. No doubt the reader has seen our technique used enough times by now to design his/her own ZKIP. This problem has applications if one wishes to keep an electronic poker face [Cr], and its solution is central to the above mentioned efficient ZKIP's for Hamiltonian circuit, clique, knapsack, etc.

10. OPEN PROBLEM

Can Bob compute encryptions of arbitrary Boolean functions of encrypted Boolean inputs *without the help of Alice*? For instance, given encryptions for the bits b_1 and b_2 , can he compute an encryption for $(b_1 \text{ and } b_2)$? If so, this might allow a dramatic improvement in our protocols, including the possibility of *publishing* ZKIP's (an idea originally investigated by Manuel Blum).

ACKNOWLEDGEMENT

We wish to thank Josh Benaloh, Joan Feigenbaum, Oded Goldreich, Shafi Goldwasser, Silvio Micali, Jean-Marc Robert, Steven Rudich and Moti Yung for fruitful discussions.

REFERENCES

- [A] Adleman, L., "Reducibility, randomness and intractability", *Proceedings of the 9th Annual ACM Symposium on the Theory of Computing*, 1977, pp. 151-163.
- [Ba] Babai, L., "Trading group theory for randomness", *Proceedings of the 17th Annual ACM Symposium on the Theory of Computing*, 1985, pp. 421-429.
- [Be] Benaloh (Cohen), J. D., "Cryptographic capsules: a disjunctive primitive for interactive protocols", *these CRYPTO 86 Proceedings*, Springer-Verlag, 1987.
- [BC] Brassard, G. and C. Crépeau, "Non-transitive transfert of confidence: a perfect zero-knowledge interactive protocol for SAT and beyond", *Proceedings of the 27th Annual IEEE Symposium on the Foundations of Computer Science*, 1986, pp. 188-195.
- [Ch] Chaum, D., "Demonstrating that a public predicate can be satisfied without revealing any information about how", *these CRYPTO 86 Proceedings*, Springer-Verlag, 1987.
- [CF] Cohen (Benaloh), J. D. and M. J. Fisher, "A robust and verifiable cryptographically secure election scheme", *Proceedings of the 26th Annual IEEE Symposium on the Foundations of Computer Science*, 1985, pp. 372-382.
- [Cr] Crépeau, C., "A zero-knowledge Poker protocol that achieves confidentiality of the players' strategy, or How to achieve an electronic Poker face", *these CRYPTO 86 Proceedings*, Springer-Verlag, 1987.
- [GHY] Galil, Z., S. Haber and M. Yung, "A private interactive test of a Boolean predicate and minimum-knowledge public-key cryptosystems", *Proceedings of the 26th Annual IEEE Symposium on the Foundations of Computer Science*, 1985, pp. 360-371.

- [G] Gill, J. "Computational complexity of probabilistic Turing machines", *SIAM Journal on Computing*, vol. 6, no. 4, 1977, pp. 675-695.
- [GMW] Goldreich, O., S. Micali and A. Wigderson, "Proofs that yield nothing but their validity and a methodology of cryptographic protocol design", *Proceedings of the 27th Annual IEEE Symposium on the Foundations of Computer Science*, 1986, pp. 174-187.
- [GK] Goldwasser, S. and J. Kilian, "Almost all primes can be quickly certified", *Proceedings of the 18th Annual ACM Symposium on the Theory of Computing*, 1986, pp. 316-329.
- [GM] Goldwasser, S. and S. Micali, "Probabilistic encryption", *Journal of Computer and System Sciences*, vol. 28, no. 2, 1984, pp. 270-299.
- [GMR] Goldwasser, S., S. Micali and C. Rackoff, "The knowledge complexity of interactive proof-systems", *Proceedings of the 17th Annual ACM Symposium on the Theory of Computing*, 1985, pp. 291-304.
- [Pe] Peralta, R., "A simple and fast probabilistic algorithm for computing square roots modulo a prime number", *IEEE Transactions on Information Theory*, to appear.
- [Pr] Pratt, V., "Every prime has a succinct certificate", *SIAM Journal on Computing*, vol. 4, 1975, pp. 214-220.
- [R] Rabin, M. O., "Probabilistic algorithms", in *Algorithms and Their Complexity: Recent Results and New Directions*, J. F. Traub (editor), Academic Press, New York, New York, 1976, pp. 21-39.
- [RSA] Rivest, R. L., A. Shamir and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *Communications of the ACM*, vol. 21, no. 2, 1978, pp. 120-126.
- [SS] Solovay, R. and V. Strassen, "A fast Monte Carlo test for primality", *SIAM Journal on Computing*, vol. 6, 1977, pp. 84-85.

All-or-Nothing Disclosure of Secrets

Gilles BRASSARD[†] and Claude CREPEAU[‡]

Département d'informatique et de R.O.
Université de Montréal
Montréal (Québec)
Canada H3C 3J7

Jean-Marc ROBERT^{*}

Département de Génie Electrique
Ecole Polytechnique de Montréal
Montréal (Québec)
Canada H3C 3A7

1. INTRODUCTION

Alice disposes of some number of secrets. She is willing to disclose one of them to Bob. Although she agrees to let him choose which secret he wants, she is not willing to allow him to gain any information on more than one secret. On the other hand, Bob does not want Alice to know which secret he wishes. This is a useful building block in crypto-protocols. For instance, it can be used to easily implement a multi-party mental Poker protocol similar to that of [Cr1], i.e.: safe against player coalitions. An *all-or-nothing disclosure* is one by which, as soon as Bob has gained any information whatsoever on one of Alice's secrets, he has wasted his chances to learn anything about the other secrets. In particular, it must be impossible for Bob to gain joint information on several secrets, such as their exclusive-or. Notice that this is crucial, because it is well-known in classical cryptography that the exclusive-or of two plaintext English messages allows easy recovery of them both, just as a running stream Vigenère would [D].

We assume that Alice is honest when she claims to be willing to disclose one secret to Bob (i.e. she is not about to send junk). The only cheating Alice is susceptible of trying is to figure out which secret is of interest to Bob. Although equally worthwhile, we do *not* address here the problem of *verifiable* secrets¹, because it is too much application dependent. However, the problem of verifiable secrets is addressed and solved in [Cr2] for its specific application to mental poker.

Let us stress that the major novelty consists in letting Bob choose which secret he obtains. This is interesting whenever the secrets are not anonymous: although Bob does not know their contents, he knows their individual purpose². Consider for instance the following situation: an international spy disposes of a large corpus of various state secrets. He sells them by the piece to whoever is

[†] Supported in part by NSERC grant A4107.

[‡] Supported in part by an NSERC postgraduate scholarship; current address: MIT.

^{*} Current address: McGill University, Montréal.

1. That is, preventing that Bob unknowingly obtains a falsified secret should Alice fail to cooperate honestly.

2. In order to get a computationally secure scheme under cryptographic assumptions, it would otherwise suffice to use a variation on oblivious transfer (attributed to Oded Goldreich in [BPT]) to allow "Alice to transfer to Bob exactly one out of two recognizable messages" so that neither has control over which message will be received.

willing to pay the price. In his catalogue, each secret is advertised with a tantalizing title, such as "where is Abu Nidal". He would not accept to give away two secrets for the price of one, or even partial information on more than one secret. On the other hand, you (the potential buyer) would not pay for a randomly chosen secret, but are reluctant to let him know which secret you wish to acquire, because his knowledge of your specific interests could be a valuable secret for him to sell to someone else (under the title : "who is looking for terrorists", for instance). Let us point out that this problem was addressed and solved more than 15 years ago by *quantum physical means*, when the number of secrets is at most three, in Wiesner's original Quantum Cryptography paper [W].

Under cryptographic assumptions, we provide in this paper a practical computationally secure solution. This solution is inspired by our work on zero-knowledge interactive protocols [BC1, BC2]. In a companion paper [BCR], we show how to efficiently reduce this general all-or-nothing disclosure of secrets problem to a much simpler problem known as the two-bit problem. The main interest of this reduction is that it is information theoretic and that it does not depend on unproved assumptions.

We assume that the reader has some number theoretic background, being familiar with the notation \mathbb{Z}_m^* , the notions of quadratic residues and Jacobi symbol, and the quadratic residuosity assumption (QRA) [GM]. We also assume the reader is familiar with the principle of zero-knowledge interactive proofs [GMR].

2. A SOLUTION BASED ON QUADRATIC RESIDUOSITY

Let x_1, x_2, \dots, x_n be Alice's t -bit secrets, and let b_{ij} be x_i 's j^{th} bit for $1 \leq i \leq n$ and $1 \leq j \leq t$. Initially, Alice randomly selects two large distinct primes p and q together with a quadratic non-residue y modulo $m = pq$ whose Jacobi symbol is +1. For each secret bit b_{ij} , she selects a random $x_{ij} \in \mathbb{Z}_m^*$ and computes $z_{ij} = x_{ij}^2 y^{b_{ij}} \pmod{m}$. Notice that z_{ij} is a quadratic residue if and only if $b_{ij} = 0$. Finally, Alice gives Bob both m and y , together with all the z_{ij} 's, keeping p and q secret. According to QRA, this does not enable Bob to obtain in polynomial time any information on Alice's actual secrets.

If Bob wanted to know bit b_{ij} for some specific i and j , and if Alice were willing to cooperate, the following protocol comes to mind: Bob chooses a random $r \in \mathbb{Z}_m^*$ and a random bit a , he computes the question $q = z_{ij} r^2 y^a \pmod{m}$ and he asks Alice for the quadratic residuosity of q . Clearly, $b_{ij} = a$ if and only if q is a quadratic residue. On the other hand, regardless of i and j , q is a random element of \mathbb{Z}_m^* with Jacobi symbol +1 and thus Alice has no idea as to which of her secret bits she has just given away. One might naively be tempted to "solve" ANDOS by allowing Bob to ask t such questions, one for each bit of the secret he wants. There are three flaws with this idea :

- Bob could ask for t bits taken from distinct secrets.
- Bob could obtain in one question the exclusive-or of several bits. For instance, he could ask the question $q = z_{ij} z_{kj} r^2 y^a \pmod{m}$ and thus learn $b_{ij} \oplus b_{kj}$. As pointed out in the introduction, this would most probably enable him to obtain two complete secrets by asking for their exclusive-or, assuming the actual secrets are in plaintext English.
- More subtly, despite the previous claim, this would open the door for Alice to cheat as well! Indeed, she could lie from the beginning and give Bob a quadratic *residue* for her y . In this case, the questions asked by unsuspecting Bob would keep the same quadratic character as the corresponding z 's, allowing Alice to figure out Bob's interests.

In order to solve these difficulties, it is imperative that both Alice and Bob convince the other of their good faith: Alice must show that the information she posted initially is genuine and Bob must convince Alice that his questions are honest. This is where zero-knowledge interactive protocols come into play. The third flaw mentioned above is solved by Alice using zero-knowledge interactive protocols of [GHY] and [GMR] to convince Bob that m has only two prime factors and that y is a quadratic *non-residue* modulo m , respectively. In a context of *verifiable* secret, this is also where Alice would convince Bob that the secrets hidden by the z_{ij} 's respect whichever conditions befit the application (a specific example is given in [Cr2]).

The first two flaws of the naive protocol above are harder to control. Although we have found several solutions, we only sketch here our favourite. Let σ be a permutation of $\{1, 2, \dots, n\}$. A σ -packet P_σ consists of one question for each bit of each secret in the following way $P_\sigma = \langle q_{kj} \mid 1 \leq k \leq n, 1 \leq j \leq t \rangle$ such that each $q_{kj} = z_{ij} r_{kj}^2 y^{a_{kj}} \pmod{m}$, where $i = \sigma^{-1}(k)$, r_{kj} is a random element of \mathbb{Z}_m^* and a_{kj} is a random bit. Moreover, a σ -packet is valid if Bob knows the corresponding σ , r_{kj} 's and a_{kj} 's (notice that any collection of nt elements of \mathbb{Z}_m^* with Jacobi symbol +1 is a σ -packet for every permutation σ , and Alice cannot distinguish a valid packet from any other such collection; however, assuming QRA, it is computationally infeasible for Bob to turn a random collection into a valid packet).

After the initialisation described previously, the ANDOS protocol proceeds as follows if x_i is the secret of interest to Bob.

- Bob randomly selects a permutation σ together with appropriate r_{kj} 's and a_{kj} 's, and forms a valid σ -packet P_σ .
- Bob gives P_σ to Alice, keeping secret his random information, and convinces her that it is a valid packet (see below).
- Bob sends $k = \sigma(i)$ to Alice as his actual request.
- Alice gives Bob the quadratic character of each q_{kj} in Bob's packet P_σ , for this specific k and each $1 \leq j \leq t$.
- Bob infers each of Alice's bits b_{ij} for $1 \leq j \leq t$, hence he obtains x_i as desired.
- If Bob wishes to obtain another secret and if Alice is willing to give (or sell) it to him, it suffices to repeat the three previous steps with the relevant new value for i ; there is no need for Bob to form another packet and convince Alice of its validity all over again (unless it is important for the application that Alice does not even know if Bob's new request is for a different secret).

It is of course crucial that Alice be convinced that Bob's packet P_σ is valid, for he could otherwise stuff it with dishonest questions and we would be back to the beginning. It is equally crucial that Bob does not give Alice a clue as to which permutation σ he chose, for she might otherwise gain information on $\sigma^{-1}(k)$, the secret of interest to Bob. This is achieved by an idea very similar to those leading to the perfect zero-knowledge interactive protocol of [BC2]. Let s be a safety parameter agreed upon between Alice and Bob. After randomly choosing s additional permutations $\sigma_1, \sigma_2, \dots, \sigma_s$ of $\{1, 2, \dots, n\}$, nts new elements of \mathbb{Z}_m^* and nts new bits, Bob creates s additional σ_i -packets P_1, P_2, \dots, P_s . He sends all these packets together with the original P_σ . At this point, Alice selects a random subset $X \subseteq \{1, 2, \dots, s\}$ and sends it to Bob as a challenge. In order to

convince her of the validity of P_σ , Bob must:

- for each $l \in X$, prove the validity of P_l to Alice by disclosing σ_l and all the random elements of \mathbb{Z}_m^* and random bits used in the creation of P_l ;
- for each $l \notin X$, prove to Alice that P_σ is valid if and only if P_l is valid by disclosing $\sigma_l^{-1}\sigma$ and showing that he is capable of transforming the questions in P_σ into the corresponding questions in P_l (we leave the details of this to the reader).

At the end of this subprotocol, Alice will be convinced that P_σ is valid, with a 2^{-s} probability of being fooled by Bob. Indeed, the only way he could convince her of the validity of an invalid P_σ would be by producing valid packets for each $l \in X$ and invalid packets for each $l \notin X$. Since he must do so before being told X , the result follows from the fact that Alice has 2^s different choices for X .

3. OUTLINE OF THE REDUCTIONS OF [BCR]

In [BCR], we give information theoretic reductions among disclosure problems. More precisely, we show that it is exactly as hard to all-or-nothing disclose one t -bit secret among n than it is to disclose one bit among two. This result is obtained by a chain of reductions that allows the collapse of an apparent hierarchy of disclosure problems. Here is a list of problems that turn out to be information-theoretically equivalent, that is even if either or both party(ies) had unlimited computing power, regardless of unproved assumptions.

The *two-bit problem* (2BP): Alice disposes of two secret bits and she is willing to disclose one of them to Bob, at his choosing. Bob must not be allowed to learn more than one bit of information on Alice's bits, but Alice will not be upset if Bob succeeds in gaining any (deterministic) one-bit function of these two bits, such as their exclusive-or. If Bob plays fair and obtains the physical bit of his choice, Alice does not know which of her two bits she disclosed.

The *all-or-nothing two-bit problem* (AN2BP): Alice disposes of two secret bits and she is willing to disclose one of them to Bob, at his choosing. Nothing Bob can do will give him more than one of these physical bits: as soon as he obtains any information on one of them, he loses all hopes to gain any information on the other. Alice does not know which of her two bits she disclosed.

The *all-or-nothing n-bit problem* (ANNBP): this is identical to the previous problem, except that Alice owns n secret bits rather than 2. She wishes to all-or-nothing disclose one of them to Bob, at Bob's choosing.

The *all-or-nothing disclosure of secrets* (ANDOS): described previously.

ACKNOWLEDGEMENTS

We wish to thank Oded Goldreich, Silvio Micali, René Peralta, Joel Seiferas and Umesh Vazirani for their valuable comments. David Chaum has independently suggested a similar protocol based on RSA rather than quadratic residuosity [Ch]. Isabelle Duchesnay has suggested the international spy application. Gilles Brassard also wishes to thank Manuel Blum for his inspiring talk at the MIT Endicott House conference in June 1985.

REFERENCES

- [BPT] R. Berger, R. Peralta and T. Tedrick, "A Provably Secure Oblivious Transfer Protocol", *Proceedings of EUROCRYPT 84*, 1984, pp. 379-386.
- [BC1] G. Brassard and C. Crépeau, "Zero-Knowledge Simulation of Boolean Circuits", these *Advances in Cryptology: Proceedings of CRYPTO 86*, A. Odlyzko, ed., Springer-Verlag, 1987.
- [BC2] G. Brassard and C. Crépeau, "Non Transitive Transfert of Confidence: A Perfect Zero-Knowledge Interactive Protocol for SAT and beyond", *Proceedings of the 27th Annual IEEE Symposium on the Foundations of Computer Science*, 1986, pp. 188-195.
- [BCR] G. Brassard, C. Crépeau and J.-M. Robert, "Information Theoretic Reduction among Disclosure Problems", *Proceedings of the 27th Annual IEEE Symposium on the Foundations of Computer Science*, 1986, pp. 168-173.
- [Ch] D. Chaum, *Private communication*, may 1986.
- [Cr1] C. Crépeau, "A Secure Poker Protocol That Minimizes the Effect of Player Coalitions", *Advances in Cryptology: Proceedings of CRYPTO 85*, H. C. Williams, ed., Lecture Notes in Computer Science 218, Springer-Verlag, 1986, pp. 73-86.
- [Cr2] C. Crépeau, "A Zero-Knowledge Poker Protocol that Achieves Confidentiality of the Players' Strategy, or How to Achieve an Electronic Poker Face", these *Advances in Cryptology: Proceedings of CRYPTO 86*, A. Odlyzko, ed., Springer-Verlag, 1987.
- [D] D. E. R. Denning, *Cryptography and Data Security*, Addison-Wesley, Reading, Massachusetts, 1982.
- [GHY] Z. Galil, S. Haber and M. Yung, "A private interactive test of a Boolean predicate and minimum-knowledge public-key cryptosystems", *Proceedings of the 26th Annual IEEE Symposium on the Foundations of Computer Science*, 1985, pp. 360-371.
- [GM] S. Goldwasser and S. Micali, "Probabilistic Encryption", *Journal of Computer and System Sciences*, **28**, 1984, pp. 270-299.
- [GMR] S. Goldwasser, S. Micali and C. Rackoff, "The knowledge complexity of interactive proof-systems", *Proceedings of the 17th Annual ACM Symposium on the Theory of Computing*, 1985, pp. 291-304.

A zero-knowledge Poker protocol that achieves confidentiality of the players' strategy or How to achieve an electronic Poker face

Claude Crépeau

Laboratory for Computer Science
M.I.T.
545 Technology Square
Cambridge Massachusetts 02141 USA

1. Introduction

Many attempts have been previously made to achieve a protocol that would allow people to play mental poker [SRA, GM1, BF, FM, Yu, Cr] (I would rather say electronic poker). Unfortunately no solution has ever come close to reality with respect to poker strategy. Poker players usually claim that luck has nothing to do with their gains. In fact, poker is a very strategic game. Often, an inexperienced player will lose a lot of money when playing against an experienced player, only because the former cannot hide so easily his emotions. The experienced player can easily know whether his opponent has a good hand or not.

Electronic poker is an ideal way of hiding one's emotions. But, in fact, every protocol proposed thus far ruins this perfect poker face since their security is based on the fact that all hands are revealed at the end of the game. This means that the strategy of the players is known to all his opponents. In particular, if one bluffs with a bad hand in the hope that all his opponents will give up, he still has to reveal his hand at the end, in order to participate in the verification part of the protocol. Moreover, when a player opens his hand, he does not want his opponents to learn the moment at which each of his cards was drawn, since this would give them some information about his strategy.

This paper proposes a new poker protocol that allows players to keep secret their strategy. This protocol is an extension of the one given by Crépeau in [Cr]. The security will not be based on the knowledge of the entire deck of card at the end of the game, but rather on some independent information linked to the entries of the deck. This protocol achieves every constraints of a real poker game. It is the first complete solution to the mental poker problem. It achieves all the necessary conditions suggested in [Cr]:

- Uniqueness of cards
- Uniform random distribution of cards
- Absence of trusted third party
- Cheating detection with a very high probability
- Complete confidentiality of cards
- Minimal effect of coalitions
- *Complete confidentiality of strategy*

2. Review of the protocol in [Cr]

Suppose that P_1, P_2, \dots, P_N want to play poker. Assume a correspondence between the standard deck of cards and the set $DECK = \{1, 2, \dots, 52\}$. Each P_i will pick a permutation π_i of $DECK$ and keep it secret. The shuffled deck will be $\pi_N \cdots \pi_2 \pi_1$, i.e.: the functional composition of these permutations.

To get a card, player P_i picks a value k in $DECK$ that nobody else has picked before, and gets his card by computing $\pi_N \cdots \pi_2 \pi_1(k)$. Since the permutations are kept secret, he will have to use a special trick in order to get this value. To do so, he may use the Hiding-Revealing protocol proposed in [Cr]. This will allow P_i to get the values $\pi_1(k), \pi_2 \pi_1(k)$ up to $\pi_N \cdots \pi_2 \pi_1(k)$ from his opponents. If everybody was getting their cards this way, all would be fine. But somebody could cheat by computing $\pi_N \cdots \pi_2 \pi_1(k')$ for some $k' \in DECK$ he does not own. This way, he may learn cards which are in the hand of another player or still in the deck. Obviously, we cannot tolerate that he gets cards that someone else has already picked. Unfortunately the protocol of [Cr] solves this problem by asking every player to disclose their π_i at the end of the game, thus revealing every hands, including those of players that would not open their hands at the end of a "real" Poker game.

How can P_i prove that he is getting a card nobody else has without revealing this card? This is the main question addressed (and solved) in this paper.

3. A first idea

To achieve this, we will first change the way by which we check that a player has been reading the entries he claims in his opponents' permutations. The main idea is to add some random information to each of the secret values in $\pi_1, \pi_2, \dots, \pi_N$. This information will be randomly chosen bit strings which are long enough to be hard to guess. When a player reads an entry in the permutation of another player, he will have to read the additional bit string linked to it. These strings will later be publicly revealed by the players who wish to open their hands, and they all should match the initial strings if nobody is cheating.

Let s be a security parameter to be chosen by the players. P_i chooses $\tau_i: DECK \rightarrow \{0, 1\}^s$. For $k \in DECK$, the string $\tau_i(k)$ is called the trace of $\pi_i(k)$.

To increase the security of π_i , we are going to link its trace τ_i to it. By linking we mean that the value of $\tau_i(k)$ will have to be read by player P_j whenever he wants the value $\pi_i(k)$ secretly. For this, we use the protocol for the *all-or-nothing disclosure of secrets*, suggested in [BCR], with the 52 secrets

$$\langle \pi_i(1), \tau_i(1) \rangle, \langle \pi_i(2), \tau_i(2) \rangle, \dots, \langle \pi_i(52), \tau_i(52) \rangle$$

instead of simply using the Hiding-Revealing protocol as before [Cr].

Whenever P_j reads one of the $\pi_i(k)$, he will get the corresponding $\tau_i(k)$ and he cannot get $\tau_i(k')$ instead. The interest is that if P_j wants some $\pi_i(k')$ instead of his legitimate $\pi_i(k)$, he will also have to get $\tau_i(k')$ instead of $\tau_i(k)$. Later in the game he will not be able to convince his opponents that he has read $\pi_i(k)$ since he do not know $\tau_i(k)$ and can guess it only with a very small probability.

4. All-or-nothing disclosure of secrets (ANDOS)

Let us first see how such a protocol works. Suppose that Alice has a set of t secrets $\{s_1, s_2, \dots, s_t\}$, and that she wishes to disclose one of them to Bob. Bob does not want Alice to know which secret he takes from the t she has offered him. Alice will choose a secret key for probabilistic encryption, that is two large primes p and q . She will give to Bob the product of them (n) and a quadratic non-residue (y) with Jacobi symbol +1. Let $b_{i,j}$ be the j^{th} bit of the secret s_i . Assume that all the secrets are L bits long. Alice sends to Bob an encrypted version of her secrets. For this, she sends $\hat{b}_{i,j}$, a random quadratic residue mod n when $b_{i,j}$ is 0 and a random non-residue otherwise.

ANDOS PROTOCOL (Encryption of Secrets)

Alice:

STEP 1 chooses p and q , two large primes and computes $n = pq$.

STEP 2 posts n and y , a quadratic non-residue such that $(y/n) = +1$.

STEP 3 chooses $R_{i,j} \in \mathbb{Z}_n^*$ at random for $1 \leq i \leq t, 1 \leq j \leq L$.

STEP 4 posts $\hat{b}_{i,j} = R_{i,j}^2 y^{b_{i,j}} \pmod{n}$, a probabilistic encryption of her secrets.

Now, Bob will build some "questions" about the secrets. To get a secret, Bob will have to ask a question to Alice for each bit of that secret. Typically, a question $Q_{i,j}$ to get bit $b_{i,j}$ looks like $\hat{b}_{i,j} \times r^2 y^m$ for some randomly selected $r \in \mathbb{Z}_n^*$ and $m \in \{0,1\}$. If Bob asks Alice to decide whether $Q_{i,j}$ is a residue or not, he will be able to compute the value of $b_{i,j}$ since he knows the quadratic relation between $Q_{i,j}$ and $\hat{b}_{i,j}$. Also, Alice will not have any idea about the bit Bob has been reading since all possible $Q_{i,j}$'s in $\mathbb{Z}_n^*[+1]$ have equal probability, independently of what $\hat{b}_{i,j}$ is.

When Bob wants a secret, he just asks enough questions to Alice to determine each bit of her secret. But how does Alice know that Bob is not cheating by reading bits in many secrets? He could very well read the first half of some secret together with the second half of another secret.

In order to avoid this, Bob will have to convince Alice that he possesses a set of t fair groups of L questions. A group of questions is fair only if all its questions apply to the same secret. Bob proves to Alice that his groups of questions Q_i are fair in the way suggested in [BCR]. With this protocol, Bob can convince her that his groups of questions are fair and the probability of achieving such a proof when they are not fair is 2^{-s} .

ANDOS PROTOCOL (Preparation of Questions)

Bob:

STEP 1 chooses ρ a permutation of $\{1, 2, \dots, t\}$.

STEP 2 chooses $r_{i,j} \in \mathbb{Z}_n^*$ and $m_{i,j} \in \{0,1\}$ at random for $1 \leq i \leq t, 1 \leq j \leq L$.

STEP 3 posts $Q_{i,j} = \hat{b}_{\rho(i),j} r_{i,j}^2 y^{m_{i,j}}$.

STEP 4 proves that his groups of questions are fair (see [BCR] for details).

Whenever Bob wants to get a secret from Alice, he just tells her which group interests him, and she will decide the quadratic character of each question in it. To convince Bob of her fairness, she also sends him a proof of the quadratic residuosity of each question: a square root of Q when Q is a quadratic residue and a square root of Qy when Q is a quadratic non-residue. From this, Bob will be able to compute the value of the secret he wishes, and Alice will be convinced that he is not getting information on more than one secret, but she will not know which secret she gave away.

ANDOS PROTOCOL (Get a Secret)

Bob:

STEP 1 chooses $i \in \{1, 2, \dots, t\}$ at his will.

STEP 2 sends $\rho^{-1}(i)$ to Alice.

STEP 3 FOR $1 \leq j \leq L$

Alice:

STEP 3.1 sets $\beta_j = \begin{cases} 0 & \text{if } Q_{\rho^{-1}(i),j} \text{ is a quadratic residue} \\ 1 & \text{otherwise.} \end{cases}$

STEP 3.2 finds r_j such that $r_j^2 \equiv Q_{\rho^{-1}(i),j} y^{\beta_j} \pmod{n}$.

STEP 3.3 sends β_j and r_j to Bob.

Bob:

STEP 3.4 computes $b_{i,j} = \beta_j \oplus m_{\rho^{-1}(i),j}$

5. Some basic difficulties

Since the final solution is still based on the use of permutations, we first consider the problem of proving to the other players that the encrypted string produced by a player is indeed a permutation of $\{1, 2, \dots, 52\}$. The problem arises from the fact that these permutations must remain secret even after the end of the game. Since they are never opened, they could in fact not be permutations at all.

One might cheat this way, for instance, by pulling out some cards from the deck and replacing them by copies of some other cards. If he does not get caught, he may learn useful information, for instance he may know that no ace of spade exists.

Suppose that P_i wants to use a permutation π_i in the protocol. He would like to convince his opponents that, indeed, π_i is a permutation of $\{1, 2, \dots, 52\}$. For this he can use a general purpose protocol proving that two encrypted permutations contain the same set of elements. So, we therefore consider first the implementation of this general protocol.

Let $X = \{x_1, x_2, \dots, x_t\}$ be a set known to Bob. Let σ and σ' be two permutations of the elements of X . Consider x_i as a bit string of length L , where $L = \max\{|x_i| : 1 \leq i \leq t\}$. Define $b_{i,j}$ to be the j^{th} bit of x_i . Let $(n=pq, y)$ be Bob's probabilistic encryption public keys. Finally, let $\hat{b}_{i,j}$ be a probabilistic encryption of $b_{\sigma(i),j}$ and $\hat{b}'_{i,j}$ be a probabilistic encryption of $b_{\sigma'(i),j}$.

PERMUTATION EQUALITY PROTOCOL (Preparation)

Bob:

STEP 1 chooses $r_{i,j}, r'_{i,j} \in \mathbb{Z}_n^*$ at random for $1 \leq i \leq t, 1 \leq j \leq L$.

STEP 2 posts $\hat{b}_{i,j} = r_{i,j}^2 y^{b_{\sigma(i),j}} \pmod{n}$ and $\hat{b}'_{i,j} = r'_{i,j}^2 y^{b_{\sigma'(i),j}} \pmod{n}$,
some probabilistic encryptions of his permutations.

Bob can then prove in zero-knowledge to Alice that for all i there exists i' such that for all j $\hat{b}_{i,j}$ and $\hat{b}'_{i',j}$ encrypt the same bit, using the following protocol. (thus proving that the $\hat{b}_{i,j}$'s and the $\hat{b}'_{i',j}$'s encrypt permutations of the same set). Let s be a security parameter agreed between Bob and Alice.

PERMUTATION EQUALITY PROTOCOL

STEP 1 FOR $1 \leq k \leq s$

STEP 1.1 Bob chooses ρ , a random permutations of $\{1, 2, \dots, t\}$.

STEP 1.2 Bob chooses $c_{i,j} \in \mathbb{Z}_n^*$ at random for $1 \leq i \leq t, 1 \leq j \leq L$

STEP 1.3 Bob posts $\bar{b}_{i,j} = c_{i,j}^2 y^{b_{\rho(i),j}} \pmod{n}$.

STEP 1.4 Alice chooses a bit c at random and tells it to Bob.

STEP 1.5 IF $c=0$ Bob reveals $r_{\sigma^{-1}(i),j}, c_{\rho^{-1}(i),j}$ for $1 \leq i \leq t, 1 \leq j \leq L$.

STEP 1.6 IF $c=1$ Bob reveals $r'_{\sigma^{-1}(i),j}, c_{\rho^{-1}(i),j}$ for $1 \leq i \leq t, 1 \leq j \leq L$.

For further details on the construction of this protocol, see [BC]. *Bob* will be able to prove to Alice that $\hat{b}_{i,j}$ and $\hat{b}'_{i',j}$ are encoded permutations of the same set, when in fact it is not, with probability 2^{-s} .

In our case, P_i simply uses $\sigma=\pi_i$ and $\sigma'=I$, where I is the identity permutation. Once the protocol is completed, P_i decrypts the $\hat{b}'_{i,j}$'s and prove that they constitute an encryption of I (by decrypting we mean that he reveals the random seed used to encrypt that information). The preparation part of the protocol may be performed only once, while the second part of the protocol should be performed with each opponent separately. Of course, each player P_i uses his personal values n_i and y_i in place of n and y in the previous protocol.

But in order for this protocol to work, n_i must be of the adequate form (with only two prime factors). In fact, the protocol works whenever $n_i = p_i^a q_i^b$ with both p_i and q_i distinct primes and a and b not both even. In order to prove that n_i is of the correct form, P_i may use the protocol given in [GHY]. By repeating this protocol, P_i can convince each of his opponents that n_i is of the good form. Also, to prove that y_i is a quadratic non-residue modulo n_i he can use the protocol given in [GMR].

Notice that all the protocols suggested so far are zero-knowledge (under the assumption that deciding quadratic residuosity is hard). This makes the following preparation protocol zero-knowledge. Initially, each player P_i uses PREPARATION(i) as suggested below:

PREPARATION(i) P_i :

- STEP 1** chooses π_i , a permutation of $DECK$.
STEP 2 chooses $\tau_i: DECK \rightarrow \{0,1\}^s$ at random.
STEP 3 chooses p_i, q_i and posts $n_i = p_i q_i$ and y_i .
STEP 4 proves that n_i and y_i are in the correct form.
STEP 5 reveals probabilistic encryptions of π_i and τ_i .
STEP 6 uses ANDOS PROTOCOL (Preparation of Questions) with each P_j for the secrets $\langle \pi_i(1), \tau_i(1) \rangle, \dots, \langle \pi_i(52), \tau_i(52) \rangle$.
STEP 7 P_i proves that π_i is indeed a permutation of $DECK$, using PERMUTATION EQUALITY PROTOCOL.

6. Getting cards

Initially, each number k in $DECK$ is marked "free". To get a new card, player P_i picks a "free" value k and mark it "used". We say that k is the identifier of the card. Then, P_i asks publicly his opponents for the values of $\pi_1(k), \pi_2 \pi_1(k)$ up to $\pi_{i-1} \cdots \pi_1(k)$. They will prove their claims by decrypting the corresponding entries of their coded permutations. Then P_i gets $\pi_i \pi_{i-1} \cdots \pi_1(k)$ by looking at his own permutation. Finally he gets the values $\pi_{i+1} \cdots \pi_1(k)$ up to $\pi_N \cdots \pi_1(k)$ by using the secret questions he has proven correct to $P_{i+1}, P_{i+2}, \dots, P_N$. When he does this, he also gets the corresponding strings $\tau_{i+1} \pi_{i+1} \cdots \pi_1(k)$ up to $\tau_N \pi_{N-1} \cdots \pi_1(k)$. These strings will allow him to prove later that he was honest when reading in $\pi_{i+1}, \pi_{i+2}, \dots, \pi_N$.

GET A CARD(i)

- STEP 1** P_i picks k a free value in $DECK$; marks it used.
STEP 2 sets $c = k$
STEP 3 FOR $p = 1$ TO $i-1$
STEP 3.1 P_i gets $\pi_p(c)$ from P_p (publicly)
STEP 3.2 sets $c = \pi_p(c)$
STEP 4[†] P_i adds $\hat{\pi}_i(c)$ to \hat{H}_i
STEP 5 P_i sets $c = \pi_i(c)$
STEP 6 FOR $p = 1$ TO $i-1$
STEP 6.1 P_p shows that he has never used his group of questions that could read $\pi_i(c)$.
STEP 7 FOR $p = i+1$ TO N
STEP 7.1 P_i gets $\langle \pi_p(c), \tau_p(c) \rangle$
 using the ANDOS PROTOCOL (Get a Secret)
STEP 7.2 sets $c = \pi_p(c)$
STEP 8 CARD = c

[†] The meaning of this step will become clear in the next section.

This protocol tolerate that a player reads a card which does not belong to him but only if this card does not belong to someone else, because this does not change the distribution of probability of the hands of the players. Getting any "free" card is equivalent. The only trouble in this case is that the lucky cheater (lucky because he won't get caught) will not be able to use this card since he cannot prove he read it honestly.

7. Opening and Closing of hands.

We have not yet discussed the way by which the players will open a card or declare it closed for the rest of the game (discarded). One might think that claiming "I discard k " for some identifier k that I own, should be sufficient to discard a card. In the same way, maybe, it would be fine to open a card to reveal $\pi_i \pi_{i-1} \dots \pi_1(k)$, $\pi_{i+1} \pi_i \dots \pi_1(k)$, ..., $\pi_N \pi_{N-1} \dots \pi_1(k)$ (since $\pi_1(k)$, $\pi_2 \pi_1(k)$, ..., $\pi_{i-1} \pi_{i-2} \dots \pi_1(k)$ are already known publicly).

But this way, some strategic information will be acquired by the players about their opponents. Suppose that my hand includes the cards of figure 1 (below). Then I may discard the first 2 cards and draw 2 new ones. Suppose I then get into the situation of figure 2.

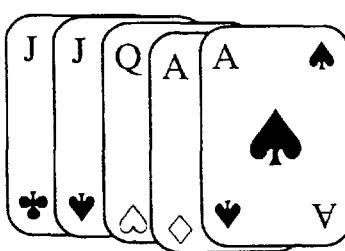


figure 1

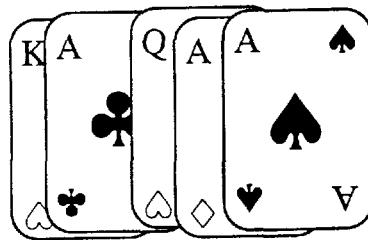


figure 2

If I open up my hand according to the above described protocol, my opponents would know which of my cards are the new ones. This way, they may learn information about my strategy.

Let K_i denote the set of values of $DECK$ owned by P_i in his own permutation π_i . To solve the above mentioned problem, the players will carry an encrypted permuted version of their K_i for the entire game. Note that this information is sufficient to determine his hand. Define $D_i \subseteq K_i$ as the subset of values in K_i which are leading to a discarded card. Clearly, $H_i = K_i - D_i$ is the subset of K_i with elements leading to a card of P_i 's hand.

Initially, H_i and D_i are empty. Whenever P_i gets a card with identifier k , he places the encryption of $\pi_i \pi_{i-1} \dots \pi_1(k)$ into \hat{H}_i , an encrypted version of H_i . Before opening or discarding a card, he will confuse his opponents about the origin of the cards in H_i by generating a new encrypted permutation of the elements in H_i and prove it so with the PERMUTATION EQUALITY PROTOCOL. He will then use this new \hat{H}_i to make his

operation. The point is that his opponents are convinced that H_i still includes the same elements, but they no longer know in which order. Moreover they know that D_i has not changed.

If P_i wants to discard a card from his hand, he transfers the corresponding element of \hat{H}_i into \hat{D}_i :

DISCARD(i,k)

STEP 1 P_i generates a new permuted version of \hat{H}_i and uses PERMUTATION EQUALITY PROTOCOL to prove that H_i has not changed.

STEP 2 P_i places the entry of \hat{H}_i corresponding to $\pi_i \pi_{i-1} \dots \pi_1(k)$ into \hat{D}_i .

On the other hand, if he wants to open it, he just decrypts the corresponding entry of \hat{H}_i and uses it to follow the corresponding values in $\pi_i, \pi_{i+1}, \dots, \pi_N$ in order to get to his card. (remember that the values in K_i are of the form $\pi_i \pi_{i-1} \dots \pi_1(k)$).

OPEN A CARD(i,k)

STEP 1 P_i generates a new permuted version of \hat{H}_i and uses PERMUTATION EQUALITY PROTOCOL to prove that H_i has not changed.

STEP 2 set $c = \pi_i \pi_{i-1} \dots \pi_1(k)$

STEP 3 P_i reveals c

STEP 4 P_i decrypts the entry of \hat{H}_i corresponding to c .

STEP 5 FOR $p = i+1$ TO N

STEP 5.1 P_i reveals $\pi_p(c)$ and $\tau_p(c)$

STEP 5.2 P_p decrypts $\hat{\pi}_p(c)$ and $\hat{\tau}_p(c)$.

STEP 5.3 set $c = \pi_p(c)$

8. General protocol

Finally, here is how all these ideas fit together in order to accomplish a fair, purely secure, game of electronic poker:

POKER PROTOCOL

STEP 1 each player P_i uses PREPARATION(i)

STEP 2 REPEAT UNTIL the end of the game

STEP 2.1 each P_i gets his cards using GET A CARD(i)

According to the rules and to their strategic decisions, the players:

STEP 2.2 bet, discard and open some cards

using DISCARD and OPEN A CARD.

9. In conclusion

We have achieved the first complete solution to the mental poker problem. Our solution cumulates all the conveniences of a real poker game *and* the elimination of the unfortunate human factor (from a strategic point of view). In order to solve even more problems of card playing or similar games (such as Scrabble), with special operations such as returning cards into the deck, the full power of Boolean circuit simulation suggested in [BC] can be used. But unfortunately, the resulting protocol is too messy to be explained here.

10. ACKNOWLEDGEMENTS

I wish to thank Gilles Brassard for the numerous discussions we had about the protocols and for the large amount of work he did with me [BCR, BC]. I would like also to thank Roxane for her support.

11. REFERENCES

- [BF] Banary, I. and Furedi, Z. "Mental Poker with Three or More Players" in *Information and Control*, 59 (1983), pp. 84-93.
- [BC] Brassard, G. and Crépeau C., "Zero-Knowledge Simulation of Boolean Circuits", presented at CRYPTO 86.
- [BCR] Brassard G., Crépeau C. and Robert J.-M., "All-or-Nothing Disclosure of Secrets", presented at CRYPTO 86.
- [Cr] C. Crépeau, "A Secure Poker Protocol That Minimizes the Effect of Player Coalitions", *Advances in Cryptology: Proceedings of CRYPTO 85*, H. C. Williams ed., Lecture Notes in Computer Science 218, Springer-Verlag, Berlin, 1986, pp73-86.
- [FM] Fortune, S. and Merrit, M., "Poker Protocols" in *Advances in Cryptology: Proc. of CRYPTO 84*, G. R. Blakley and D. Chaum, eds., Lecture Notes in Computer Science 196, Springer-Verlag, Berlin, 1985, pp.454-464.
- [GHY] Galil, Z., S. Haber and M. Yung, "A Private Interactive Test of a Boolean Predicate and Minimum-Knowledge Public-Key Cryptosystems" *Proceedings of the 26th Annual IEEE Symposium on the Foundations of Computer Science*, 1985, pp. 360-371.
- [GM1] Goldwasser, S. and Micali S., "Probabilistic Encryption and How to Play Mental Poker Keeping Secret All Partial Information" in *Proceedings of the 14th Annual ACM symp. on Theory of computing*, ACM-SIGACT, May 1982, pp. 365-377.
- [GM2] Goldwasser, S. and Micali S., "Probabilistic Encryption" in *J. Comput. System Sci.*, 28 (1984), pp. 270-299.
- [GMR] Goldwasser, S., S. Micali and C. Rackoff, "The Knowledge Complexity of Interactive Proof-Systems" *Proceedings of the 17th Annual ACM Symposium on the Theory of Computing*, 1985, pp. 291-304.
- [SRA] Shamir, A., Rivest R. and Adleman L., "Mental Poker" MIT Technical Report, 1978.
- [Yu] Yung, M., "Cryptoprotocols: Subscription to a Public Key, The Secret Blocking and the Multi-Player Mental Poker Game" in *Advances in Cryptology: Proc. of CRYPTO 84*, G. R. Blakley and D. Chaum, eds., Lecture Notes in Computer Science 196, Springer-Verlag, Berlin, 1985, pp.439-453.

Secret Sharing Homomorphisms: Keeping Shares of a Secret Secret (Extended Abstract)

Josh Cohen Benaloh*

Abstract

In 1979, Blackley and Shamir independently proposed schemes by which a secret can be divided into many shares which can be distributed to mutually suspicious agents. This paper describes a homomorphism property attained by these and several other secret sharing schemes which allows multiple secrets to be combined by direct computation on shares. This property reduces the need for trust among agents and allows secret sharing to be applied to many new problems. One application described here gives a method of verifiable secret sharing which is much simpler and more efficient than previous schemes. A second application is described which gives a fault-tolerant method of holding verifiable secret-ballot elections.

1 Introduction

Suppose that Alice holds a secret A and distributes shares of her secret to n agents, using Shamir's secret sharing (threshold) scheme ([Sha79]), such that any k agents can construct A . Suppose further that Bob holds a secret B and distributes shares of B to the same n agents in the same way as Alice. Finally suppose that k of the agents decide that they want to determine $A + B$ while revealing as little information about A and B as possible. (Of course, revealing $A + B$ yields some partial information about A and B .) How can this be done?

It is not hard to see that if each of the k agents releases the sum of the two shares it holds, each of these sums is itself a share of the sum of the secrets $A + B$. In short, the sum of the shares of the secrets are shares of the sum of the secrets. It is also the case that release of these share sums gives no information about A and B other than that contained in the release of their sum $A + B$.

In general, suppose each of m parties holds a "sub-secret", and there exists a "super-secret" which is the composition of the sub-secrets under some known function (such as the sum or the product of the sub-secrets). The parties want

*This work was supported in part by the National Security Agency under Grant MDA904-84-H-0004.

to determine the super-secret without revealing their sub-secrets and without depending upon cryptographic assumptions.

Cryptographic techniques for computing with encrypted data have been studied in [RAD78], [DLM82], [Yao82], [BLMe85], and [Fei85], for example. This approach to the problem, however, depends heavily upon cryptographic assumptions such as the difficulty of factoring. In this paper, we shall consider an alternate approach to such problems in which no cryptography or cryptographic assumptions are required (although the data used may be encrypted for other reasons).

With an appropriate secret sharing homomorphism, shares of the sub-secrets can be distributed to n agents such that any k can determine each of the sub-secrets. Each agent can then compose its “sub-shares” into a single “super-share” such that any k of the super-shares are sufficient to determine the super-secret.

The advantage of such a homomorphism is that k of the n agents can, by revealing their super-shares, determine the super-secret without sharing any information about the constituent sub-secrets. Information about the sub-secrets can only be obtained if k or more agents agree to merge their sub-shares to reconstruct the sub-secrets.

At this point, we assume that there are no attempts at subversion. The information is assumed to be correct, and the only concern is that some of the agents may surreptitiously collaborate in order to obtain secret information. In section 4, we shall see examples of how interactive proofs and cryptographic methods can be used to verify both the validity of the shares given to the agents and the accuracy of the composite results returned by the agents.

Two applications of this homomorphism will be seen.

The first allows the validity of secret shares to be verified without their being revealed. Here, a shareholding agent can obtain very high confidence that it holds a valid share of the secret rather than a useless random number. A share is valid if it, when combined with *any* other $k - 1$ shares, yields the same secret as does any subset of k of the shares.

The second application is in the domain of elections. Here a voter can distribute shares of his or her vote to n agents. Each agent can then compose its vote-shares to form a share of the election tally. If k or more of the agents reveal their composite tally-shares, then the election tally is publically revealed. A conspiracy of at least k dishonest agents is required, however, in order to obtain information about an individual vote.

2 The Homomorphism Property

Shamir in [Sha79] defines a (k, n) *threshold scheme* to be a division of a secret D into n pieces D_1, \dots, D_n in such a way that:

- (1) knowledge of any k or more D_i pieces makes D easily computable;

- (2) knowledge of any $k - 1$ or fewer D_i pieces leaves D completely undetermined (in the sense that all its possible values are equally likely).

Let S be the domain of possible secrets, and let T be the domain of legal shares. Every instance of a (k, n) threshold scheme determines a set of functions $F_I : T^k \rightarrow S$ defined for each $I \subseteq \{1, 2, \dots, n\}$ with $|I| = k$. These functions define the value of the secret D given any set of k values D_{i_1}, \dots, D_{i_k} :

$$D = F_I(D_{i_1}, \dots, D_{i_k}),$$

where $I = \{i_1, \dots, i_k\}$.

Definition Let \oplus and \otimes be binary functions on elements of the secret domain S and of the share domain T , respectively. We say that a (k, n) threshold scheme has the (\oplus, \otimes) -homomorphism property (or is (\oplus, \otimes) -homomorphic) if for all I , whenever

$$D = F_I(D_{i_1}, \dots, D_{i_k})$$

and

$$D' = F_I(D'_{i_1}, \dots, D'_{i_k}),$$

then

$$D \oplus D' = F_I(D_{i_1} \otimes D'_{i_1}, \dots, D_{i_k} \otimes D'_{i_k}).$$

This property implies that the composition of the shares are shares of the composition.

It is easy to see that Shamir's polynomial based secret sharing scheme is $(+, +)$ -homomorphic, but it is not quite so apparent that Shamir's scheme satisfies another property which is also necessary to capture the intuition described earlier. We want it to also be the case that up to $k - 1$ sets of sub-shares together with *all* of the super-shares (and therefore the super-secret) give no more information about the sub-secrets than does the super-secret alone.

Shamir's definition of a (k, n) threshold scheme does not allow for such partial information, but Kothari in [Kot84] generalizes Shamir's definition slightly to allow for the possibility of *a priori* information about a secret. Kothari's definition can be summarized by replacing condition (2) above with

- (2') $\text{Prob}(D = x) = \text{Prob}(D = x \mid D_{i_1} = x_{i_1}, D_{i_2} = x_{i_2}, \dots, D_{i_{k-1}} = x_{i_{k-1}})$ for an arbitrary set of $k - 1$ indices $\{i_1, i_2, \dots, i_{k-1}\}$ for all $x \in S$ and all $x_i \in T$.

This says that even with partial *a priori* information about the secret D , possession of up to $k - 1$ shares of D gives no additional information about the value of D .

It is now possible to give a formal definition to capture the intuition that no extraneous information is released by a secret sharing homomorphism.

Definition We define a (\oplus, \otimes) -composite (k, n) threshold scheme to be a division of a set of m sub-secrets d_1, \dots, d_m into sub-shares $d_{i,j}, 1 \leq i \leq n, 1 \leq j \leq m$ ($d_{i,j}$ is the i^{th} share of the j^{th} sub-secret d_j) such that

- (1) The super-secret $D = d_1 \oplus d_2 \oplus \dots \oplus d_m$ is easily computable given k or more distinct super-shares $D_i = d_{i,1} \otimes d_{i,2} \otimes \dots \otimes d_{i,m}$;
- (2) For all possible values $X \in S$ of the super-secret D and for every possible value x_j of each sub-secret d_j ($1 \leq j \leq M$),

$$\text{Prob}(d_j = x_j \mid D = X) =$$

$$\text{Prob}(d_j = x_j \mid D = X; \forall i \in I, D_i = X_i; \forall i \in I', \forall j \in J, d_{i,j} = x_{i,j})$$

where $I = \{1, 2, \dots, n\}$, $J = \{1, 2, \dots, m\}$, and I' is an arbitrary subset of size up to $k - 1$ of $\{1, 2, \dots, n\}$ and for all possible values $X_i \in T$ of the super-shares and for every possible value $x_{i,j} \in T$ of the sub-shares.

Intuitively, the first property says that any k of the n agents can together determine the super-secret D . The second property asserts that no conspiracy of fewer than k agents can gain any information at all about any of the sub-secrets d_j (other than that already given by the super-secret D) even when given all of the super-shares D_i .

The following theorem is somewhat surprising,

Theorem 1 If the secret domain S and the share domain T are finite and of the same cardinality, then every (\oplus, \otimes) -homomorphic (k, n) threshold scheme is a (\oplus, \otimes) -composite (k, n) threshold scheme.

Proof: (sketch)

The definition of (\oplus, \otimes) -homomorphism implies condition (1) immediately.

To prove condition (2), it is simpler to consider only the case when $m = 2$ (two sub-secrets). The case for arbitrary m follows straightforwardly.

Consider a table of the form

S	s_1	s_2	\dots	s_n
A	a_1	a_2	\dots	a_n
B	b_1	b_2	\dots	b_n

where $S = A \oplus B$ and for all i , $s_i = a_i \otimes b_1$. S is the secret defined by the shares s_1, \dots, s_n , A is the secret defined by the shares a_1, \dots, a_n , and B is the secret defined by the shares b_1, \dots, b_n ,

Assume that a set of up to $k - 1$ conspirators are willing to share some of their information in order to try to gain information about A and B . Without loss of generality, assume that these conspirators are among the first $k - 1$ shareholders.

By the definition of a (k, n) threshold scheme, A and B remain completely undetermined even if $k - 1$ shares are known. Therefore, we may assume that

all of a_1, a_2, \dots, a_{k-1} and b_1, b_2, \dots, b_{k-1} are known to the conspirators. With this information, the conspirators are able to compute s_1, s_2, \dots, s_{k-1} without assistance. It is already assumed that the “super-secret” S is known. Therefore, since $|S| = |T| < \infty$, the “super-shares” s_k, s_{k+1}, \dots, s_n are completely determined and can be computed by the conspirators. Thus, their release to the conspirators gives them no additional information. ■

Remark The condition that the secret domain S and the share domain T are of the same finite cardinality was not strictly required, and the following weaker property will suffice. For a given super-share D_k and sub-secrets d_1, d_2, d'_1, d'_2 , such that $d_1 \oplus d_2 = D = d'_1 \oplus d'_2$, let p be the conditional probability that $D_k = d_{1,k} \otimes d_{2,k}$ for some $d_{1,k}$ and $d_{2,k}$ which imply sub-secrets d_1 and d_2 , respectively, and let p' be the conditional probability that $D_k = d'_{1,k} \otimes d'_{2,k}$ for some $d'_{1,k}$ and $d'_{2,k}$ which imply sub-secrets d'_1 and d'_2 , respectively. If $p = p'$ for all such d_1, d_2, d'_1, d'_2 , and D_k , then the conclusion of the theorem is true.

For simplicity of exposition (and to keep the notation under control), this generalization has not been incorporated into the theorem. Its inclusion is straightforward, but cumbersome, and appears to offer no additional insights.

3 Some Examples

It is easy to see that the properties of polynomials give Shamir’s (k, n) threshold scheme the $(+, +)$ -homomorphism property, and since the secret domain and the share domain consist of the same finite set (namely the integers modulo p), Shamir’s scheme is a $(+, +)$ -composite (k, n) threshold scheme and enjoys all of the properties thereof.

Some other techniques can also be easily seen to produce $(+, +)$ -composite (k, n) threshold schemes. See [Bla79], [AsBl80], and [Kot84] for some further examples.

What if the super-secret is not the sum of the sub-secrets? Shamir’s scheme is *not* (\times, \times) -composite. This is because the product of two non-constant polynomials is of higher degree than the factors.

By using a homomorphism between addition and discrete logarithms, for example, it is possible to transform Shamir’s scheme into a $(\times, +)$ -composite (k, n) threshold scheme. Thus, if the desired super-secret is the product of the sub-secrets, Shamir’s scheme can still be used. This method can be summarized by the following adage. *The sum of the shares of the discrete logs of the secrets are shares of the discrete log of the product of the secrets.*

In general, discrete logarithms may be difficult to compute. However, if p is small or of one of a variety of special forms, the problem is tractable (see [PoHe78], [Adl79], [COS86]). It should be emphasized that such special cases for p do not in any way weaken the security of our schemes. The security is *not* cryptographic,

but rather is information theoretic. Therefore, there need be no assumptions about the difficulty of solving any special problems.

4 Applications

The applications described here rely on the encryption of shares both to facilitate their distribution and allow for a mechanism which ensures that certain properties of the shares are attained. Since the encryptions of shares are made public, the security is no longer information theoretic, but rather depends upon a cryptographic assumption.

The encryption function used here was introduced in [CoFi85] and draws upon the ideas of probabilistic encryption found in [GoMi84]. The function is also described in [BeYu86].

Before beginning, a prime number r is fixed such that $r \geq |S|$ — the size of the secret domain. To develop an encryption function E , one selects primes p and q such that $r|(p - 1)$ and $r \nmid (q - 1)$. Let N be the product $N = pq$. The developer releases the pair (N, y) where y is relatively prime to N and y is not an r^{th} residue modulo N .¹ It is necessary in most applications for the developer of such an encryption function to convince others that y is, in fact, not an r^{th} residue modulo N . This may be accomplished by interactive proof techniques described in [CoFi85] and [BeYu86].

To use E to encrypt a value s , one randomly selects an x and forms $E(s, x, y, N) = y^s x^r \pmod{N}$. The holder of the trapdoor factors of N can easily determine s from $E(s, x, y, N)$. However, there is no known efficient method for determining s from its encryption when the factors of N are not known.

4.1 Verifiable Secret Sharing

The first application gives a simple and efficient method for verifiable secret sharing. This problem was first described in [CGMA85] and the application of secret sharing homomorphisms to this problem was developed as a result of an observation made by Oded Goldreich.

Definition We say that a set of n shares s_1, s_2, \dots, s_n is k -consistent if every subset of k of the n shares defines the same secret.

The problem of verifiable secret sharing is to convince shareholders that their shares (collectively) are k -consistent.

It is easy to see that in Shamir's scheme, the shares s_1, s_2, \dots, s_n are k -consistent if and only if the interpolation of the points $(1, s_1), (2, s_2), \dots, (n, s_n)$ yields a poly-

¹ y is an r^{th} residue modulo N if and only if there exists an x such that $y \equiv x^r \pmod{N}$.

nomial of degree *at most* $d = k - 1$. It is also useful to observe that if the sum of two polynomials is of degree at most d , then either both are of degree at most d or both are of degree greater than d .

This suggests the following outline of an interactive proof that a polynomial P , given by its (encrypted) values at n distinct points, is of degree at most d (see [FMR84] and [GMR85] for a description of interactive proofs and applications).

1. Encryptions of the values of the points that describe P are released by the prover.
2. Encryptions of many (say 100) additional random polynomials again of degree at most d are also released by the prover.
3. A random subset of the random polynomials is designated by the verifier(s).
4. The polynomials in the chosen subset are decrypted by the prover. They must all be of degree at most d .
5. Each remaining random polynomial is added to P . (Note that pointwise addition gives the same polynomial as the coefficientwise addition.) Each of these sum polynomials is decrypted by the prover. They must also all be of degree at most d .

The encryption of the values of each point must be probabilistic (to prevent guessing of values) and satisfy a homomorphism property (so that an encryption of the sum of two values can be developed directly from the encryptions of the two values). These properties are satisfied by the encryption function E described above.

In more detail, a secret s is divided into n shares s_1, s_2, \dots, s_n such that the polynomial P interpolated through the points (i, s_i) has degree at most $k - 1$ and passes through the point $(0, s)$. (So far, this is precisely Shamir's scheme). Each (future) shareholder selects and makes public an (N_i, y_i) pair to develop an encryption function E_i as above. The i^{th} share, s_i , is transmitted to the i^{th} shareholder by selecting a random x_i and releasing $E_i(s_i, x_i, y_i, N_i) = y_i^{s_i} x_i^r \bmod N_i$.

To prove interactively that the (encrypted) points released describe a polynomial with degree no more than d , prepare (say) 100 more random polynomials, each of degree at most d , in exactly the same way. The values of these random polynomials at 0 (the secrets they describe) are also selected randomly.

The verifiers randomly select a subset A of these random polynomials. Each polynomial in A is opened by revealing the corresponding s_i and x_i . For each polynomial P' not in A , the (pointwise) sum $P + P'$ is opened by releasing $s_i + s'_i \bmod r$ and $x_i \cdot x'_i \cdot y^{[(s_i + s'_i)/r]}$ where the i^{th} point of P' is given by $E_i(s'_i, x'_i, y_i, N_i)$. All points released should describe polynomials of degree at most d .

It is not hard to see that a set of random polynomials of degree at most d together with a set of sums of P and other random polynomials of degree at most

d gives no useful information about P (other than that its degree is bounded by d).

4.2 Secret-Ballot Elections

The motivating application for this work is in the domain of cryptographic elections. In [CoFi85], an election scheme is presented which allows a government to hold an election in which the legitimacy of the votes and the tally is verified by means of interactive proofs.

Although, there is high confidence in the correctness of the tally in such an election, the government is a “trusted authority” with the ability to see every vote and thereby compromise every voter’s privacy.

In [BeYu86], the government is replaced by a set of “tellers” such that it is necessary for all tellers to conspire in order to compromise a voter’s privacy. In that scheme, however, if even one of the tellers fails to complete its protocol properly, the entire election fails and no tally is produced.

The basic election scheme described in these papers can, however, be embedded within a $(+, +)$ -composite (k, n) threshold scheme (in particular, in Shamir’s scheme) as suggested by the outline below. This extension is also described in [Coh86].

Instead of a single government, n sub-governments (or *tellers*) each hold a sub-election. Each voter chooses either 0 or 1 as a secret value (0 indicating a **no** vote, 1 indicating a **yes** vote) and distributes one share of the secret vote to each of the n tellers. The tally of the election will be the sum of the voters’ secrets.

After votes are cast, each teller simply adds the vote-shares it has received using the (single government) verifiable election scheme of [CoFi85]. Since the (k, n) threshold scheme has the $(+, +)$ -homomorphism property, this sum of vote-shares is itself a share of the sum (tally) of the votes. Thus, once k or more tellers release their sub-tallies, the overall election tally can be determined. Furthermore, since the secret domain and the share domain consist of the same finite set, the conditions of Theorem 1 are satisfied, and k or more conspiring tellers are required to determine any individual voter’s secret vote.

The interactive proof techniques used in section 4.1 can be generalized slightly to allow verification of the vote-shares. Here, each voter participates in an interactive proof to demonstrate to all participants that the vote-shares it distributes are *legitimate* in the sense that every set of k vote-shares derives the same secret vote and that this vote is either a 0 or a 1.

Thus, as long as at least k of the n designated tellers participate through to conclusion, an election can be conducted such that each participant has very high confidence in the accuracy of the resulting tally and no set of fewer than k tellers (together with any number of conspiring voters) can (without breaking the underlying cryptosystem and thereby solving an open number theoretic problem)

gain more than a polynomially small advantage at distinguishing between possible votes of honest voters.

Acknowledgements

The author would like to express many thanks to Mike Fischer, Oded Goldreich, Gregory Sullivan, and David Wittenberg for their help in developing this work.

References

- [Adl79] **Adleman, L.** “Subexponential Algorithm for The Discrete Logarithm Problem.” *Proc. 20th IEEE Symp. on Foundations of Computer Science*, San Juan, PR (Oct. 1979), 55–60.
- [AsBl80] **Asmuth, C.** and **Bloom, J.** “A Modular Approach to Key Safeguarding.” *Texas A&M University, Departement of Mathematics*, College Station, TX (1980).
- [BeYu86] **Benaloh, J.** and **Yung, M.** “Distributing the Power of a Government to Enhance the Privacy of Voters.” *Proc. 5th ACM Symp. on Principles of Distributed Computing*, Calgary, AB (Aug. 1986).
- [Bla79] **Blakley, G.** “Safeguarding Cryptographic Keys.” *Proc. AFIPS 1979 National Computer Conference*, New York, NY (June 1979), 313–317.
- [BlMe85] **Blakley, G.** and **Meadows, C.** “A Database Encryption Scheme Which Allows the Computation of Statistics Using Encrypted Data.” *Proc. IEEE Symposium on Computer Security and Privacy*, Oakland, CA (Apr. 1985).
- [CGMA85] **Chor, B., Goldwasser, S., Micali, S., and Awerbuch, B.** “Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults.” *Proc. 26th IEEE Symp. on Foundations of Computer Science*, Portland, OR (Oct. 1985), 383–395.
- [CoFi85] **Cohen, J.** and **Fischer, M.** “A Robust and Verifiable Cryptographically Secure Election Scheme.” *Proc. 26th IEEE Symp. on Foundations of Computer Science*, Portland, OR (Oct. 1985), 372–382.
- [Coh86] **Cohen, J.** “Improving Privacy in Cryptographic Elections.” *TR-454, Yale University, Departement of Computer Science*, New Haven, CT (Feb. 1986).
- [COS86] **Coppersmith, D., Odlyzko, A., and Schroeppel, R.** “Discrete Logarithms in $GF(p)$.” *Algorithmica*, 1 (1986), 1–15.

- [DLM82] DeMillo, R., Lynch, N., and Merritt, M. “Cryptographic Protocols.” *Proc. 14th ACM Symp. on Theory of Computing*, San Francisco, CA (May 1982), 383–400.
- [Fei85] Feigenbaum, J. “Encrypting Problem Instances or Can You Take Advantage of Someone Without Having to Trust Him”, *Proc. Crypto '85*, Santa Barbara, CA (Aug. 1985), 477–488. Published as *Advances in Cryptology*, ed. by H. Williams in *Lecture Notes in Computer Science*, vol. 218, ed. by G. Goos and J. Hartmanis. Springer-Verlag, New York (1985).
- [FMR84] Fischer, M., Micali, S., and Rackoff, C. “A Secure Protocol for the Oblivious Transfer.” Presented at *Eurocrypt84*, Paris, France (Apr. 1984). (Not in proceedings.)
- [GMR85] Goldwasser, S., Micali, S., and Rackoff C. “The Knowledge of Complexity of Interactive Proof-Systems.” *Proc. 17th ACM Symp. on Theory of Computing*, Providence, RI (May 1985), 291–304.
- [GoMi84] Goldwasser, S. and Micali, S. “Probabilistic Encryption.” *J. Comput. System Sci.* 28, (1984), 270–299.
- [Kot84] Kothari, S. “Generalized Linear Threshold Scheme.” *Proc. Crypto '84*, Santa Barbara, CA (Aug. 1984), 231–241. Published as *Advances in Cryptology*, ed. by G. Blakely and D. Chaum in *Lecture Notes in Computer Science*, vol. 196, ed. by G. Goos and J. Hartmanis. Springer-Verlag, New York (1985).
- [PoHe78] Pohlig, S. and Hellman, M. “An Improved Algorithm for Computing Logarithms Over GF(2) and Its Cryptographic Significance.” *IEEE Trans. on Information Theory* 24, 1 (Jan. 1978), 106–110.
- [RAD78] Rivest, R., Adleman, L., and Dertouzos, M. “On Data Banks and Privacy Homomorphisms.” *Foundations of Secure Computation*, ed. by R. A. DeMillo, et. al. Academic Press, New York (1978), 169–179.
- [Sha79] Shamir, A. “How to Share a Secret.” *Comm. ACM* 22, 11 (Nov. 1979), 612–613.
- [Yao82] Yao, A. “Protocols for Secure Computations.” *Proc. 23rd IEEE Symp. on Foundations of Computer Science*, Chicago, IL (Nov. 1982), 160–164.

How to Share a Secret with Cheaters

Martin Tompa

IBM Thomas J. Watson Research Center

P. O. Box 218

Yorktown Heights, New York 10598

Heather Woll

Department of Computer Science, FR35

University of Washington

Seattle, Washington 98195

Abstract

This paper demonstrates that Shamir's scheme ("How to share a secret", *Communications of the ACM*, vol. 22, no. 11, November 1979, 612-613) is not secure against cheating. A small modification to his scheme retains the security and efficiency of the original, is secure against cheating, and preserves the property that its security does not depend on any unproven assumptions such as the intractability of computing number-theoretic functions.

1. How to Cheat when Sharing a Secret

Shamir [7] proposed and solved a problem in which a secret known only to one party is to be divided among n other participants. This is to be done in such a way that a certain number k of these participants is necessary and sufficient to reconstruct the secret. Each individual participant knows n , k , and the set of possible values of the secret. The problem is stated more precisely as follows:

Inputs:

- Nonnegative integers n , s , and $k \leq n$.
- A "secret" $D \in \{0, 1, \dots, s - 1\}$.

Problem: Divide D into "shares" D_1, D_2, \dots, D_n such that

- (a) knowledge of any k shares is sufficient to efficiently reconstruct D , and
- (b) knowledge of any $k - 1$ shares provides no more information about the value of D than was known before.

This material is based in part upon work supported by the National Science Foundation under grants DCR-8301212 and DCR-8352093.

Such a scheme would be useful, for example, when some data must be replicated over n locations (say, for convenience or fault tolerance), and simultaneously must be protected from $k - 1$ security violations (for example, due to sensitivity of the data or mistrust among the participants).

Shamir's solution is simple, elegant and, unlike most other protocols related to cryptography, not dependent on any unproven assumptions about the complexity of computing certain number-theoretic functions. Shamir's scheme for dividing D into shares is as follows:

1. Choose any prime $p \geq \max(s, n + 1)$. Let Z_p represent the field of integers modulo p .
2. Choose $a_1, a_2, \dots, a_{k-1} \in Z_p$ randomly, uniformly, and independently.
3. Let $q(x) = D + a_1x + a_2x^2 + \dots + a_{k-1}x^{k-1}$.
4. Let $D_i = q(i)$, for all $1 \leq i \leq n$. (The evaluation of $q(i)$ is done over Z_p .)

Properties (a) and (b) now follow from the interpolation theorem, which states that k points are necessary and sufficient to determine $q(x)$. (Details are given in the next section.)

Since the scheme is intended to be useful in applications involving mistrustful participants, the following property is desirable in addition to (a) and (b):

- (c) There is only a small probability $\epsilon > 0$ that any $k - 1$ participants i_1, i_2, \dots, i_{k-1} can fabricate new shares $D'_{i_1}, D'_{i_2}, \dots, D'_{i_{k-1}}$ that deceive a k th participant i_k . Here, deceiving the k th participant means that, from $D'_{i_1}, D'_{i_2}, \dots, D'_{i_{k-1}}$, and D_{i_k} , the secret D' reconstructed is "legal" (i.e., $D' \in \{0, 1, \dots, s - 1\}$), but "incorrect" (i.e., $D' \neq D$).

The desirability of condition (c) is particularly clear when $k = 2$. Without condition (c), a cheater can obtain D while simultaneously, and without being detected, convincing a second participant of an incorrect secret. Notice that the stronger version of condition (c) resulting when $\epsilon = 0$ is unattainable. This is due to the fact that condition (b) implies that for any share D_{i_k} of the secret D and any legal but incorrect secret $D' \neq D$ there must exist $D'_{i_1}, D'_{i_2}, \dots, D'_{i_{k-1}}$ such that the collection of shares $\{D'_{i_1}, D'_{i_2}, \dots, D'_{i_{k-1}}, D_{i_k}\}$ represents the secret D' , thus deceiving the k th participant.

Unfortunately, Shamir's scheme is not secure against such cheating. Firstly, if $p = s$ then all reconstructed secrets are legal, so that it is impossible for the k th participant to detect cheating. One might guess from this that Shamir's scheme can be made secure by choosing p much greater than s , since then there would be only a slight chance of the reconstructed secret being legal. The following example shows that this is not the case. In fact, with high probability a *single* participant can deceive $k - 1$ others.

Suppose that participants i_1, i_2, \dots, i_k agree to pool their shares. Participant i_1 , who decides to cheat, uses interpolation to find a polynomial $\Delta(x)$ of degree at most $k - 1$ such that $\Delta(0) = -1$ and $\Delta(i_2) = \Delta(i_3) = \dots = \Delta(i_k) = 0$. Having been given the share D_{i_1} , participant i_1 announces instead the share $D_{i_1} + \Delta(i_1)$. Now the interpolation theorem guarantees that the k participants will reconstruct the polynomial $q(x) + \Delta(x)$, which has constant term $q(0) + \Delta(0) = D - 1$. Thus, the deception will go undetected unless the original secret happened to be $D = 0$.

In the next section, it is shown that a small modification of Shamir's scheme has all 3 of properties (a), (b), and (c). (In fact, even knowledge of both the secret D and the polynomial $q(x)$ does not increase the probability of successful deception.) Furthermore the running time is polynomial in $k, n, \log s$ and $\log(1/\epsilon)$.

One straightforward solution to the problem of cheating is to have the distributor of shares sign each share D_i with an unforgeable signature (such as that proposed in [5]). This is, in fact, exactly the solution that Rabin [6] chose when he used Shamir's scheme to solve the problem of agreement among distributed processes that might cheat. There are 2 advantages of our scheme over the use of Shamir's scheme plus signatures:

1. All currently known signature schemes depend upon such unproven hypotheses as the intractability of integer factorization, whereas our secret sharing scheme, like Shamir's, does not. In fact, our scheme is secure even if the conspirators have unlimited computational resources.
2. Our scheme is exactly as easy to implement as Shamir's, thus avoiding the complications of implementing an additional signature scheme.

A recent paper [2] introduced a related problem called "verifiable secret sharing". This problem is in some sense more general than ours, since the distributor of secrets, like the other participants, is not above cheating. In particular, the problem requires that the distribution of inconsistent pieces be detected. All known solutions, including the best so far [3], rely on unproven assumptions such as the intractability of integer factorization or the existence of secure encryption schemes. Thus, they have the disadvantages mentioned previously in the discussion of signature schemes.

2. You Can Fool Some of the People All of the Time

This section shows how to modify Shamir's scheme so that the probability of undetected cheating is less than ϵ , for any $\epsilon > 0$.

1. Choose any prime $p > \max((s - 1)(k - 1)/\epsilon + k, n)$.
2. Choose a_1, a_2, \dots, a_{k-1} in Z_p , randomly, uniformly, and independently.
3. Let $q(x) = D + a_1x + a_2x^2 + \dots + a_{k-1}x^{k-1}$.
4. Choose (x_1, x_2, \dots, x_n) uniformly and randomly from among all permutations of n distinct elements from $\{1, 2, \dots, p - 1\}$. Let $D_i = (x_i, d_i)$, where $d_i = q(x_i)$.

Note that the key difference between this and Shamir's scheme occurs in step 4. The proofs of properties (a) and (b) were given by Shamir, and are presented here for completeness.

- (a) Any k participants can determine the secret uniquely by interpolation, since the points x_1, x_2, \dots, x_n are distinct.
- (b) Suppose participants i_1, i_2, \dots, i_{k-1} conspire to determine the secret without consulting participant i_k . When the values of D and $x_{i_1}, x_{i_2}, \dots, x_{i_{k-1}}$ are fixed, $q(x_{i_1}), q(x_{i_2}), \dots, q(x_{i_{k-1}})$ are functions of the random variables a_1, a_2, \dots, a_{k-1} . Using the interpolation theorem and the mutual independence of a_1, a_2, \dots, a_{k-1} , it is straightforward to show that those $k - 1$ values $q(x_{i_1}), q(x_{i_2}), \dots, q(x_{i_{k-1}})$ are uniformly distributed and mutually inde-

pendent. Hence, the secret shares D_1, D_2, \dots, D_{k-1} provide no more information about the value of D than do random numbers. (This proof is somewhat more general than Shamir's, since his assumes that D is chosen by some random process, or at least viewed that way by the conspirators.)

- (c) It remains to explore the probability of deceiving another participant. It will be shown that property (c) holds even if the $k - 1$ cheaters know $q(x)$, and hence know the secret. Suppose participants i_1, i_2, \dots, i_{k-1} fabricate values $(x'_{i_1}, d'_{i_1}), (x'_{i_2}, d'_{i_2}), \dots, (x'_{i_{k-1}}, d'_{i_{k-1}})$ to send to participant i_k . Each possible secret $D' \in \{0, 1, \dots, s - 1\}$ defines a distinct polynomial $q_{D'}(x)$ of degree at most $k - 1$ passing through the point $(0, D')$ and the fabricated points above. If $D' \neq D$, such a polynomial $q_{D'}(x)$ can intersect $q(x)$ in at most $k - 1$ points. Participant i_k will reconstruct the incorrect secret D' only if $q_{D'}(x_{i_k}) = q(x_{i_k})$ and $D' \neq D$. Recall that x_{i_k} is a random element of $\{1, 2, \dots, p - 1\} - \{x_1, x_2, \dots, x_{k-1}\}$. Thus for each polynomial $q_{D'}(x)$ with $D' \neq D$ the probability that $q_{D'}(x_{i_k}) = q(x_{i_k})$ is at most $(k - 1)/(p - k)$. There are $s - 1$ legal but incorrect secrets, so the fabricated values yield $s - 1$ corresponding polynomials. Any one of these polynomials would deceive participant i_k with probability at most $(k - 1)/(p - k)$. Thus the probability of deceiving participant i_k is at most $(s - 1)(k - 1)/(p - k) < \epsilon$.

It will now be shown that this scheme runs in expected time polynomial in $k, n, \log s$, and $\log(1/\epsilon)$. It suffices to demonstrate that the expected time is polynomial in k, n and $\log p$, since p may always be chosen so that $\log p$ is linear in $\log k, \log n, \log s$, and $\log(1/\epsilon)$. A certified prime p of this magnitude can be found in expected time polynomial in $\log p$ [4]. The random choice of a_1, a_2, \dots, a_{k-1} and (x_1, x_2, \dots, x_n) can be done in expected time polynomial in k, n , and $\log p$, as can the evaluation of $q(x)$ at n points over Z_p . Finally, interpolating k points over Z_p can be done in time polynomial in k and $\log p$ [1].

3. How to Keep a Secret from Cheaters

Unfortunately, although cheaters are detected with high probability, they obtain the secret while the other participants gain no information about the secret. The reader can probably imagine applications in which this would be unacceptable.

A simple solution is to introduce a dummy legal value, say s , that is never used as the value of a real secret. The true secret D is now encoded as a sequence $D^{(1)}, D^{(2)}, \dots, D^{(n)}$ where $D^{(i)} = D$ for some i chosen randomly, and $D^{(j)} = s$ for all $j \neq i$. Each element of this sequence is then divided into shares using the scheme of section 2.

When k participants agree to pool their shares, they reconstruct $D^{(1)}, D^{(2)}, \dots$ one at a time, until some $D^{(i)} \neq s$ is obtained. If $D^{(i)}$ is not legal, then cheating has occurred. The probability of cheating on the one crucial round while going undetected at any possible earlier cheats is less than

$$\epsilon^{-1} + \epsilon t^{-1} + \epsilon^2 t^{-1} + \dots = (1 - \epsilon)^{-1} t^{-1}.$$

(This can be proven more formally by induction on t . Recall that even if the cheater suspects the secret is s , the probability of undetected cheating is at most ϵ .)

Acknowledgements

We thank Don Coppersmith for several helpful conversations. Oded Goldreich kindly provided constructive criticism on an earlier draft.

Bibliography

1. A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
2. B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. *Proc. 26th IEEE Symp. Found. Comp. Sci.*, pages 383-395, October 1985.
3. O. Goldreich, S. Micali, and A. Wigderson. How to prove all NP statements in zero-knowledge and a methodology of cryptographic protocol design. *These Proceedings*, 1987.
4. S. Goldwasser and J. Kilian. Almost all primes can be quickly certified. *Proc. 18th Annual ACM Symp. Theory Comput.*, pages 316-329, May 1986.
5. S. Goldwasser, S. Micali, and R. Rivest. A 'paradoxical' solution to the signature problem. *Proc. 25th IEEE Symp. Found. Comp. Sci.*, pages 441-448, October 1984.
6. M. Rabin. Randomized Byzantine generals. *Proc. 24th IEEE Symp. Found. Comp. Sci.*, pages 403-409, November 1983.
7. A. Shamir. How to share a secret. *Comm. ACM*, 22(11):612-613, November 1979.

SMLLEST POSSIBLE MESSAGE EXPANSION IN THRESHOLD SCHEMES

G. R. Blakley
Department of Mathematics
Texas A&M University
College Station, Texas 77843-3368

R. D. Dixon
Department of Computer Science
Wright State University
Dayton, Ohio 45435

A k out of n threshold scheme of any sort known to date involves at least n fold message expansion at the source (where the shadows of the original message are produced). Also, at least k times as much text must be input to the recovery process as is output from it. Linear ramp schemes are more economical but they give only Shannon relative security [BL85]. Is it possible to retain Shannon perfect security and yet cut down the message expansion from at least n fold at the source and at least k fold at the time and place of message recovery? No. In fact the message expansion attained by a Shamir scheme [SH79] or the rigid version [BL85] of a Blakley linear scheme (these two are merely duals of each other) is, in a sense, best possible. Moreover this best possible expansion is slightly larger than just n fold and k fold. The actual expansion factors involve an additive log term. We assume that the reader is familiar with [SH79] and [BL85], and their terminology.

Let k be a positive integer. Let P and N be finite sets such that

$$1 \leq k \leq n = \text{card}(N) \leq \text{card}(P) - 1$$

Here $\text{card}(N)$ stands for the cardinality of the set N . Similarly $\text{card}(P)$. Fix any α which does not belong to N . Let

$$V = \bigcup P^I \cup \{\alpha\}$$

$$W = \bigcup P^H$$

where the unions are over all $(k-1)$ -member subsets I of N , and over all k -member subsets H of N , respectively. A k out of N threshold scheme over P is a pair (E, D) of maps

$$E : V \rightarrow P^{N \cup \{\alpha\}}$$

$$D : W \rightarrow P$$

with the properties that

$$E(\phi)|_{I \cup \{\alpha\}} = \phi$$

$$D(E(\phi)|_G) = \phi(\alpha)$$

for every k -entry list ϕ belonging to V , and every k -member subset G of N . As usual, the restriction $E(\phi)|_G$ of the function

$$E(\phi) : N \cup \{\alpha\} \rightarrow P$$

is the k -member sublist of the n -member list $E(\phi)$ which consists of only those pairs $(t, E(\phi)[t])$ for which t belongs to G . Similarly $E(\phi)|_{I \cup \{\alpha\}}$.

Comment: If the k -entry list ϕ belongs to V then its E -image $E(\phi)$ is a member of $P^{N \cup \{\alpha\}}$ and, thus, amounts to a list with $n+1$ entries. The list $\lambda = E(\phi)$ can have two equal entries, i.e. it is possible to have $\lambda(i) = \lambda(j)$ for two distinct members i, j of $N \cup \{\alpha\}$. But when you consider λ as a set of exactly $n+1$ ordered pairs belonging to $(N \cup \{\alpha\}) \times P$ then no two members of this set of ordered pairs can coincide. For any choice of ϕ belonging to V we will define a shadow of $\phi(\alpha)$ belonging to P to be a member of $\lambda|_N$, considered as an n -member subset of $N \times P$. Thus when enough random material has been chosen so that the substance (i.e. message) $\phi(\alpha)$ has given rise to n shadows, no two of these shadows can coincide. This is not a subtlety. Consider a rigid Shamir 3

out of {1, 2, 3, 6, 8} scheme on GF(13). Let the substance be $\phi(0) = 2$ and suppose two appeals to the random number generator yield

$$\phi(2) = \phi(8) = 2.$$

Then the quadratic polynomial ϕ happens to be the constant polynomial

$$\phi(x) = 2 = 2 + 0x + 0x^2$$

As we have noted, the five shadows of $\phi(0) = 2$ are not the numbers

$$\phi(1) = 2$$

$$\phi(2) = 2$$

$$\phi(3) = 2$$

$$\phi(6) = 2$$

$$\phi(8) = 2$$

but are, instead, the five ordered pairs

$$(1, \phi(1)) = (1, 2)$$

$$(2, \phi(2)) = (2, 2)$$

$$(3, \phi(3)) = (3, 2)$$

$$(6, \phi(6)) = (6, 2)$$

$$(8, \phi(8)) = (8, 2).$$

And no two of these five ordered pairs are equal.

The reader might feel that our definition admits Shamir and Blakley schemes but rules out the one-time pad. For in this 2 out of {pad, transmission} case it would indeed seem possible to have

substance = 0

pad = 0

transmission = 0.

Thus it would seem that the two shadows (i.e. the pad and the transmission) are both equal to zero. We will not address this point directly. We will, instead, say two things. First, one-time pads obey the conclusions of our theorem even if they do not obey its hypotheses. Second, a decoder could not place any reliance on the inference

$$\begin{aligned} 0 &= \text{substance} \\ &= \text{pad XOR transmission} \\ &= 0 + 0 \end{aligned}$$

with only two bits of information lying around: a 0 and a 0. The decoder would only feel confident if at least one more bit of information were available, namely a yes answer to the question: "Is this 0 really the pad and that 0 really the transmission?" We will return to this idea shortly.

The deeper question of how to formulate a definition of threshold scheme which clearly describes and utilizes all the information really available to the decoder and which, perhaps, leads to a more inclusive theorem with conclusions as strong as ours will be left to the reader.

For every $(k-1)$ -member subset I of N , we define probability density functions

$$\mu : P^{\{a\}} \rightarrow [0,1]$$

$$v_I : P^I \rightarrow [0,1]$$

$$\lambda_I = \mu \times v_I : P^{I \cup \{a\}} \rightarrow [0,1]$$

in such a way that v_I is a uniform pdf. Such a threshold scheme is called Shannon perfectly secure through the disclosure of $k-1$ shadows. The reason for this is that, on the basis of the probability assessment [BL81] based on these measures, we have the equality

$$\begin{aligned} &\text{a posteriori probability that } \phi(a) = \pi, \text{ given that } \phi|_G = \psi \\ &= \text{a priori probability that } \phi(a) = \pi \end{aligned}$$

for every π belonging to P , every subset G of N such that $\text{card}(G) = k-1$, and every function $\psi: G \rightarrow P$.

Theorem 1: Let (E, D) be a k out of N threshold scheme on P . Suppose that it is Shannon perfectly secure through the disclosure of $k-1$ shadows. Suppose there is a way of representing shadows as bit strings. Then the average length of a shadow is no less than $\log ([\text{card}(P)][\text{card}(N) - k + 1])$ bits.

Example 1: In Shamir's scheme a shadow is a pair $(x, p(a(x)))$. The nonnegative integer x tells which shadow it is. The substance is $p(0)$. $a(x)$ is a member of $GF(2^L)$ which has been fixed and published in advance for all x belonging to $\{1, 2, \dots, n\}$. p is a polynomial function $p: GF(2^L) \rightarrow GF(2^L)$ of degree $k-1$. Now it is possible that $p(a(x)) = p(a(y))$ for $x \neq y$. But no two shadows can coincide. This is true because the equality $(x, p(a(x))) = (z, p(a(z)))$ simply means that you are comparing two copies of the same shadow, not two shadows, since $x = z$. So Shamir's scheme satisfies our hypotheses. And it is right at the lower bound if $L = 3$, $k = 4$, $n = 7$. A pair $(w, p(w))$ could be formatted as a 3-bit string prefixed by as many bits as needed to identify one of 7 shadows. If $p(w) = b(3) b(2) b(1)$ is always a string of 3 bits then the shadows are of the forms:

```

b(3) b(2) b(1);
1 b(3) b(2) b(1);
1 0 b(3) b(2) b(1);
1 1 b(3) b(2) b(1);
1 0 0 b(3) b(2) b(1);
1 0 1 b(3) b(2) b(1);
1 1 0 b(3) b(2) b(1).

```

their average length is $(3+4+5+5+6+6+6)/7 = 5$ bits. But the bound here is equal to $L + \log(n-k+1) = 3 + \log(7-4+1) = 5$. Somebody might say that you could also omit a high order bit $b(3)$ if it equals 0. But then you would actually have to symbolize the comma in the expression $(x, p(a(x)))$ some way. And this would add

bits to each word. You could go at it the other way and write out all the x values as 3-bit numbers and let the field elements be variable in length. In this approach you have

$b(3) \ b(2) \ b(1) \ x(3) \ x(2) \ x(1)$

dropping high order b bits which are zero. But here again the possible shadows are:

```

x(3) x(2) x(1);
1 x(3) x(2) x(1);
1 0 x(3) x(2) x(1);
1 1 x(3) x(2) x(1);
1 0 0 x(3) x(2) x(1);
1 0 1 x(3) x(2) x(1);
1 1 0 x(3) x(2) x(1);
1 1 1 x(3) x(2) x(1).

```

The average length is now $(3+4+5+5+6+6+6+6)/8 = 5.125$ bits.

This is one example of a general phenomenonon. A Shamir scheme over $GF(2^L)$ achieves the theorem's bound, $L + \log(n-k+1)$ when $n-k+1$ is an integer power of 2.

Example 2: Consider a one-time pad for transmitting messages belonging to a set P of cardinality 4. Here $k = \text{card}(N) = 2$. On the face of things it would appear that the average shadow size is

$$(1 + 1 + 2 + 2)/4 = 3/2$$

which is smaller than the bound

$$\log(4) * (2-2+1) = 2$$

in the theorem. Does this mean that the theorem is merely a curiosity with so many hypotheses that it cannot usefully apply to the simplest cases? Quite the

contrary. When we look more closely at this example we will find that the decoder must have, on the average, four bits of information when he applies the decode process to the two shadows he requires (i.e. the secret pad and the nonsecret transmitted message. We may as well let $P = \{0, 1, 10, 11\}$. There are 16 possibilities

		secret	shared	pad
	0	1	10	11
public trans- mitted message	0	0	1	10
	1	1	0	11
	10	10	11	0
	11	11	10	1
secret message to be communicated				

A naive observer might conclude that the 16 possible outcomes lead to 16 decodes with a total number of input bits equal to

$$(2+2+3+3) + (2+2+3+3) + (3+3+4+4) + (3+3+4+4) = 48.$$

Such an observer would believe the 1.5-bits-per-shadow average alluded to above. The error in such a viewpoint lies in not looking at the whole picture. When two shadow words (e.g. 1 and 11) are ready to be XORed together to produce the reconstructed plaintext word 10 (i.e. the substance 10), the decoder does not have merely three bits of information. Some person or device has inspected the pad and found it to contain a 1, and has monitored the channel and verified that a 11 was actually received in what looks like a legitimate transmission. Thus there is at least one more bit of information available at the decoder. This bit corresponds to a yes answer to the question "Does the pair consisting of 1 and 11 constitute a valid input, one consisting a word from the pad and a word transmitted in the agreed manner down the channel? The decoder thus has 4 bits of information when it forms $1 \oplus 11 = 10$. When we take this into account and

average over the 16 possible pairs we find the average number of bits available at decode to be 1/16th of the sum

$$(3+3+4+4) + (3+3+4+4) + (4+4+5+5) + (4+4+5+5) = 64.$$

Hence the average number of bits per shadow is 2.

It follows that the one time pad, which does not seem to obey the hypotheses of Theorem 1 (because $\text{pad} = 0$, $\text{transmission} = 0$ is allowable, in violation of the assumption that no two shadows are equal), nevertheless obeys its conclusions.

The purpose of Example 2 is to make the following point. We believe that the bound in Theorem 1 cannot be bettered if one takes into account all the information available to a decoder. In other words we believe that our hypotheses are unduly restrictive and that the message bandwidth expansion attained by, for example, certain rigid Shamir schemes [SH79] is best possible no matter how you define a shadow. Decode isn't possible if you merely have a few members of a message space. You must have some further information. And the amount of further information needed is as much as if you knew where each of your message space members occurred in the output stream of the encoder.

The proof of the theorem.

Now fix any $(k-1)$ -member subset I of N . Note that α is not a member of I . Fix any $\pi \in P$. Fix any $\psi \in P^I$ and let $\beta[\psi, \alpha, \pi] \in P^I \cup \{\alpha\}$ be the k -entry list such that

$$\beta[\psi, \alpha, \pi](t) = \psi(t)$$

for every t belonging to I , and such that

$$\beta[\psi, \alpha, \pi](\alpha) = \pi.$$

Define $f: P \rightarrow P$ by requiring that

$$f(\pi) = D(\beta[\psi, \alpha, \pi]).$$

Because of Shannon perfect security, f must be a surjection. For, otherwise, possession of $k-1$ shadows would enable somebody to rule out some values of the substance as impossible. In fact for every z belonging to P and every s belonging to $N \setminus I$ there exists $\Upsilon[z,s]$ belonging to P such that

$$D(\delta[\psi, s, \Upsilon[z, s]]) = \pi$$

where

$$\delta[\psi, s, \Upsilon[z, s]](t) = \psi(t)$$

for every $t \in I$ and such that

$$\delta[\psi, s, \Upsilon[z, s]](s) = \Upsilon[z, s].$$

Since no two shadows of π can be equal for any k -member subset $H = I \cup \{s\}$ of N it follows that there are at least $n-k+1$ such shadows $(s, \Upsilon[z, s])$. This is true for every $\pi \in P$. Hence, for any choice of a k -member subset H of N there are at least $\text{card}(P) * (n-k+1)$ preimages.

References

- BL81 G. R. Blakley and L. Swanson, Security proofs for information protection schemes, Proceedings of the 1981 Symposium on Security and Privacy, IEEE Computer Society, Los Angeles (1981), pp. 75-88.
- BL85 G. R. Blakley and C. Meadows, Security of Ramp Schemes, in G. R. Blakley and D. Chaum, Editors, Advances in Cryptology: Proceedings of Crypto '84, Springer-Verlag, Berlin (1985), pp. 242-268.
- SH79 A. Shamir, How to share a secret, Communications of the ACM, vol. 22 (1979), pp. 612-613.

VLSI implementation of public-key encryption algorithms

G.A. Orton*, M.P. Roy**, P.A. Scott*,
L.E. Peppard* and S.E. Tavares*

* Department of Electrical Engineering
Queen's University
Kingston, Ontario, Canada K7L 3N6

** Bell Northern Research Ltd.
P.O. Box 3511, Station C
Ottawa, Ontario, Canada K1Y 4H7

Abstract. This paper describes some recently successful results in the CMOS VLSI implementation of public-key data encryption algorithms. Architectural details, circuits, and prototype test results are presented for RSA encryption and multiplication in the finite field $GF(2^M)$. These designs emphasize high throughput and modularity. An asynchronous modulo multiplier is described which permits a significant improvement in RSA encryption throughput relative to previously described synchronous implementations.

1. Introduction

The RSA algorithm provides a well known, secure implementation of a public-key cryptosystem [1,2,3]. The arithmetic operations required are exponentiation and modulo reduction involving numbers represented by several hundreds of bits. A VLSI approach is justified but presents challenging problems in terms of control generation and distribution circuitry, minimization of storage register size and achieving an adequate throughput rate. Rivest has given a recent review of other attempts to design an RSA chip [4]. Kochanski [5] has described a cascadable chip which implements 32-bit operations on each chip at a rate of 5kbits/sec for 512-bit encryption; however, it appears that considerable redesign is required to compress the implementation to one or a few chips. CYLINK has recently introduced a chip which can perform 512-bit encryption at 6.4kbits/sec in 2um CMOS [6]. A faster design is currently under development at Sandia National Laboratories [7], which uses delayed carry adders to avoid carry propagation delay. This approach is said to be capable of 25kbits/sec in 2um CMOS but has added complexity due to the difficulty of performing comparisons, storage of two K-bit numbers for intermediate results, where K is the number of bits in the modulus, and conversion of the result from the delayed carry representation to binary. Also, the MSB of the modulus must be justified (the message is shifted equally) and the ciphertext returned to LSB justification at the end of the encryption and, as well, the modulo multiplication results need to be shifted 11 bits.

In this paper we describe a bit-slice architecture which incorporates the RSA control functions in the slice along with the arithmetic (modulo multiplication) functions. Registers longer than the modulus are avoided using concurrent modulo reduction. A 32-bit prototype has been fabricated in 3um CMOS and successfully tested. Based on test results and simulations, a throughput rate of 1kbits/sec should be possible for 512-bit encryption with a 2um CMOS process.

In an effort to substantially improve the throughput rate of the bit-slice implementation, a new bit-slice design has been developed employing asynchronous (self-timed) ripple adders [8]. With a small penalty in extra control circuitry, an increase in throughput of up to 40 times can be obtained. A 22-bit prototype in 3um CMOS has been successfully fabricated and tested and a 64-bit version is currently being fabricated.

Multiplication in the finite field $GF(2^m)$ is employed in several data encryption algorithms as well as other areas of communications [7,9]. Recent work has shown that cryptographic algorithms based on arithmetic in $GF(2^m)$ require very large values of m for security [10,11]. In particular, values of m in the range of 1500 bits are recommended. However, for large m, efficient VLSI implementation of the multiplication function requires careful algorithm design to provide modularity and concurrency as well as simplified control requirements. A new multiplication algorithm will be described along with a suitable bit-slice VLSI architecture [12]. Test results from an 8-bit prototype will be presented.

2. Modulo multiplication algorithms

The RSA encryption and decryption transformations involve exponentiation and modulo reduction of a text data block possibly of several hundred bits. The arithmetic process involved is modulo multiplication which requires addition, subtraction and shifting.

Brickell [7] and Blakely [13] have proposed modulo multiplication algorithms in which multiplication is performed concurrently with modulo reduction. This differs from the algorithms used by Rivest

[14] and Simmons and Tavares [15] where multiplication of two K-bit numbers is first performed and then the resulting 2K-bit number is modulo reduced. The maximum word length is $(K+1)$ bits using concurrent modulo reduction. Concurrent algorithms save storage space, reduce adder carry propagation time and require fewer clock periods. Only algorithms of this type will be considered in the remainder of this section.

All of the concurrent algorithms which restrict number lengths to $(K+1)$ bits perform multiplication in one of two ways. The most familiar way of multiplying two numbers is to add shifted versions of the multiplicand or zero depending on the value of the multiplier bits. An example of this technique is shown below in Example 1(a). The second way involves adding the multiplicand or zero to the running total and shifting the running total, as shown in Example 1(b).

Binary multiplication	
(a)	(b)
$ \begin{array}{r} 1010. \\ \times 1101 \\ \hline 1010. \\ 0000. \\ 10100. \\ + \underline{1010000} \\ 10000010. \end{array} $	$ \begin{array}{r} 1010. \\ \times 1101 \\ \hline 1010. \\ 1010. \\ 0000. \\ + \underline{1010.} \\ 10000010. \end{array} $

Most techniques of modulo reduction rely on adding some positive or negative multiple of the modulus. With the following concurrent modulo reduction algorithms, the number being reduced is smaller in magnitude than twice the modulus. The modulus is either added or subtracted to reduce the absolute value of the number below the magnitude of the modulus. Modulo reduction can also occur indirectly. An example of indirect modulo reduction of the running total is to first add or subtract the modulus from another number such as the intermediate product (IP) and then add the adjusted IP to the running total. With the two methods of performing multiplication illustrated in Example 1, a variety of methods for concurrent modulo reduction can be employed all of which must prevent overflow by finishing with a number less in magnitude than the modulus. The algorithms operate correctly with starting values less in magnitude than the modulus, so if overflow occurs, the magnitude would continue to increase in subsequent periods. The conditions for concurrent modulo reduction, without overflow, are summarized below.

If a number, A, which is less in magnitude than the modulus, n, is multiplied by 2 or added to another number, B, which is also less in magnitude than the modulus, the intermediate result can be modulo reduced in the time for one addition. In the case of a positive intermediate result the modulus is subtracted and in the case of a negative intermediate result the modulus is added.

$$\begin{array}{lll}
 \text{i.e.} & \text{if } 0 \leq A < n & \text{if } 0 \leq A < n \text{ and } 0 \leq B < n \\
 & \text{then } 2A - n < n & \text{then } A + B - n < n \\
 & \text{and } 2(-A) + n > (-n) & \text{and } (-A) + (-B) + n > (-n)
 \end{array}$$

A number of useful modulo multiplication algorithms will now be discussed.

Algorithm A

A flow graph of the algorithm is shown in Fig. 1 which is a modification of Blakely's algorithm [13]. Multiplication is done by shifting the intermediate product (IP) and modulo reducing if the IP is greater than the modulus [16]. Then, if the multiplier bit is a 1, the IP is added to the running total. After this, the running total is modulo reduced if necessary. Both the IP and the running total are always positive. A disadvantage of this algorithm is that the running total may require two consecutive additions per multiplier bit.

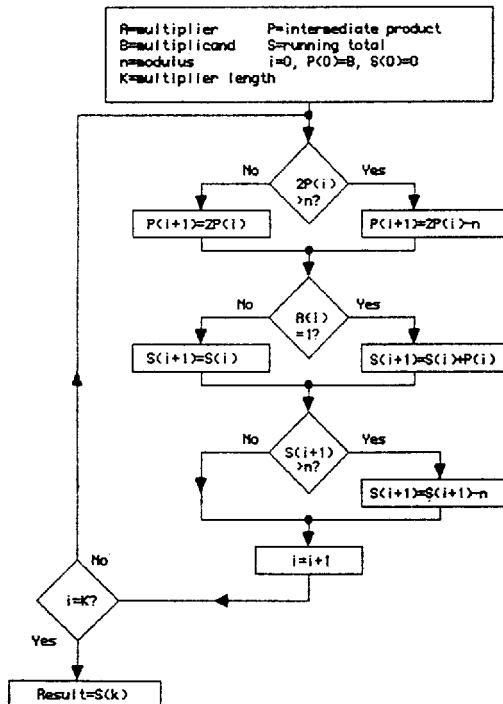


Fig. 1. Modulo multiplication algorithm A. Three adders are used with an average of 1.5 addition phases per multiplier bit.

Algorithm B

A concurrent modulo multiplication algorithm was suggested by Simmons and Tavares [15] which uses multiplication with the running total multiplied by two each period as shown in Fig. 2. A normal cycle starts with the running total being multiplied by 2, followed by adding the IP. Then the running total is modulo reduced using an add/subtract scheme. However, overflow occurs because the combination of multiplication by 2 followed by addition of the multiplicand cannot always be modulo reduced with one addition or subtraction. A necessary modification is to add a negative IP if the running total is positive. This negative IP is generated during the 1st period by adding the positive multiplicand to the running total and then subtracting the modulus to produce a negative result. The negative IP is then stored in a separate register for future use. With this method the final result must be adjusted positive by adding the modulus if necessary. A maximum of one period is required for this

step. Algorithm B requires only 2 adders but, as with algorithm A, two consecutive additions may be required per multiplier bit.

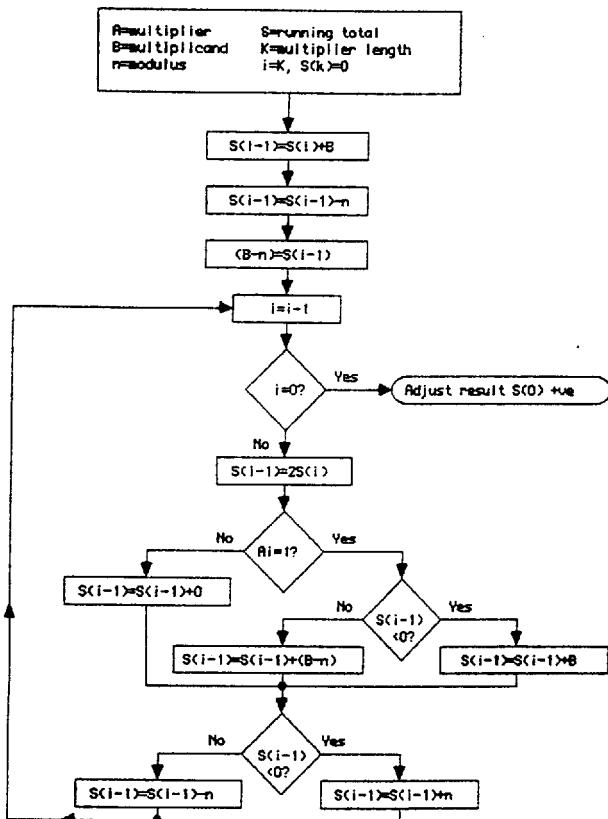


Fig. 2. Algorithm B for modulo multiplication. Two adders are used with an average of 1.5 addition phases per multiplier bit.

Algorithm C

It was thought desirable to consider a modulo multiplication algorithm which would perform all additions during one addition phase per multiplier bit and with a reduced number of adders. An algorithm which uses only 2 adders, which operate concurrently, is algorithm C shown in Fig. 3. In this algorithm, the running total is multiplied by 2 each period. Then if the multiplier least significant bit (MLT1sb) is 0, the running total is modulo reduced. If the MLT1sb is 1, two additions are performed and one of the results is selected as the new modulo reduced running total. This algorithm requires more area because four intermediate products, P, P+n, P-n, and P-2n, need to be first generated then stored. Due to the increase in area required for algorithm C, it was not considered further.

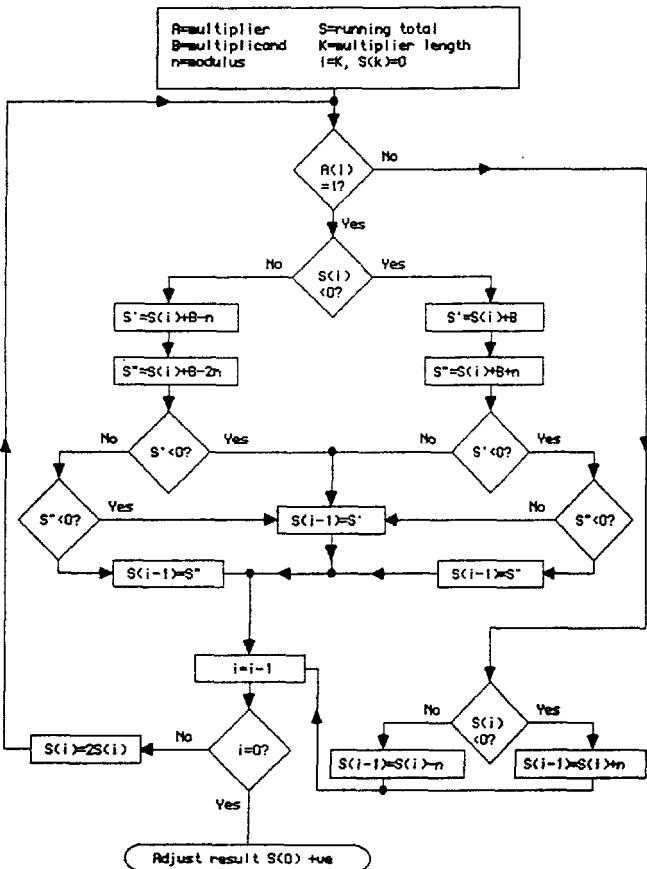


Fig. 3. Algorithm C for modulo multiplication. Two adders are used with one clock phase per multiplier bit.

Algorithm D

Algorithm D performs three additions in a single phase per multiplier bit as shown in Fig. 4. Asymmetrical clock phases are generated with a shortened phase used to set up the adder inputs. As with algorithm A, the previous IP is multiplied by 2 each period. Algorithm D differs in running total generation. If the running total is positive, a negative IP is added to the running total. On the other hand, if the running total is negative, a positive IP is added. This keeps the running total from overflowing and allows it to be generated in one step. The positive IP is generated exactly the same as the IP in algorithm A. At the same time as the positive IP is modulo reduced, twice the modulus is subtracted from twice the previous positive IP. This results in an IP between 0 and -2n. After K periods, where K is the number of bits in the modulus, the multiplication is finished but the running total may need to be adjusted positive by adding the modulus. This takes a maximum of two periods. Algorithm D uses three adders with 1 concurrent addition per multiplier bit and provides a useful compromise between speed and area.

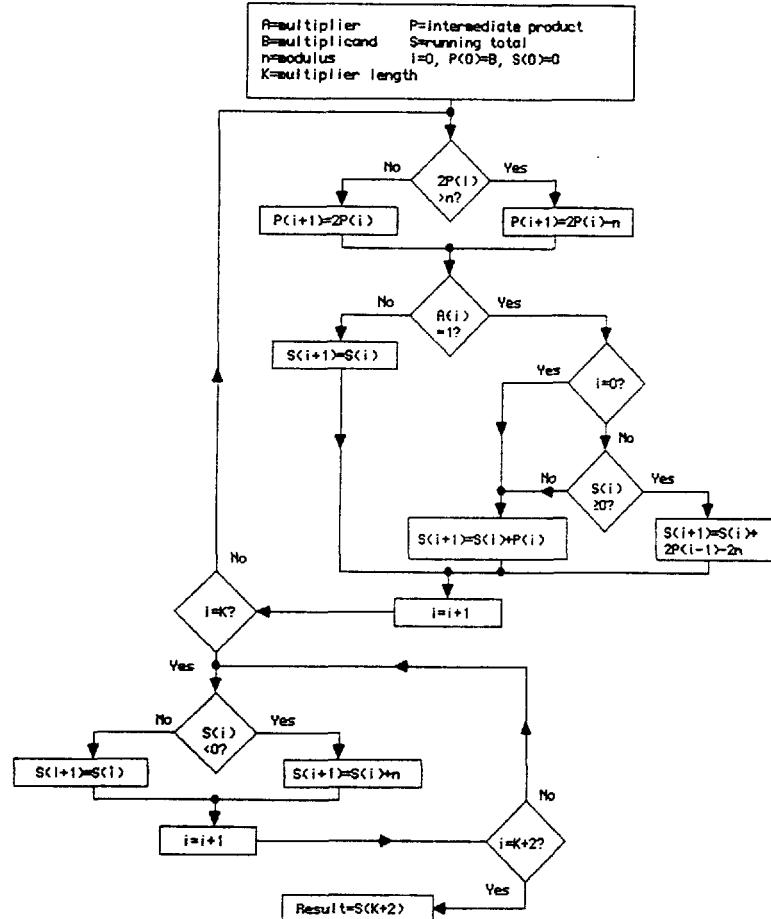


Fig. 4. Modulo multiplication algorithm D. Three adders are used with one phase per multiplier bit.

Algorithm E

An efficient concurrent modulo multiplication algorithm may be devised using the modified Booth's Algorithm (MBA), where the multiplier is shifted two bits at a time. Two consecutive additions per clock period would be required, but the number of clock periods would be reduced by half. This algorithm will be faster if the constant circuit delays in a period are larger than the average addition time which would be the case with a fast adder. In the case of the adder to be described later, approximately a 10% to 15% increase in area would result along with 50% improvement in speed compared to algorithm D.

The modified Booth's Algorithm (MBA) is frequently used to improve the speed of multipliers [17]. Through encoding of the multiplier bits, the number of intermediate products to be added is reduced by half. Booth's Algorithm works by skipping over any contiguous string of all 1's or all 0's. A string of all 0's does not require any IP's to be added, but a string of 1's requires an addition and a subtraction. For example, if the multiplier is 11100, (100000 X multiplicand) is added and (10 X multiplicand) is subtracted. The MBA looks at

the three least or most significant multiplier bits at a time depending on the direction that the multiplier is being shifted and shifts the multiplier by 2 bits each clock period. The intermediate products are multiplied by 0, ± 1 , and ± 2 before accumulation as shown in Table 1.

Table 1. Encoding of multiplier for the modified Booth's Algorithm. The centre bit of the three bits being encoded is referred to as the i_{th} bit.

Multiplier bits			Factor of IP accumulated	Operation
$i+1$	i	$i-1$		
0	0	0	0	no string
0	0	1	+1	end of string
0	1	0	+1	a string
0	1	1	+2	end of string
1	0	0	-2	beginning of string
1	0	1	-1	$-2 + 1 = -1$
1	1	0	-1	centre of string
1	1	1	0	middle of a string

Algorithm E is diagrammed in Fig. 5 and uses a total of four adders with two adders operating in each phase of a two phase clock. The intermediate products are generated as in algorithm A, but multiplied by 0, ± 1 , or ± 2 before accumulation. Two positive IP's are generated each period consecutively, corresponding to 2 and 4 times the previous IP. Each IP is calculated by shifting the previous IP by 1 bit and subtracting the modulus if necessary. Each period, the appropriate IP is selected, inverted if a negative IP is required and added to the running total. The running total is then modulo reduced by either adding or subtracting the modulus. Two additions are performed each period to generate the IP's and two additions are used to generate the running total. An algorithm which uses fewer additions could be devised at the expense of more memory.

Comparison of modulo multiplication algorithms

The algorithms presented in this section employ concurrency of multiplication and modulo reduction to improve the bit throughput rate. A comparison of six modulo multiplication algorithms is given in Table 2. The selection of the "best" algorithm depends on system parameters such as the delay required for additions relative to constant circuit delays, the availability of non-symmetric clock phases, asynchronous timing and memory. Algorithm D was chosen for implementation because it is almost as fast as algorithm C and occupies about the same area as algorithm A. Algorithm E has only been considered recently. Algorithms A and B are closely matched in speed and area. Long addition times relative to constant delays result in algorithms D and E operating at the same speed, while short addition times make algorithm E twice as fast. With the pulse-timed adder to be described in section 4, the constant circuit delays are at least twice as large as the average addition time, which would make algorithm E at least 50% faster than algorithm D.

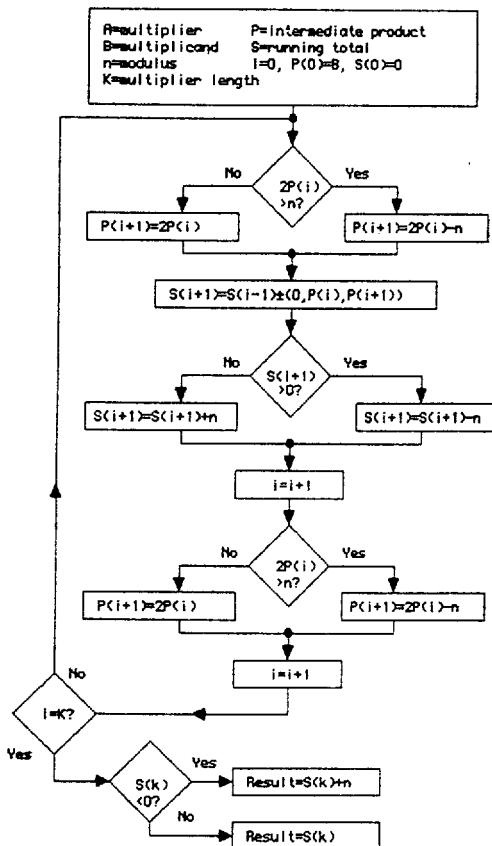


Fig. 5. Modulo multiplication algorithm E. Four adders are used with an average of 1.75 clock phases and the modified Booth's Algorithm.

3. RSA implementation

Architectural aspects

Modulo multiplier architecture. A bit slice architecture for algorithm D is shown in Fig. 6. With a fast adder, communication delays become more significant. Signal flow within the bit slice is less time consuming than the propagation of signals, such as clock signals, MLT1sb, START, ADDER1 carryout, and BEGIN which must be sent to all slices. A completion signal generator subsystem (CSG) is added if pulse-timed adders are used (described in section 4).

A sum term generator controller (STGC) is required to select the input to the running total adder. This subsystem is a 4 to 1 multiplexer, which selects from Cn, Gnd, IP, and (2IP(i-1)-2n) as shown in Fig. 6. The STGC is controlled by logic outside the bit slice which has inputs: MLT1sb, SSRsign, start, phi2, and K or K+1. The signals K or K+1 are from the shift register counter which flags the multiplier to adjust the result positive.

Algorithm	Number of K bit adders	Number of addition phases /multiplier bit	No. of periods per M	No. of extra registers	Maximum bit rate	Comments
Simmons et.al.	1, (2K bits)	1	2K	0	~20kb/s	Slow, large, and complex.
A	3	Average 1.5	K	0	~35kb/s	Simplest control logic.
B	1 or 2	Average 1.5	K+1	1	~35kb/s	Simplest signal flow in bit-slice.
C	2	1	K+3	3	~42kb/s	Large with extra storage registers.
D	3	1	K+2	0	~40kb/s	Some control logic required outside bit-slice array.
E	4	2	K/2	0	~60kb/s	15% larger than algorithm D.

Table 2. Comparison of modulo multiplication (M) algorithms.

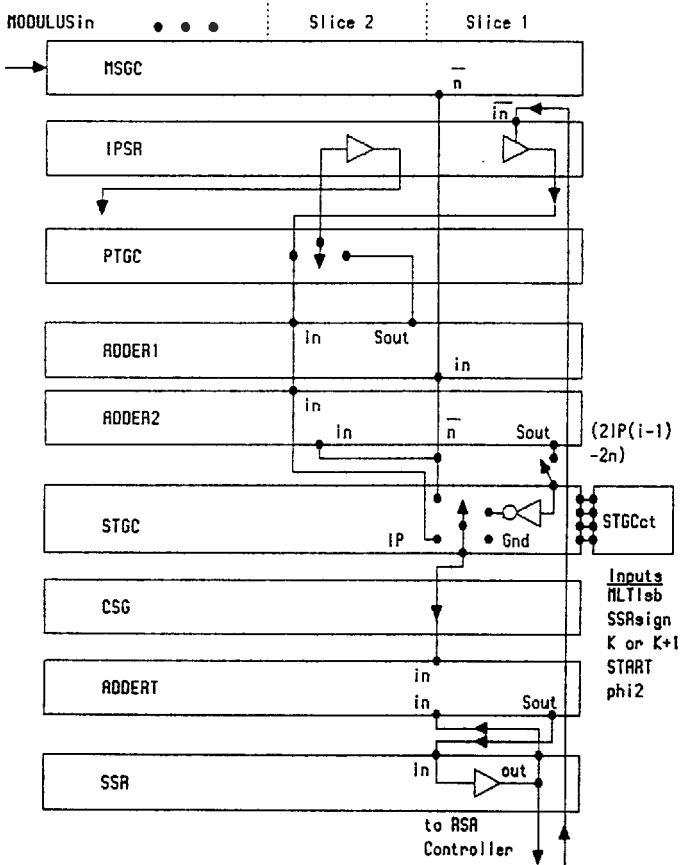


Fig. 6. A bit-slice architecture for algorithm D.

RSA architecture. Three slices of a new bit-slice RSA architecture which includes the implementation of modulo multiplication algorithm D described above is shown in block diagram form in Fig. 7. Modulo exponentiation is performed as a series of modulo multiplications. The message is repeatedly squared and modulo reduced and a running-product modulo multiplication is performed if the corresponding exponent bit is a one. Modulo multiplications are carried out by the subsystems in the upper half of the bit slice while the RSA control functions are implemented in the lower half. The same architecture can be used with other modulo multiplication algorithms. Input and output of data is synchronous and concurrent.

A general purpose register (STRSR) acts as a shifting or storage register with its function determined by control logic outside the bit slice as shown in Fig. 8. The use of this dual purpose subsystem considerably reduces the number of custom subsystems required. The circuitry is compact because control logic for the gates is outside the bit slice. This saves area because one set of control logic is used for all slices. Control logic delay is not a factor since the storage operations are not speed limiting. Standard cells would be suitable for the control logic outside the bit slice. For RSA operation, the message is shifted into both the square term storage

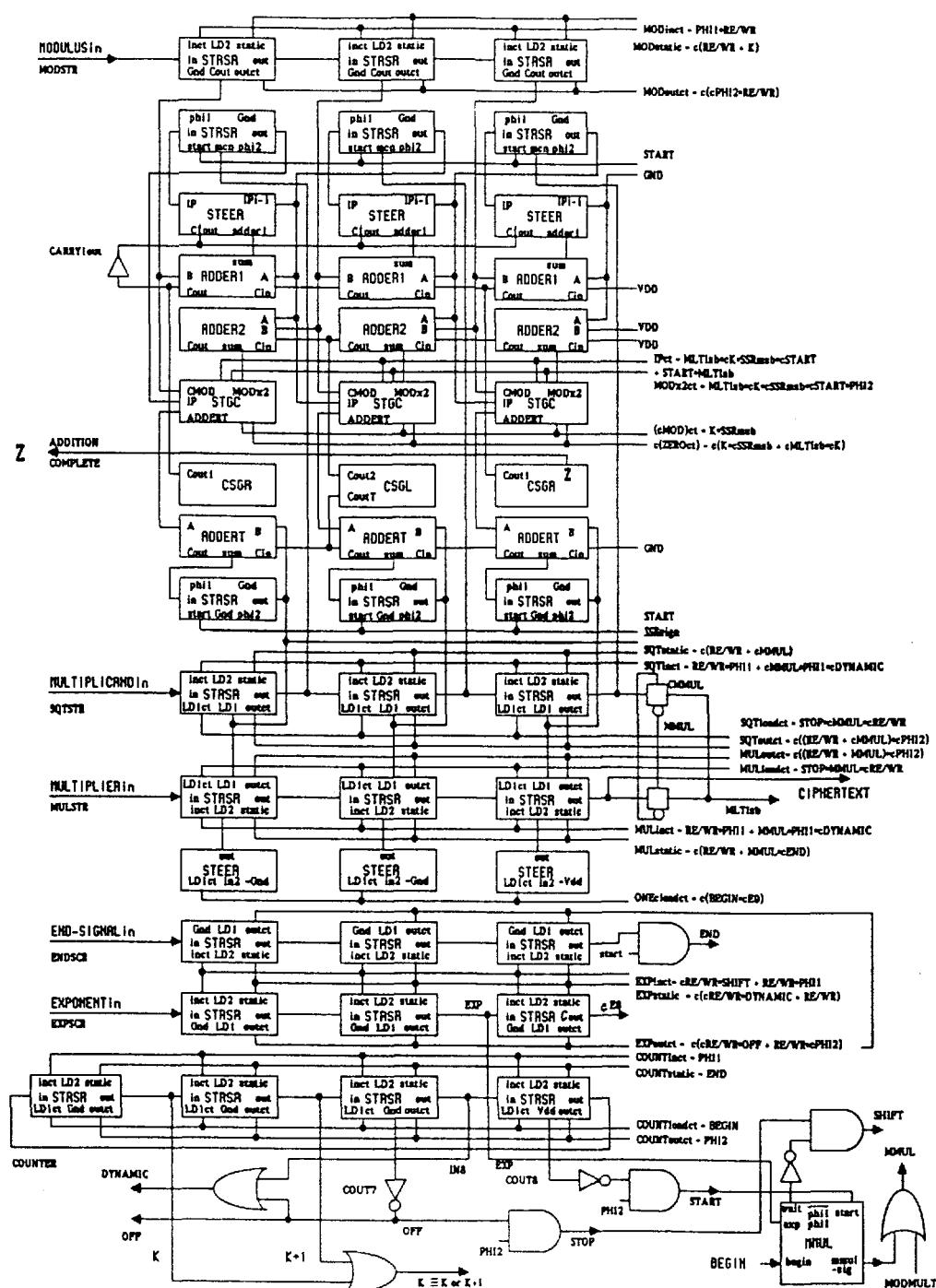


Fig. 7. Bit-slice architecture block diagram (3 slices).

register (SQTSTR) and the multiplication running total storage register (MULSTR). A single modulo multiplication can be performed by loading the multiplicand into SQTSTR and the multiplier into MULSTR.

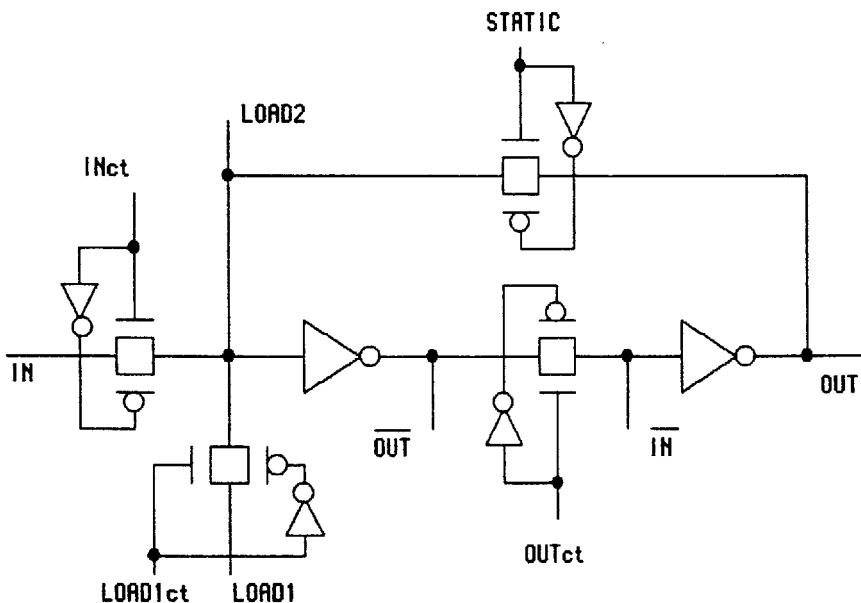


Fig. 8. A multiple function data register (STRSR).

Modulo multiplications are timed with a shift register counter in the bottom row of Fig. 7. The completion of an RSA encryption transformation is set by the END-SIGNAL which allows for exponents of different lengths. The end and exponent registers (ENDSCR and EXPSCR) shift after 1 or 2 modulo multiplications, depending on the exponent bit. Several external control signals are needed: BEGIN starts the encryption; MODMULT sets the chip for a single modulo multiplication; EXT sets the chip to external synchronous timing for data I/O or synchronous testing. The total number of pins required is about 12 depending on the actual application.

Asynchronous operation of the adders can be accommodated using a completion signal generator subsystem (CSG). Fig. 7 shows a pair of CSG subsystems, CSGleft and CSGright which detect all three carry outputs every second slice. For synchronous operation, the CSG subsystems are not required.

Asynchronous aspects

A self-timed adder. Several synchronous adders were considered such as carry lookahead, carry select, and the binary lookahead carry adder [18]. These adders had disadvantages such as high area, irregular layout, slower speed, or the difficulty of providing non-symmetric synchronous clock phases. Past approaches to self-timed adders have been speed independent or Muller circuits which use double rail logic. The disadvantages of this method are slower carry propagation and several times greater implementation area. Pulses have been used successfully to time asynchronous operations, such as asynchronous access of stored state registers [19]. When access is

requested, an edge detector / pulse generator circuit forms a pulse to time the operation.

A new pulse-timed adder which borrows ideas from Hayes [19], is shown in Fig. 9 in which carry propagations are detected in a precharged ripple adder. Carry outputs are reset to 0 during the precharge phase, so only propagations of 1 have to be detected. An edge detector / pulse generator circuit provides enough delay for the carry signal to propagate to the next pulse subsystem. A pulse subsystem is only half as large as a low area precharge adder slice. Pulses are combined to create the completion signal with a single active load pullup NOR gate.

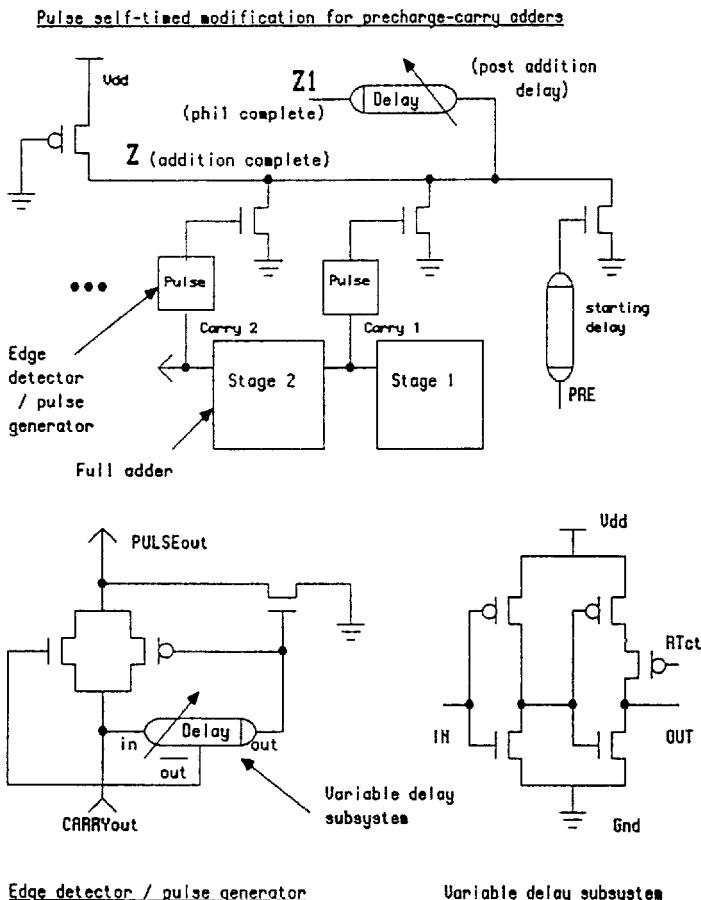


Fig. 9. Self-timed adder circuit details.

Several features of the adders make this scheme feasible. Overlap of the pulses prevents premature generation of the completion signal. The pulses are made several times wider than necessary since the resulting delay at the end of the additions is absorbed by subsequent RSA operations. The mean of the maximum number of consecutive carries in a 512 stage adder is only 8.2 with a variance of 3.3 (almost 60 times faster than

a ripple adder alone, on average!). The probability of less than 5 carries is negligible so a starting delay equivalent to 4 carries can be provided to overlap the first pulses. A pulse subsystem is not required every slice and all three adders calculating during the same phase can signal with the same NOR gate pulldown.

Optimization of the adder speed must be balanced against area considerations. A pulse subsystem every slice is the fastest on average. However too many pulldowns slow down the large NOR gate. The number of pulldowns can be reduced by grouping pulses with smaller NOR and NAND gates first. These variables were adjusted to achieve low area and a regular layout.

Clocking. In combining the modulo multiplication algorithm of Fig. 4 with the self-timed adder of Fig. 9 in the bit-slice architecture of Fig. 7, several control and timing considerations must be addressed.

The clock has to be capable of switching between asynchronous and synchronous timing to allow synchronous I/O and testing. Also, when the RSA encryption is finished, the clock should stop with the ciphertext safely in a storage register. These functions are implemented with random logic as shown in Fig. 10. A Muller C element is used to prevent the PHI2 clock signal from going low until the PHI1 clock phase has risen to prevent a race condition. This allows the PHI2 falltime to be set to the minimum value and only the risetime of the variable delay elements have to be adjustable.

Driver delays form a significant part of the clock period. Delays for generating some control signals cannot be avoided but the delay of the clock drivers can be largely prevented from adding to the clock period. Most of the falltime of clock phases does not contribute to the clock period because the clock phase widths can be externally adjusted. Also the non-overlap time can be externally minimized as shown in Fig. 10. In the 8 and 24 slice implementations, the inverted driver outputs, cPHI1 and cPHI2, were fed back to the clock controller rather than delayed versions of cPHI1sig and cPHI2sig. This guarantees that there is sufficient non-overlap time but it is not adjustable and results in an approximately 30% larger clock period.

For some parts of the circuit, considerable area would be required to generate completion signals logically. Delay elements were used instead with active load resistors to time these circuits. Active loads can provide sufficient delay in a small area and can be controlled by an external DC voltage (RTct in Fig. 9). The new data rate control scheme employs a single pin to control all delay elements. An intermediate DC voltage is first selected, say, 2.5 Volts. Then the gate aspect ratio of each active load is chosen to provide the expected circuit delay. During testing, the DC voltage is reduced to find the maximum operating rate (similar to finding the maximum clock rate of a synchronous chip). The accuracy and stability of the active loads can be improved by increasing the gate length and width while keeping the gate aspect ratio constant. This asynchronous timing method has the advantages of rate controllability, low area, elimination of global clock distribution, and allows different processes to be timed at their own rate. Lastly, it uses only 1 pin. Correct chip timing is ensured since the delay of each variable delay element can be increased arbitrarily.

Synchronization failure can occur when gating an asynchronous signal to a synchronous system. Only latching the END signal is prone to this type of failure. In an encryption environment, a host processor would periodically sample the END signal and there is a small probability that a metastable state would be detected. Increasing the settling time rapidly decreases the failure rate to an acceptable level [20]. The required settling time is negligible compared to the encryption time.

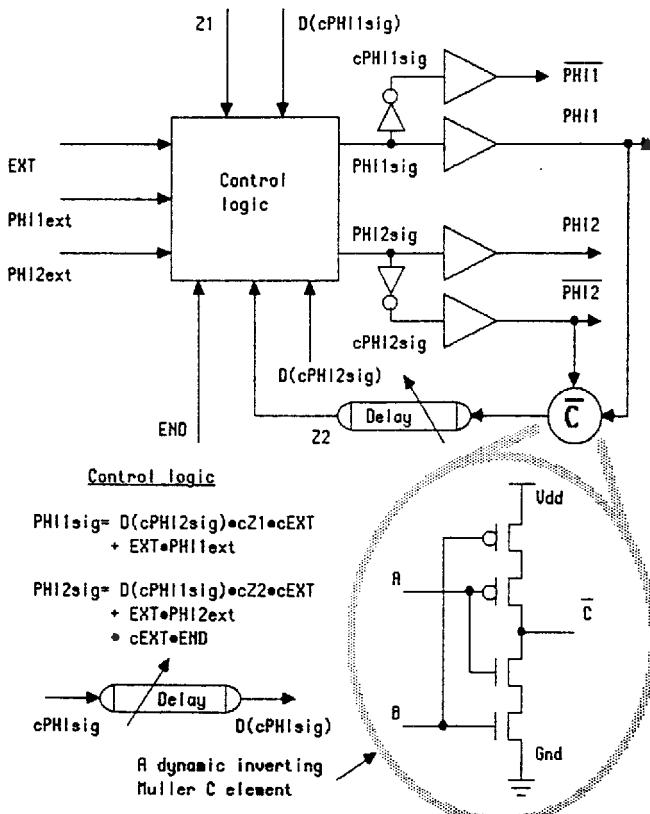


Fig. 10. Clock controller for the RSA chip with algorithm D and asynchronous adders.
The non-overlap time is adjustable.

Implementation

Output speed limits. Output driver current will often limit a synchronous clock rate while an asynchronous clock is not I/O limited since there is no I/O during the RSA transformation. The estimated average asynchronous clock rate in 2um CMOS is as high as 30Mhz. This advantage of asynchronous timing will become more pronounced as processes scale down further. Circuit speed scales down as A^2 [21], where A is the scaling ratio, if the power supply voltage is held constant. However, the driver speed scales down as $A/\ln(A)$ [22]. Asynchronous techniques could also be applied to any synchronous algorithm to allow the RSA encryption to proceed faster than the I/O speed.

The data rate of algorithm D. Algorithm D requires one clock phase for addition plus a shorter phase to set up the adder inputs. The throughput rate is effected by the on-chip communication delays which are hard to estimate accurately since they depend on the particular manufacturing process. Estimates based on SPICE simulations of signal propagation delays can be made. A calculation of the bit rate for algorithm D with the pulse-timed adder yields a rate of 40kbits/sec. Details of this calculation are provided in Appendix A. This corresponds to an average asynchronous clock rate of 30Mhz. A slower synchronous clock rate would be used for I/O, but a negligible number

of clock periods are required for I/O. At the expected clock rates, small variations in the circuit speed have a large effect on the throughput rate, but a conservative estimate of 30kbits/sec appears reasonable.

Asynchronous bit-slice implementation. Fig. 11 is a photomicrograph of a 32-bit prototype chip executed in 3um CMOS. Algorithm A has been used for modulo multiplication. A different architecture than that shown in Fig. 7 is employed in the synchronous implementation, which simplifies the control logic external to the slice at the expense of more custom registers. The bit slices run horizontally and are comprised of 14 subsystems which implement modulo multiplication, exponentiation and storage functions. Input and output data flow is serial which minimizes the total pin count. This chip has been tested and shown to correctly perform RSA encryption (or decryption) at a synchronous clock speed of 200kHz, which corresponds to a rate of 4kbits/sec. The synchronous, pre-charged adder delay per bit has been measured to be 8 ns in 5um CMOS from which a throughput rate of 1kbits/sec for 512-bit encryption is predicted for a 2 micron CMOS process. For 3um CMOS and 32-bit encryption, a rate of 5MHz and 100kbits/sec encryption is predicted. The low measured speed is difficult to explain since a 7-bit prototype in 5um CMOS was found to operate at 2MHz. More samples are being bonded for testing which may indicate if process parameter variations are involved.

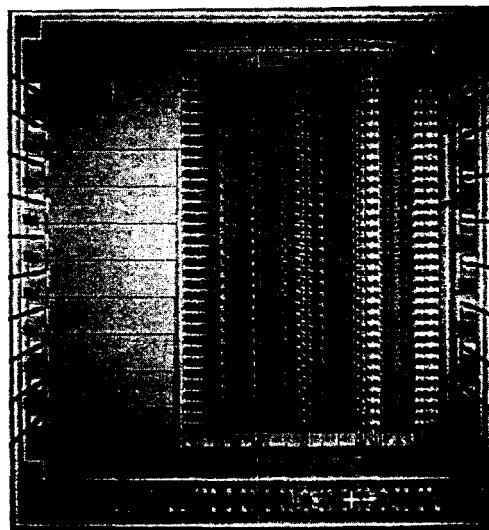


Fig. 11. Photomicrograph of synchronous 32-bit RSA prototype implemented in 3um CMOS.

Asynchronous implementation. Fig. 12 is a photomicrograph of a 24 slice implementation of an asynchronous RSA design based on algorithm D, the architecture described in Fig. 7 and the pulse-timed adder. The data analyser display for a 22 bit encryption is shown in Fig. 13. Input and output of data are overlapped, so both input and the previous output can be seen at the same time. Both inputs and outputs start least significant bit first.

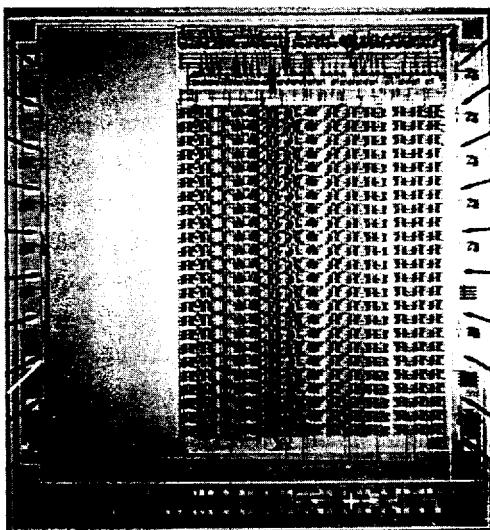


Fig. 12. Photomicrograph of asynchronous 22-bit RSA prototype implemented in 3um CMOS.

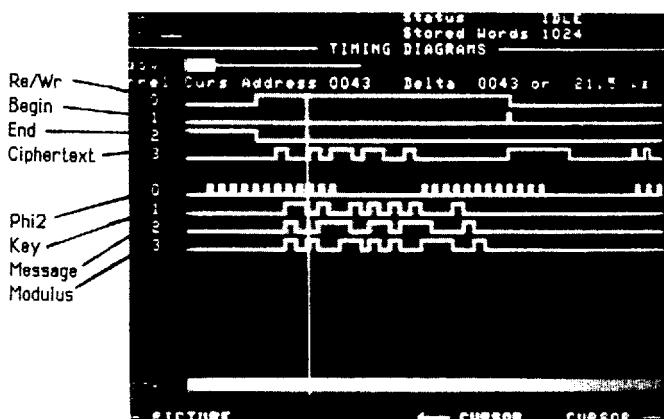


Fig. 13. Data analyzer display for a 22-bit computation: $584,932^{283,948} \text{ mod } (1,283,476) = 19,876$. Due to the slow sampling rate PHI2ext appears to stay at 0 sometimes. A single input set was cycled, so this ciphertext corresponds to this input set.

The average asynchronous clock rates of these designs provide a good indication of the accuracy of the speed extrapolations made in Appendix A. In 2um CMOS for 512-bit encryption, the estimated optimized throughput was 40kbits/sec with an average clock rate of 30MHz. In 5um CMOS for an 8 slice prototype, the average clock rate was found to be 3MHz and the encryption rate was 300kbits/sec, while in 3um CMOS for 24 slices the average clock rate was found to be 5MHz and the encryption rate 150kbits/sec. The estimated optimized average clock frequencies for these processes were 6MHz for 5um CMOS (8 slices) and 13MHz for 3um CMOS (24 slices). Additional samples are being bonded to determine if the slower than predicted clock rate is related to process variations. In any case, there are some further steps which can be taken to increase speed, including use of double metalization, so that it seems possible that the predicted performance can be attained.

Work in progress. Expansion of the asynchronous design to perform transforms involving many hundred bits is necessary to verify the speed advantages of the architecture and algorithms described here. The 64-bit chip presently being fabricated in 3um CMOS will help to verify the extrapolations which were made to predict the speed of a 512 bit design in 2um CMOS. A 128-bit version has also been designed and will be fabricated in the near future.

Significant further improvements in the throughput rate of RSA encryption are not likely to come from faster adders. With the asynchronous pulse adder, constant circuit delays take about twice as long as the three concurrent additions. The constant delays which result from signal propagation delays (excluding additions) are difficult to reduce in this style of architecture. Thus, a faster adder could only achieve about a 30% speed improvement at the most. Future improvements may be possible with new architectures.

Higher bit rates can be achieved by interconnecting chips in several patterns. One suggested architecture is a systolic arrangement of modulo multipliers [23]. This design cascades at least K modulo multipliers with a systolic data flow, where K is the number of bits. New systolic arrangements of asynchronous encryption units are faster and can be built with any number of encryption units. Binary tree input distribution, with token ring chip selection is the most efficient and achieves the same performance as a single encryption chip.

4. A multiplier for the finite field GF(2^m)

Arithmetic operations in the finite field GF(2^m) are quite different from ordinary integer arithmetic operations. Addition does not involve carries and is thus easier to perform than integer addition, but multiplication is still a fairly complex and difficult task. Most circuits proposed [9,24] are not suited for use in VLSI systems. They require excessive silicon area, complicated control schemes, complex wire routing, have nonmodular structures, or lack concurrency [12].

The systolic multiplier developed by Yeh, Reed and Truong [25] is suitable for VLSI implementation although it is only moderately compact and has a latency of 2m time units which may be undesirably long for some applications. The implementation of the Massey-Omura multiplier [26] is simpler than the systolic version and operates with a smaller latency, but is less modular and has a circuit structure and operating speed which is dependent on the size of the field.

The architecture to be described here uses an approach similar to the one outlined by Laws and Rushforth [27]. It is modular and therefore easily expanded, compact, and requires few control signals. The multiplication time and latency are m time units.

The algorithm

It is assumed that the reader has a basic knowledge of finite fields. If $A(x) = a_{m-1}x^{m-1} + \dots + a_1x + a_0$ and $B(x) = b_{m-1}x^{m-1} + \dots + b_1x + b_0$ are two elements of GF(2^m), then their product, $A(x)B(x)\text{mod } F(x)$, is $P(x) = p_{m-1}x^{m-1} + \dots + p_1x + p_0$, where $F(x) = f_{m-1}x^{m-1} + \dots + f_1x + 1$ is an irreducible polynomial.

The multiplication, $A(x)B(x)\text{mod}F(x)$, can be expanded by multiplying each term of $B(x)$ by $A(x)$:

$$\begin{aligned} P(x) &= A(x)B(x)\text{mod}F(x) \\ &= \{A(x)b_{m-1}x^{m-1}\text{mod}F(x) + \dots + A(x)b_1x\text{mod}F(x) \\ &\quad + A(x)b_0\text{mod}F(x)\} \text{ mod}F(x) \end{aligned}$$

The first term $A(x)b_{m-1}x^{m-1}\text{mod}F(x)$ is computed, followed by each successive term which is added to it and the sum reduced $\text{mod}F(x)$ until all the terms have been used.

If $A = [a_{m-1}, \dots, a_1, a_0]$ is the vector of coefficients of $A(x)$ and similarly for $B(x)$, $P(x)$ and $F(x)$, then this algorithm can be represented by the flowchart in Fig. 14. Element A is added to the intermediate product, P , whenever the current bit of B , b_i , is a 1. F is added whenever the most significant bit (MSB) of P is 1, which indicates that modulo reduction is necessary. These two decisions are carried out simultaneously. If the field is of degree m , then m steps are needed to complete a multiplication.

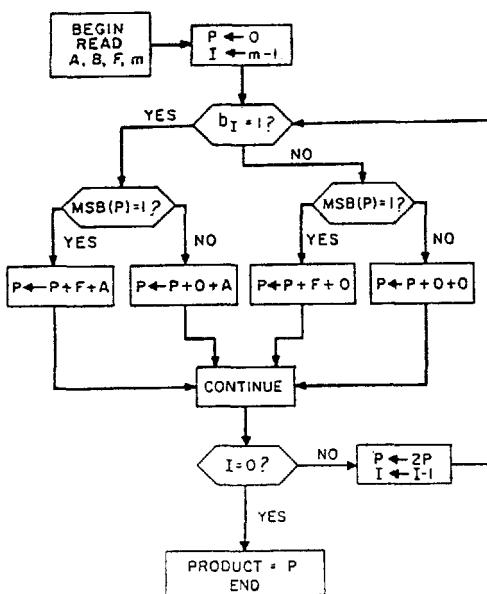


Fig. 14. Flowchart of $\text{GF}(2^m)$ multiplication algorithm.

Architecture

The multiplier architecture is shown in Fig. 15 for the field $\text{GF}(2^4)$. Registers a_i , f_i and p_i hold A , F and the intermediate product P , respectively. The MSB of F , which is always 1, is actually not used in the

calculation. The state of b_i , latched with a flip-flop, and the MSB of P constitute the two primary control signals. The left shift is performed by loading the output of stage L_i into the product register P_{i+1} of the next stage L_{i+1} . The final product is transferred to the output shift register (OSR) and shifted out serially once the multiplication is complete. Note that the worst case delay path from the f_i register to the OSR is independent of the multiplier size. The number of L_i and register stages is equal to the degree of the field, in this case four.

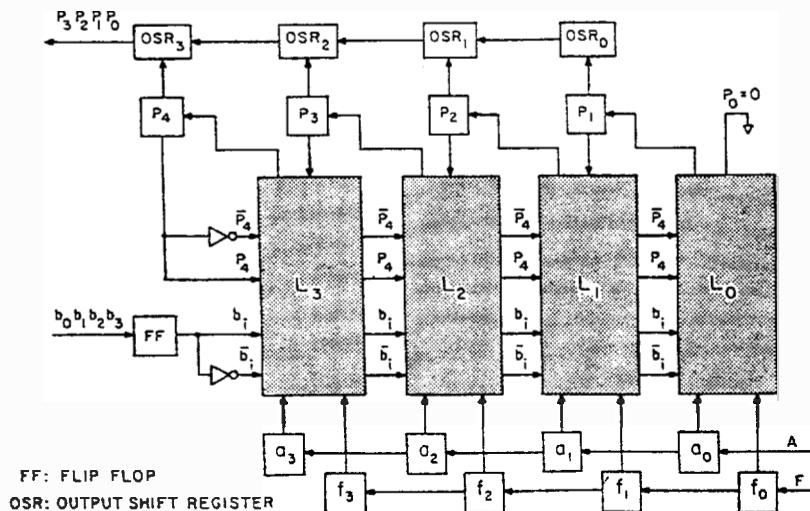


Fig. 15. The multiplier for $GF(2^4)$

Each stage L_i contains one 3-input modulo, two transmission gates and two NMOS transistors, as detailed in Fig. 16. The transmission gates and transistors are configured to perform the AND function (MSB(P) AND f_i , and b_i AND a_i). For example, if $b_i = 1$ then a_i is passed to the adder; otherwise that adder input line is grounded (set to 0).

Implementation and testing

A multiplier for $GF(2^8)$ was implemented using a combination of static and dynamic logic and fabricated in 5 micron CMOS. It was found to be fully functional, capable of operating at speeds up to 7 Mbits/sec [28]. As SPICE simulations predicted, the data rate was limited by the speed of the output pad drivers, not the worst case delay path on the chip. Optimization of the pad drivers should improve the speed by about 30–40%.

Each of the eight slices occupied an area of 185 microns by 1459 microns, of which 10% was allocated to test structures. Subsequent chips have been modified to incorporate a more structured design for testability approach, the Scan Path technique [29]. In addition, the pad drivers and adders were replaced with faster versions. The new slice occupies 23% less area.

This enhanced 8-bit version was submitted for fabrication in 3 micron CMOS in September 1986, along with a 128-bit multiplier. A 512-bit chip, with a total area (multiplier and I/O pads) of 6912 microns by 6980 microns, will be submitted at the end of 1986. From these two larger designs it is hoped that more information about the performance of the algorithm will be obtained.

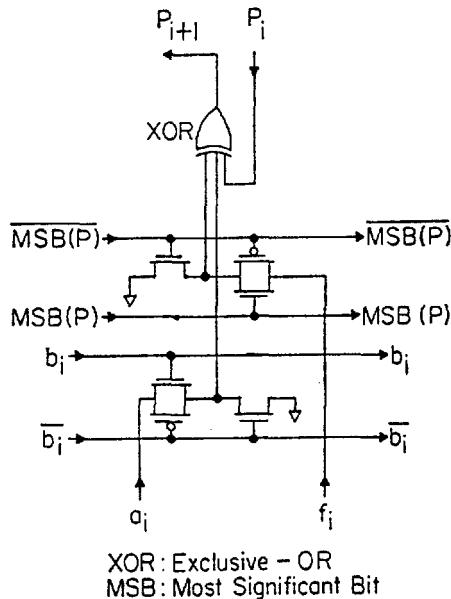


Fig. 16. The circuit for each block L_i shown in Fig. 15.

5. Conclusion

RSA architecture. A 22-bit, 3um CMOS prototype of an asynchronous RSA chip has been fabricated and found to function correctly with a throughput rate of 150kbits/sec. A conservative estimate for the 512 bit encryption rate in 2um CMOS is 30kbits/sec with optimization of the present design and 40kbits/sec with algorithm E. The asynchronous clock rate during encryption is not I/O limited nor is it limited by the clock rate in other components.

Concurrent modulo multiplication algorithms provide the most efficient implementations known for RSA encryption. Multi-adder algorithms such as algorithms D and E are efficiently implemented with the asynchronous pulse-timed adder. Minimization of constant circuit delays is important since they several times larger than the addition time in a clock period.

GF(2^m) multiplier. To evaluate the performance of the finite field multiplication algorithm, an 8-bit prototype has been fabricated in 5um CMOS and tested. It was found to operate correctly for data rates up to 7Mbits/sec. A new 8-bit version with faster adders and pad drivers, and a more structured approach to testing is currently being fabricated in 3um CMOS along with a 128-bit multiplier. A 512-bit chip will be implemented at the end of 1986.

and should provide some useful information about large VLSI multipliers.

6. Appendix A: Calculation of the RSA throughput rate with algorithm D

Constant on-chip communication delays in Sum CMOS:

Non-overlap time = $2 \times 10\text{ns}$ (if adjustable)

Signal flow after addition: drive carryout line plus signal flow within bit slice = 30ns

Clock transition time = $2 \times 10\text{ns}$ (crossing threshold only)

phi2: control generation (15ns) plus driving of control lines plus • signal flow within bit slice = 50ns

Total = 120ns

1 clock period in a Sum process = $\text{TD5} = (\text{Nc} + \text{L})\text{Tp} + 120\text{ns} = 201.2\text{ns}$

where Nc = average number of carries for an average of 2.5 adders of 500 bits each = 9.6

L = No. of slices separating pulse subsystems of adders = 2

and Tp = carry propagation speed in a Sum process = $7\text{ns}/\text{slice}$

1 clock period in a Zum process = $\text{TD2} = \text{TD5} \bullet (2/5)^2 = 32.2\text{ns}$

RSA transform execution time = $\text{Texe} = (\text{the number of modulo multiplications})(\text{the number of periods per multiplication})(\text{the length of a period})$

$$\text{Texe} = (1.5 \bullet K) \bullet (K+2) \bullet \text{TD2} = .0127 \text{ sec}$$

where K = number of bits in exponent and modulus = 512

Bit rate = $K/\text{Texe} = 40 \text{ kbits/sec}$

7. Acknowledgements

The research reported here was supported in part by strategic grants 60893, 60894 and 61364 and Operating Grants from the Natural Sciences and Engineering Research Council of Canada. VLSI design and testing equipment was provided under the loan program of the Canadian Microelectronics Corp. (CMC). Chip fabrication was carried out by Northern Telecom Electronics Ltd. under the fabrication program of the CMC.

8. References

- [1] W. Diffie and M. Hellman, "New Directions in Cryptography", IEEE Trans. Info. Theory, Vol. IT-22 (6), pp. 644-659, Nov. 1976.
- [2] R.L. Rivest, A. Shamir and L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems", Comm. of the ACM, Vol. 21, No. 2, pp 120-126, Feb. 1978.
- [3] D. Denning, "Cryptography and Data Security", Reading, Mass.: Addison-Wesley Publ. Co., 1982.
- [4] R.L. Rivest, "RSA Chips (Past/Present/Future)", Advances in Cryptology, Proc. of EUROCRYPT 84, pp. 159-165, Springer-Verlag, Berlin, 1985.
- [5] M. Kochanski, "Developing an RSA Chip", Proc. of CRYPTO 85, Santa Barbara, CA., Aug. 1985.
- [6] CYLINK, "Advance Data Sheet: CY1024 Key Management Processor", CYLINK, 920 West Fremont Ave., Sunnyvale, California 94087, 1986.
- [7] E.F. Brickell, "A Fast Modular Multiplication Algorithm with Application to Two Key Cryptography", Proceedings of CRYPTO 82, Santa Barbara, California, pp. 51-60, August 1982.

- [8] G.A. Orton, L.E. Peppard, and S.E. Tavares, "A Fast Asynchronous RSA Chip", IEEE Custom Integrated Circuits Conference, Rochester, N.Y., pp. 439-443, May 12-15, 1986.
- [9] W.W. Peterson and E.J. Weldon, "Error-Correcting Codes", Cambridge, MA: MIT Press, 1972.
- [10] A.M. Odlyzko, "Discrete Logarithms in Finite Fields and Their Cryptographic Significance", Advances in Cryptology, Proc. of EUROCRYPT 84, pp. 225-314, Springer-Verlag, Berlin, 1985.
- [11] I.F. Blake, R. Fuji-Hara, R. Mullin and S. Vanstone, "Computing Logarithms in Finite-Fields of Characteristic Two", SIAM J. Alg. Discr. Methods, Vol. 5, pp. 276-285, 1984.
- [12] P.A. Scott, S.E. Tavares and L.E. Peppard, "A Fast VLSI Multiplier for $GF(2^m)$ ", IEEE Journal on Selected Areas in Comm., Vol. SAC-4, pp. 62-66, January 1986.
- [13] G.R. Blakely, "A Computer Algorithm for Calculating the Product AB Modulo M", IEEE Trans. Computers, Vol. C-32, pp. 497-500, May 1983.
- [14] R.L. Rivest, "A Description of a Single-Chip Implementation of the RSA Cipher", Lambda (Fourth Quarter 1980) pp. 14-18.
- [15] D. Simmons and S.E. Tavares, "An NMOS Implementation of a Large Number Multiplier for Data Encryption Systems", Proc. 1983 Custom Integrated Circuits Conf., Rochester, N.Y., pp. 262-266, May 1983.
- [16] M.P. Roy, L.E. Peppard and S.E. Tavares, "A CMOS Bit-Slice Implementation of the RSA Public-Key Encryption Algorithm", 1985 Canadian Conference on Very Large Scale Integration, Toronto, Canada, pp. 52-56, November 1985.
- [17] S. Weser and A. Peterson, "Real-time Processing Gains Ground with Fast Digital Multiplier", Electronics, pp. 93-99, September 29, 1977.
- [18] N. Weste and K. Eshraghian, "The Principles of CMOS VLSI Design: A Systems Perspective", Addison-Wesley, 1985.
- [19] A.B. Hayes, "Self-Timed IC Design with PPL's", Third Caltech Conference on VLSI, Computer Science Press, Inc., Rockville, Maryland, 1983, pp. 257-274.
- [20] T.J. Chaney and F.U. Rosenberger, "Characterization and Scaling Of MOS Flip Flop Performance in Synchronizer Applications", Proceedings of the First Caltech Conference on VLSI, 1979.
- [21] C.L. Seitz, "Self-Timed VLSI Systems", Proceedings of the First Caltech Conference on VLSI, pp. 345-354, January 1979.
- [22] D.R. Brown, "Optimization of On-Chip Input/Output Interfacing Circuitry for VLSI Systems", M.Sc. Thesis, Department of Electrical Engineering, Queen's University, July 1985.
- [23] K. Culik II, Jürgensen, K. Mak, "Systolic Tree Architecture for some Standard Functions", Report 140, Dep. of Computer Science, University of Western Ontario.
- [24] T.C. Bartee and D.I. Schneider, "Computation with Finite Fields", Inform. and Control 6, pp. 79-98, 1963.
- [25] C.S. Yeh, I.S. Reed, and T.K. Truong, "Systolic Multipliers for Finite Fields $GF(2^m)$ ", IEEE Trans. Comput., vol. C-33, pp. 357-360, April 1984.
- [26] C.C. Wang, T.K. Truong, H.M. Shao, L.J. Deutsch, J.K. Omura, and I.S. Reed, "VLSI Architectures for Computing Multiplications and Inverses in $GF(2^m)$ ", IEEE Trans. Comput., vol. C-34, pp. 709-717, Aug. 1985.

- [27] B.A. Laws, Jr. and C.K. Rushforth, "A Cellular-Array Multiplier for GF(2^m)", IEEE Trans. Comput., vol. C-20, pp. 1573-1578, Dec. 1971.
- [28] G.A. Orton, M.P. Roy, P.A. Scott, L.E. Peppard, and S.E. Tavares, "New Results in Mapping Data Encryption Algorithms into VLSI", presented at the Fourth Int. Workshop on VLSI in Comm., Ottawa, Ont., June 1986.
- [29] T.W. Williams and K.P. Parker, "Design for Testability - a Survey", Proc. IEEE, vol. 71, pp. 98-112, Jan. 1983.

Architectures for exponentiation in GF(2^n)

T.Beth^{*}, B.M.Cook^{**}, D.Gollmann^{*}

Abstract.

We investigate different data structures in GF(2^n) and their correspondence to silicon architectures to examine possible hardware implementations of the Diffie-Hellman key exchange system.

1. Introduction.

We want to analyse possible MOS implementations of the Diffie-Hellman key exchange system. This system is based on exponentiation in GF(2^n). Exponentiation will be performed using a square and multiply algorithm. In this paper we will point out how different algebraic structures in the representation of GF(2^n) give rise to different architectures for the MOS implementation of this algorithm. We will use examples to demonstrate the relative merits of the different architectures while giving only references for their mathematical foundations. (A survey of the mathematical background is given in [1]).

* University of Karlsruhe, Faculty of Computer Science

** University of Surrey, Departm. of Electronic and Electrical Engineering

2. Multiplication.

2.1. Polynomial basis multiplier

This architecture is based on the standard way of representing elements in $GF(2^n)$ by polynomials. Multiplication is performed by convolution and reduction modulo an irreducible polynomial $p(x)$ of degree n (see e.g. [2],[3]). Fig.1 gives a serial input architecture for a polynomial basis multiplier.

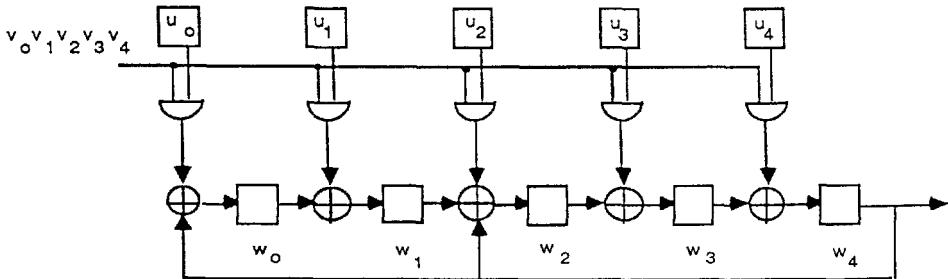


Fig.1. A polynomial basis multiplier for $GF(2^5)$ using $p(x)=x^5+x^2+1$.

2.2. Normal basis multiplier (Massey-Omura)

In a normal basis representation squaring is a single shift in a cyclic shift register. A multiplication algorithm can be found e.g. in [4]. Fig.2 gives a parallel input architecture for a normal basis multiplier ([4]), Fig.3 a serial input architecture ([5]).

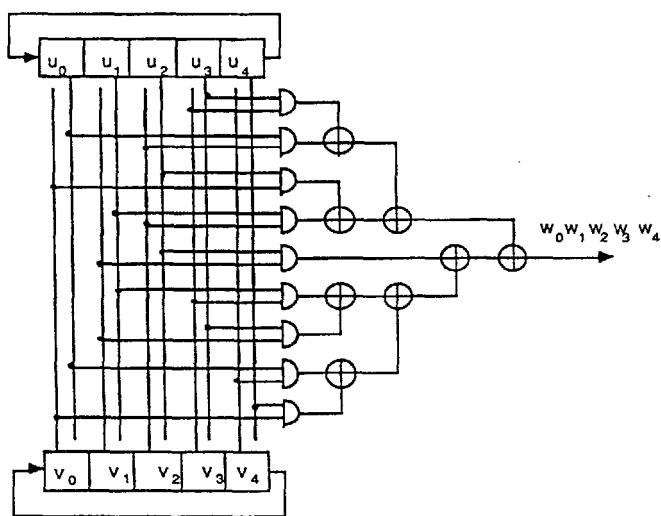


Fig.2. A parallel input normal basis multiplier for $\text{GF}(2^5)$ using
 $p(x)=x^5+x^4+x^2+x+1$

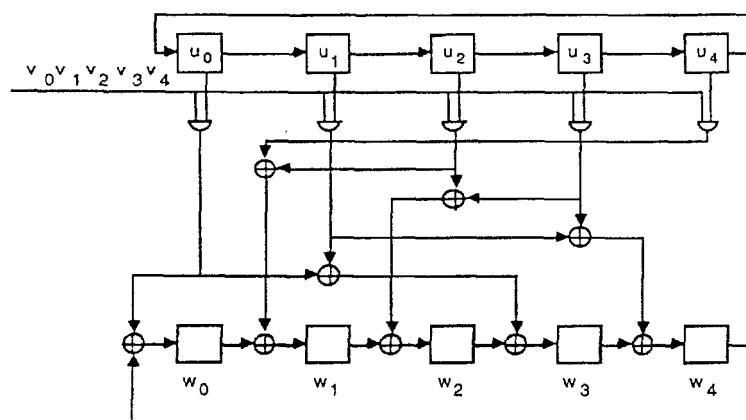


Fig.3. A serial input normal basis multiplier for $\text{GF}(2^5)$ using
 $p(x)=x^5+x^4+x^2+x+1$

2.3. Dual basis multiplier.

Dual basis multipliers for multiplication with a constant were introduced in [6]. A general dual basis multiplier is discussed in [5], in this algorithm one of the factors and the result are represented in the dual basis and the other factor in the polynomial basis. Fig.4 and Fig.5 give architectures for a parallel input and a serial input dual basis multiplier.

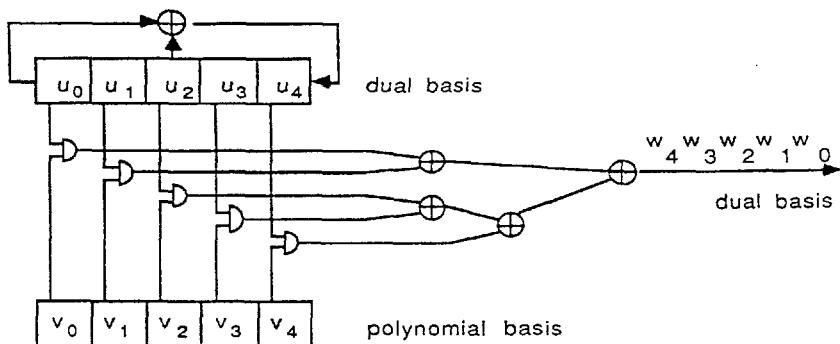


Fig.4. A parallel input dual basis multiplier for $GF(2^5)$ using $p(x)=x^5+x^2+1$.

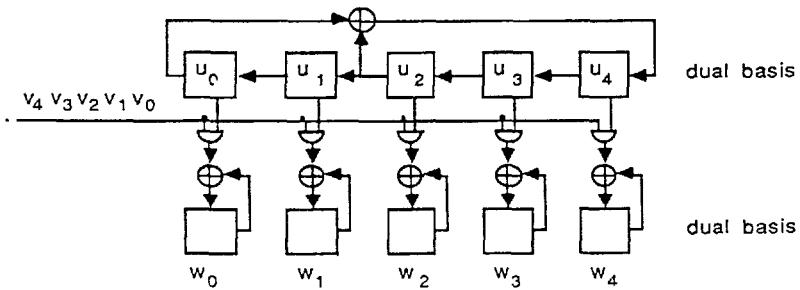


Fig.5. A serial input dual basis multiplier for $GF(2^5)$ using $p(x)=x^5+x^2+1$.

2.4. Evaluation and comparison.

The proposed polynomial basis multiplier performs a multiplication in n steps. The area required grows linearly in n . The architecture does not use straightforward shift registers. This disadvantage for standard hardware implementations had been a reason to look for new architectures. Normal basis and dual basis multipliers can be built from standard shift registers (Fig.2 and Fig.4). Both perform a multiplication in n steps. In the parallel input designs there is an additional delay to the XOR-tree. It may be noted that the serial input designs avoid the XOR-tree and use 'non-standard' reduction registers similar to the polynomial basis multiplier.

The size of a normal basis multiplier is mainly determined by the size of the PLA. In general it requires $O(n^2)$ AND-gates [4]. In MOS implementations additional problems occur due to the number of crossing wires in the PLA. The regular PLA of the dual basis multiplier is more convenient for MOS implementations. In the parallel input design area requirement however is not linear in n when we consider the wiring to the XOR-tree.

We have already pointed out that the serial input design is not too different from the polynomial basis multiplier. In a full custom MOS design it is always possible to define basic cells and bit slices tailored towards a given architecture. In such an environment the above argument against the polynomial basis multiplier is no longer valid. We conclude that the polynomial basis architecture is superior to the other architectures with regard to a full custom VLSI implementation.

3. Squaring.

The normal basis representation is obviously most appropriate for squaring. Squaring in the polynomial basis could be done by a multiplier. For a particular choice of the reduction polynomial $p(x)$ we can find a special architecture for the squarer. Take an irreducible trinomial $p(x) = x^n + x^k + 1$, n odd, k even, $k < n/2$. Observe for some field element u

$$u^2 = \left(\sum_{i=0}^{n-1} u_i x^i \right)^2 = \Sigma_1 + \Sigma_2 + \Sigma_3$$

with

$$\Sigma_1 = \sum_{i=0}^{(n-1)/2} u_i x^{2i},$$

$$\Sigma_2 = \sum_{i=(n+1)/2}^{n-1} u_i x^{2i-n},$$

$$\Sigma_3 = \sum_{i=(n+1)/2}^{n-1-k/2} u_i x^{2i-n+k} + \sum_{i=n-k/2}^{n-1} u_i (x^{2i-2n+k} + x^{2i-2n+2k}).$$

Σ_1 corresponds to putting coefficients $u_0, \dots, u_{(n-1)/2}$ into the even positions of a register,

Σ_2 corresponds to putting the coefficients $u_{(n+1)/2}, \dots, u_{n-1}$ into the odd positions of this register,

Σ_3 corresponds to shifting the coefficients $u_{(n+1)/2}, \dots, u_{n-1}$ by k steps before adding them into the register (again in odd positions). The top $k/2$ coefficients are added into even positions starting from position 0 and starting from position k .

Fig.6 gives the architecture of such a squarer. It computes the square of an element u in $n+k/2$ shifts. With a minor modification of the design this can be achieved in $(n+k+1)/2$ shifts.

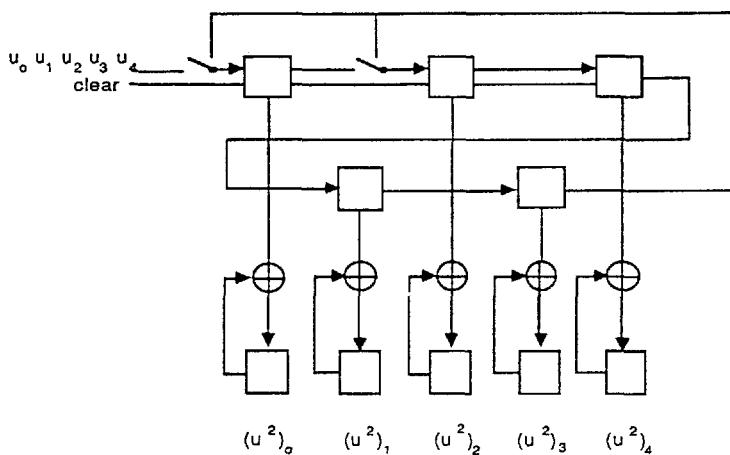


Fig.6. Squarer for $\text{GF}(2^5)$ using $p(x)=x^5+x^2+1$.

4. Square and multiply.

Squaring and multiplication can be performed in parallel if square and multiply starts at the least significant bit of the exponent ("right to left"). Thus the squarer of Fig.6 can be combined with either the polynomial basis multiplier or the dual basis multiplier. Fig.7 gives the architecture for the latter case. Note that we do not need an extra register to store intermediate results. An exponentiation is performed in $n(n+k/2)$ steps.

(n^2 steps with the modified squarer). If a serial input multiplier is used we arrive at a pipeline architecture where we can load the next inputs while shifting out the result.

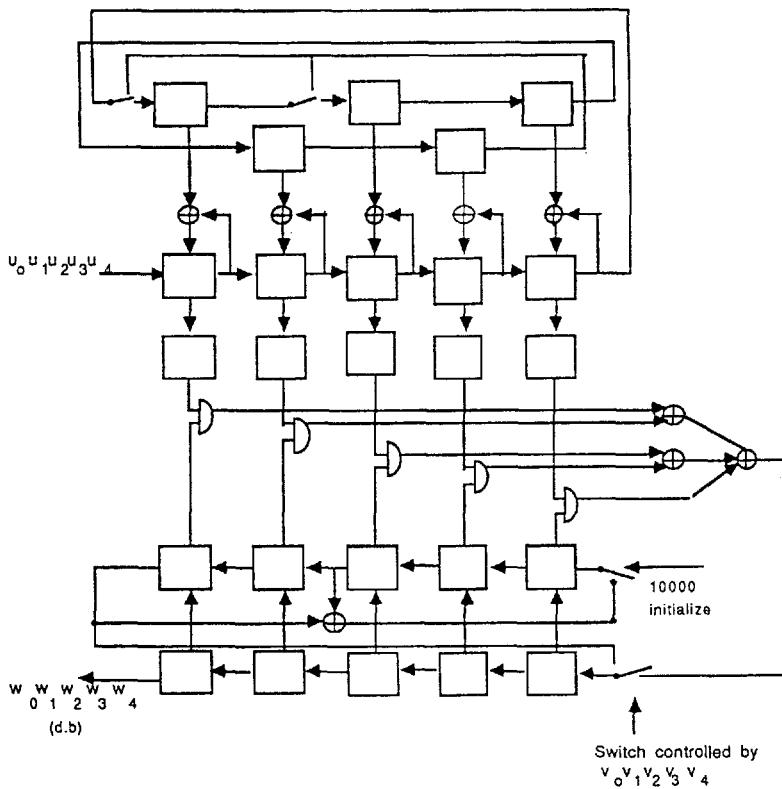


Fig.7. Exponentiator for $GF(2^5)$ using a dual basis multiplier.

If exponentiation starts at the most significant bit of the exponent ('left to right') multiplication and squaring have to be performed serially. In the polynomial basis or dual basis architectures we could use the multiplier alternatively for multiplication and squaring. In this case we need an extra register for storing the base of the exponentiation. Let m denote the number of 1's in the exponent. Both implementations require $(n+m)n$ steps for an exponentiation. In a normal basis architecture squaring can be done in a single additional step. Thus we need $n+mn$ steps for one exponentiation.

5. Conclusion.

Normal basis exponentiators need less steps than polynomial basis or dual basis exponentiators though the average number of steps is still of order $O(n^2)$. They are however not very well suited to VLSI implementations especially when area constraints are considered. Dual basis exponentiators show an advantage against polynomial basis exponentiators if TTL or standard cell technology is employed. If full custom MOS design is available the use of polynomial basis exponentiators seems to be the best choice .When both area and time requirements are demanding, a bit-slice architecture based on principles of Algebraic Algorithm Engineering shows an improved behaviour, particularly as unusual geometric configurations can be generated.

Literature.

- [1] T.Beth, On the Arithmetics of Galoisfields and the Like,
Proc. AAECC3, LNCS 229, Springer, 1986
- [2] F.J.MacWilliams, N.J.A.Sloane, The Theory of Error-Correcting Codes,
North-Holland, New York, 1977
- [3] B.A.Laws, C.K.Rushforth, A cellular-array multiplier for $GF(2^m)$,
IEEE Trans.Comput., vol.C-20, pp.1573-1578, 1971
- [4] C.C.Wang et al., VLSI Architectures for Computing Multiplications and Inverses in $GF(2^m)$,
IEEE Trans.Comput., vol.C-34, pp.709-717, 1985
- [5] W.Fumy, Über orthogonale Transformationen und fehlerkorrigierende Codes,
Ph.D.Thesis, Erlangen, 1986
- [6] E.R.Berlekamp, Bit-Serial Reed-Solomon Encoders,
IEEE Trans.Inf. Theory, vol. IT-28, pp.869-874, 1982

IMPLEMENTING THE
RIVEST SHAMIR AND ADLEMAN
PUBLIC KEY ENCRYPTION ALGORITHM
ON A
STANDARD DIGITAL SIGNAL PROCESSOR

Paul Barrett, MSc (Oxon)
COMPUTER SECURITY LTD
August 1986

ABSTRACT

A description of the techniques employed at Oxford University to obtain a high speed implementation of the RSA encryption algorithm on an "off-the-shelf" digital signal processing chip. Using these techniques a two and a half second (average) encrypt time (for 512 bit exponent and modulus) was achieved on a first generation DSP (The Texas Instruments TMS 32010) and times below one second are achievable on second generation parts. Furthermore the techniques of algorithm development employed lead to a provably correct implementation.

WHY DSP?

At the time we started work we considered several implementation options:

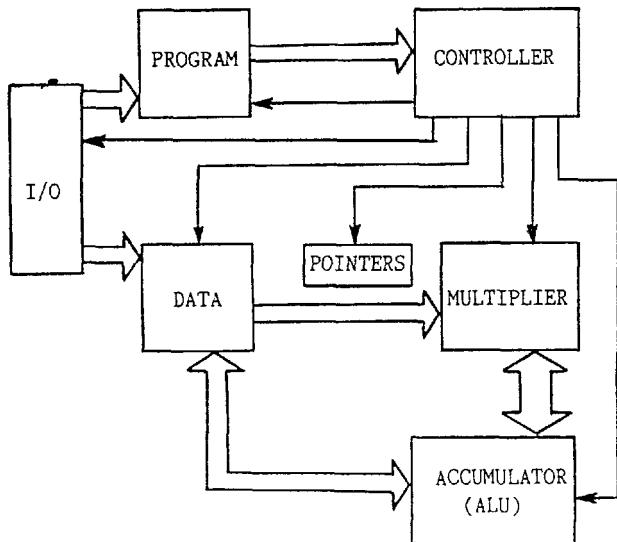
1. The first and most available option was an eight bit micro-processor - best estimates of 512 bits in 4 minutes (ie. 2 bits per second) did not seem very promising.
2. A 16 bit micro-processor - might make it in 50 seconds - but that's still too slow.
3. Discrete logic - was going to be extremely complex and messy.
4. A bit slice system would be very expensive to develop and implement.
5. And although a custom/semi-custom chip would be cheap to manufacture, it would be expensive to develop and would be too inflexible to allow commitment to the high volumes necessary to make this approach economically viable.

One thing we did know about implementing the RSA algorithm is that it involved lots of multiplication and so we decided to see if we could utilise a dedicated hardware multiplier/accumulator or MAC.

6. A MAC taking 100 ns for a 16×16 multiply was available and looked very promising. However, we quickly realised that we needed some fairly specialised hardware to drive it and feed it with data. Certainly no ordinary micro-processor would be able to keep up with the MAC's performance.

Just as we were beginning to despair the answer came to us courtesy of Texas Instruments who announced a new type of chip : the Digital Signal Processor or DSP.

DIAGRAM ONE - DSP ARCHITECTURE



7. The DSP - is a MAC and a fast microprocessor on a single chip which seemed to be the ideal combination..... The first one available was the TMS320 which has a 200ns cycle time for most instructions including multiply. Our early performance estimates suggested that with this chip five seconds for a 512 bit exponentiation should be fairly easily achievable.

THE IMPLEMENTATION

Having decided to use a DSP we have to develop a program for it. The first problem is that there are no suitable DSP compilers available and, although we might expect to eventually have to tune the assembler code to take full advantage of the DSP architecture and optimise performance, assembler is no good as a design language. Furthermore, our choice of implementation technique must take into consideration the nature of the application and in particular the requirement for integrity. With this in mind we chose to use the program development and validation techniques expounded by Prof. David Gries of Cornell University. The notation used is a combination of predicate logic and the "guarded command" form of computation guru Edsger Dijkstra.

THE ALGORITHM

In our notation the RSA algorithm can be specified in terms of pre- and post- conditions thus:

```
spec fastexp.0 (in: A,E,M; out: c);
  { pre: 0 ≤ A < M & 0 ≤ E }
  { post: c = AE mod M }
endspec
```

Where the pre conditions require that: the input data A is in the range 0 to M, the modulus minus one and the exponent E is positive; and post: the output data c equals A to the power E modulo M.

The basic algorithm we will work with to satisfy these conditions is Knuth's 'square and multiply' exponentiation method with modulo reduction incorporated. Thus:

```
proc fastexp.1 (in: A,E,M; out: c);
  { pre: 0 ≤ A < M & 0 ≤ E }
  a, e, c := A,E,1 ;
```

```

{ inv: c * ae mod M = AE mod M }
{ bound: t = 2 * log2e + 1 }
do e ≠ 0 & e mod 2 = 0 →
    e,a := e div 2, a * a mod M
    ♦ e mod 2 ≠ 0 →
        e,c := e-1, c * a mod M      od
{ post: c = AE mod M }

endproc

```

Notice that after initialisation of the variables the executable portion of this fastexp has been reduced to a single loop command albeit with two branches. Writing the algorithm in this very concise form which may not at first seem natural, allows us to prove its correctness more easily at a later stage.

Obviously this basic algorithm will need to be written in a substantially different form before our target DSP can execute it and in order to arrive at an assembler code version we go through a process of step-wise refinement. At each step of refinement the algorithm is re-written in a form which can be proven to be equivalent to its predecessor. In the case of our RSA algorithm most of the refinement is necessary in order to be able to represent and operate on the several hundred bit long integers within the constraints of a 16 bit architecture; the implementation of conditions, loops and other program constraints being fairly straightforward on the micro-processor-like DSP.

In order to keep our top level program simple and well structured we introduce two procedures (subroutines) which we call 'longmult' and 'longmod' to handle respectively the long integer multiplication and modulo reduction.

Here is the specification of these procedures, once again using the pre/post condition form:

```

spec longmult.0 (in: u,v; out: w);
{ pre: 0 ≤ u,v < bn }

```

```

{ post: w = u * v }

endspec

spec longmod.0 (in: w,m; out: v);
    { pre: 0 ≤ w < m2 }
    { post: v = w mod m }

endspec

```

THE HEART OF THE ALGORITHM

These two procedures really are the heart of the algorithm; and the key to performance is going to be their design. First let us consider what algorithm to use for long multiplication. The problem we have is similar to one we learned to solve at school. There we knew, from a memorised table, how to multiply up to 12 times 12 but faced with a larger multiplication (and assuming that we all went to school before the advent of the pocket calculator) we used a paper and pencil algorithm which went something like this (referring to diagram two): 6 times 2 is 12, 2 down carry 1, 2 times 2 is 4 plus one is 5 and so on repeating for each row, shifting one column left each time and finishing with a final addition sum. This is a fairly convenient method of hand calculation but how efficient is it?

Taking the general case of an n by n digit multiply - for each row we have to do n multiplications, $2n$ fetches, $n + 1$ stores and, n carry and add operations. Plus the final additions which require n^2 fetches and adds plus carries etc. Assuming all operations are equivalent to execute that makes in the order of $6n^2$ instructions.

Let's try it another way using the same principle but working in columns not rows and saving all the carries till we sum each column.

DIAGRAM TWO - LONG MULTIPLICATION AT SCHOOL

	3	0	7	8	2	6	
	4	1	5	1	3	2	
							x
		6	1	5	6	5	2
		2	3	4	7	8	-
		9	2	3	-	-	-
		0	7	8	2	6	-
		3	1	3	0	-	-
1	5	3	9	1	-	-	-
3	0	7	8	2	6	-	-
1	3	0	4	-	-	-	-
1	2	7	7	8	8	4	2
1	2	7	7	8	8	4	2

DIAGRAM THREE - ALTERNATIVE LONG MULTIPLICATION

	3	0	7	8	2	6	x
	4	1	5	1	3	2	
		6	0	14	16	4	12
		9	0	21	24	6	18
		3	0	7	8	2	-
	15	0	35	40	10	30	-
	3	0	7	8	2	6	-
12	0	28	32	8	24	-	-
1	2	7	7	8	8	4	2
						3	2

Referring to diagram 3: here 6 times 2 is 12, 2 down 1 to carry, 2 times 2 is 4, 6 times 3 is 18, 18 plus 4 is 22 plus 1 is 23, 3 down 2 to carry and so on for the other columns. This time we have the same number of multiplies and adds but have saved a set of fetches and carries leaving an order of $4n^2$ instructions, ie a saving of 33% over the previous method. A further 50% saving can be obtained at implementation by taking advantage of a feature of the TMS320 DSP which allows auto increment and decrement of data pointers during multiply and accumulate operations - this effectively gives us the data fetching for free. Using this feature the core of our multiply program is as shown in diagram four.

In the DSP we have two auxiliary registers AR0 and AR1 which we use as data pointers and a T register which contains the multiplicand for any multiplication instructions.

The MPY * star instruction multiplies the contents of the T-register by the data pointed to by the current auxiliary register. The LTA * star instruction loads the T register (with new data pointed to by the current auxiliary register) and adds the result of the previous multiply into the accumulator.

DIAGRAM FOUR - MULTIPLICATION PROGRAM CORE

```
MPY * +, 1
```

```
LTA * -, 0
```

```

MPY * +, 1
LTA * -, 0
MPY * +, 1
LTA * -, 0
MPY * +, 1
-----
```

The + and - respectively increment and decrement the current auxiliary register and the 0 or 1 at the end selects a new auxiliary register as current for the next instruction. Both arguments for each successive multiply can thus be changed for no overhead while we multiply and add; which is what we need for the column based multiplication procedure just described.

With this method we do have to ensure that we don't overflow the accumulator before the end of a column. However, it is a fairly simple calculation to work out the optimum word length to satisfy this condition.

In practice we are prevented from using 16 bit words (on the early DSP's anyway) because they take all data as being in two's compliment form. Some of the more recent DSP's do help out by providing 40 bit accumulators and unsigned arithmetic.

MODULO REDUCTION

Next let's consider the modulo reduction operation. We have an intermediate value (say W) which is the result of a long multiply calculation and we want to find the remainder when W is divided by the modulus M. That is we want:

$$X = W \bmod M = W - M * (W \text{ div } M)$$

where 'div' is normal integer division.

Division on a DSP is hard (that is to say expensive in time) but given that throughout any single exponentiation we will always be using the same modulus and that we have available easy or 'cheap' multiplication, we can calculate (once only for each M) R equals the reciprocal of M and subsequently obtain our result, X, by two

multiplications and a subtraction:

$$X = W - M * (W * R)$$

The problem is that R in this case is a real number considerably smaller than one.

Thus, if we are to use this method we need to approximate and scale R . That is multiply R by some power of 2 and round off in order to represent R as an integer.

The trade off in this is fairly clear - the more accurately we represent R (and other intermediate values) the longer it will take to do the multiplications, the less accurately the greater the error we will have to correct at the end.

The mathematics of this trade-off are more complex than it would at first appear so I will just assume the results that we proved in our paper at Oxford.

LONGMOD PROCEDURE (refer to Diagram Five)

If M is represented as n base b digits (and therefore W is $2n$ base b digits) then R should be represented as the integer

$$R := b^{2n} \text{ div } M$$

Note that R here will have $n + 1$ digits as a result of the second precondition defining the range of M .

Next we multiply the most significant $n + 1$ digits of W by R and then multiply the n most significant digits of this result by M and subtracting the $n + 1$ least significant digits of this from the corresponding part of W . Our calculations show that the result x so obtained will always be in the range 0 to $3M - 1$. In other words at most two further subtractions of M are required to give us the result we are looking for.

It is possible to show that for about 90% of the values of W and M , the initial value of X obtained will be less than M and that only in 1% of cases will X exceed $2M$ and thus require two correcting subtractions.

DIAGRAM FIVE - LONGMOD

```

proc longmod.1 (in: w, m; out: x);
{ pre:  $0 \leq w < m^2$  &  $b^{n-1} < m < b^n$  &  $3 < b$  }
r :=  $b^{2n}$  div m ;
y := w div  $2^{n-1} * r$  ;
x := w mod  $b^{n+1} - m * y$  div  $b^{n+1}$  ;
{  $0 \leq x < 3m$  }
do x  $\geq m \rightarrow x := x - m$  od
{ x = w mod m }
endproc
-----
```

It can be seen from all this that for large n this modulo reduction method takes about the same time to execute as two long multiplications. Actually we can do almost twice as well as this by only calculating half the product in each long multiplication since the other half of each product is not required.

Thus, apart from the small overhead of calculating the reciprocal R (which could of course be done in advance and stored with its corresponding M as part of the RSA key) the modulo calculation is not much slower than the long multiplication.

FASTEXP CONTINUED

Returning now to the top level Fastexp algorithm. If we represent the exponent E as a sequence of n base b digits where $b = 2^f$ then our next requirement of the algorithm will require two nested loops to take care of respectively the digits and bits of E. Skipping a couple of refinement steps, our fastexp procedure is as shown in Diagram six,

DIAGRAM SIX - PROC FASTEXP.4

```

proc fastexp.4 (in: A E M; out: c);
{ pre:  $0 \leq A < M$  &  $0 \leq E$  }
( $e_{n-1} \dots e_0$ )b := E;
a,c,i := A,I,O;
```

```

do ( $e_{n-1} \dots e_1$ )b = 0 →
    ( $e_{f-1} \dots e_{i_0}$ )2 :=  $e_i$  ;
    j := 0;
    do j < f →
        if  $e_{ij} = 0$  → skip
         $\otimes$   $e_{ij} = 0$  → c := c * a mod M
        fi;
        a := a * a mod M;
        j := j + 1
    od;
    i := i + 1
{ post: c = AE mod M }

endproc
-----
```

which with a few further refinements, including insertion of our subroutines longmult and longmod and globalisation of the data (to save on parameter passing), can be translated almost directly into the TMS320 assembler code listed in Diagram seven. Notice how simple the program appears.

DIAGRAM SEVEN - PROC FASTEXP.7

```

*      proc fastexp.7(var A,E,M,R,C)
*
EXP      LARP 1      use AR1 as a counter
         LAR AR1,N   to initialize C
         MAR *-       AR1 := N-1
         LACK C0      C0 is XRAM relative address of C0
         ADDS DATA0   DATA0 is XRAM data page address
         ADDS N       ACC is pointer to CAR1
*
LOOP1    SUBS ONE    decrement ACC
         TBLW ZERO   " $C_{N-1} \dots C_0 := 0$ "
         BANZ LOOP1  repeat LOOP1 while AR1>0 and dec AR1
ENDL1    TBLW ONE   " $C_0 := 1$ "
         ZAC
         SACL I     "i := 0"
```

```

*
LOOP2 LACK E0      E0 is XRAM relative address of E0
ADDS DATA0
ADDS I
TBLR EI      "EI := Ei"
LACK F
SUBS ONE
SACL J      "j := f-1"
*
LOOP3 ZALS EI
AND ONE      "ACC := EI0"
BZ LSB0      "if ACC = 0 → skip" (to LSB0)
             "if ACC = 1 →"
LSB1   LACK C0
ADDS DATA0
SACL X      X := address of C0
CALL LONMUL  "call longmult(C)"
CALL LONMOD  "call longmod(C)"

*
LSB0   LACK A0
ADDS DATA0
SACL X      X := address of A0
CALL LONMUL  "call longmult(A)"
CALL LONMOD  "call longmod(A)"

*
LAC EI,15
SACH EI      "EI := EI div 2"
ZALS J
SUBS ONE
SACL J      "j := j-1"
BGEZ LOOP3  "repeat LOOP3 while j>0"

ENDL3
ZALS I
ADDS ONE
SACL I      "i := i+1"
SUBS NE
BLZ LOOP2    "repeat LOOP2 while i<ne"
*
endproc

```

There are only 43 machine code instructions required apart from the multiplication and modulo procedures.

This simplicity is, another direct benefit of the rigorous development methodology employed.

PERFORMANCE AND SECOND GENERATION DSP'S

This implementation of 'fastexp' takes on average (that is with an

exponent composed of half 0's and half 1's) 2.6 seconds to execute with 512 bit modulus and exponent on a Texas Instruments TMS32010 running at its maximum clock rate of 20 MHz. The 32010 (originally just called the TMS320) was the first general purpose DSP on the market but second generation DSP's are appearing now from most manufacturers and speed calculations using our algorithm suggest that times below 1 second will be possible on the TMS320C25 and below one quarter of a second on the Motorola DSP56200 which has a 24 x 24 multiplier and 56 bit accumulator.

The third (or is it fifth?) generation DSP from Inmos (the IMSA 100) which is part of the Transputer family, has on board no less than 32 16 x 16 multiplier/accumulators and should prove to be the fastest yet once we have refined our algorithm into the OCCAM parallel processing language which is executed directly by the transputer hardware.

CUSTOM CHIPS

Finally, I know that I started this presentation by stating that we decided against a custom silicon RSA implementation on the grounds of development cost and inflexibility, but a number of developments have taken place since we originally came to that conclusion. Most importantly the advent of silicon compilers and low volume custom silicon processes has reduced the turnaround time and development cost to a point where manufacture of a few hundred chips is a viable proposition. Furthermore, the increase in demand for fast RSA solutions plus the ultimate unit cost and performance advantages has led Computer Security Limited's sister company, RAANND Systems Ltd, to develop a custom RSA chip. Dr Gordon Rankine, the Managing Director of RAANND and the architect of this RSA chip, code named Thomas, has documented his presentation of this design elsewhere in the proceedings.

REFERENCES

- R L Rivest, A Shamir and L Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", Communications ACM Vol 21 (2) (Feb 1978)
- Dorothy E R Denning, "Cryptography and Data Security", Addison-Wesley (1983)
- Texas Instruments, "TMS 32010 User's Guide" (1983)
- Donald E Knuth, "The Art of Computer Programming Volume 2-Seminumerical Algorithms", Addison-Wesley (second edition - 1981)
- P D Barrett, "Communications Authentication and Security using Public Key Encryption - A Design for Implementation." (Oxford University Programming Research Group MSc Thesis (1984))
- C A R Hoare, "Notes of Communicating Sequential Processes", Oxford University Computing Laboratory (1983)
- David Gries, "The Science of Computer Programming", Springer-Verlay (1981)
- Edsger Dijkstra, "A Discipline of Programming", Prentice Hall (1976)

A HIGH SPEED MANIPULATION DETECTION CODE

Robert R. Jueneman
Computer Sciences Corp.
3160 Fairview Park Drive
Falls Church, VA 22042
(703) 876-1076

Abstract

Manipulation Detection Codes (MDC) are defined as a class of checksum algorithms which can detect both accidental and malicious modifications of an electronic message or document. Although the MDC result must be protected by encryption to prevent an attacker from succeeding in substituting his own Manipulation Detection Code (MDC) along with the modified text, MDC algorithms do not require the use of secret information such as a cryptographic key. Such techniques are therefore highly useful in allowing encryption and message authentication to be implemented in different protocol layers in a communication system without key management difficulties, as well as in implementing digital signature schemes. It is shown that cryptographic checksums that are intended to detect fraudulent messages should be on the order of 128 bits in length, and the ANSI X9.9-1986 Message Authentication Standard is criticized on that basis. A revised 128-bit MDC algorithm is presented which overcomes the so-called Triple Birthday Attack introduced by Coppersmith. A fast, efficient implementation is discussed which makes use of the Intel 8087/80287 Numeric Data Processor coprocessor chip for the IBM PC/XT/AT and similar microcomputers.

Key words: Manipulation Detection Code (MDC), Message Authentication Code (MAC), checksums, birthday problem attacks, authentication, encryption, digital signature, cryptography, numeric data processor chip, math coprocessor chip, 8087, 80287, IBM PC.

1 Introduction

A common theme throughout a series of papers^{1,2,3} by the author and his colleagues, Dr. S. M. Matyas and Dr. C. H. Meyer of IBM, has been the desirability of separating the function of encryption from that of authentication, so that they could operate at different architectural layers or levels in a communications system. In the context of the ISO Open System Interconnect reference model, for example, it was suggested that link encryption might be applied to all of the communications from a host, using a stand-alone link encryption device operating at ISO OSI layer 1, the data link layer. In this case the appropriate place for authentication would probably be in the Presentation or Application layers (layer 6 or 7), implemented in an application program inside the host. We have also suggested that since the mode of encryption might change depending on the physical medium involved, it would be desirable if the method of authentication were independent of the encryption scheme used.

The recently announced decision of the National Security Agency not to endorse new DES equipment for certification in accordance with Federal Standard 1027 after 1988, and in general to move on to a new family of encryption algorithms for both Unclassified, National-Security Related traffic as well as classified data, should serve to underscore the advisability of such a separation of function, as it will result in an increased requirement for "keyless" Manipulation Detection Code algorithms. Until the new Commercial COMSEC Endorsement Program (CCEP) algorithms are widely available (and perhaps for an even longer period, in the case of international circuits which may have to continue running DES), application programs might be supported by two or even three different link encryption algorithms (DES, an unclassified CCEP Type 2 algorithm, and a classified CCEP Type 1 algorithm, depending on the destination), but should require only one authentication algorithm. It should be observed that there is a fundamental difference between encryption and authentication with respect to the need to change algorithms, for in the case of encryption it is very difficult to know whether your traffic is being broken surreptitiously. In the case of authentication, however, it usually becomes obvious sooner or later if you have been spoofed. The objective is to minimize the amount of time required to detect the spoofing. It would therefore seem that authentication algorithms would not have to be changed nearly as often as encryption algorithms, and that there is perhaps less need for secrecy in their design.

In the papers presented to date, our primary concern was to find an authentication algorithm that would be more efficient than a MAC (especially when implemented in software on a microprocessor), and/or would not require a traditional encryption operation. Only secondarily did we focus on what this author now believes to be the fundamental distinction between an MDC and a MAC, i.e., that whereas a MAC involves one or more secret keys, an MDC makes use

-
1. Jueneman, Robert R., "Analysis of Certain Aspects of Output Feedback Mode", *Advances in Cryptology: Proceedings of Crypto82*, Plenum Press, New York, 1983, pp 99-127.
 2. Jueneman, R. R., C. H. Meyer, and S. M. Matyas, "Message Authentication With Manipulation Detection Codes", *Proceedings of the 1983 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, 1984, pp 33-54.
 3. Jueneman, R. R., C. H. Meyer, and S. M. Matyas, "Message Authentication", *IEEE Communications Magazine*, Sept. 1985 - Vol. 23, No. 9, pp 29-40.

of only publicly known quantities, and is therefore considerably more convenient from the standpoint of key management.

1.1 Cryptographic Checksum Requirements

Let us assume that we wish to apply a cryptographic seal to some electronic message or document, and that we will either use a digital signature approach, or else use link or end-to-end encryption to protect the MDC result. We must assure that the set of all checksums is very nearly one to one with respect to the set of all message texts, so that we can easily check the checksum (for example in the digital signature) instead of having to process the entire text. That is, given two messages A and B with checksums, we desire that checksum (A) and checksum (B) be identical if and only if the messages A and B are themselves identical. Assuming a good checksum algorithm, the chances that A and B are *not* identical given that checksum (A) equals checksum (B) should be 2^{-k} , where k is the number of bits in the checksum and the probabilities are averaged over all possible messages.

More specifically, the algorithm should have the following properties:

1. If two different texts (of arbitrary length) are checksummed, the probability that the two checksums will be the same when the two documents are not identical should be a uniformly distributed random variable that is independent of the text, with an average value over all possible texts of 2^{-N} where N is the number of bits in the checksum.
2. The checksum must be sensitive to permutations, so that the message ABC will produce a different value than ACB, etc.
3. As will be seen, the resulting checksum must be on the order of 128 bits in length, in order to resist a so-called "birthday attack" against the text itself.
4. Finally, all of the bits of the checksum must be an over-determined function of all of the bits of the text and all of the bits of the checksum of the previous block, in order to defeat several attacks that will be discussed below.

In addition, in a number of applications it is necessary to add a random Initialization Vector to the text itself, and to chain the blocks of messages together by including the checksum of the previous block in the checksum of the current block, so that one properly authenticated value cannot be substituted for another in a *playback attack*. For example, if a particular dialog occurs frequently, and the answer to some question is either "Yes" or "No", without the appropriate chaining the attacker could easily substitute the entire contents of a previous message, together with its valid checksum, and the message would be accepted. A 64-bit random Initialization Vector will suffice to initialize the authentication, but message chaining may still be required. It should be noted that an Initialization Vector may also be necessary to ensure that the same text is encrypted differently each time it is transmitted, in order to prevent a so-called dictionary attack. In general it appears that the same Initialization Vector (sometimes called a Message Indicator) could be used for both purposes, but it would be necessary to carefully examine both the encryption and the authentication scheme before making a blanket statement.

Finally, we must point out that although a DES-based Message Authentication Code or MAC could be used to authenticate either an encrypted or unencrypted text without further encryption because it makes use of a secret key⁴, that is not true of a Manipulation Detection Code. Although the text itself does not need to be encrypted, the MDC must be, so that the attacker cannot substitute his own MDC with any significant probability of success. In most cases, the MDC can simply be appended to the message, and if the entire message is encrypted together with the MDC, that will provide adequate protection. If the MDC is easier to calculate than an MAC, then if the message would be encrypted for secrecy in any case the MDC technique would be more efficient than a MAC.

2 Attacks Against Checksum Techniques

In the three previous papers in this series, we have addressed different aspects of the problem of authenticating the contents of a message against possible modification or corruption. In the first, a flaw in a draft of a federal standard regarding Manipulation Detection Codes was pointed out briefly, and a quadratic residue technique suggested as an alternative form of checksum. That paper also pointed out the need for two independent keys for encryption and authentication if a Message Authentication Code (MAC)⁵ is generated through the use of a secret (DES) key and appended to the message, for it was shown that the errors introduced in the plaintext by an error or by manipulation were exactly the errors needed to cause the MAC to be erroneously computed so as to validate the manipulated text.

The second paper presented an extensive analysis of various forms of Manipulation Detection Codes, including block XOR and linear addition techniques, when used in combination with Cipher Block Chaining, Cipher Feedback, and Output Feedback modes. That paper also discussed the architectural advantages of a Manipulation Detection Code that was independent of an encryption algorithm, particularly in those cases where low-level link encryption may be used to protect the traffic flowing into or out of a main-frame host processor, yet it is desired for an application program in the host to verify the authenticity of the messages received. In addition, the potential speed advantages of an MDC technique compared to the calculation of a MAC were discussed.

During the course of writing that paper and reviewing it with our peers, a number of attack scenarios were identified that must be considered whenever new schemes are proposed. In particular, Dr. Don Coppersmith introduced several attacks which he called under-determined knapsack attacks. These have also been called "birthday" attacks, because they generally involve generating random variations in the text and calculating a MAC or an MDC, then working

4. This is not recommended, however, because an unencrypted MAC reveals something about the message itself, and may form the basis for a dictionary attack.

5. As defined in Federal Information Processing Standard FIPS PUB 46, "DES Modes of Operation" published by the National Bureau of Standards, "A MAC may be generated using either the CFB [Cipher Feedback] or CBC [Cipher Block Chaining] mode. In CFB authentication, a message is encrypted in the normal CFB manner except that the cipher text is discarded. After encrypting the final K bits of data and feeding the resulting cipher text back into the DES input block, the device is operated one more time and the most significant M bits of the resulting DES output block are used as the MAC, where M is the number of bits in the MAC. In CBC authentication, a message is encrypted in the normal CBC manner but the cipher text is discarded. Messages which terminate in partial data blocks must be padded on the right (LSB) with zeros. In CBC authentication, the most significant M bits of the final output block are used as the MAC."

forward and backward until two matching MACs or MDCs are found. Making random variations in the text in two places and then sorting and comparing the results for a match allows the attacker to take advantage of the so-called Birthday Problem in statistics to reduce the work required to approximately the square root of the effort required to match a particular given MAC or MDC.

2.1 The Fundamental Birthday Attack.

The third paper abstracted the second for a more general audience, but also added some new information. In particular, it was recognized that *any* Manipulation Detection Code (MDC) or Message Authentication Code (MAC) is susceptible to a birthday attack against the text itself, unless the MDC or MAC is on the order of 128 bits in length. This fundamental attack proceeds as follows, and assumes that one user is attempting to defraud another by devising a version of a bogus or unfavorable contract or agreement which would have an identical checksum as would an acceptable version of a legitimate one, having the other party digitally "sign" the legitimate version, and then produce the bogus version in front of a judge and claim that the other party has defaulted on his obligations:

1. Assume that a 64-bit MAC or MDC is used, and that if necessary the attacker can exercise the authentication system *ad infinitum* to generate a MAC or an MDC, even if a secret key which he does not know is used in the case of the MAC.
2. The attacker secretly prepares a number of subtle variations of the legitimate text in advance, and calculates (or has the system calculate) the MDC or MAC for each one. In the case of an electronic mail message or document, for example, suppose that a number of lines contain the ASCII character sequence "space-space-backspace"⁶ between selected words. The attacker might prepare a set of variations of that document in which the sequence in selected lines would be "space-backspace-space". The length of the text would not be altered thereby, and all of the variations of the document would appear to be identical, both when printed and when displayed on the normal video display, unless "dumped" in hexadecimal format. Other, more consequential changes to the text could also be made, of course. By systematically altering or not altering the text in each of say 32 different lines, 2^{32} or 4.3 billion variations could be generated. A file of records consisting of the MAC or MDC plus a 32-bit permutation index could be used to summarize what lines were altered by a given variation, and what MAC or MDC resulted.
3. The attacker then prepares an equally large number of variations on the bogus text he would like to substitute for the legitimate text, and calculates (or has the system calculate) the MDC or MAC for each one of those variations as well, producing another file of MAC/MDC results plus the permutation index records.
4. The attacker then compares the two files, searching for a pair of identical MACs or MDCs and noting the permutation indices. (If no match is found, the attacker can simply generate a few more random variations of the legitimate and the bogus texts until a match

6. Other combinations, such as null-character, or carriage return - line feed would also work, as well as less subtle variations such as changing "the" to "an", or inserting or deleting commas or spaces in a numeric field.

is found.) He then recreates the full text of both the acceptable and the unacceptable documents with the specific modifications necessary to produce the matching MACs or MDCs, based on the permutation indices.

5. Finally, he offers the appropriate variation of the legitimate contract to the other party and both "sign" it. At some time in the future the attacker substitutes the unfavorable contract, and tells the judge that the digital signature containing the MAC/MDC "proves" it was that version that was signed by both parties.

This is Yuval's⁷ classic "How to Swindle Rabin" form of a so-called "Birthday Problem" attack.

According to the famous birthday paradox⁸ problem in statistics, this kind of an attack is likely to succeed if the number of variations of each document that are generated and compared approaches the square root of the total number of possible MAC/MDC values. That is, if a 32-bit checksum were used, the probability of a successful attack would be about 50% after only 2^{16} or 65536 variations were computed, and would increase rapidly after that point. If a 64-bit MAC or MDC were used, then the 4.3 billion iterations produced by systematically varying 32 lines of text would be likely to suffice.

In order to see whether this attack would be computationally feasible against a 64-bit MAC, let us assume that the variations all occur at the end of the text and that exactly one variation occurs in 8 bytes of text, so that only one DES iteration would be required to account for that variation. The brute-force way to calculate the resulting MAC for the entire text would be to recalculate the last 32 DES blocks for each variation, which would require $2 \times 32 \times 2^{32}$ DES iterations for the two sets of variations of the text. However, by only encrypting those blocks that have changed and those for which earlier blocks have changed, the number of DES iterations can be reduced to $2 \times (2^{33}-1)$. A hardware DES implementation running at 10 microseconds per iteration could complete the task in just under 2 CPU days.

However, the amount of I/O required to sort and compare the data must not be neglected. A 64 bit MAC and a 32 bit permutation index per variation would require 12 bytes per entry times 2^{32} entries, or 51.5 gigabytes per file. At an effective rate of 20 microseconds per variation (including encrypting due to the requirement to reencrypt blocks after a change), data would be generated at the rate of 4.8 Mbps or 600 kilobytes per second, which is well within the channel capacity of a mainframe computer to record. The process of comparing two files consisting of 340 reels each of 6250 bpi high-density tape (151 megabytes per reel), searching for any one value on one file that matches any one value on the other file, would admittedly be a lengthy task even for a mainframe computer, but it is not infeasible. One approach would be to presort the information by distributing the data across 22 tape drives while the information is being generated, producing 22 files of approximately 15 to 16 reels each for each variation. Each of those files could in turn be distributed onto 20 reels of tape at maximum tape speed, and then those approximately 680 individual reels could be sorted one at a time using a conventional tape

7. Yuval, G., "How to Swindle Rabin", *Cryptologia*, Vol 3., No. 3, July 1979, pp 187-190.

8. How many people must there be in a room in order to have a good chance that at least two people in the room will have the same birthday.

or disk sort routine, and finally compared. Assuming each reel requires 15 minutes to sort, the total process could be completed in about a week.

An interesting alternative technique was suggested by Caron and Silverman's distributed processing approach to factoring⁹. Let us assume that the attacker has at least the occasional use of 256 Intel 80386-based microprocessors or similar machines which are connected via a high-speed LAN. Each of these slave machines will be assumed to have two boards of 8 megabytes¹⁰ each of the new 1 meabit memory chips. In addition, a master station will be equipped with a hardware DES implementation, four 8-megabyte memory boards, and two 85 megabyte hard disks.

The total amount of memory in the 256 slave processors would be 4.295 gigabytes, or 2^{35} bits. Let us assume that after each calculation of a MAC in the first set of variations, the master workstation sends 24 bits (bits 8 through 32) of the MAC to the appropriate slave processor based on bits 0 to 7 of the MAC. Each slave processor would then use those 24 bits to address a particular bit within its memory, and would turn on that bit. At the end of the first pass through all of the variations of a single document (requiring about 24 hours), the contents of the first 32 bits of all 2^{32} MACs calculated would be represented as a set of bits turned on in all of the memories. Because there are 2^{32} bits turned on out of 2^{35} bits total, the probability that a particular bit will be on after the first pass is 1/8, with many bits having been turned on multiple times within this pass. At the end of the first pass, all of the slave processors would dump memory to a hard disk, then zero all of the bit storage area.

The master processor would then begin processing the second set of variations and would again send 24 bits of the MAC to all of the slave processors. This time, however, the slave processors would check to see if that particular bit had already been turned on. If it had, it would signal the master CPU, which would record that permutation index. Because the probability of a particular bit being turn on in both the first and the second passes is 1/64, a 1 byte increment from the previous permutation index would normally suffice and there would be approximately $2^{32}/64$ or 67,108,859 values to record, so one 85 megabyte hard disk would be sufficient to contain one set of permutation indices.

The master CPU would then repeat the calculations of the first pass in a third pass, again broadcasting 24 bits of the MAC to the appropriate slave stations, which would replay whenever a collision was found. The master station would then record the permutation indices associated with those collisions on the second 85 megabyte hard disk.

This entire three pass process would then be repeated, but instead of examining the first 32 bits of the MAC the last 32 bits would be used. The fourth pass would initially turn a set of bits based on the first document, and the fifth pass would check for a possible collision. However, the master CPU would not have to generate all 2^{32} variations, but would only process the variations that were previously recorded as potential matches after the second and third passes. Therefore, instead of taking two days for this processing, it would only take about 45 minutes.

During the fifth and sixth passes, the various slave processors would send back acknowledgements as before, and the master station would erase any permutation index that did not produce a collision. This time, the probability of a false alarm collision is only 1/4096, so the expected number of collisions remaining to be processed is 1,048,576.

The master station would then make two internal passes over the remaining permutation indices for the two different documents, using a hash table lookup scheme to store/search the 64-bit MAC and 32 bit permutation indices.

2.2 Other Opportunities For Birthday Attacks.

Similar attacks could potentially succeed against command and control systems, especially if the attacker is able to send bogus commands and random variables over a channel that cannot be shut down without denying service to the legitimate users as well. An example would be an attacker who attempts to take over or disrupt a communications satellite by sending spurious commands via the Telemetry, Tracking, and Control channel to the satellite in an attempt to get

9. Caron, Thomas R. and Robert Silverman, "Parallel Implementation of the Quadratic Sieve", *Advances in Computer Science - CRYPTO '86 Proceedings*, Springer-Verlag, Berlin, 1987.

10. Sixty-four microprocessors with 64 megabytes of memory would be significantly cheaper, but that would be a very specialized system, as opposed to a configuration that might be used for other purposes and could be "borrowed" for our purposes.

it to move out of position, use up all of the maneuvering fuel, go into a spin, etc. There is no easy way that the attacker can be located, and if he is operating out of a foreign country there may be nothing that can be done to stop his transmissions. The attacker can simply send random data, and even if the command link were encrypted there is a possibility that the decrypted information might be accepted as a valid command. Unless a sufficiently long checksum is used, random data and a random MDC or MAC will eventually result in a random command being accepted¹¹.

Another instance could arise in a multilevel-secure system, where a cryptographic "seal" is applied to an "object", in order to prevent classified information from being disclosed or modified without proper authorization. For example, if the security classification associated with the object could be manipulated by a Trojan Horse program, a classified object's label could be changed to "unclassified", and the information released. Similarly, the contents of a properly marked, unclassified object could be changed and classified information inserted. Because the sensitivity label must be very closely associated with the contents of the object (to prevent a simple cut-and-paste attack), the security seal of the object typically includes both the sensitivity label and the contents of the object as well. In this case, the Trojan Horse program could conceivably manipulate the label together with some innocuous portion of the data, and repeatedly present the information to the cryptographic seal mechanism until two versions, one good and one bad, happened to produce the same cryptographic checksum. The substitution would then be prepared.

2.3 Recommended Length For Cryptographic Checksums.

Based on these attacks, we conclude that it is essential that any MAC or MDC checksum be on the order of 128 bits in length, in order to protect against situations where the opponent could systematically change both the text and the MAC/MDC until he finds a combination that works.

A 128-bit checksum is sufficient, because in addition to the sorting and searching problem rapidly becoming insurmountable (after about 80 bits), the 2^{65} basic MAC/MDC calculations required by the birthday problem attack would not be computationally feasible, even if they were to take only 1 nanosecond apiece. It must be stressed that this attack has nothing to do with the cryptographic strength of the MAC or MDC algorithm, or whether conventional keys, public keys, or no keys at all are used, but only whether the length of the result is sufficient to withstand any computationally feasible number of random "birthday attack" trials.

In this connection, it is worth observing that the recently revised ANSI X9.9-1986 authentication standard¹² specifies the use of a 32-bit MAC, although the future use of a 48-bit or 64-bit MAC is also discussed. In analyzing the protection afforded by that standard, we should consider both external attacks and internal fraud. With respect to an external threat in this environment, a 32-bit MAC is arguably sufficient. Even though an attack against such a system would be likely to

11. Actually, satellite command processors typically echo the command received back to the ground, and then require an "Execute" command within a certain period to make the received command take effect. Assuming that the Execute command is also encrypted and authenticated it is much less likely that this particular attack would succeed, but the point is clear.

12. Financial Institution Message Authentication (Wholesale) X9.9-1986 (Approved August 15, 1986), published by the X9 Secretariat, American Bankers Association, 1120 Connecticut Avenue, Washington, D.C. 20036.

succeed after only 65 thousand attempts, hopefully all of the false MACs should generate some alarm, and the investigative agencies would be called in to stop the perpetrator before he (or she!) was successful.

With respect to a possible internal threat or Trojan Horse program, however, it is obvious that if the security of the system were to rest solely on the authentication provided by the MAC, then a 32-bit MAC is grossly inadequate. It should be apparent from the preceding discussion that even a 64-bit MAC would provide inadequate protection from a member bank or insider who might attempt to defraud another institution, if that were the only mechanism used to protect against such attacks. In the banking environment, of course, there are all sorts of reconciliation processes that would presumably uncover such attempts at fraud sooner or later, but in other environments this might not be the case. *System developers are therefore cautioned not to apply the X9.9-1986 authentication standard outside of the specific wholesale banking environment for which it was developed.*

2.4 The Need For Super-Authentication.

It should be noted that if an MDC technique were used to authenticate a message that is protected by Output Feedback (OFB) mode (or worse yet, not protected at all), the opponent could easily calculate a valid MDC to go with the modified text, and append the new MDC to the text at will, since there is no separate cryptographic key used to protect the authentication information. Even though the attacker doesn't know the key used to encrypt the message, if we assume that he does know the plaintext (perhaps because he generated it) he can determine the keystream output from OFB by XORing it with the plaintext, and can then change the keystream to suit his purposes. This particular attack can be defeated by having the system introduce a secret, varying, random component which the opponent doesn't know (an Initialization Vector) into every message, and including that random value in the MDC calculation. The Initialization Vector is not a key, since it doesn't have to be known in advance by either party. It doesn't even have to be deterministic, and it can be discarded by the receiver after the MDC is checked. However, the random value should be at least 64 bits long, so that the attacker cannot discover its value and then the true value of the MDC and therefore the corresponding bits of the key stream by exhaustively trying all possible values of the initial random component.

With this in mind, let us reconsider the delayed transmission OFB attack that was discussed in the second and third papers. That attack made use of a lengthy message whose plaintext was known to the attacker, so that an extensive amount of keystream would become known. The beginning and end of the message would then be jammed, and an invalid message substituted based on the keystream. The invalid message could even contain a random component, since the attacker would have already recovered the keystream bits for that portion of the output.

In order for this attack to succeed, it is necessary for the attacker to precisely synchronize the plaintext and the ciphertext, know the current message sequence number, intercept the ciphertext and block it, jam the portion of the message containing the secret, random component to make it look like a noise burst on the transmission medium, and then fabricate any desired random value, bogus message, and a corresponding fraudulent MDC, and follow it with a valid HDLC

frame check sequence. Finally, the end of the message containing any remaining message text, the old MDC, frame check, and the start of the next message would be replaced with random characters to cause another noise burst to be simulated, which would then be rejected by the standard HDLC error recovery mechanism at the receiver.

It should be clear that this real-time interception and modification technique, although difficult to put into practice, could theoretically be applied to *any* MDC scheme that does not involve the use of a secret key for authentication, if the message text being sent is known to the attacker.

Although this attack was previously considered legitimate, and a potentially serious obstacle to the use of an MDC technique, it can only succeed if the message being attacked is considered in isolation, as if it were the only message being sent. In order to defeat the attack it is only necessary to chain the individual messages together in such a manner that a change in one message will affect the MDC in the *next* message. Therefore, instead of the MDC in a given message pertaining to that message, it should instead pertain to the previous message. The MDC contained in the first message should cover the Call Request/Call Acknowledgement or other session establishment message *sent by the other correspondent*, and containing a secret, random component known to that correspondent, or the system at that end. By MDCing something that the other correspondent already knows, the chain is anchored at the beginning, defeating an attack that would systematically change every message in the sequence.

Each MDC should therefore cover not only the data contents of the previous message, but the previous MDC as well, so that changing a single bit of a message will affect all of the MDC results from then on. The MDC for the previous message then satisfies the requirement for a secret, random component in each message if OFB is used. In order to detect an attempt to delete the final message of a session, a unique end-of-session message should be sent that includes the MDC of the previous message, plus the MDC of the end-of-session message itself. If a digital signature capability is implemented, it would be desirable to sign this final message. If the final MDC is digitally signed, then the initial MDC could be a constant. This would avoid the necessity of having a session established in real time so that the other correspondent can check the original value of the MDC at the time of session startup. This would be particularly useful in store-and-forward message systems, including electronic mail and bulletin board systems, where the receiver is not in direct contact with the originator and the intermediate system may be a public or untrusted system. It would also apply to unidirectional transmission systems, including some command and control systems as well as systems that transmit to destinations operating under radio silence rules.

Finally, it should be noted that in some cases the communications system may employ some device such as an Automated Teller Machine to screen the messages being sent, allowing only the "good" messages through. But in this case the system (the ATM machine and the bank) and the user do not necessarily share common interests. The user may wish to ensure that his messages are kept secret, and the legitimate user may also be interested in assuring the end-to-end integrity of his messages. But the system, in this case the ATM machine, may also have a role to play in assuring that the user does not compromise the integrity of his own messages.

We should not try to satisfy both of these possibly diverging requirements through one mechanism. Instead, just as we sometimes use super-encipherment (for example using end-to-end

DES encryption to ensure writer-to-reader privacy, plus link encryption using classified algorithms to protect against an external threat), we should talk about super-authentication. That is, *if the system has a requirement to assure that messages are not modified after they exit a secure processing facility, then the system must independently provide that assurance without depending upon the user's mechanisms.*

3 A Quadratic Congruential MDC

Now that we have developed the rationale for the use of an MDC algorithm, we should certainly try to define a suitable implementation:

3.1 The Original QCMDC.

The original Quadratic Congruential Manipulation Detection Code (QCMDC) function proposed in the second paper in this series was defined as:

$$Z_0 = C = \text{MDC initial value}$$

$$Z_i = (Z_{i-1} + X_i)^2 \bmod N$$

$$\text{MDC} = Z_n,$$

where C , Z_i , and MDC are all 32-bit integers in two's-complement notation, and N was the Mersenne prime $2^{31}-1$, chosen so that the modulo result would fit in a 32-bit word.

In order to prevent an attack against the MDC in the case of Output Feedback Mode (where both the text and the MDC could easily be changed), it was first proposed to make the first 32 bits of the message a secret seed, S , withheld even from the message originator, so that if the opponent attempted to attack his own message he would not know the secret seed and would therefore not be able to intelligently modify the MDC.

However, a variation of the under-determined knapsack attack of Coppersmith involving the taking of square roots modulo N and working backwards from the MDC in a meet-in-the-middle attack showed that the use of the secret seed, S , was not sufficient; and that either a secret quantity C would have to be introduced into the accumulator or the MDC would have to be extended to 80 bits or more.

When the QCMDC algorithm was first implemented on the 8087, some variations were also coded and tested which used an Exclusive OR operation (denoted \oplus or XOR). These variations were intended to defeat Coppersmith's technique of working backwards taking square roots modulo P . Although these operations were felt at the time to increase the cryptographic strength of the algorithm by denying the attacker the opportunity to work backwards (by making the algorithm non-invertible), the additional operations were quite time consuming.

However, we concluded in the third paper that the MDC must be on the order of 128 bits long in order to foil the birthday problem attack in any case, and for that reason it was recommended

that four separate iterations of the MDC algorithm be performed over the text resulting in a 124-bit MDC. It was therefore thought that Coppersmith's attack on the QCMDc would be defeated because of the difficulty of generating the requisite 2^{62} different variations. We then concluded that none of the variations on the basic QCMDc approach were necessary.

3.2 The Triple Birthday Attack

Ironically, one week before the publication of the third paper, Coppersmith¹³ pointed out a weakness in a double-iteration DES signature scheme by Davies and Price which also applied (to a somewhat lesser degree) to the quadruple-iteration MDC scheme, as follows:

- Assuming the use of an arbitrary *invertible* function $F(X, H)$ as a checksum function operating over the message $M = (M_1, M_2, \dots, M_n)$, intermediate results H_1, H_2, \dots, H_n are produced from the relation $H_i = F(M_i, H_{i-1})$, or alternately from the inverse of F , $H_{i-1} = F^{-1}(M_i, H_i)$.
- During a precomputation phase, select some arbitrary n -bit quantity Z , which is going to be the value of $H_2, H_4, H_6, \dots, H_{18}$. Then randomly select approximately 2^{36} values X , compute the values $F(X, Z)$, and store these values. Then randomly select 2^{36} values Y , compute the inverse function $F^{-1}(Y, Z)$, and store those values as well. Then compare all of the Y values to all of the X values searching for a matching pair, using a sort and compare technique as required. This constitutes the first birthday problem. We expect to find 256 such matching pairs, and if not, we will examine a few more values of X or Y or both. Note that each such pair (X_i, Y_i) can be used as a message pair $(M_3, M_4), (M_5, M_6), \dots$, or (M_{17}, M_{18}) such that if $H_2 = Z, M_3 = X_i, M_4 = Y_i$ then $H_4 = Z$, etc.
- Given a message $M^* = (M_{19}, M_{20}, \dots, M_n)$, the chosen value of Z , and the 256 pairs (X_i, Y_i) obtained during the precomputation, our task is to select values of M_1, M_2, \dots, M_n which will make H_{2n} a valid hash of $M = (M_1, M_2, \dots, M_n)$. We therefore find values of M_1 and M_2 such that $F(M_1, Z) = F^{-1}(M_2, Z)$ to put ourselves in a standardized position. This takes on the order of 2^{33} hashing operations and 2^{32} storage. This is the second birthday problem.
- Working backwards from H_{2n} (note that this requires the checksum function to be invertible), using the values $M_n, M_{n-1}, \dots, M_{19}$, we find the value of H_{n+18} , the value of the hash function on the second iteration. Finally, we make use of the precomputed pairs (X_i, Y_i) . For each of the $256^4 = 2^{32}$ choices of the four pairs (X_i, Y_i) to be the values of $(M_3, M_4), (M_5, M_6), (M_7, M_8)$, and (M_9, M_{10}) , we compute the value of H_{n+10} that would result then do the same thing with the values of $(M_{11}, M_{12}), (M_{13}, M_{14}), (M_{15}, M_{16}), (M_{17}, M_{18})$, computing backwards from H_{18} to get a value for H_{10} . We again sort and compare these values as the third birthday problem. We expect one match, and the corresponding values of M_3 through M_{18} finish our task for a two-pass checksum process.
- The process could be extended to attack a triple-pass hash algorithm by constructing eight "super-pairs" consisting of M_{19} through M_{35} plus M_{36} through M_{52} , etc., up to M_{258} . Each

13. Coppersmith, D., "Another Birthday Attack", *Advances in Cryptology - CRYPTO '85 Proceedings, Lecture Notes in Computer Science, Vol. 218*, Springer-Verlag, Berlin, 1986, pp 14-17.

super-pair would be manipulated during the precomputed phase to continue to produce the value of Z, even on the third pass. Only slightly more computation would be required, but obviously 258 blocks of the message M would be constrained, limiting the messages that could be attacked to fairly long ones. Finally, this process could be extended even further to attack a quadruple-pass hash algorithm by computing eight "super-dooper" pairs consisting of 512 blocks each, or a total of 4098 blocks.

The multiple birthday attack therefore serves to reduce the strength of an N-pass signature scheme from an apparent $2^{N^*k/2}$ to an almost trivial $N^*2^{k/2}$.

It is worth mentioning that the Coppersmith's attack also applies to attempts to extend the MAC of FIPS PUB 46 or ANSI X9.9 to 128 bits (in order to try to overcome Yuval's attack against the plaintext) by simply concatenating two or more MACs using two or more different authentication keys. The reason is that the MAC function, i.e., DES Cipher Feedback mode encryption, is invertible, and in addition the components are separable and individually too small to resist a birthday attack¹⁴. As a result, and *contrary to the advice in the second and third papers in this series, the 64-bit Message Authentication Code technique by itself cannot be considered sufficiently strong, and is not recommended* if there is any possibility that the originator may attempt to defraud the message recipient, or if a Trojan Horse could circumvent security controls through such a mechanism. In addition, the use of a MAC in certain command and control situations where the attacker may attempt to spoof computer-controlled equipment or processes is also not recommended.

In practice, the likelihood of all of these blocks of being substituted without being noticed may be remote, for in the case of the quadruple-iteration QCMDC routine this amounts to 16392 bytes that would have to be inserted in the text. However, in the previous papers we had committed ourselves to detecting even a single inserted, deleted, or manipulated bit, regardless of the amount of text and independent of any internal syntactical or semantic content. After all, if we were to rely solely on internal consistency checks to detect such manipulations we would first have to invent a suitable manipulation detection scheme!

It should therefore be observed that Coppersmith's triple-birthday attack will succeed against a multiple-iteration QCMDC routine if two conditions are true:

1. If the checksum function is *invertible*, so that it is possible to work both forwards and backwards to produce matching values in a birthday-problem attack.
2. If the checksum function is subject to *decomposition* into separate and independent elements, each of which is sufficiently small that the birthday-problem attack is feasible from the standpoint of computation time and storage. If the checksum function were to involve a 128-bit result that could not be broken down into something smaller, then the birthday attack would be infeasible because it would involve generating, storing, and comparing on the order of 2^{64} 128-bit checksums and 64-bit permutation indices, or about 8.8×10^{20} bytes of storage, or 5 quadrillion reels of 6250 bpi magnetic tape.

14. This is not to say that a suitable 128-bit checksum could not be constructed using DES or some other 64-bit block cipher, but only to caution that the task is not nearly as trivial as it may appear at first glance.

In the case of the simple QCMDC routine (where $H_i = ((H_{i-1} + M_i)^2) \bmod N$), the addition of H_{i-1} and M_i makes the function technically non-invertible from the standpoint of exactly and uniquely reproducing the input F_{i-1} given some F_i , since the H_i is a function of two independent variables. But it is sufficient if the attacker can construct a value $Y_{i-1} = F^{-1}(X_i)$ which, when computed in the forward direction, will produce the desired result for H_i . To do this, note that $(K^*N + X) \bmod N = X$. Therefore, multiply the modulus N by some variable K such that the result is a perfect square, and take the square root of the result. Then $Y_{i-1} = H_i - K^*N$, and the value of X_i that will satisfy this relation is $\text{SQRT}(K^*N) - Y_i$.

This suggests a variation of the QCMDC routine that would involve XOR(s) or some other non-linear combining function that would not be susceptible to a square root attack. If in addition the routine involved all 128 bits of the text and all 128 bits of the MDC of the previous block, then neither of the two conditions would be true and the triple-birthday attack would therefore be defeated. However, as the indefatigable Dr. Coppersmith pointed out, this is not necessarily a trivial task.

In order to make the MDC function non-invertible it is necessary to introduce a history function, i.e., some value that would not yet be known when working in the backwards direction, calculated in some non-linear manner so that the square root attack will not work. In addition, it appears necessary to incorporate multiple references to both the text to be authenticated and to the previous MDC result, so that the only value that would satisfy the forward relationship is the proper one. Not only must each bit of the checksum function be a function of all of the bits in the full 128-bit text block together with all of the bits in the MDC of the previous block, but additional dependencies should be introduced to ensure that the function is not just minimally dependent on those bits but is over-constrained instead.

Finally, as stated previously, the MDC function must produce a value on the order of 128 bits in length in order to defeat the various birthday attacks against the text itself.

3.3 The New, Improved QCMDCV4 Algorithm

The following algorithm, dubbed the Quadratic Congruential Manipulation Detection Code, Version 4 (QCMDCV4) for brevity, is proposed to satisfy these requirements:

Consider a 128-bit (16 byte) block of text, divided into four 32-bit words, T_1, \dots, T_4 . For reasons that will be explained later, we will be operating on a 31-bit subset of each of those 32-bit words which consists of the sign bit and the low-order 30 bits, i.e., $T^*_i = T_i \text{ AND } \text{BFFFFFFF}$. In addition, we will define a 30-bit fifth component, T^{**} , consisting of the 6 high-order bits of T_1 (with the 6 bits shifted right two bits and 2 leading zero bits introduced on the left or most-significant-bit position), concatenated with the high order 8 bits of T_2, T_3 , and T_4 , to make a 32 bit word with two high order zero bits.

Let the 128 bits of the MDC result (obtained from the previous block of text) also be divided into four 32-bit integer components M_1, M_2, M_3, M_4 ; and let the 32-bit components of the new MDC result be designated as M^*_i .

Finally, define a set of moduli $N_1 \dots N_4$, consisting of the four largest prime numbers less than the maximum 32-bit integer, namely 2147483629 ($2^{31}-19$), 2147483587 ($2^{31}-61$), 2147483579 ($2^{31}-69$), and 2147483563 ($2^{31}-85$).

Then calculate:

$$M^*_1 = [(M_1 \oplus T^*_1) - (M_2 \oplus T^*_2) + (M_3 \oplus T^*_3) - (M_4 \oplus T^*_4) + T^{**}]^2 \bmod N_1$$

$$M^*_2 = [(M_2 \oplus T^*_1) - (M_3 \oplus T^*_2) + (M_4 \oplus T^*_3) - (M^*_1 \oplus T^*_4) + T^{**}]^2 \bmod N_2$$

$$M^*_3 = [(M_3 \oplus T^*_1) - (M_4 \oplus T^*_2) + (M^*_1 \oplus T^*_3) - (M^*_2 \oplus T^*_4) + T^{**}]^2 \bmod N_3$$

$$M^*_4 = [(M_4 \oplus T^*_1) - (M^*_1 \oplus T^*_2) + (M^*_2 \oplus T^*_3) - (M^*_3 \oplus T^*_4) + T^{**}]^2 \bmod N_4$$

Several features of this algorithm should be noted. First, each of the 16 different XOR combinations is unique. Second, even if a significant amount of the text contains all zeroes (with the result that the XOR does nothing), the alternating signs for the M_i and T^{**} components operate in such a manner that the contribution of the various terms will be different in each case. Finally, the M^*_i values are introduced into the computation of the subsequent components as soon as they are available, so that there is a great deal of inter-dependency and mixing. As a result, each 32-bit component of the MDC result is an over-constrained function of all of the text and all of the prior MDC.

The previous papers had proposed a constant value for the modulus, N , equal to the Mersenne prime $2^{31}-1$ (2147483647), for all four of the 32-bit M^*_i results. But as Don Coppersmith pointed out when reviewing a draft of the current procedure, because $2^{31}-1$ is the largest number that can be contained in a four byte integer in two's complement form, XORing the hexadecimal bit-string 80000001 has the effect of inverting the sign and the low order bit, which can be the equivalent of adding or subtracting the modulus. As a result, even when the intermediate sum is squared, the division by the $2^{31}-1$ modulus frequently produces no change in the result, depending on the sign of the T_i and whether a carry would be required, and a modification to the text could thereby escape detection.

Coppersmith proposed picking up the text only 24 bits at a time to avoid this problem, using additional iterations to get back to around 128 bits. In an attempt to overcome this problem without the overhead of an additional iteration, the four different primes for the moduli N_i

were introduced, all of them smaller than 2^{31} . However, it was found that if the text consisted of one 32-bit word of random bits and three words of zeroes, then in about 10% of the cases it was possible to either add or subtract the value of the first modulus and have the change go undetected in the corresponding 32-bit word of the MDC result. Although the use of four different values for the moduli means that the substitution does affect the remaining 3 words, or at least 96 bits, it was felt that the full 128-bit strength should be preserved.

For this reason, only 30 bits plus the sign bit of each 32-bit word of text is used in forming the intermediate sum. Since the moduli are all greater than 2^{30} , it is impossible to add or subtract the modulus from the text without detection. The final addition or subtraction of T^{**} ensures that all of the bits in the text affect all of the bits of the result.

One further improvement is possible. Because of the squaring operation, each 32-bit MDC component will be positive, producing a 124-bit result. But we can calculate the parity of the intermediate MDC result, just prior to the multiplication, and then change the sign of each 32-bit result if the parity is even.

Finally, because the algorithm operates on 16-byte blocks, it is necessary to somehow differentiate between a text string that is say 1 byte long and one that consists of the same byte extended with up to 15 bytes of zeroes. For that reason the last few bytes (less than 16), if any, are moved to a 16-byte buffer, the rest of the buffer zeroed, and the MDC algorithm executed $N+1$ times on that same buffer, where N is the number of the last few bytes. $N+1$ is used instead of N , because a block that is 16 bytes long has to be processed once, and therefore a 1 byte block has to be processed twice in order to be distinguished from the previous case. If improved performance is needed, the length code of the text can be prefixed to the text, and the size of the buffer extended to be an exact multiple of 16 bytes. This technique *must* be used if it is necessary to deal with text strings that are not multiples of 8 bits in length.¹⁵

In order to avoid a strong correlation between the text and the MDC result in the case where the text is very sparse (contains mostly zero bits), it is desirable to use different values for the starting values of M_i . For purposes of standardization the values 141421356, 271828182, 314159265, and 57721566 are suggested.

4 Implementation Considerations

The QCMDCV4 algorithm has been implemented and tested on the IBM PC and AT microcomputers and the Compaq 286 Portable, and should run correctly on any similar machine which uses the Intel 8088, 8086, 80188, or 80286 CPU chip in combination with the 8087 or 80287 Numeric Data Processor chip. The 8087/80287 is used to significantly speed up the calculation of the various arithmetic operations, in particular the division modulo the large primes.

15. It may be worth mentioning that the ANSI X9.9-1986 authentication standard and the definition of the MAC in FIPS PUB 46 do not take this problem into account, and therefore do not differentiate between a short message (one that is not a multiple of 8 bytes in length) that must be padded with zeroes, and one that is a multiple of 8 bytes in length and happens to contain zeroes at the end. Although binary zeroes would be interpreted as ASCII null characters and would not be confused with the ASCII "0" (hexadecimal 30) character in coded text, formatted binary information is allowed by paragraph 5.1 of that standard, which does not specify that a length indicator field must be used. The confusion therefore could occur in this case.

During the calculations the results are kept in IEEE Binary Floating Point 80-bit Temporary Real format with a 64-bit mantissa, and T_i and M_i are in Intel 32-bit integer (IBM/Microsoft Pascal INTEGER4) format. (Note that the Intel format loads and stores register contents in "reversed" order, i.e., with the low order byte coming first in memory, so that the text bytes are processed in the order 4, 3, 2, 1, 8, 7, 6, 5, etc.)

In the worst case, the total resulting from the alternating sign terms could range from -2^{33} to $2^{33} \cdot 4$, in which case the squaring operation would produce a value as large as 2^{66} . Because the operation is carried out in floating point an overflow cannot occur, but a number that large cannot be represented in the 64-bit mantissa without loss of precision. If the 8087/80287 control word status were set to enable the precision interrupt then an interrupt would occur in that event, but the normal Pascal setting is to disable such interrupts. The result in the normal case will therefore be to round up or down to the nearest even value as appropriate (assuming the normal setting for the rounding mode), and discarding up to four low order bits of the sum. It should be noted that for precision loss to occur, the signs of the 32-bit result of the XOR must be +, -, +, -, to match the order of operations. As a result, it would be extremely unlikely for a loss of precision to occur on all four of the 32-bit intermediate result computations because of the way the text is cycled through the algorithm. In addition, if the intermediate result is viewed as the sum $2x + y$, where x represents the 31 high order bits and y the two low order bits, then the square is $4x^2 + 4xy + y^2$. Therefore, even though the low order y^2 bits are dropped after the multiplication this does not mean that the low order bits of the original quantity are ignored, since they affect the mid-square ($4xy$) component of the result. For this reason it is not possible for the low order bit or bits of one or more of the 32-bit words of text to be changed without causing a change in all 128 bits of the result.

The 8087/80287 FPREM instruction computes an exact remainder by successive subtractions the way division is done by hand, instead of using the more usual technique of dividing, rounding, multiplying, and subtracting from the original. The FPREM instruction is as fast as a divide, and is guaranteed to be accurate, without any roundoff. However, because the modulus is slightly less than 2^{31} and the maximum value of the result after the squaring operation is 2^{66} , the FPREM operation is not guaranteed to be completed in one operation (since the difference in magnitude between the dividend and the divisor may be larger than 2^{64} and FPREM shifts at most 64 bits in one operation), but it will always be complete in two operations. For this reason, the 8087/80287 condition code is tested after each FPREM and an additional FPREM performed if necessary.

In order to produce the fastest possible implementation, the XORs and other CPU instructions are executed in parallel with the coprocessor addition, subtraction, multiplication, and FPREM operations whenever possible. The FWAIT instructions necessary to ensure that the coprocessor has finished with its computations before the CPU reads the results are delayed as long as possible to permit the maximum possible overlap. Although the original version was coded using a macro that was invoked four times for the four different iterations within one block, in the final version the code was "unwound" and hand-optimized to permit maximum overlap.

On an IBM-PC with an 8088 & 8087 and a 4.77 MHz clock, the time to MDC check 1,000 512-byte blocks was 43.5 seconds, or 1359.5 microseconds per 16 bytes. This corresponds to 94.2 kilobits

per second. By comparison, the time for the fastest known software implementation of DES for the PC is 2801 microseconds per 8 bytes for the PC (22.8 Kbps, or 171K bytes per minute). With an 80287 speedup kit (consisting of an 8 MHz 80287 with its own clock crystal on a plug-in daughter-board) installed in an IBM AT with the standard 6 MHz 80286, the same test took 813.6 microseconds for 16 bytes (157.3 Kbps), or 1.18 megabytes per minute, compared to the DES time of 933 microseconds per 8 bytes. We are currently awaiting the availability of the new Intel 80386 CPU together with the 80387 coprocessor to time that configuration. We expect to recode the algorithm to take advantage of the new 386/387 instructions, and anticipate that the result will run about 4 times faster than on the IBM AT. Depending on the clock speeds of the processors involved, then, the 128-bit MDC technique is anywhere from 4.6 to 8.1 times faster than computing two independent 64-bit Message Authentication Codes in software using the fastest known software DES implementation for the IBM PC or AT.¹⁶ From a human factors standpoint, this means that the entire contents of a floppy disk (362K bytes) can be authenticated to the most stringent standards in less than 15 to 30 seconds on current microprocessors, without benefit of any special cryptographic hardware.

4.1 MDC Test Program

The following program, written in IBM/Microsoft Pascal for the IBM PC, can be used to verify the proper operation of the QCMDCV4 algorithm:

```

{ $TITLE:'CHECKMDC' - Verify MDC algorithm.}
{ $FLOATCALLS- (Generate native 8087/80287 code.)}

PROGRAM checkmdc(input,output);

TYPE
  checksums=      ARRAY[1..4] OF INTEGER4;

VAR [PUBLIC]
  text: PACKED ARRAY[1..33] OF CHAR;
  text_p:          ADSMEM;
  n_bytes:         WORD;
  result:          checksums;
  i,j:             INTEGER;

VAR [EXTERN]
  mdc_name:        PACKED ARRAY[1..8] OF CHAR;
  {"QCMDCV4 "}

CONST
  mdc_init =      checksums(
                  141421356,
                  271828182,
                  314159265,
                  57721566);
  check =          checksums(
                  -1900412449,
                  676867420,
                  -689076088,
                  1333643940);

```

16. In addition, two independent 64-bit MACs are not believed to be nearly as secure as a single 128-bit MDC.

```

PROCEDURE mdc(  text_ptr:ADSMEM;
                n_bytes:WORD;
        VARS result:checksums);
        EXTERN;

BEGIN;

WRITE(output,
      'Verifying MDC routine (',
      mdc_name,')...');

FOR i:= 1 TO 33 DO text[i] := CHR(0);
text[1] := CHR(1);
text_p := ADS text;
result := mdc_init;

FOR i:= 1 TO 50 DO
BEGIN;
  IF i<34 THEN n_bytes := WRD(i)
  ELSE n_bytes := 32;

  mdc(text_p,n_bytes,result);

  FOR j:= 32 DOWNTO 1 DO
    text[j+1] := text[j];

  text[1] :=
  CHR(LOBYTE(LOWORD(result[4])));

END;

IF result[1]=check[1] AND THEN
  result[2]=check[2] AND THEN
  result[3]=check[3] AND THEN
  result[4]=check[4]
THEN WRITE('OK.')
ELSE
BEGIN;
  WRITE('MDC is INCORRECT!');
  WRITELN(result[1],result[2],
          result[3],result[4]);
END;
WRITELN;

END.

```

5 Summary and Conclusions

Several architectural justifications have been presented an authentication algorithm which does not require a traditional crypto "black box" approach using secret cryptographic keys, with all of the key management difficulties that entails. In particular, the relatively common practice of using link encryption for secrecy at the OSI Data Link layer and implementing end-to-end authentication at the Presentation Layer would profit from "keyless", non-cryptographic means of authentication that could be easily implemented in both PCs and general-purpose main-frame computers.

The need for a checksum on the order of 128 bits in length was reaffirmed, both in the case of two mutually suspicious, potentially deceitful users where one may attempt to defraud the other, and in the command and control case where the attacker may have an almost unlimited ability to attempt to spoof the system. Contrary to the author's previous position, it was concluded that the 64-bit Message Authentication Code (MAC) approach of FIBS PUB 46 cannot be considered sufficiently strong in the case where the originator of a message may attempt to defraud the recipient, as well as in some command and control and multi-level security situations.

The MAC checksum technique used by ANSI X9.9-1986 is viewed as particularly unfortunate, both because of the inadequate 32-bit length and because no provision was made to distinguish between short block that was padded and a block that is a multiple of 8 bytes that happens to end with the same characters.

Coppersmith's Triple Birthday attack as it applied to the original QCMDC algorithm was summarized, and it was concluded that in order for that attack to be defeated it was necessary to ensure that the checksum function is not invertible, and that the length of the checksum be on the order of 128 bits in length.

The QCMDCV4 algorithm was described, which uses XORs plus a history function to ensure that the function is not invertible. The function computes a 128-bit result that is an over-determined function of 128 bits of the text and the 128-bit MDC result of the previous text block than cannot be decomposed. A "birthday attack" against the QCMDCV4 result cannot succeed, because of the enormous number of variations that would have to be computed, sorted and compared. In order to ensure that a message that is not an even multiple of 128 bits can be distinguished from the same message extended with zeros, the algorithm is executed $N + 1$ times on the last buffer, which contains the last N bytes of data extended with zeros.

The QCMDCV4 algorithm is recommended for use in microcomputer and main-frame applications where encryption will be provided separately and it is desirable not to have to replicate the encryption function for authentication. It is also suitable for use in combination with a public-key algorithm when implementing a digital signature function to protect against fraud.

Electronic Funds Transfer Point Of Sale In Australia

*Ralph Gyoery
Jennifer Seberry*

University of Sydney

ABSTRACT

The Australia wide eftpos systems was developed by the Australian Retail Banks to meet Australian conditions including a small population, which overwhelmingly uses cash for transactions, a small number of banks capable of "exchange of value" settlements and enormous distances. This paper discusses the system that has evolved first involving only ATM's and banks, then extending to POS systems for retailers and non bank financial institutions.

1. An Introduction to Australian Banking.

1.1 Before Deregulation.

The control of the Australian money supply was the responsibility of the Reserve Bank, a statutory body which had many powers such as setting the foreign exchange rate, issuing currency, issuing Australian Government bonds (thus determining interest rates), monitoring the monetary supply, determining the statutory deposit rate (the proportion of bank deposits which had to be deposited with the reserve bank) and generally guaranteeing the stability of the currency and banking system.

There were seven national and a small number of state banks which had licences to issue cheques and access to cheque clearing facilities and thus the exchange of value. All these banks had a large number of outlets so that in the cities there would be an outlet every one or two kilometres.

All these national banks had associated savings banks with which the general public could deposit funds, generally at a very low interest rate, and borrow funds for purchasing housing. Housing interest rates were controlled by the Government via the Reserve Bank. There were a small number of savings banks not associated with cheque issuing banks.

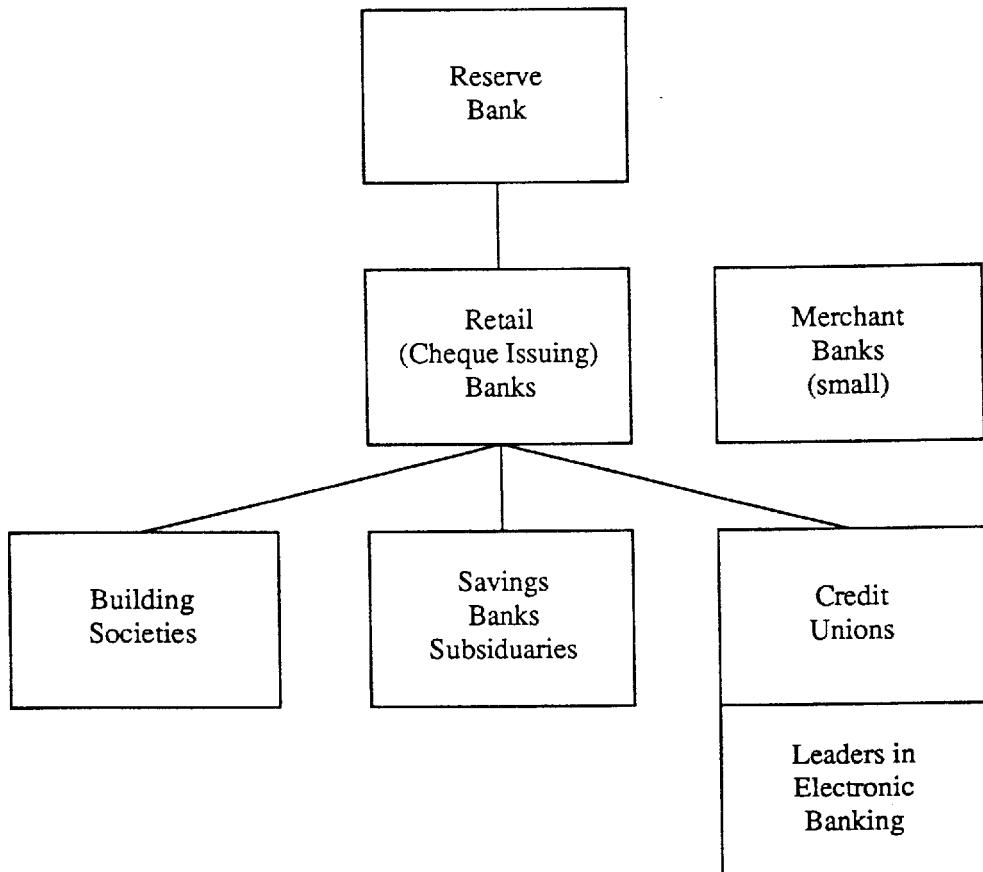
Savings banks only lent a maximum of 65% of the value of a house and so non-bank financial institutions, building societies arose. The general public made deposits with building societies by buying shares in the society. Societies lent up to 95% of the valuation of a house, at a slightly higher interest rate than savings banks, paid a higher interest rate on deposits and wrote customers cheques on the society account as required. They marketed their services aggressively and won considerable public support. Building societies also had a large number of outlets and longer opening hours than banks.

Still there was a gap in the market for the provision of consumer finance other than via finance companies, mostly also associated with banks, which often charged extremely high interest rates. The banks met this opening by issuing an Australia wide indigenous credit card - the Bankcard.

The other main move into this market segment was via Credit Unions which were originally organisations based on a membership with some commonality of interest, e.g. working for a University or the public service. Members joined the Credit Unions by buying a share and thereafter could deposit and borrow funds at attractive rates. The Credit Unions, via EFT Pty Ltd, forced the pace of electronic banking in Australia, they market aggressively via T.V. and have wide consumer acceptance.

There was a small number of merchant or wholesale banks but these usually had only at most a handful of offices in Australia and were invisible to the general public.

Figure 1: Australian Banking before Deregulation.



1.2 After Deregulation.

In the early 1980's it became apparent that there was considerable speculation against the Australian currency at a time when there was considerable volatility between the exchange rate of foreign currencies and high domestic inflation. This ultimately led to the Government's decision to deregulate the exchange rate and allow the Australian dollar to float in the international currency market.

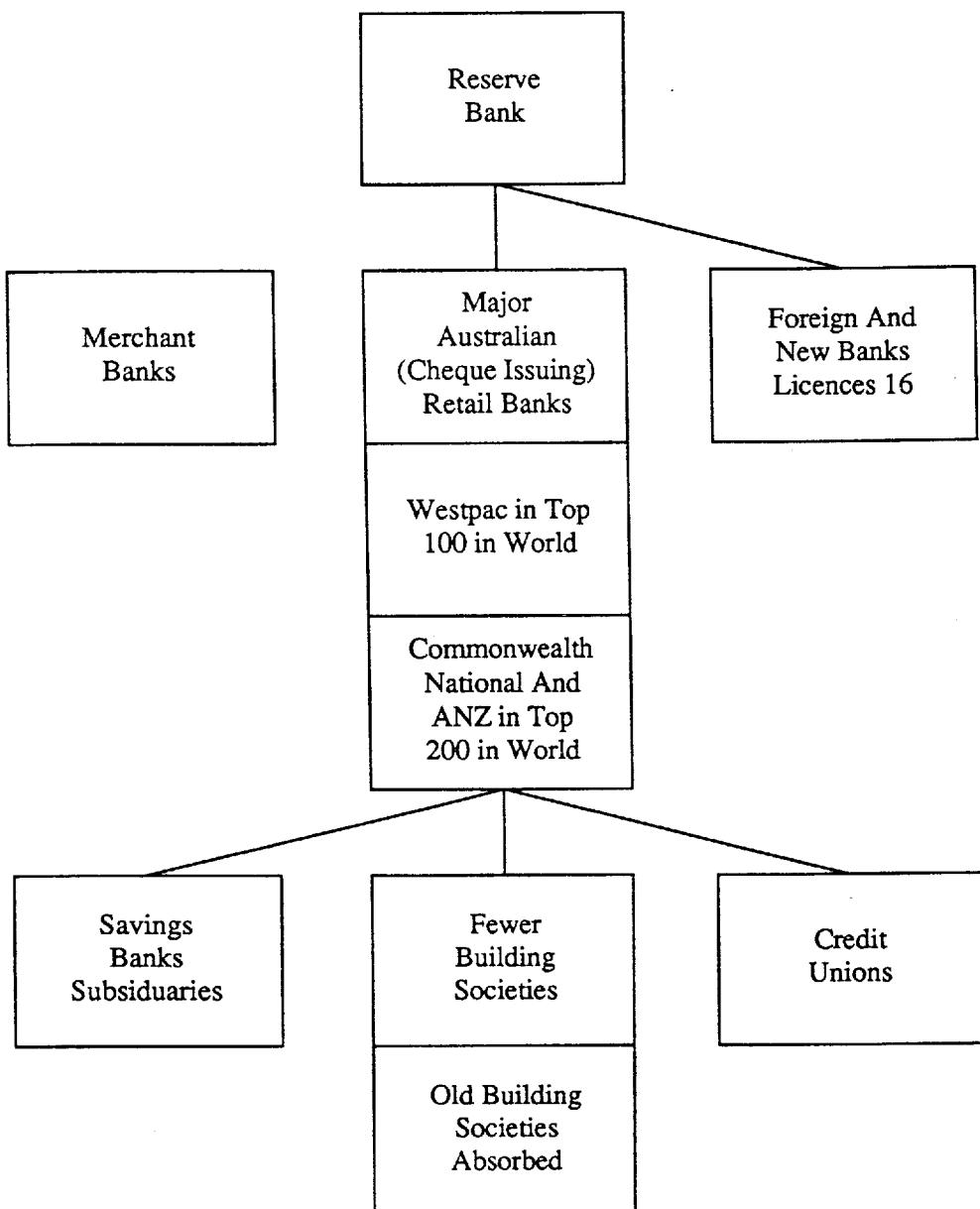
At the same time the banks within Australia had an effective cartel so there was no competition. The Government announced that it would issue 16 new banking licences and that foreign banks would be able to apply for licences.

The indigenous banks would have to become bigger in order to survive in a world wide competitive market and there was a number of mergers leaving four major Retail Banks. Another cheque issuing bank (the State bank) which did not have a savings bank subsidiary acquired a building society, a building society issued a banking licence became a bank (the Advance bank) and other organisations issued licences are trying to acquire building societies in order to obtain retail outlets.

At the moment the four major Retail Banks, the ANZ, the Commonwealth, the National Australia and Westpac have extensive retail outlet networks in place gaining a major advantage in the Australian market.

Many building societies, some very large, survive as have credit unions and merchant banks.

Figure 2: Australian Banking after Deregulation.



1.3 Debit And Credit Cards.

Bankcard, the indigenous credit card, has gained wide acceptance in Australia with 5.5 million cards issued to a population of 15 million.

Mastercard and Visa made an abortive entry to the Australian market in 1979-81 but were withdrawn. They re-entered the market, with different conditions attached in 1984 and have been aggressively marketed by some banks who wanted to discontinue Bankcard. Nevertheless Bankcard, which has very extensive market penetration in Australia and New Zealand survives.

All banks and non-bank financial institutions have issued debit cards for use with ATMs and POS.

2. Development of eftpos in the Australian Context.

In the development of any commercial activity there are obviously a range of factors which are not only crucial to the establishment of the activity, but also its ongoing success. For convenience of classification, such factors can be defined as:

1.Commercial/Strategic considerations

2.Technical considerations

3.Socio-political considerations

The financial industry in Australia, like those in other countries, has undergone a process of rapid transformation over recent years. This transformation can, in large part, be attributed to two factors.

- The first is the move towards the deregulation of financial markets and,
- the second, increased utilisation and application of technology to a range of banking activities.

Within this environment, Australian Retail Banks have had to face two major challenges.

First, the need to meet the intensification of competition between a range of financial institutions by providing customers with the services they demand. In particular at the retail end, convenient and speedy access to customer funds when and where they want them.

Second, Australian Retail Banks have had to seek to contain and, over a period of time, reduce the cost pressures which operate on the organisations of their size.

Within Retail banking, eftpos plays a major role in both of these areas. The level of branch banking representation in Australia per head of population is significantly higher than most other countries. Eftpos is seen as an important means of reducing the reliance of the customer base on an expensive, labour intensive, and until currently, paper dominated, branch banking system.

The marketing thrust in this respect is to achieve a much lesser reliance, for simple debit and credit transactions, on the expensive one to one banking operation with its attendant reliance on paper documentation. The Australian Retail Banks have sought, therefore, to achieve a higher level of direct crediting of salaries to bank accounts. Once the Australian Retail Banks have this money, they are seeking to ensure that further transactions - whether they be cash debits for food purchases; further credits or payment of regular accounts - occur electronically. The Australian Retail Banks are now looking to encourage their customer base to also to use Point Of Sale.

The importance of succeeding in this strategy of achieving a higher level of direct crediting of wages is revealed by the fact that approximately 70% of employees in Australia are still paid in cash. As a comparison - less than 5% of wage and salary earners in the USA, Canada and France are paid in cash. We note that in 1985 Australians used cash for 90% of all transactions.

For people to utilise a new form of financial delivery system - which is what eftpos is - some change in banking habits is required. It has been found that Australians - particularly young Australians - are not hesitant to use new technology in banking - eftpos provisors have had to ensure that the placement of POS terminals and the promotion of the overall eftpos system is well planned and executed to encourage utilisation.

In this respect, market research has revealed that the largest outgoings from the household income, apart from mortgage or rent payments are for

- food and household goods, and for,
- petrol and other motor vehicle related expenses,

Further, research has indicated that people shop for food or buy petrol at least once a week.

As a result of this market research, locating eftpos terminals, particularly during the development phase, in retail outlets such as supermarkets and petrol stations, has been the most appropriate means of achieving the aim of providing customers with a delivery system which enabled the customers to gain access to their funds at convenient times and convenient locations.

When you consider that all four major Australian Retail Banks are actively involved in eftpos developments it becomes very clear that Australia has made a substantial commitment to this form of banking technology and is a significant way down the road to achieving a fully integrated system.

We will now consider the structure of the eftpos system in Australia. This is a subject of considerable importance and one which has excited considerable debate not only internationally but also in Australia.

There are a number of options or alternatives which were proposed in relation to the development of eftpos systems. While there are a range of variations, two of the major options can be defined as follows.

First, there is a centrally owned and operated system, often called a single access system. This system could be controlled, for example, by a public utility; be owned by a private company or established as a consortium of various interests and groups. It is basically this structure which the British have been pursuing and which, we believe, is being developed in Singapore. In Australia it would have to be achieved by using Telecom Australia's Austpac Switching Network or a private switching network and be jointly administered by all participants.

Second, there is the approach initially being strongly advocated by the four major Australian Retail Banks. It is a decentralised system, known as the "gateway" structure, in which management and control of the machinery and switching system is in the hands of a number of players - in Australia's case the banks - that cooperate for the exchange of value. As initially proposed it was to be a closed system with retailers tied to their bank.

A central feature of the gateway approach to the development of EFT, particularly with reference to POS, is that it is a component in the exchange of value. Inherent in such an arrangement is the ability to guarantee to a receiving party that an exchange of value will occur consistent with a message transaction through the EFT system. The very concept of EFT is to transfer funds and this consideration had a bearing on the insistence of the four major Retail Banks that only they could guarantee the exchange of value.

The four major Australian Retail Banks were looking for an eftpos system which would, as far as practicable, be driven by market considerations and be market sensitive. They saw such a precondition as essential to the fulfillment of the requirements of the main participants in any eftpos system.

The Australian Retail Banks' belief that the gateway approach offered the best means for actually getting eftpos established in Australia was probably well founded. They foresaw that intractable delays and immense administrative, logistical and financial problems were associated with an approach which sought to bring together a range of interests and participants to

develop a complex set of commercial arrangements.

The Banks had observed the United Kingdom experience, and believed it was both untenable and ultimately not in the best interests of the banks or their customers to attempt to duplicate what this country had set out to do. The Banks went into eftpos in an effort to gain a market advantage. Some of the Australian Retail Banks have obtained this market advantage, and at the same time, not sacrificed the ultimate integrity and viability of a fully operational and interlinked system between a wide range of financial institutions, retailers and consumers.

A major disadvantage to the gateway approach is the restrictions imposed on the market by excluding from or giving only limited access to new retail banks, non bank financial institutions such as building societies (which are a major source of housing finance), credit unions (which provide extensive services to members of cooperatives), merchant banks and insurance companies.

The gateway concept as initially advocated has now been modified in light of the vocal and valid criticisms by the non Retail Bank sector that it was being effectively excluded from the market and the Retail Traders Association that it did not adequately service all their customers. Agreement has now been reached that the eftpos terminals will accept all bankcards, debit and credit, as well as other cards such as American Express.

The major Retail Banks saw the "gateway" leading to more rapid implementation breaking through the log jam of establishing a central system. They also saw the gateway as building incrementally on existing relationships between participants. The very essence of the gateway structure is a modular or incremental set of stages leading to full integration. The three main modular stages can be briefly described as follows.

3. Modular Stages.

3.1. Stage One.

Stage one is a simple single connection between a retailer and a bank with only the cards issued by the participating bank being accepted. This is the stage Australia has already gone through. It is basically a process of experimentation, testing and verification.

3.2 Stage Two.

Stage two is an extension of the network to other **settlement capable** Australian Retail Banks, with the cards issued by those banks also being accepted. Note that we are talking about **settlement capable** financial institutions, which relates to our earlier point that eftpos is about the exchange of value. In Australia, it is only the banks which have this settlement capability, through our central bank - the Reserve Bank.

3.3 Stage Three.

Stage three further extends the network, this time to include other **non settlement capable** financial institutions. In Australia's case these are building societies and credit unions. Such institutions gain access to the system on an agency or sponsorship basis, whereby a bank undertakes to settle on their behalf.

Returning to the point of building on existing commercial arrangements between participants, one must realise that eftpos is first and foremost a set of complex commercial arrangements between the participants. One of the key strengths of the gateway structure is that it utilises existing relationships. These relationships can be identified as those:

- between a bank and retailer
- between a bank and the clearing house

- an inter settlement capability between banks
- a relationship between a retailer and a non bank financial institution and, finally
- between an non bank financial institution and bank,

To illustrate this point, let us look at the initial part of the process - the relationship between a bank and retailer. In the case for Australian Retail Banks, one particular bank, Westpac, utilised an existing banking relationship with a supermarket chain, Woolworths Ltd, and extended it to the area of eftpos. The supermarket chain were confident that the eftpos system being introduced would be secure, reliable and, above all else, guarantee payment at the end of the day. Further, the supermarket chain was confident that if the bank entered into a relationship with another Retail Bank or non bank financial institution for the use of their plastic cards in the eftpos system, the same standards would apply and that they would still be assured of receiving value for transactions.

This incremental approach also offered a high degree of flexibility. It recognised that POS would ebb and flow as market forces determine the balance between demand and cost of provision. It also recognised - especially during the initial stages - that the different participants would wish to determine investment decisions, and to make decisions on longer term involvement on the basis of observing actual operation and development. The gateway structure allowed this to happen. It was not designed to lock participants in at the outset.

Another feature of the gateway system which is quite important to the private sector is it allows for the development of commercial relationships between participants based on market considerations. A fee structure could then be based on negotiation. Many Retailers, however, feared that the gateway, which controls access for other financial institutions, would effectively act as a Retail Bank cartel limiting free market operations. They foresaw that the banks would want them to pay for eftpos arguing that they provided retailers with access to their customer base. The Retailers argued that they

made the sale and the banks should pay for the system. Neither wanted to pass the cost on to the customer who might then revert to paying cash.

The Australian Retail Banks expressed the view that the gateway could only be conducive to innovation and to the effective servicing of all participants - whether they were retailers, customers or the financial institutions.

Very little technical difficulty has been experienced with the majority of the terminals in operation in Australia. Customer usage of the system is increasing steadily, indicating that people are adapting to this form of technology. For example, the Combined Credit Unions Rediteller network, which was the first network of ATM's in Australia, was handling 7000 transactions per month in October 1985. Banks with ATM networks were handling 2-3000 transactions per month at that time.

Also in Australia we have the development of bilateral arrangements between financial institutions, such that the card of one institution is accepted at the terminal of another institution. In the case of Australia's non settlement capable institutions - the building societies and credit unions - agency arrangements have been actively progressed between such institutions and banks. These arrangements were to ensure that the customers of non settlement institutions gained access to the eftpos system while the central element of that system - the guaranteed exchange of value - was maintained.

Let us now outline a few details of a particular eftpos system's operation in Australia. The system gives its customers access to their cheque account and savings account at retail outlets. At present about 4000 outlets are available in supermarket chains, chain stores and petrol retailers.

The customers can access their accounts to pay for goods and obtain some cash (if desired). The amount of cash that can be obtained is set by the individual retailer. For example, at the supermarket the sum of two hundred dollars cash can be obtained (US \$130), whereas at the petrol outlets the amount is thirty dollars (US \$20). In the supermarkets the terminals are

located at the check-out lanes, while at the petrol outlets they are positioned at the service desks.

To operate the system the customer requires a plastic card particular to that financial institution, together with a four digit PIN (Personal Identification Number).

To use the system the customer hands his/her card to the cashier/operator who wipes the card through the terminal and then enters the amount of the transaction. The customer then authorises the transaction by entering his/her PIN through a separate hand-held key pad which we call the Pin Pad. The terminal then communicates with the financial institution's computer and the funds transferred from the account of the customer to the account of the retailer. This takes on average only fifteen seconds although transactions may be travelling up to 10,000 kilometres.

Upon successful completion of the transaction, the customer is issued with a printed receipt, details of the transaction (duplicate of which is held on the tally role in the terminal) are recorded on the customer's statement of account.

It should be noted that reliability and speed of the system is essential to both customer usage and retailer acceptance.

Up until March 1985, the financial institutions only provided a debit facility to its customers. Now there is the option for customers in States where regulations permit, to have access to their line of credit.

4. Technical Aspects Of Eftpos in Australia.

The quality of any system is dependent on the quality of the constituent parts and the level of investment and expertise devoted to its development. This is certainly the case with eftpos which involves the transmission of financial information and which is, as such, an element of the payments process of a country.

The Australian Retail Banks have invested heavily in the area and required equipment and a system which guaranteed high reliability, high integrity and security of customer financial information.

LM Ericsson Pty Ltd., Burroughs and Fortronics have been responsible for fulfilling many requirements in relation to terminals which were required not only to be compact in size, but to be easy to use, fast in response time between terminal and host computer and have very secure in its encryption of raw data.

Transmission of messages between the terminal and the bank's computer system utilises either Telecom Australia's Austpac Switching Network, for the Commonwealth Bank and Westpac, or a private switching network, for the ANZ and National Australia Bank. Each Bank uses its own facilities as a frontend to large mainframes. The host provides on-line enquiry and updating facilities on customers' fund positions. The frontend also performs other functions, such as :

- Verification of customer Personal Identification Numbers
- Accumulation of individual merchant's settlement position, and
- The interchange of information between financial institutions.

Figure 3: Commonwealth - Westpac System.

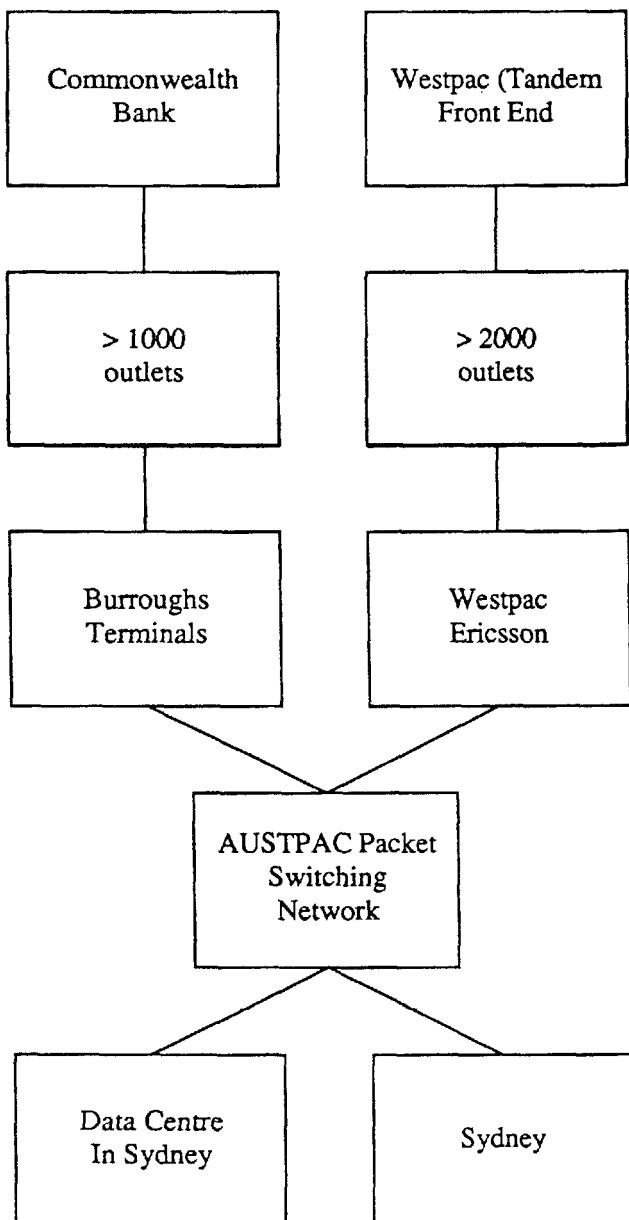
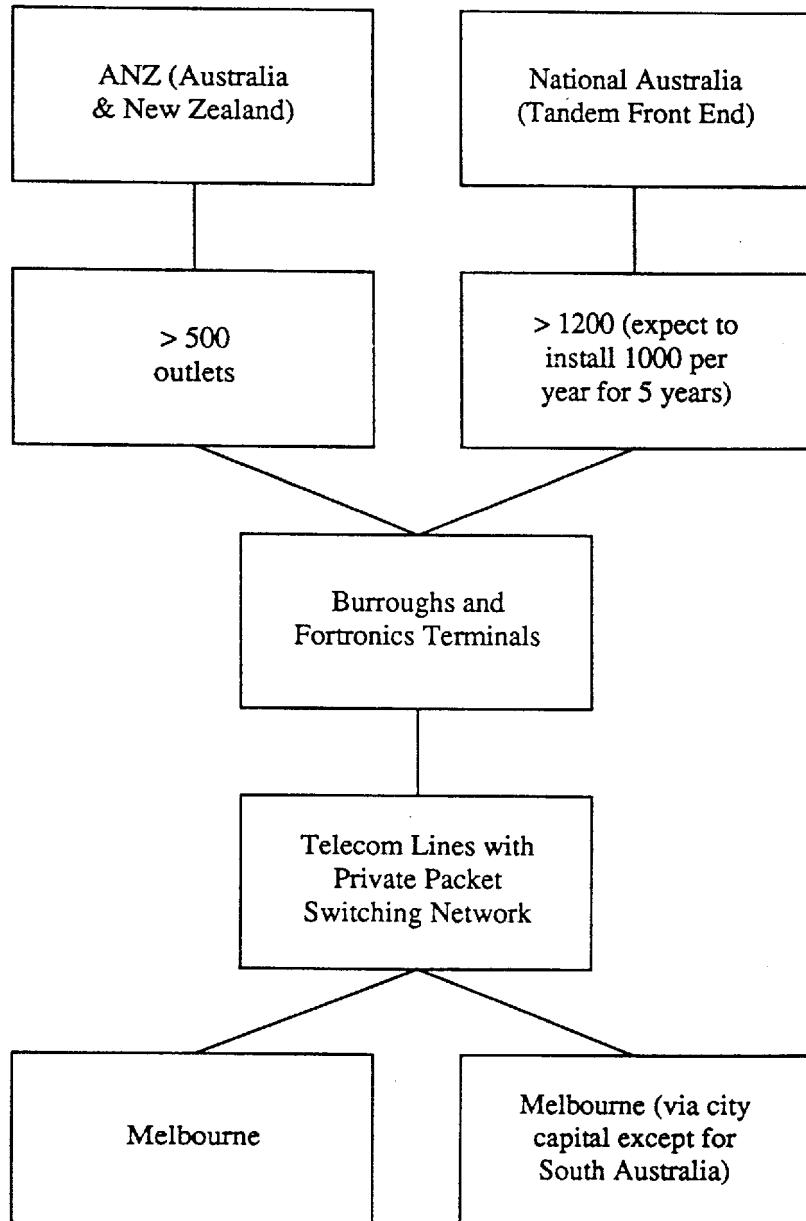


Figure 4: ANZ - National System.



5. Case Study - Westpac.

The development of the Westpac eftpos system will be now given as a case study. The establishment of Westpac's eftpos system resulted from detailed planning and development over a nine month period. The first step in this exercise was the selection of a project manager and team encompassing key people from all areas :

- bank personnel from technical, operational and retail areas
- Telecom Australia responsible for the National Telecommunication Links; and
- Westpac's retail partners - Woolworths (Supermarket chain) and BP (petrol company).

The selection of a suitable equipment supplier was of prime importance in completing the project team. Although a number of POS systems (usually credit authorisation) were in operation in other countries, none met the requirements of the Australian environment. Westpac were looking for a supplier who would build a terminal to their specifications instead of offering a modified off-the-shelf solution.

LM Ericsson Pty Ltd had the requirements for the development of a terminal according to Westpac's specifications, which detailed the hardware and software requirements. Some of the major requirements were :

- compact in size (height and footprint to enable easy location at service desks and check-out lanes)
- in built modem
- dot integrated printer
- low noise levels during operation

- PIN pad no larger than a small calculator
- clear and easily understood display/prompts to both operator and customer
- quick response times between terminal and host computer to ensure fast service
- in short, Westpac wanted a terminal and PIN pad that was simple to use for both the operator and customer.

In addition, the overall security of the system was a key issue and a high standard of encrypted message formats were adopted to ensure "raw" data was not transmitted through communication links to Westpac's host computer. At no time during a transaction was any form of confidential customer detail to be displayed at the terminal.

The overall project was monitored with the use of critical path charts and action plans detailing the tasks of all parties. Development was not without problems but the planning and monitoring process highlighted shortcomings and enabled contingent actions to counteract potential delays.

A pilot launch of 20 terminals (10 in Woolworths and 10 in BP sites) commenced in April 1984, enabling Westpac to test the system in a live environment and iron out and errors before Westpac moved to any expansion of the network. The store managers and service station managers at the 20 pilot sites cooperated fully and were happy to use their respective sites as controls to trial this new technology.

From the experience of establishing the Westpac eftpos system, a few key points have surfaced, which should not be overlooked during the planning of any eftpos project :

- installation of communication link between retailer and bank. The physical connection is by the use of a dedicated line for the sole use of EFT messages and at certain retail premises, major alterations were required

to permit installation. To avoid delays and unforeseen costs, availability of access should be considered, as well as allowing sufficient time for installation.

- testing of the system and equipment - prior to installation of any equipment, the entire system was subjected to extensive testing to eliminate the problems of a "live" environment. The live situation presented some problems. However with the cooperation of the 20 pilot sites, these were overcome in a short space of time (approximately three months). A close involvement and mutual commitment between retailer, Telecom and financial institution is essential for success.
- adequate training of financial institution and retailer staff is also an important pre-requisite. Customer usage of any form of "hands on" technology requires a degree of customer confidence. An encouraging employee, and associated educational campaigns, can greatly enhance this confidence, or, alternatively deflate it. This is backed up by a central "help desk" staffed by trained bank personnel. A phone call to the "help desk" provides immediate action to most situations.
- Site selection of POS terminals is clearly an important consideration. To ensure the best return on an investment, utilisation must be maximised. This can be achieved by careful site selection based on analysis of demographic movements and shopping habits.
- Adequate backup in the case of system malfunction is also something that must be closely considered in any new system. In Westpac's case, a simple paper-based back-up approach was introduced to deal with any system failures or disruptions.
- Final, the marketing and promotion of eftpos is essential. As with any retail product a clear marketing strategy must accompany the launch. Without this, eftpos starts well behind the barrier and will suffer the fate of any poorly marketed product. Incentive campaigns targeted at the card holders have been successful in improving awareness and encouraging use.

5.1 Key Factors For Success.

In this section we will highlight some of the key issues of the planning, development and launching process as experienced by Westpac :

- Development Plan - This must include stretching but realistic time frames and objectives to ensure deadlines are achieved. Loose time frames will lead to delays.
- Total Commitment From All Parties - The absence of support from all concerned will inevitably result in delays and possibly failure.
- Retailer Involvement - As these people have a large influence on the success or otherwise of a project of this type, their input to the development process is considered mandatory.
- Reliable Equipment Supplier - Delivery where and when required is a must.
- Simplistic Design of Terminal - It is considered that ease of operation of the terminal can mean the difference between success or failure. Above all, the need for strong project management and control cannot be over emphasised.

5.2 The Future For Westpac.

Westpac is currently developing :

- Automated fuel pumps - the oil industry has expressed the need to relocate eftpos transactions from the shop to the forecourt of a petrol filling station. A card reader and PIN pad installed on the forecourt would provide customers with a fully self serviced facility - both petrol and eftpos
- Integrated cash registers - in this development the eftpos terminal would be fully integrated with the cash register and the electronic price

scanning technology currently being developed. This has obvious advantages for retailers in terms of reducing space utilised and ensuring comprehensive inventory and financial control.

A major move by all Australian Retail Banks is to integrate their networks with the other bank networks thus allowing card holders of one bank to use the Automatic Teller machine of another bank. This is also an active area of development and installment for Westpac.

5.3 Westpac Handyway System.

- June,82: Firmed up on concept
- December,82: Negotiations with potential partners
Westpac wanted
 - National Supermarket Chain.
 - National Service Station Chain.
- March,83: Firmed up negotiations with
 - Woolworths and BP.
 - Talked with potential terminal suppliers.
 - Commenced negotiations with Telecom.
- June,83: Project formally approved by bank's executive committee and board. Project team set up.
- July,83: Ericsson selected as terminal supplier. Tandem selected as front end.
- December,83: Press release re POS project Westpac, BP, Woolworths.
- April,84: Pilot launch 20 terminals in Sydney area.
 - Expansion of Network
 - Open access agreed to by all interested parties, interchange discussions, plans.

6. Socio-Political Aspects of Eftpos.

Every commercial activity has the ability to affect the social and political environment or in turn be affected by it. Eftpos is certainly no exception and is perhaps a good example of how social and governmental factors can impact upon a commercial initiative.

Within Australia, the development of EFT, most notably in the area of POS, has been associated with the expression of a number of concerns within sections of the community and from some interest groups which seek to represent consumer interests. It has been suggested, often quite vociferously, that eftpos poses a serious threat to consumers in areas such as :

- security and confidentiality of financial information,
- liability in the event of a disputed transaction or system failure, and
- fraudulent usage.

The Australian Retail Banks have accepted the basis of such concerns as well intentioned even if they are not always based on an objective understanding of eftpos operations.

For example, it has been claimed that eftpos would allow a financial institution to build up a detailed picture of the purchasing habits of individuals. For example, in the event of an individual buying a device for the use of illicit substances, such as a water pipe for smoking marijuana, this information it has been claimed, could be made available to drug enforcement authorities.

The Australian Retail Banks have sought to face the social and governmental aspects of eftpos development responsibility and comprehensively. Discussions with consumer groups, administrators and politicians at all levels of government have been established to understand better their concerns and convey the views of the Australian Retail Banks.

It has been argued by some that there is no established set of legislative arrangements to define rights and obligations for the providers and users of eftpos systems.

The Australian Retail Banks' view is that it is concerned to maintain the same, if not higher standards, relating to their relations with customers with EFT as with paper based transaction methods.

The Australian Retail Banks believe that legislation should only be contemplated if there is, or is likely to be, a demonstrable public need. To the extent that problems from a consumer point of view may arise, they would certainly prefer that these problems be tackled on a voluntarily or on the basis of an established and uniform industry code.

So far problems which have arisen such as

- consumers being held liable in the case of errors caused by mechanical and systems failures of the computer
- consumers being held liable for money issued in excess of their balances

have not always been perceived to have been handled fairly.

If there are any lessons to be learnt on this aspect of eftpos operation from Australia, it is that it is essential that the technical development and planning for eftpos be closely associated with a consideration of the policy and political aspects. To do this averts, or at least minimises, the potential for misunderstandings between the private sector and government and the consequent possibility of inhibiting legislative arrangements.

7. Advantages Of Eftpos To The Main Parties.

7.1 For The Australian Retail Banks.

- Replacing paper with E.F.T (cost reducing)
- Better customer service
- Australian retail banks representation at retail outlets where customers shop and spend money
- Opportunities to broaden its customer base.

7.2 For The Retailer.

- Replacing paper with E.F.T (cost saving)
- Reduced cash (cost saving and better security)
- Better customer service
- Higher ticket sizes (increased sales)
- Same day value
- No bad debts
- No separate authorisation required
- Less physical security
- Access to Australian retail banks customer base.

7.3 For The Customer.

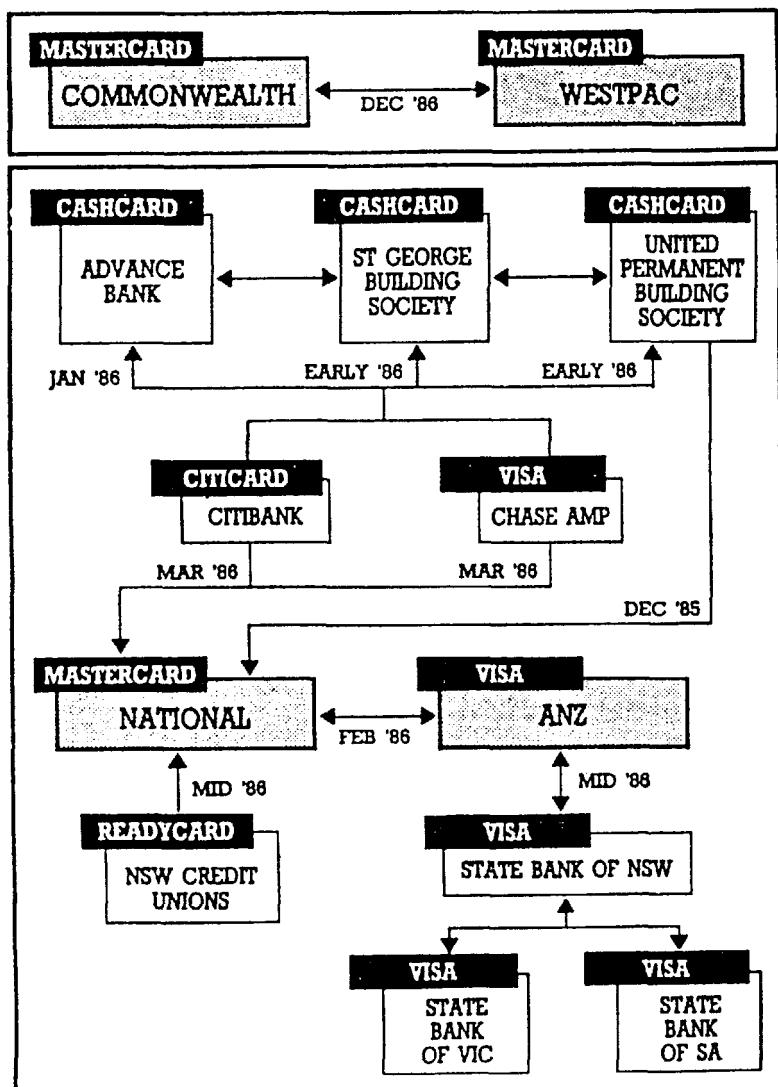
- The major benefit for the customer is convenience - being able to access their accounts where and when they spend.
- Another important benefit is seen as less need to carry cash.

8. The Future of Eftpos in Australia.

Agreement has now been reached so banks, non bank financial institutions, and retailers will all have complete access to the Australia wide eftpos network.

The linking up of ATMs is described in the diagram

Figure 5: Linking Up the ATMs (Sydney Morning Herald), 12/2/86.



9. Concluding Remarks.

It is our opinion that eftpos marks a challenging and exciting development in the provision of retail banking services. After two years of debate, in mid July 1986, all the banks and the retailers agreed to eftpos. It is the future of retail banking and as such an integral means of improving customer services.

Security companies alarmed at the inroads to their business have been putting out TV commercials that try to illustrate the pure joy and beauty of cash. But we believe that the high acceptance of plastic money, with 5.5 million Bankcards for a population of 15 million people, and the acceptance of ATM's especially by the young, means they are too late to dam the flood.

Westpac and the Commonwealth Bank both have programs to allow children as young as 12-13 key cards to allow access to savings accounts. Unfortunately some families, who have not kept their PIN number secure from their children, have found their accounts drawn upon.

We expect to see in the future

- more terminals
- more merchants/retailers using the system
- extended POS interchange
- automated fuel pumps or driveway card acceptors
- electronic cash registers more widely used
- universal access to line of credit
- an extension of the facilities available via the eftpos network to include items such as brokerage, loans, investments, travel services, certificates of deposit .

References.

- [1] Gareth Powell and Gene Stephan, **Banks, macro battle of the micro chips**, Sydney Morning Herald, 14 October 1985.
- [2] Anonymous, **Eftpos standards pose challenge**, Sydney Morning Herald, 14 October, 1985.
- [3] Kester Cranswick, **How Banks solve the cashless equation**, Computing Australia, 18 November, 1985.
- [4] Ben Bremner, **Credit card fraud - it's growing**, Sydney Morning Herald, 9 October 1985.
- [5] Anonymous, **Plastic money row on the cards as retailers challenge Banks**, Sydney Morning Herald, 16 April 1986.
- [6] Tony Healy, **Fears ebb as eftpos banks on co-operation**, Computing Australia, 7 July 1986.
- [7] Keith Dunstan, **Our cashless society heads back to the rum days**, Sydney Morning Herald, 22 July 1986.
- [8] Malcolm Wilson, **Coming soon the card that goes anywhere**, Sydney Morning Herald, 12 February 1986.
- [9] Stephen Hutcheon, **Pro-Bankcard forces plan services link**, Sydney Morning Herald, 24 April 1986.
- [10] Bill Paget, **Implementing the Electronic Funds Transfer Point Of Sale System (EFTPOS) - Factors For Success**, Speech delivered in Singapore, 1985.

The Notion of Security for Probabilistic Cryptosystems

(Extended Abstract) *

Silvio Micali[†]
MIT

Charles Rackoff
University of Toronto

Bob Sloan[‡]
MIT

Abstract

Three very different formal definitions of security for public-key cryptosystems have been proposed—two by Goldwasser and Micali and one by Yao. We prove all of them to be equivalent. This equivalence provides evidence that the right formalization of the notion of security has been reached.

1 Introduction

The key desideratum for any cryptosystem is that encrypted messages must be secure. Before one can discuss whether a cryptosystem has this property, however, one must first rigorously define what is meant by security. Three different rigorous notions of security have been proposed. Goldwasser and Micali[5] suggested two different definitions, polynomial security and semantic security, and proved that the first notion implies the second. Yao[8] proposed a third definition, one inspired by information theory, and suggested that it implies semantic security.

Not completely knowing the relative strength of these definitions is rather unpleasant. For instance, several protocols have been proved correct adopting the notion of polynomial security. Are these protocols that are secure with respect to a particular definition or are they secure protocols in a more general sense? In other words, a natural question arises: Which of the definitions is the “correct” one? Even better: How should we decide the “correctness” of a definition?

The best possible answer to these questions would be to find that the proposed definitions—each attempting to be as general as possible—are all equivalent. In this case, one obviously no longer has to decide which one definition is best. Moreover, the equivalence suggests that one has indeed found a strong, natural definition.

In this paper, we show that these notions are *essentially* equivalent. The three originally proposed definitions were not equivalent. However, as we point out, this inequivalence was caused only by some minor technical choices. After rectifying these marginal choices, we succeed in proving the desired equivalences, keeping the spirit of the definitions intact. We believe this to be an essential step in developing theory in the field of cryptography.

*The full version of this work will appear in the *SIAM Journal of Computing*. In the meantime, the full version is available from the authors.

[†]Supported by NSF grant DCR-8413577 and an IBM Faculty Development Award.

[‡]Supported by a General Electric Foundation Fellowship and NSF grant DCR-8509905.

2 Public Key Scenarios

Let us briefly review what is meant by the notion of public-key cryptography, first proposed by Diffie and Hellman [4] in 1976. As with all cryptography, the goal is that A(lice), by using an encryption algorithm E , becomes able to securely send a message m to B(ob). What is meant by “securely” is that it is impossible for any party T who’s tapped A and B’s line to figure out information about m from $E(m)$. The distinguishing feature of public-key cryptography is that we require this security property to hold even when T knows the encryption algorithm E .

We believe that until now, the notion of public-key cryptography has not been fully understood. In fact, it is crucial to consider exactly how the communication between A and B establishes the algorithm E . Therefore, we introduce the fundamental notion of a *pass*. We will first explain what passes are, and then explain their implications for security.

2.1 Passes

Within the public-key model, A and B can alternate communicating back and forth as many times as they feel are necessary to achieve security. Call each alternation a *pass*.

Any number of passes are, of course, permissible. We concentrate on what we believe are the two most interesting and important cases, one and three passes. We do not consider more than three passes, because, if trapdoor permutations exist, a well designed probabilistic encryption scheme can achieve as much security as is possible using only three passes.

Three-pass systems

The three-pass case is, perhaps, the most natural to think about. It corresponds to a telephone conversation. A has a message m that she wants to securely communicate to B. A calls up B and says, “I have a message I’d like to send to you.” B, so alerted, proceeds to generate an encryption/decryption algorithm pair, (E, D) , and tells A, “Please use E to encrypt your message.” A then uses E to encrypt her message and tells B $E(m)$.

Notice the key property of a three-pass system: The message and the encryption algorithm are selected independently of one another. We are nevertheless in a public-key model, since anyone tapping the phone line gets to hear B tell E to A.

One-pass systems

A one-pass system corresponds to what is commonly called a public file system. In the one-pass model, A simply looks up B’s public encryption algorithm, E , in a “phone book” and uses it to encrypt her message. (One pass is a slight misnomer. At some point, in what we may view as a preprocessing stage, B must have communicated his encryption algorithm, presumably by telling it to whomever publishes the phone book of encryption algorithms, and thus indirectly to A. “One and a half passes” might be more accurate. “Half” refers to the preprocessing stage that needs to be performed only once.) In this case, the choice of message *can* depend on E .

2.2 Passes and Security

The main result of this paper is:

GM-security, semantic security, and Y-security (all formally defined in section 3) are equivalent both for one-pass and three-pass cryptosystems.

Interestingly, the equivalence still holds in the one-pass scenario, but the notions of security vary between the one-pass and three-pass scenarios. This point has not been given the proper attention, because people frequently confuse the notion of one-pass public-key cryptography with public key cryptography in general.

The distinction, however, is crucial for avoiding errors, particularly in cryptographic protocols. Let us informally state the two definitions of security that are achievable in the two scenarios if trapdoor permutations exist.

A 3-pass cryptosystem is secure if, for every message m in the message space, it is impossible to efficiently distinguish an encryption of m from random noise.

A 1-pass cryptosystem is secure if, for every message m that is efficiently computable on input the encryption algorithm alone, it is impossible to efficiently distinguish an encryption of m from random noise.

In other words, in the one-pass scenario one cannot just blithely write, “For all messages m .¹” For instance, if one closely analyzes all known public-key cryptosystems, it is *conceivable* that if (E, D) is an encryption/decryption pair, then D can be easily computed from $E(D)$. For instance, the constructive reduction of security to quadratic residuosity given by Goldwasser and Micali [5] for their cryptosystem would *vanish* if the encrypted message is allowed to be D itself.¹

Such problems cannot arise in the three-pass scenario because the encryption algorithm E is selected after and independently of the message m .

In this paper we concentrate on providing the details for the three pass case, and sketch the results for the one pass case in the final section. The reason for this choice is that the definitions of security are much more easily stated for three-pass systems. It is much more convenient to say, “For all messages m ,” than “For all messages m that are efficiently computable given the encryption algorithm as an input.”

3 Notions of security for three-pass systems

In this section we will formally specify our cryptographic scenario, and define the three notions of security. These definitions are the same in spirit as those originally chosen by Goldwasser and Micali and Yao; therefore, we will use either the names they chose or their initials. We will point out explicitly at the end of this section the minor changes we needed to make to reach the right level of generality.

¹Notice that if Bob publishes an encryption algorithm E in the public file while keeping its associated decryption algorithm D secret, then any other user, being limited to efficient computation and ignorant of D , necessarily selects her message m efficiently from the input E —maybe without even looking at E —and perhaps other inputs altogether independent of (E, D) . However, in designing cryptographic protocols, one would often like to be able to transmit things like $E(D)$. For instance, if that type of message were allowed, one would have a trivial solution to the problem of verifiable secret sharing [3].

3.1 Notation and Conventions for Probabilistic Algorithms.

We introduce some generally useful notation and conventions for discussing probabilistic algorithms. (We make the natural assumption that all parties may make use of probabilistic methods.)

We emphasize the number of inputs received by an algorithm as follows. If algorithm A receives only one input we write " $A(\cdot)$ ", if it receives two inputs we write " $A(\cdot, \cdot)$ " and so on.

"PS" will stand for "probability space"; in this paper we only consider countable probability spaces. In fact, we deal almost exclusively with probability spaces arising from probabilistic algorithms.

If $A(\cdot)$ is a probabilistic algorithm, then for any input i , the notation $A(i)$ refers to the PS which assigns to the string σ the probability that A , on input i , outputs σ . Notice the special case where A takes no inputs; in this case the notation A refers to the algorithm itself, whereas the notation $A()$ refers to the PS defined by running A with no input. If S is a PS, denote by $\text{Pr}_S(e)$ the probability that S associates with element e . Also, we denote by $[S]$ the set of elements which S gives positive probability. In the case that $[S]$ is a singleton set $\{e\}$ we will use S to denote the value e ; this is in agreement with traditional notation. (For instance, if $A(\cdot)$ is an algorithm that, on input i , outputs i^3 , then we may write $A(2) = 8$ instead of $[A(2)] = \{8\}$.)

If $f(\cdot)$ and $g(\cdot, \dots)$ are probabilistic algorithms then $f(g(\cdot, \dots))$ is the probabilistic algorithm obtained by composing f and g (i.e. running f on g 's output). For any inputs x, y, \dots the associated probability space is denoted $f(g(x, y, \dots))$.

If S is any PS, then $x \leftarrow S$ denotes the algorithm which assigns to x an element randomly selected according to S ; that is, x is assigned the value e with probability $\text{Pr}_S(e)$. If F is a finite set, then the notation $x \leftarrow F$ denotes the algorithm which assigns to x an element randomly selected from the PS which has sample space F and the uniform probability distribution on the sample points. Thus, in particular, $x \leftarrow \{0, 1\}$ means x is assigned the result of a coin toss.

The notation $\text{Pr}(p(x, y, \dots) \mid x \leftarrow S; y \leftarrow T; \dots)$ denotes the probability that the predicate $p(x, y, \dots)$ will be true, after the ordered execution of the algorithms $x \leftarrow S$, $y \leftarrow T$, etc. We use analogous notation for expected value— $\text{Ex}(f(x, y, \dots) \mid x \leftarrow S; y \leftarrow T; \dots)$ —where now f is a function which takes numerical values.

Let \mathcal{RA} denote the set of probabilistic polynomial-time algorithms. We assume that a natural representation of these algorithms as binary strings is used.

By 1^n we denote the unary representation of integer n , i.e.

$$\underbrace{11\dots1}_n$$

3.2 Cryptographic Scenario

Here we specify those elements that are necessary for all public-key cryptography.

A *cryptographic scenario* consists of the following components:

- A *security parameter* n which is chosen by the user when he creates his encryption and decryption algorithms. The parameter n will determine a number of quantities (length of plaintext messages, overall security, etc.).

- A sequence of *message spaces*, $M = \{M_n\}$ from which all plaintext messages will be drawn. M_n consists of all messages allowed to be sent if the security parameter has been set equal to n . In order to make our notation simpler, (but without loss of generality), we'll assume that $M_n = \{0, 1\}^n$. There is a probability distribution on each message space, $\Pr_n : M_n \rightarrow [0, 1]$ such that $\sum_{m \in M_n} \Pr_n(m) = 1$.
- A *public-key cryptosystem* is an algorithm $C \in \mathcal{RA}$ that on input 1^n outputs the description of two polynomial-size circuits E and D such that:
 1. E has n inputs and $l(n)$ outputs, and D has $l(n)$ inputs and n outputs. (l is some polynomial that gives the length of the ciphertext.)
 2. E is probabilistic; D is deterministic.
 3. For all $m \in \Sigma^n$, $\Pr(D(\alpha) = m | (E, D) \leftarrow C(1^n); \alpha \leftarrow E(m)) = 1$.

Notice that $[E(m)]$ is a set which is typically quite large. Our notation requires us to write $\alpha \in [E(m)]$ to refer to α , particular encryption of m . Nevertheless, we will sometimes sloppily write $E(m)$ for a particular encryption of m when the meaning is clear.

3.3 GM-security

This definition is essentially what Goldwasser and Micali [5] called polynomial security.

A *line tapper* is a family of polynomial-size probabilistic circuits $T = \{T_n\}$. Each T_n takes four strings as input and outputs either 0 or 1. However, to make our next equation more readable, we will treat T_n 's output as being either its second or third input (0 or 1 respectively).

Definition Let C be a public-key cryptosystem. C is *GM-secure* if for all line tappers T and $c > 0$, for all sufficiently large n , for every $m_0, m_1 \in \{0, 1\}^n$

$$\Pr(T_n(E, m_0, m_1, \alpha) = m | m \leftarrow \{m_0, m_1\}; E \leftarrow C(1^n); \alpha \leftarrow E(m)) < \frac{1}{2} + n^{-c}. \quad (1)$$

Remark: In reading the above definition, one should pay close attention to our notation. Upon casual consideration of Equation 1, one might conclude that there aren't any GM-secure cryptosystems! After all, the definition says that the encryption E must be secure for *any* m_0 and m_1 , both of which are given as inputs to the line tapper. What happens if we put $m_0 = D$, a description of the decryption algorithm? The answer to this question is that our notation specifies that *first* we choose m from $\{m_0, m_1\}$ (and thus m_0 and m_1 already had been set), and *then* we choose our encryption algorithm. If C is GM-secure, then the probability that $C(1^n)$ assigns to any given output is quite small, say $O(2^{-n})$. Thus there's little worry that C will just happen to output a decryption algorithm $D = m_0$. Notice how the above definition (via our notation) models the three-pass scenario.

3.4 Semantic Security (3-pass)

Again, this definition is essentially the same as in [5]. It can be viewed as a polynomial-time bounded version of Shannon's "perfect secrecy" [7]. Informally, let f be *any* function defined on a

message space sequence. $f(m)$ constitutes information about the message m . Intuitively, f should be thought of as some particular information about the plaintext that the adversary is going to try to compute from the ciphertext—say the first seventeen bits of the plaintext. A cryptosystem is semantically secure if no adversary, on input $E(m)$ can compute $f(m)$ more accurately than by random guessing (taking into account the probability distribution on the message space).

Definition Let \mathcal{C} be a public-key cryptosystem, and let $M = \{M_n\}$ a sequence of message spaces. Let $\mathcal{F} = \{f_E : M_n \rightarrow \Sigma^* \mid E \in [\mathcal{C}(1^n)], n \in \mathbb{N}\}$ be any set of functions on the message spaces. For any value $v \in \Sigma^*$, we denote by $f_E^{-1}(v)$ the inverse image of v ; that is, the set $\{m \in M_n \mid f_E(m) = v\}$. Then the probability of the most probable value for $f_E(m)$ is $p_E = \max \left\{ \sum_{m \in f_E^{-1}(v)} \Pr_n(m) \mid v \in \Sigma^* \right\}$. p_E is the maximum probability with which one could guess $f_E(m)$ knowing only the probability distribution from which m has been drawn.

\mathcal{C} is *semantically secure* if for all message space sequences M , for all families of functions \mathcal{F} , for every family of polynomial-size probabilistic circuits $A = \{A_n(\cdot, \cdot)\}$, for all $c > 0$, and for all sufficiently large n

$$\Pr(A_n(E, \alpha) = f_E(m) \mid m \leftarrow M_n; E \leftarrow \mathcal{C}(1^n); \alpha \leftarrow E(m)) < p_E + \frac{1}{n^c}. \quad (2)$$

Notice that p_E implicitly depends on n , because E depends on n . Notice also that we quantify over message spaces in order to take into account all possible probability distributions on the messages.

3.5 Y-security (3-pass)

Yao's definition [8] is inspired by information theory, but its context differs from classical information theory in that the communicating agents, A(lice) and B(ob), are limited to probabilistic polynomial-time computations.

An intuitive explanation of Yao's definition is the following: A has a series of n^k messages, selected from a probability space known to both A and B, and an encryption of each message. She wishes to transmit enough bits to B so that he can (in polynomial time with very high probability) compute all the plaintexts. A cryptosystem is Y-secure if the average number of bits A must send B is the same regardless of whether B possesses a copy of the ciphertexts.

We now make this notion precise, first by defining “Alice and Bob,” and then eventually defining Y-security itself.

Let $M = \{M_n\}$ be a sequence of message spaces. Each M_n is $\{0, 1\}^n$ with a fixed probability distribution. (Note that an information theorist would consider M to be a sequence of sources.)

Let $\epsilon(n)$ be any function that vanishes faster than n^{-c} for all positive c .

For the sake of compactness of notation, the expression \vec{m} will denote a particular series of n^k messages. That is, \vec{m} stands for m_1, m_2, \dots, m_{n^k} .

Let f be any positive function such that $f(n) \leq n$. Intuitively, $f(n)$ is the number of bits per message that A must transmit to B in order for B to recover the plaintexts. Recall that all the messages in M_n have length n .

Definition An $f(n)$ compressor/decompressor pair (hereinafter c/d pair) for M is a pair of families of probabilistic polynomial-size circuits, $\{A_n\}$ and $\{B_n\}$, satisfying the following three properties for some constant k and all sufficiently large n :

1. “ B_n understands A_n .”

$$\Pr(\vec{m} = y \mid m_1 \leftarrow M_n; \dots; m_{n^k} \leftarrow M_n; \beta \leftarrow A_n(\vec{m}); y \leftarrow B_n(\beta)) = 1 - O(\varepsilon(n)). \quad (3)$$

2. “ A_n transmits only $f(n)$ bits per message.”

$$\text{Ex} \left[\frac{|\beta|}{n^k} \mid m_1 \leftarrow M_n; \dots; m_{n^k} \leftarrow M_n; \beta \leftarrow A_n(\vec{m}) \right] \leq f(n). \quad (4)$$

3. “The output of A_n can be parsed.”

For all polynomials Q there exists a probabilistic polynomial-time Turing machine S^Q such that S^Q takes as input n and a concatenated string of $Q(n)$ β s, each of which is a good output from A_n , and separates them. That is, its input is $\beta_1 \beta_2 \dots \beta_{Q(n)}$ and its output is $\beta_1 \# \beta_2 \# \dots \# \beta_{Q(n)}$. We require that

$$\Pr(S^Q \text{ correctly splits } \beta_1 \beta_2 \dots \beta_{Q(n)}) = 1 - O(\varepsilon(n)). \quad (5)$$

Remark: The requirement that S^Q exist is a technical requirement. It creates a finite analogue of classical information theory’s requirement that messages be transmitted one bit at a time, in an infinite sequence of bits.

We say that the *cost of communicating* M is less than or equal to $f(n)$, in symbols $C(M) \leq f(n)$, if there exists an $f(n)$ c/d pair for M .

We define $C(M) > f(n)$ to be the negation of $C(M) \leq f(n)$ —that is, *any* circuits “communicating M ” must use at least $f(n)$ bits. The definition of $C(M) = f(n)$ is analogous.

Let \mathcal{C} be a cryptosystem. We define $C(M \mid E_{\mathcal{C}}(M))$, the *cost of communicating* M given encryptions from \mathcal{C} in a manner analogous to $C(M)$. The only difference is that now both A_n and B_n also get E and the n^k values of some encryption function $E \in [\mathcal{C}(1^n)]$ as inputs. That is, for this definition we must rewrite Equation 3 above to read:

$$\begin{aligned} \Pr(\vec{m} = y \mid m_1 \leftarrow M_n; \dots; m_{n^k} \leftarrow M_n; E \leftarrow \mathcal{C}(1^n); \\ \alpha_1 \leftarrow E(m_1); \dots; \alpha_{n^k} \leftarrow E(m_{n^k}); \\ \beta \leftarrow A_n(E, \vec{m}, \vec{\alpha}); y \leftarrow B_n(E, \beta, \vec{\alpha})) = 1 - O(\varepsilon(n)). \end{aligned} \quad (6)$$

An analogous change must also be made to Equation 4.

Notice that for this definition, the probabilities involved must be taken over the different choices of E from \mathcal{C} as well as everything else.

Definition Let \mathcal{C} be a public-key cryptosystem. Fix a sequence of message spaces $M = \{M_n\}$ (and thus the probability distribution on each M_n). We say that \mathcal{C} is *Y-secure with respect to* M if

$$C(M) = C(M \mid E_{\mathcal{C}}(M)) + O(\varepsilon(n)). \quad (7)$$

We say that \mathcal{C} is *Y-secure* if for all M , \mathcal{C} is Y-secure with respect to M .

3.6 The original definitions vs. ours

As we pointed out in the introduction, we made minor changes in cryptographic scenario from [5] and [8]. Here we will spell out what those changes are and why they were made.

Changes to Goldwasser and Micali's Definition

There are two ways a cryptosystem (the server that generates encryption/decryption algorithm pairs) can achieve security:

1. The cryptosystem gets a description of a message space M (and thus its probability distribution) as one of its inputs and will output an encryption/decryption algorithm pair to securely encrypt M .
2. The cryptosystem is told nothing about the message space. The encryption algorithms it outputs are supposed to be secure for every possible message space.

We will call the former cryptosystems *adaptive* and the latter *oblivious*.

Goldwasser and Micali consider adaptive cryptosystems for both of their definitions of security [5]; Yao doesn't make it clear which type of cryptosystem he is assuming for his definition of security [8]. We believe it makes more sense to consider oblivious cryptosystems, for both theoretical and applied reasons.

The theoretical reason for preferring oblivious cryptosystems is that all three definitions of security are equivalent. (See Section 4.) This is a desirable property that fails to hold for adaptive cryptosystems, as we will show in the next section.

The practical reason for preferring oblivious cryptosystems is that, although it is certainly conceivable that having knowledge of the message space would allow one to design a better encryption algorithm, cryptographers have in fact normally tried to design cryptosystems that are secure for all message spaces. For example, consider the cryptosystem based on arbitrary trapdoor predicates proposed by Goldwasser and Micali [5]. Although they only considered security in the adaptive cryptosystem sense, their cryptosystem is in fact secure in the stronger, oblivious sense.

Changes to Yao's Definition

In [8], Yao assumes deterministic private key cryptography, but the definition is immediately extended to probabilistic public-key cryptography.

Yao defines the compressor A and decompressor B to be Turing machines, not circuits. We have switched to circuits because it is not clear that there are *any* secure cryptosystems with respect to probabilistic Turing machines. It might be that one can always achieve greater polynomial-time compression given the ciphertext simply because having a *shared* random (enough) string (in this case the ciphertext!) helps. If it does help, however, having made the compressor and decompressor nonuniform circuits, we can always hardwire in a shared random string of bits.

3.7 Inequivalence of the original definitions

In this section, we point out that, for adaptive cryptosystems, GM-security is a notion stronger than either semantic security or Y-security. We do this in the following two claims, each supported by an informal argument. These claims can be easily transformed to theorems after formalizing the discussed security notions in terms of adaptive cryptosystems, a tedious effort once we have realized that the adaptive setting is not the “right” one.

Claim 1 *If any GM-secure adaptive public-key cryptosystem exists, then there exist adaptive public-key cryptosystems that are semantically secure but not GM-secure.*

Let $\mathcal{C}(\cdot, \cdot)$ be any GM-secure (and thus semantically secure) adaptive cryptosystem. We’ll construct a $\mathcal{C}'(\cdot, \cdot)$ that is still semantically secure, but is not GM-secure.

\mathcal{C}' behaves identically to \mathcal{C} for all message spaces, except for the message space $\{0, 1\}^n$ with uniform probability distribution. In this case, \mathcal{C}' runs \mathcal{C} to compute an encryption algorithm E , and then outputs the algorithm E' defined by:

$$E'(x) = \begin{cases} 0^n & \text{if } x = 0^n \\ E(x) & \text{otherwise} \end{cases} \quad (8)$$

\mathcal{C}' is clearly not GM-secure, because, for the special message space described above, there is a message, 0^n , which is easily distinguished from other messages by its encryption. However, \mathcal{C}' is still semantically secure. The message 0^n has such a low probability weight that it won’t give an adversary any significant advantage—on average—in computing a function of the plaintext on input the ciphertext. \square

Claim 2 *If any GM-secure adaptive public-key cryptosystem exists, then there exist adaptive public-key cryptosystems that are Y-secure but not GM-secure.*

We construct exactly the same \mathcal{C}' as we did for the previous claim. \mathcal{C}' is of course not GM-secure. However, the “weak message” has such low probability that it basically doesn’t affect the average number of bits necessary to communicate messages from the message space. Thus \mathcal{C}' is Y-secure. \square

4 Main Results

GM-security, semantic security, and Y-security are all equivalent. We have chosen to prove this equivalence by showing that GM-security is equivalent to Y-security and that GM-security is equivalent to semantic-security. In this extended abstract, we simply state our theorems. The proofs may be found in the full paper. In fact, we prove only three of the four necessary implications. The proof that GM-security implies semantic security may be found in [5].

Theorem 1 *Let \mathcal{C} be a public-key cryptosystem. If \mathcal{C} is semantically secure, then \mathcal{C} is GM-secure.*

The proof of this theorem is quite simple. The key idea is that if a cryptosystem is not GM-secure, then there exist two messages, m_1 and m_2 , which we can easily distinguish. If we make a new message space in which these are the only messages, then given only a ciphertext, one has a better than random chance of figuring out which of the two plaintext messages this ciphertext represents.

Theorem 2 *Let \mathcal{C} be a cryptosystem. If \mathcal{C} is Y-secure, then \mathcal{C} is GM-secure.*

Theorem 3 *Let \mathcal{C} be a public-key cryptosystem. If \mathcal{C} is GM-secure, then \mathcal{C} is Y-secure.*

5 One-Pass Scenarios

In this section we present the proper definitions for one-pass cryptography, and then go on to show that these definitions are all equivalent to one another. (They are *not* equivalent to the three-pass definitions.) These definitions are all considerably more complicated than the analogous definitions for the three-pass scenario.

5.1 GM-security (1-pass)

As discussed above in Section 2.2, for a one-pass cryptosystem, we must change from requiring security “for all messages m ,” to requiring security for every message m that is efficiently computable on input the encryption algorithm alone. In order to do this, we introduce an adversary called a message finder.

A *message finder* is a family of polynomial-size probabilistic circuits $F = \{F_n(\cdot)\}$ each of which takes the description of an encryption algorithm as its input and has two messages of length n as its output. Intuitively, on input E , F_n tries to find m_0 and m_1 such that it’s easy for a fellow adversary (a line tapper) to distinguish encryptions of m_0 from encryptions of m_1 .

Definition Let \mathcal{C} be a public-key cryptosystem. \mathcal{C} is *GM-secure (one-pass)* if for all message finders F , line tappers T , and $c > 0$, for all sufficiently large n ,

$$\Pr(T_n(E, m_0, m_1, \alpha) = m \mid E \leftarrow \mathcal{C}(1^n); m_0, m_1 \leftarrow F_n(E); m \leftarrow \{m_0, m_1\}; \alpha \leftarrow E(m)) \leq \frac{1}{2} + n^{-c}. \quad (9)$$

5.2 Semantic Security (1-pass)

To change the definition of semantic security to fit the one-pass scenario, we need to introduce something like the message finders of the previous section. For semantic security, however, we’re concerned not with finding two “weak” messages, but rather with the probability distribution of the entire message space. Thus our second adversary will not pick out particular messages, but instead set the probability distribution of the message space. Furthermore, we now explicitly give the other adversary a description of that probability distribution.

A *message space enemy* is a family of polynomial-size probabilistic circuits $B = \{B_n(\cdot)\}$. Each B_n takes the description of an encryption algorithm as its input, and outputs the description of a probabilistic Turing machine $N()$. N outputs elements of $\{0, 1\}^n$ with some probability distribution.

As in the three-pass definition, we let V be any set and let $\mathcal{F} = \{f_n^E : M_n \rightarrow V \mid E \in [C(n)]\}$ be any set of functions. Again set p_n^E to be the probability of the most probable value for $f(m)$; set $p_n^E = \max\{\sum_{m \in f_n^E(v)} \Pr_n(m) \mid v \in V\}$.

Definition Let C be a public-key cryptosystem. C is *semantically secure* if for every message space enemy B , family of polynomial-size probabilistic circuits $A = \{A_n(\cdot, \cdot, \cdot)\}$, and $c > 0$, for all sufficiently large n

$$\Pr(A_n(E, N, \alpha) = f_n^E(m) \mid E \leftarrow C(1^n); N \leftarrow B_n(E); m \leftarrow N(); \alpha \leftarrow E(m)) < p_n^E + \frac{1}{n^c}. \quad (10)$$

5.3 Y-security (1-pass)

The changes that must be made to the definition of Y-security are completely analogous to the changes we made to the definition of semantic security.

5.4 Equivalence

The proofs that the three definitions of security are all equivalent are quite similar to the proofs for the three-pass case.

Acknowledgment

We wish to thank Shafi Goldwasser and Shimon Even for insight and helpful discussions about the notions of security, and general help in preparing this paper.

References

- [1] R. Boppana, private communication.
- [2] H. Chernoff, "A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations," *Annals of Mathematical Statistics* 23 (1952).
- [3] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch, "Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults," *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, 1985, pp. 383-95.
- [4] W. Diffie and M. E. Hellman, "New direction in cryptography," *IEEE Trans. Inform. Theory*, Vol. IT-22, No. 6, 1976, pp. 644-54.
- [5] S. Goldwasser and S. Micali, "Probabilistic Encryption," *JCSS*, vol. 28, No. 2, April 1984, pp. 270-99.

- [6] W. Rudin, *Principles of Mathematical Analysis*, 3rd ed., McGraw-Hill, New York, 1976.
- [7] C. E. Shannon, "Communication theory of secrecy systems," *Bell System Tech. J.*, 28 (1949), pp. 656–749.
- [8] A. C. Yao, "Theory and Applications of Trapdoor Functions (extended abstract)," *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, 1982, pp. 80–91.

LARGE-SCALE RANDOMIZATION TECHNIQUES

Neal R. Wagner¹

Drexel University

Paul S. Putter

The Pennsylvania State University

Marianne R. Cain¹

Drexel University

Abstract.

This paper looks at a collection of especially simple conventional cryptosystems that use a very large blocksize. One variation uses a single xor randomization followed by a single bit permutation. Tight upper and lower bounds are obtained on the number of bits of matching plaintext/ciphertext needed to break the systems. These results follow from two interesting combinatorial theorems. The cryptosystems are not practical because the number of bits above is about the same as the keysize. We can make the systems practical by introducing key-dependent pseudo-random numbers, though we then lose any proofs of the difficulty of cryptanalysis.

1. Introduction.

Recent work in cryptography has focused on proving that the difficulty of breaking a cryptosystem is equivalent to the difficulty of solving some other mathematical problem; for example, the problem of factoring composite integers.

We have been investigating two other approaches to conventional cryptography:

- Very simple encryption/decryption algorithms -- perhaps so simple that one can prove average-case lower bounds [Mas85] [Wag86].
- Very complex encryption/decryption algorithms -- perhaps so complex that mathematical analysis is not feasible [Wol85].

¹ Research supported in part by NSF grant DCR-8403350 and by a Research Scholar award from Drexel University.

This article considers the first approach -- looking for simple algorithms. To compensate for the simplicity of the algorithm we propose using a very large blocksize, or even encrypting an entire file at once. We also use randomization.

As an example, we present a code that uses one randomization step and one bit-permutation with a blocksize in the range of 1K to 1M bits. We present the simple motivating examples and their generalizations in Section 2. Section 3 looks at cryptanalysis of these various systems. We are able to obtain tight bounds on the number of plaintext/ciphertext pairs needed to break the systems. These bounds show exactly how much matching plaintext/ciphertext is needed to break the system, but the systems investigated in Section 3 are not at all practical. In fact, the keysize must be very large -- equal roughly to the amount of information that must be transmitted before the system can be broken. Finally, Section 4 considers several practical variations using pseudo-random numbers. The pseudo-random number algorithm is not assumed to be at all secure.

Another example of this basic approach is considered in [Wag 86]. Here in what might be called *global randomization*, we encrypt an entire relation of a relational database as a unit.

2. The Rip van Winkel cipher and generalization.

2.1. The Rip van Winkel cipher.

An especially simple though impractical cipher called the *Rip van Winkel* cipher was recently introduced [Mas85]. It is interesting for a provable lower bound on the average amount of computation needed to break the system. The system is illustrated in Figure 1.

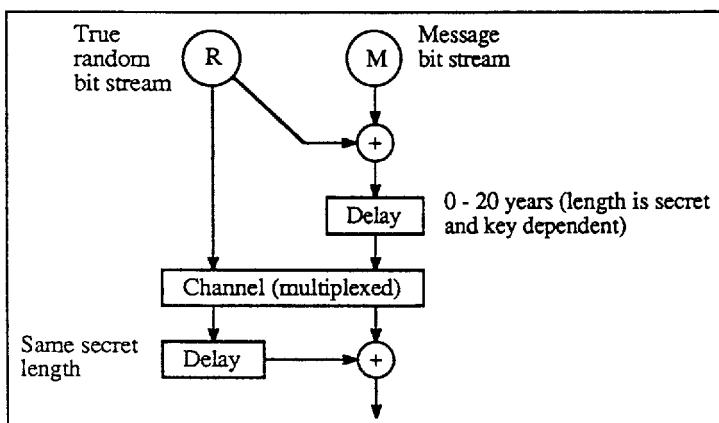


Figure 1. The Rip van Winkel cipher.

This cipher has the following interesting properties:

- The only secret is the *delay length*, a key-dependent number from 0 to the maximum delay.
- There is a provable average-case lower bound of $2\sqrt{N}$ comparisons needed to break the system, where N is the maximum delay length.
- Encryption and decryption require only a single xor per bit.
- The delay hardware must contain buffer storage.
- The system is completely impractical.

The rest of this paper looks at generalizations of this cipher.

2.2. Pseudo-random selection from queues.

During a study of randomization techniques [Wag85], we looked at ways to generalize the concept of concatenation to work with bit streams. One method employed a pseudo-random bit-sequence to control selection from two bit streams; the first, a random stream and the second, a randomization of a message stream (see Figure 2).

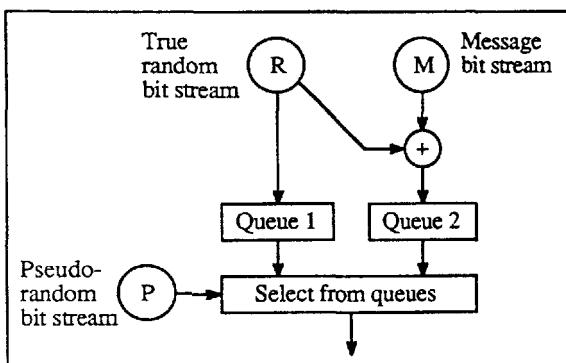


Figure 2. Pseudo-random selection from queues.

It is interesting that this system is a generalization of the Rip van Winkel cipher of Section 2.1. In fact, one can take $P = "01010101..."$ and initialize Queue 2 to hold 0 - 20 years worth of true random bits.

Section 2.3 and 2.4 consider further generalizations of this system, while Sections 3.1 and 3.3 give cryptanalytic attacks on the generalizations.

2.3. Rip van Winkel in RAM.

In implementing some version of the Rip van Winkel cipher, efficiency considerations almost force one into the use of RAM storage. Once RAM storage is chosen, one might as well plan to use arbitrary locations within RAM. This leads to the type of generalization we consider in this section.

Start with m plaintext bits P_1, P_2, \dots, P_m and m true random bits R_1, R_2, \dots, R_m . Use a RAM buffer C of size $n = 2m$. Initially load C with

$$C = (R_1, R_2, \dots, R_m, R_1 \oplus P_1, R_2 \oplus P_2, \dots, R_m \oplus P_m).$$

Then apply an arbitrary permutation of these n bits. This permutation is the encryption/decryption key, requiring $n \log_2 n$ bits of storage in the obvious representation.

Notice that any pair of bits R_i and $R_i \oplus P_i$ is a two-bit randomization of P_i , so that the xor of these two yields P_i . There is nothing that distinguishes the $R_i \oplus P_i$ bit from the R_i bit.

This is a randomized cryptosystem. There are $n!2^{n/2}$ distinct encryption functions and $n!2^{-n/2}$ distinct decryption functions. The average number of bits needed to represent a decryption key is

$$\log_2(n!2^{-n/2})$$

or approximately (using Stirling's formula)

$$n \log_2 n - n \log_2 e - (1/2)n + (1/2)\log_2 n.$$

Thus we could represent the decryption key with slightly fewer than the $n \log_2 n$ bits needed in just listing n addresses, but any gain would go asymptotically to zero as n increases.

Note that there are

$$\frac{n!}{(n/2)! 2^{n/2}}$$

ways to choose pairs of bits, where each pair represents a randomization of a plaintext bit. For each choice of pairs, there are $(n/2)!$ permutations of the pairs, giving $n!2^{-n/2}$ decryption functions altogether.

Cryptanalysis of this cipher will be considered in Section 3.3.

2.4. The matrix cipher.

Suppose in the previous cipher we wish to protect against chosen plaintext attacks. We could add an extra randomization step, in which one xors the plaintext with new random bits *before* carrying out the encryption of Section 2.3. The extra random bits would be concatenated onto the ciphertext. More generally, we could think of xorring triples together – two random bits and one plaintext bit -- before applying the permutation. From here it is natural to think of xorring general subsets of bits. This leads to the general form

$$KC = P,$$

where

- $P = (P_j)$ is a $m \times 1$ matrix of *plaintext bits*,
- $C = (C_j)$ is an $n \times 1$ matrix of *ciphertext bits*,
- $K = (K_{ij})$ is an $m \times n$ matrix of *key bits*,
- $n \geq m$,
- all arithmetic is over GF(2), and
- the matrix K has rank n .

The matrix K is the *secret key*. For encryption, one solves m equations in the n unknowns C_1, C_2, \dots, C_n . For decryption, just multiply K by C . If $n > m$, then encryption is randomized and non-linear, but decryption is a linear transformation. We are picturing n in the range from 1K to 1M bits. Assuming $n = 2m$, the keysize (size of matrix K) will be in the range from 500K to 500000M bits. Note that the Hill cipher [Den82] is a special case of this when $n = m$.

For cryptanalysis of this system, refer to Section 3.1.

3. Cryptanalysis -- upper and lower bounds.

3.1. The matrix cipher.

It is easy to break this system using enough known plaintext/ciphertext pairs. Suppose we have l ciphertexts C_1, C_2, \dots, C_l and corresponding plaintexts P_1, P_2, \dots, P_l . Write the ciphertexts as an $n \times l$ matrix. We will be able to break the system once this matrix has rank n . Assuming the entries of (C_1, C_2, \dots, C_l) are random, Appendix I gives probabilities for the matrix to have rank n .

Theorem. Given n known plaintext/ciphertext pairs and assuming random ciphertexts, the probability of being able to break the system is 0.288.... After $n + 10$ pairs, the probability is 0.999....

See Appendix I for probabilities of being able to break the system after $n + i$ known plaintext/ciphertext pairs, $i = 0, 1, 2, \dots$.

We can also state a lower bound.

Theorem. We must process at least m plaintext/ciphertext pairs on the average in order to break the matrix cipher.

Proof. This follows from an information theory argument. The keysize is mn . If $m = n$, only 29% of all matrices can serve as keys, but the percentage tends rapidly to 1 as n gets greater than m (see Appendix I). Thus the number of bits needed to represent the key is at least $mn - 2$. Each chosen of known plaintext/ciphertext pair adds at most n bits of information about the key. Thus m pairs must be processed before cryptanalysis is possible.

3.2. Bit permutations.

Assume $m = n$ and restrict to ciphers that are just permutations. Thus the matrix K is a permutation matrix -- exactly one 1 in each row and column.

Theorem. We can always uniquely determine a permutation of m bits using $\lceil \log_2 m \rceil$ chosen plaintext/ciphertext pairs, and no fewer number of pairs will suffice to uniquely determine a permutation.

Proof. The proof that $\lceil \log_2 m \rceil$ pairs will suffice is very similar to the Hamming code construction.

Next we show that at least $\lceil \log_2 m \rceil$ pairs are required. Suppose we have fewer than $\lceil \log_2 m \rceil$ plaintexts, say k of them. For each i , $1 \leq i \leq m$, regard the i^{th} positions of each plaintext as a k -bit number. There are m such numbers, but fewer than $\lceil \log_2 m \rceil$ bits. Thus we do not have enough bits for each of these m numbers to be distinct. In other words, there exist positions i and j such that each of the k plaintexts have the same value at i and j . Then given these k plaintexts and the corresponding ciphertext, we cannot tell whether the permutation leaves i and j fixed, or interchanges i and j , or does something more complex with positions i and j . Thus the permutation is not uniquely determined, completing the proof.

It is interesting to note that we lose guaranteed security just as we finish transmitting $m \log_2 m$ bits of plaintext, and $m \log_2 m$ is the keysize. So as we might expect, we would be better off just using a one-time pad. Later we will attempt to refine examples like the one in this section into a practical cryptosystem.

Suppose we have *known* plaintext/ciphertext pairs available, rather than *chosen* pairs.

Theorem. Let $P_{m,k}$ denote the probability that a permutation of m bits is uniquely determined by k random known plaintext/ciphertext pairs, i.e., the probability that k pairs will suffice to break the system. Then $P_{m,k}$ is the same as the number $p(m, 2^k)$ of Appendix I, the probability that m k -bit numbers will be distinct. In particular $P_{m,k} > 0.606$ for $k \geq 2\lceil \log_2 m \rceil$, and $P_{m,k} > 0.9995$ for $k \geq 2\lceil \log_2 m \rceil + 10$.

Proof. From Appendix I, once you believe that the m k -bit numbers must be distinct to uniquely determine a permutation.

3.3. The Rip van Winkel cipher in RAM.

Refer to Section 2.3 for the encryption/decryption algorithms.

Let us mount a chosen plaintext attack on this system using $P = (0\ 0 \dots 0)$, i.e., all m bits equal to 0. Repeated attacks will yield corresponding ciphertexts C_1, C_2, \dots , each $n = 2m$ bits long. For a given C_i , each pair R_i and $R_i \oplus P_i$ of bits will be the same, either both 0 or both 1, whereas all other pairs have probability 1/2 of differing. For a given bit position i ($1 \leq i \leq n$) and for a given ciphertext C_i , we expect half of the other positions j will be eliminated as candidates for the position paired with i . Thus intuitively, for fixed bit position i , we expect with probability 1/2 to have identified the unique position paired with i after processing $\log_2 m$ of the C_i . For *all* bit positions i , we intuitively expect with probability 1/2 to have identified all pairs after processing $2\log_2 m$ of the C_i . This intuition turns out to be correct, as we show below.

We need an additional $\log_2 m$ chosen plaintext/ciphertext pairs to uncover the permutation of the pairs, using a variation of the Hamming code.

Theorem. The system described here can be broken with probability at least 0.606... using $3\lceil \log_2 m \rceil$ chosen plaintext/ciphertext pairs. With $3\lceil \log_2 m \rceil + 10$ pairs, the probability of breaking the system is greater than 0.9995.

Proof. Let Q_i be the n -bit vector obtained by applying the inverse of the encryption permutation to C_i . Then Q_i consists of a random m -bit string followed by the identical m -bit string. Suppose we have k such n -bit vectors. Look at the columns of the Q_i rather than the rows, so that we have m random k -bit quantities followed by a repetition of these. By an argument similar to that used for the Hamming code, these k ciphertexts will uniquely determine the pairings of bit positions if and only if the m k -bit numbers are all distinct. Thus from Appendix I, the probability of uniquely determining the pairings becomes 0.606... for $k = 2\lceil \log_2 m \rceil$. The remainder of the proof is the same as the method for breaking a permutation cipher.

If we look at the proof, we see that for this type of attack we cannot do any better.

Theorem. Assuming we first determine the bit pairings using a chosen plaintext of all zero bits or all one bits and then the permutation of pairs, $3\lceil \log_2 m \rceil$ chosen plaintext/ciphertext pairs are necessary and sufficient to have probability 0.606... of breaking this system. Exact probabilities for breaking the system using this attack with fewer or with more chosen plaintext/ciphertext pairs are given in Appendix I for $m = 2^{10}$ and $m = 2^{20}$.

We are not able to prove the lower bound for an arbitrary chosen plaintext attack on this system, though we believe that no attack uses fewer pairs than the one we presented.

Notice that it seems much more difficult to break this system given only *known* plaintext/ciphertext pairs -- particularly if the plaintext contains roughly equal number of zero and one bits. In fact, the only attack we know of in this case is the one of Section 3.1, requiring at least n known plaintext/ciphertext pairs.

4. Practical variations using pseudo-random numbers.

4.1. Rip van Winkel in RAM.

For a RAM buffer of size $n = 1\text{M}$ bits, the Rip van Winkel in RAM cipher introduced and studied in Sections 2.3 and 3.3 has two fatal flaws:

- The keysize is relatively large, namely $n \log_2 n$, or 20M bits = 2.5M bytes for $n = 1\text{M}$.
- Relatively few chosen plaintext/ciphertext pairs are needed to break the system with high probability, namely $3\log_2 n + 10$, or 70 for $n = 1\text{M}$. (However, it appears that *known* plaintext attacks would require many more pairs.)

We can make these difficulties go away by

- generating the bit permutation pseudo-randomly, using a key-dependent pseudo-random number generator, and
- changing the key to the pseudo-random number generator with each encryption step. (We can use some of the random bits used to encrypt the plaintext for the key for the next encryption.)

Several difficulties then emerge:

- We must choose a suitable key-dependent pseudo-random number generator.
- We no longer have a required $3\log_2 n$ chosen plaintext/ciphertext pairs in order to break the system. In fact, a single known plaintext attack might succeed using a brute-force approach and arbitrarily large computing resources.

Though we no longer have any provable security, we feel that almost any pseudo-random number generator with a long key and very long period should provide good security. For example, the Tausworthe generator in [Bri79] with $n = 521$ has key (=seed) 521 bits long and period of 2^{521} 64-bit numbers. Since there is a key change at each stage, the opponent effectively has just one plaintext/ciphertext pair to work with. Best for an opponent would be the ciphertext corresponding to a chosen plaintext of all 0 bits (or all 1 bits), and this might be sufficient information to uniquely determine the key. However, the information is in a very diffuse and vague form that should make anything but brute-force cryptanalysis difficult. Basically, this information breaks the buffer into two subsets, and bit pairings are known to be confined to one or the other subset.

The actual algorithm is very simple to state:

```

(* Encryption algorithm *)
(* initial load of buffer *)
for i := 1 to m do
begin
  C[i] := truerandom;
  C[i+m] := C[i] xor P[i]
end;
(* pseudo-random permutation *)
for i := n downto 2 do
  interchange ( C[i], C[pseudorandom ( key, i )] )

```

Here C and P are bit arrays of ranges respectively $1 \dots n$ and $1 \dots m$, where $n = 2m$. The procedure "truerandom" returns a true random bit and the procedure "pseudorandom" returns an integer in the range $1 \dots i$. The permutation algorithm is described in [Knu81], and it is easy to see that if this algorithm is supplied uniformly distributed random integers, then the permutations generated are also uniformly distributed. Notice that between the two halves of the algorithm, the key for the *next* encryption would be extracted from the lower half of the buffer.

This algorithm seems interesting for its simplicity. It requires very few computational resources per plaintext bit: 1 true random bit, 1 call to the pseudo-random number generator, 1 xor, and several accesses and stores. Thus a larger buffer size requires no extra computation per message bit, though a larger buffer means that a longer pseudo-random integer must be returned. (Of course a larger buffer increases the overhead for short messages, since the entire buffer must still be sent.)

Decryption can be carried out in general without much difficulty (see [Knu73] for an algorithm for inverting a permutation). Decryption becomes especially easy if the pseudo-random number generator will run backwards.

4.2. Other practical variations.

Many variations and extensions of the basic system presented in Section 4.1 are possible. If we have fewer than $m = n/2$ message bits, we can fill up the buffer with true random bits. We can also enlarge the buffer size to fill up the channel. Finally, instead of xoring just two bits together, we can xor as large a subset as we can afford computationally. Thus we can plan to use up the resources of personal workstation in order to enhance security.

Consider the following extreme case with $n > m$, a generalization of the system in Section 4.1:

```

(* Encryption -- general form *)
(* initial load of buffer *)
for i := 1 to n-m do
  C[i] := truerandom;

```

```

(* xor of subsets *)
for i := 1 to m do
begin
  PR[1 .. (n-m+i -1)] := pseudo-random bit stream;
  C[n-m+i ] := P[i] xor (PR[ 1 .. (n-m+i -1) ]
                           innerproduct C[ 1 .. (n-m+i -1) ]
end;
(* pseudo-random permutation, as before *)

```

Acknowledgment.

Doren Zeilberger and Jet Wimp gave suggestions about the material in the appendices.

References.

- [Bri79] H. S. Bright, and R. L. Enison, "Quasi-random number sequences from a long-period TLP generator with remarks on application to cryptography," *ACM Computing Surveys*, Vol. 11, No. 4, (Dec. 1979), pp. 357-370.
- [Den82] D. Denning, *Cryptography and Data Security*, Addison-Wesley, 1982.
- [Knu81] D. E. Knuth, *The Art of Computer Programming, Vol. II, Seminumerical Algorithms*, 2nd Edition, Addison-Wesley, 1981.
- [Knu73] D. E. Knuth, *The Art of Computer Programming, Vol. I, Fundamental Algorithms*, 2nd Edition, Addison-Wesley, 1973.
- [Mas85] J. L. Massey, and I. Ingemarsson, "A simple and provable computationally secure cipher with a finite key," abstract submitted to ISIT-85 and presented at Eurocrypt 85.
- [Wag86] N. R. Wagner, P. S. Putter, and M. R. Cain, "Encrypted database design: Specialized approaches," *Proceedings of the 1986 IEEE Symposium on Security and Privacy*, IEEE Computer Society, 1986, pp. 148-153.
- [Wag85] N. R. Wagner, P. S. Putter, and M. R. Cain, "Using algorithms as keys in stream ciphers," *Proceedings of Eurocrypt 85, Lecture Notes in Computer Science*, Springer Verlag (to appear).
- [Wol85] S. Wolfram, "Random sequence generation by cellular automata," presentation at Crypto 85.

Appendix I. Probability that m k -bit random numbers are distinct.

Choose m k -bit numbers at random, $k \geq \lceil \log_2 m \rceil$. Let $p(m, 2^k)$ denote the probability that all m of the numbers are distinct. (If $k < \lceil \log_2 m \rceil$, then it is impossible for them all to be distinct.) More generally, consider $p(m, N)$, the probability of picking m distinct items out of N , where the selection is done at random with replacement. It is easy to see that

$$p(m, N) = \prod_{i=0}^{m-1} \left(1 - \frac{i}{N}\right) = \frac{N!}{(N-m)! N^m}.$$

Values of $p(2^{10}, 2^k)$ and $p(2^{20}, 2^k)$ are tabulated below for various k . Note that for increasing k , $p(m, 2^k)$ first gets greater than 0.5 for $k = 2\lceil \log_2 m \rceil$. As k becomes large, the value $p(2^{k/2}, 2^k)$ tends to 0.6065306597....

The entries in the table were all calculated using the exact formula. However, we can set $m = \alpha N$ and use Stirling's formula to obtain the approximation

$$p(m, N) = p(\alpha N, N) \approx \left[\frac{e^{-\alpha}}{(1-\alpha)^{1-\alpha}} \right]^N (1-\alpha)^{-1/2}.$$

Though it is only an approximation, this formula is accurate enough to give every digit of every entry in the table, except for the entry $p(2^{10}, 2^{10})$.

As shown Table 1, two lists of $p(m, 2^k)$ values for two different values of m are nearly the same when shifted so that the entries $k = 2\lceil \log_2 m \rceil$ are lined up. This property follows from the approximation

$$p(2m, 4N) \approx p(m, N),$$

which is quite accurate as long as N is not too small. Other interesting approximate recurrences include

$$\begin{aligned} p(m, 2N) &\approx p(m, N)^{1/2} \\ p(2m, N) &\approx p(m, N)^4 \end{aligned}$$

and more generally,

$$p(2^a m, 2^b N) \approx p(m, N)^{4^a a / 2^b b}.$$

To prove one variation of these recurrences, use the formula for $p(\alpha N, N)$ with the $(1-\alpha)^{-1/2}$ part dropped out (accurate for α small). Then $p(2\alpha N, 2N) = p(\alpha(2N), 2N) = p(\alpha N, N)^2$, so that

$$p(2m, 2N) \approx p(m, N)^2.$$

	$m = 2^{10} = 1024$	$m = 2^{20} = 1048576$	
k	$p(2^{10}, 2^k)$	$p(2^{20}, 2^k)$	k
10	1.5371837 E -443	3.7069749 E -223	30
15	0.00000009652	0.00000011251	35
16	0.0003242093	0.0003354515	36
17	0.018196318	0.018315522	37
18	0.13524703	0.13533519	38
19	0.36799933	0.36787955	39
20	0.60672822	0.60653085	40
21	0.77895928	0.77880093	41
22	0.88259566	0.88249699	42
23	0.93946801	0.93941311	43
24	0.96926219	0.96923326	44
25	0.98451130	0.98449645	45
30	0.99951231	0.99951183	50
35	0.99998475	0.99998474	55

Table 1.

n-m	Probability that random $m \times n$ matrix over GF(2) has rank m (accurate to 12 digits for $m \geq 40$)
0	0.288788095086
1	0.577576190173
2	0.770101586897
3	0.880116099311
4	0.938790505932
5	0.969074070639
6	0.984456198745
7	0.992207822357
8	0.996098833425
9	0.998048146211
10	0.999023755347 = 1 - 0.976244 E -3
20	1 - 0.953674 E -6
30	1 - 0.931322 E -9
40	1 - 0.909494 E -12
50	1 - 0.888124 E -15

Table 2.

Appendix II. Probability that an $m \times n$ matrix over GF(2) with random entries has rank m .

Theorem: Consider an $m \times n$ ($n \geq m$) matrix over GF(2) with all entries randomly chosen. For $m = n$, the approximate probability that the matrix is invertible is given by the constant $q_0 = 0.28878809508660242\dots$ (The approximation is accurate to about 12 digits for $m \geq 40$.) More generally, for $n \geq m$, the approximate probability that the matrix has rank m is given by Table 2. (The probability of rank m is very roughly $1 - 10^{-0.3(n-m)}$.)

Proof: Consider first the $m \times m$ case. There are 2^{m^2} matrices altogether over GF(2). Let $F(m)$ denote the number of invertible matrices. To calculate $F(m)$, we just count the number of ways to build up m linearly independent rows. This gives $F(m) = (2^m - 2^0)(2^m - 2^1)(2^m - 2^2)\dots(2^m - 2^{m-1})$. $F(m)$ satisfies the recurrence $F(m) = (2^{2m-1} - 2^{m-1})F(m-1)$. Since $G(m) = 2^{m^2}$ satisfies $G(m) = 2^{2m-1}G(m-1)$, we suspect that $F(m)$ looks like 2^{m^2} .

$$\frac{F(m)}{2^{m^2}} = 1/2 * 3/4 * 7/8 * 15/16 * \dots * (2^m - 1)/2^m.$$

The corresponding infinite product is given by a special Theta function known as a Q-function.

$$Q_0(q) = \prod_{m=1}^{\infty} (1 - q^{2m}), \text{ where we want } Q_0(1/\sqrt{2}).$$

This product converges rapidly to $q_0 = 0.28878809508660242\dots$ Thus $F(m)$ is approximately $q_0 * 2^{m^2}$. (The approximation is accurate to 3 digits when $m = 10$, 6 digits when $m = 20$ and generally about $0.3 * m$ digits.) The other figures in the table of the theorem are obtained by dropping the initial $n - m$ terms of the infinite product.

**On the Linear Span of Binary Sequences
Obtained from Finite Geometries**

AGNES HUI CHAN, RICHARD A. GAMES

The MITRE Corporation
Bedford, Massachusetts 01730

ABSTRACT

A class of periodic binary sequences that are obtained from the incidence vectors of hyperplanes in finite geometries is defined, and a general method to determine their linear spans (the length of the shortest linear recursion over $GF(2)$ satisfied by the sequence) is described. In particular, we show that the projective and affine hyperplane sequences of odd order both have full linear span. Another application involves the parity sequence of order n , which has period $p^n - 1$ and linear span $vL(s)$ where $v = (p^n - 1)/(p - 1)$ and $L(s)$ is the linear span of a parity sequence of order 1. The determination of the linear span of the parity sequence of order 1 leads to an interesting open problem involving primes.

1. INTRODUCTION.

Binary sequences which satisfy recursions over $GF(2)$ are easy to generate and have many applications in modern communication systems. If the recursions involved are linear, then the sequences can have several desirable properties, e.g., long periods, useful correlation properties, and balanced statistics. Binary sequences of maximum period $2^n - 1$ that are generated by linear recursions over $GF(2)$ of degree n are called binary *m-sequences* of span n [1].

These linear recursive sequences suffer from one drawback: only relatively few terms of the sequence are needed to solve for the generating recursion; i.e., their *linear span* (the length of the shortest linear recursion over $GF(2)$ satisfied by the sequence) is short relative to their period. Such easy predictability makes binary *m-sequences* unsuitable for some applications requiring pseudorandom bits.

This work was supported by the MITRE-Sponsored Research Program. A. H. Chan is also with the College of Computer Science, Northeastern University, Boston MA 02115

In this paper, we consider a class of periodic binary sequences that are obtained from the incidence vectors of hyperplanes in finite geometries. From another point of view, these sequences can be obtained from q -ary m -sequences of span n through a mapping ρ from $GF(q)$ to $GF(2)$, where q is a power of an odd prime. We show that the linear span of these sequences is comparable to their large periods. In fact, the linear span of such a sequence S of period $q^n - 1$ is given by $vL(s)$ where $v = (q^n - 1)/(q - 1)$ and $L(s)$ denotes the linear span of a sequence s of period $q - 1$. The binary sequence s is obtained by applying the defining mapping ρ of S to a listing of the nonzero elements of $GF(q)$ according to the powers of some primitive element. If q is of moderate size, then the linear span of s is easily computed by the Berlekamp-Massey algorithm [2].

The next section introduces the notions involving the linear span of a sequence. Section 3 describes the construction of the binary sequences obtained from finite geometries. Section 4 establishes the upper bound of the linear span of these sequences, while section 5 shows that this bound is always attained. Finally, section 6 gives four examples of sequences obtained from finite geometries and their associated linear spans. These include the hyperplane sequences for both projective and affine spaces. One of these sequences, called the parity sequence, gives rise to an interesting open problem involving primes.

2. THE LINEAR SPAN OF A SEQUENCE.

This paper considers binary sequences and linear recursions over $GF(2)$. All operations used are those of $GF(2)$ unless otherwise stated. Let E denote the sequence shift operator: Es is the sequence with i^{th} term $(Es)_i = s_{i+1}$. A sequence $s = (s_0, s_1, \dots, s_i, \dots)$ satisfies a linear recursion of degree m if, for $a_j \in GF(2)$,

$$s_{i+m} + \sum_{j=1}^m a_j s_{i+m-j} = 0, \quad i \geq 0.$$

This recursion can be expressed in terms of the shift operator,

$$(E^m + \sum_{j=1}^m a_j E^{m-j})s = 0.$$

The polynomial $f(E) = E^m + \sum_{j=1}^m a_j E^{m-j}$ is called the *characteristic polynomial* of the recursion. If $m(E)$ denotes the unique monic polynomial of least degree such that $m(E)s = 0$, then the *linear span* of s , denoted by $L(s)$, equals the degree of $m(E)$; $m(E)$ is called the *minimal polynomial* of s . If $f(E)s = 0$, then from the division algorithm, $m(E)|f(E)$. In particular, if s has period N , then $(E^N + 1)s = 0$, so that $m(E)|E^N + 1$, and $L(s) \leq N$.

The linear span of a sequence is one measure of its predictability. Any “good” pseudorandom sequence must have large linear span relative to its period [3], [4]. If a sequence has linear span L , then its linear recursion can be determined from $2L$ successive elements of the sequence. The remaining elements then can be produced from the recursion.

3. SEQUENCES FROM FINITE GEOMETRIES.

In this section, we give the construction of a class of periodic binary sequences that are obtained from finite geometries. Let $q = p^r$, where p is an odd prime, and α be a primitive element of $GF(q^n)$. The nonzero elements of $GF(q^n)$ can be ordered using the primitive element $\alpha : \{\alpha^i : i = 0, 1, \dots, q^n - 2\} = GF(q^n)^*$. Elements of $GF(q^n)$ can also be considered as points of an n -dimensional affine space, denoted by $EG(n, q)$. We shall first establish the geometric structure of an affine space $EG(n, q)$ defined by a q -ary m -sequence of span n .

Let $Tr : GF(q^n) \rightarrow GF(q)$ be the trace function defined by $Tr(x) = x + x^q + \dots + x^{q^{n-1}}$. It is well known that the sequence $R = (R_i)$, obtained by $R_i = Tr(\alpha^i)$, is a q -ary m -sequence of span n with period $q^n - 1$. Furthermore, one period of R has the form $R = (T, \beta T, \dots, \beta^{q-2} T)$ where T is a q -ary vector of length $v = (q^n - 1)/(q - 1)$, and $\beta = \alpha^v$ is the corresponding primitive element of $GF(q)$ [5]. The sequence R partitions the elements of $GF(q^n)^*$ into q subsets H_a^* , $a \in GF(q)$, where $H_a^* = \{\alpha^i : R_i = Tr(\alpha^i) = a\}$. If we consider $H_0 = H_0^* \cup \{0\}$ and $H_a = H_a^*$ for $a \neq 0$, then $\Pi = \{H_a : a \in GF(q)\}$ forms a parallel class of affine hyperplanes in $EG(n, q)$. In general, corresponding to every cyclic shift $E^k R$, $k = 0, 1, \dots, v - 1$, of R there is in $EG(n, q)$ a corresponding parallel class of hyperplanes $\Pi^{(k)}$ consisting of $H_a^{(k)} = \{\alpha^{i-k} : \alpha^i \in H_a\}$, $a \in GF(q)^*$, and $H_0^{(k)} = \{0\} \cup \{\alpha^{i-k} : \alpha^i \in H_0^*\}$. Thus all parallel classes of hyperplanes in $EG(n, q)$ can be obtained from R .

We define periodic binary sequences, which are indexed by the elements of $EG(n, q) \setminus \{0\}$, by considering the incidence vectors of subcollections of $\Pi^* = \{H_a : a \in GF(q)^*\}$. In particular, for $I \subseteq GF(q)^*$ and corresponding subcollection $\Sigma_I = \{H_a : a \in I\}$, the sequence $S(\Sigma_I)$ has period $q^n - 1$ and i^{th} term, corresponding to α^i given by

$$S_i = \begin{cases} 1, & \text{if } \alpha^i \in H_a \in \Sigma_I \\ 0, & \text{otherwise.} \end{cases}$$

Equivalently, if $\rho_I : GF(q) \rightarrow GF(2)$ is defined by

$$\rho_I(a) = \begin{cases} 1, & \text{if } \alpha \in I \\ 0, & \text{otherwise,} \end{cases}$$

then $S(\Sigma_I)$ has i^{th} term given by $S_i = \rho_I(R_i)$. Note that points in H_0 are always mapped to 0 in $GF(2)$. For simplicity of notation, we shall use Σ and ρ to denote Σ_I and ρ_I whenever I is understood. The purpose of this paper is to determine the linear span of $S(\Sigma_I)$.

Example 1: For $p = 3, n = 2$ and primitive polynomial $x^2 + x + 2$, the ternary m -sequence R of span 2 is given by

$$R = \begin{pmatrix} \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & \alpha^7 \\ (2 & 2 & 0 & 2 & 1 & 1 & 0 & 1) \end{pmatrix}$$

The parallel class of hyperplanes corresponding to R consists of

$$H_0 = \{0, \alpha^2, \alpha^6\},$$

$$H_1 = \{\alpha^4, \alpha^5, \alpha^7\},$$

$$H_2 = \{\alpha^0, \alpha^1, \alpha^3\}.$$

For $I = \{1, 2\}, S(\Sigma_I) = (01110111)$ has linear span 4. For $I = \{1\}, S(\Sigma_I) = (01000011)$ has linear span 8.

4. UPPER BOUND ON THE LINEAR SPAN OF SEQUENCES FROM FINITE GEOMETRIES.

For fixed $n \geq 1, R = (R_0, R_1, \dots, R_{q^n-2})$ will denote an m -sequence of span n over $GF(q)$. For any collection Σ of hyperplanes obtained from $R, S(\Sigma) = (S_0, S_1, \dots, S_{q^n-2})$ will denote a binary sequence determined by Σ ; i.e., $S_i = 1$ if $\alpha^i \in H_a \in \Sigma$ and $S_i = 0$ otherwise. Although the sequence $S(\Sigma)$ depends on the choice of primitive polynomial, the results concerning its linear span will not. This is because any sequence obtained using a different primitive polynomial is related to $S(\Sigma)$ by a decimation by a value relatively prime to the period.

We introduce an array operator A which takes any sequence X of period $q^n - 1$ and arranges it into the $(q - 1)$ by v array:

$$A(X) = \begin{pmatrix} X_0 & X_1 & \dots & X_{v-1} \\ X_v & X_{v+1} & \dots & X_{2v-1} \\ \vdots & \vdots & \ddots & \vdots \\ X_{(q-2)v} & X_{(q-2)v+1} & \dots & X_{(q-1)v-1} \end{pmatrix}.$$

When applied to R , this operator produces the array $A(R)$ with column i of the form $(R_i, \beta R_i, \dots, \beta^{q-2} R_i) = R_i(1, \beta, \dots, \beta^{q-2})$. The sequence $(1, \beta, \beta^2, \dots, \beta^{q-2})$ is a q -ary m -sequence of span 1 and will be denoted by $r = (r_0, r_1, \dots, r_{q-2})$. A binary sequence $s(\Sigma)$ of period $q - 1$ is obtained by applying ρ to each term of r , where ρ is the mapping from $GF(q)$ to $GF(2)$ determined by Σ . The main result of this paper is that the linear span $L(S(\Sigma))$ of $S(\Sigma)$ is given by $vL(s(\Sigma))$. The value of $L(s(\Sigma))$ is relatively easy to compute.

If $R_i = 0$, then column i of $A(R)$ is the zero sequence 0. Otherwise $R_i = \beta^{e_i}$ for some $e_i \in \{0, 1, \dots, q-2\}$ and column i is $E^{e_i}r$. In general, we define the *shift sequence* of R to be $e = (e_0, e_1, \dots, e_{q^n-2})$ where $e_i = \infty$ if $R_i = 0$ or $R_i = \beta^{e_i}$ if $R_i \neq 0$. Likewise, the finite terms of $(e_0, e_1, \dots, e_{q^n-2})$ give the shifts of the sequence $s(\Sigma)$ occurring as columns of the $(q-1)$ by v array $A(S(\Sigma))$. $A(S(\Sigma))$ contains columns of all zeros corresponding to the ∞ positions in $(e_0, e_1, \dots, e_{v-1})$. By convention, we write $E^\infty s = 0$.

Example 2: For $p = 5$, $n = 3$ and primitive polynomial $x^3 + x^2 + 2$ the p -ary m -sequence R of span 3 is given in its array form as

$$A(R) = \begin{pmatrix} 0014120332224243340432042342201 \\ 0032310441112124420241021421103 \\ 0041430223331312210123013213304 \\ 0023240114443431130314034134402 \end{pmatrix}.$$

If $\Sigma = \{H_a : a \equiv 1 \pmod{2}\}$ then

$$A(S(\Sigma)) = \begin{pmatrix} 0010100110000001100010000100001 \\ 0010110001110100000001001001101 \\ 000101000111110010101011011100 \\ 000100011000101110110010110000 \end{pmatrix},$$

$r = (1, 3, 4, 2)$, $s(\Sigma) = (1, 1, 0, 0)$, and the first $v = 31$ terms of the shift sequence e are:

$$(\infty \infty 0203 \infty 11333232112 \infty 213 \infty 231233 \infty 0).$$

Throughout, let $M(E)$ and $m(E)$ denote the minimal polynomials of $S(\Sigma)$ and $s(\Sigma)$, respectively. We shall use the notation S and s to denote the sequences $S(\Sigma)$ and $s(\Sigma)$ respectively whenever the collection Σ of hyperplanes is assumed. We begin to investigate the properties of $M(E)$ and $m(E)$, obtaining upper bounds on the linear spans of S and s . The *weight* $wt(X)$ of a binary sequence $X = (X_0, X_1, \dots, X_{N-1})$ is the sum of any N consecutive terms of X . Thus, $((E^N + 1)/(E + 1))X = (E^{N-1} + E^{N-2} + \dots + 1)X = (wt(X) \pmod{2})$.

LEMMA 1. If Σ contains an odd number of hyperplanes, then $wt(S(\Sigma)) \equiv 1 \pmod{2}$. If Σ contains an even number of hyperplanes, then $wt(S(\Sigma)) \equiv 0 \pmod{2}$.

PROOF.

$$wt(S) = |\Sigma| / (\text{number of elements in a hyperplane}).$$

Since each hyperplane in an affine space $EG(n, q)$ contains q^{n-1} elements and q is odd, $wt(S) \equiv |\Sigma| \pmod{2}$, and the result follows. ■

THEOREM 2. If Σ contains an odd number of hyperplanes in $EG(n, q)$, then the highest power of $(E + 1)$ that divides $(E^{q^n-1} + 1)$ divides $M(E)$.

PROOF. Since S has period $q^n - 1$, $M(E)$ divides $(E^{q^n-1} + 1)$. But,

$$((E^{q^n-1} + 1)/(E + 1))S = (\text{wt}(S) \pmod{2}) = (1)$$

and so $M(E)$ does not divide $(E^{q^n-1} + 1)/(E + 1)$. The result follows. ■

COROLLARY 3. If Σ contains an odd number of hyperplanes, then

$$(E + 1)^2 | m(E).$$

PROOF. Apply the theorem to the case $n = 1$, and note that since q is odd, $(E + 1)^2 | (E^{q-1} + 1)$. ■

Our next theorem establishes an upper bound on the linear span of the sequence S .

THEOREM 4. $M(E) | m(E^v)$ so that $L(S(\Sigma)) \leq vL(s(\Sigma))$.

PROOF. We show that if $m(E) = E^{j_1} + E^{j_2} + \dots + E^{j_k}$, where $j_1 = 0$, is the minimal polynomial of s , then $m(E^v)S = 0$. For $i \in \{0, 1, \dots, q^n - 2\}$,

$$\begin{aligned} (m(E^v)S)_i &= S_i + S_{i+j_2v} + \dots + S_{i+j_kv} \\ &= (m(E)(S_i, S_{i+v}, \dots, S_{i+(q-2)v}))_0. \end{aligned}$$

If $R_i = \beta^{e_i}$, then $(S_i, S_{i+v}, \dots, S_{i+(q-2)v}) = E^{e_i}s$, and

$$(m(E^v)S)_i = (m(E)(E^{e_i}s))_0 = (E^{e_i}m(E)s)_0 = (E^{e_i}0)_0 = 0.$$

If $e_i = \infty$, then $(S_i, S_{i+v}, \dots, S_{i+(q-2)v}) = 0$, and certainly $(m(E^v)S)_i = 0$. Thus, $m(E^v)S = 0$, so $M(E) | m(E^v)$ and $L(s) \leq \text{degree } m(E^v) = v(\text{degree } m(E)) = vL(s)$. ■

Two questions naturally arise when determining the linear span of S :

- (1) Is the bound in theorem 4 attained, that is, does $L(S) = vL(s)$?
- (2) What is $L(s)$?

In the next section, we show that the answer to question 1 is always yes. Section 6 describes four particular choices of Σ and the respective values of $L(s)$.

5. LINEAR SPAN OF BINARY SEQUENCES OF FINITE GEOMETRIES.

In the previous section we established the upper bound $((q^n - 1)/(q - 1))L(s(\Sigma))$ on the linear span of the binary sequence $s(\Sigma)$ obtained from affine space $EG(n, q)$. Here, we show that this upper bound is always attained. In fact, we show that the minimal polynomial of $S(\Sigma)$ is $M(E) = m(E^v)$ where $m(E)$ is the minimal polynomial of $s(\Sigma)$.

First, we group the terms of $M(E)$ which have exponents congruent to the same value modulo v , and let $M(E)$ be expressed as follows:

$$M(E) = f_0(E^v) + f_1(E^v)E + \dots + f_{v-1}(E^v)E^{v-1}$$

where each $f_i(x)$ is a polynomial of degree d_i . The constant term 1 must appear in $M(E)$ and so $f_0(x) \neq 0$. We shall show that $f_i(x) = 0$ for all $i = 1, \dots, v - 1$.

LEMMA 5. *If $f_0(E)s(\Sigma) = 0$, then $M(E) = m(E^v)$.*

PROOF.

$$\begin{aligned} f_0(E)s &= 0 \Rightarrow m(E)|f_0(E) \\ &\Rightarrow \text{degree } m(E) \leq \text{degree } f_0(E) \\ &\Rightarrow \text{degree } m(E^v) \leq \text{degree } f_0(E^v) \leq \text{degree } M(E) \end{aligned}$$

By theorem 4, $M(E)|m(E^v)$, so $M(E) = m(E^v)$. ■

To show that the upper bound on theorem 4 is attained, we need to show that $f_0(E)s = 0$. Involved in the proof are properties of the shift sequence $e = (e_0, e_1, \dots, e_{q^n-2})$ introduced in section 4.

THEOREM 6. *Let $e = (e_0, e_1, \dots, e_{q^n-2})$ be the shift sequence associated with the primitive polynomial $f(x)$ which generates a q -ary m -sequence R of span n . Then any v consecutive terms of e contain exactly $(q^{n-1} - 1)/(q - 1) \infty$ terms.*

PROOF. This is the number of zeros in any v consecutive terms of R . ■

COROLLARY 7. *There are exactly q^{n-1} finite terms in $(e_0, e_1, \dots, e_{v-1})$.*

PROOF. $v - (q^{n-1} - 1)/(q - 1) = q^{n-1}$. ■

In the following theorem, the elements $\{0, 1, \dots, q - 2\}$ are identified with the elements of $\mathbb{Z}(q - 1)$, the integers modulo $q - 1$. We use the convention that if $e_i \in \mathbb{Z}(q - 1)$, then $e_i - \infty = \infty - e_i = \infty$.

THEOREM 8. Let e be the shift sequence associated with the primitive polynomial $f(x)$ which generates a q -ary m -sequence R of span n . For fixed $k \in \{1, 2, \dots, v-1\}$, the list of differences $(e_{i+k} - e_i \pmod{q-1} : i \in \{0, 1, \dots, v-1\})$ contains each element of $\mathbb{Z}(q-1)$ exactly q^{n-2} times.

PROOF. See [6, theorem 2, but with $m = 1$]. The results in [6] are stated for the case $q = 2$, however, the proofs remain valid for any prime power q . ■

Recall that

$$M(E) = f_0(E^v) + f_1(E^v)E + \dots + f_{v-1}(E^v)E^{v-1}.$$

The next lemma establishes a relationship between the polynomials $f_i(x)$ and the sequence s .

LEMMA 9. $f_0(E)s = (f_1(E) + f_2(E) + \dots + f_{v-1}(E))(E^0 + E + \dots + E^{q-2})s$.

PROOF. For a polynomial $f(E)$, applying $f(E^v)$ to S is equivalent to applying $f(E)$ to every v^{th} term of S , that is, applying $f(E)$ to columns of $A(S)$. Since the i^{th} column of $A(S)$ is $E^{e_i}s$, the i^{th} column of $A(f(E^v)S)$ is given by $f(E)E^{e_i}s$. Recall the convention that $E^\infty s = 0$.

Now, for each $k, 0 \leq k \leq v-1$, consider the sequence $E^k S$ as represented by the array $A(E^k S)$. Every column i of the array now has a leading term S_{i+k} . Hence, every column i of the array $A(f_k(E^v)E^k S)$ is given by $f_k(E)E^{e_{i+k}}s$. Now, $M(E)S = 0$ implies that every column i in $A(M(E)S)$ is 0, that is, for $i \in \{0, 1, \dots, v-1\}$,

$$(f_0(E)E^{e_i} + f_1(E)E^{e_{i+1}-e_i} + \dots + f_{v-1}(E)E^{e_{i+v-1}-e_i})s = 0.$$

For every i , with $e_i \neq \infty$

$$E^{e_i}(f_0(E) + f_1(E)E^{e_{i+1}-e_i} + \dots + f_{v-1}(E)E^{e_{i+v-1}-e_i})s = 0$$

if and only if

$$(f_0(E) + f_1(E)E^{e_{i+1}-e_i} + \dots + f_{v-1}(E)E^{e_{i+v-1}-e_i})s = 0.$$

Summing over $i, 0 \leq i \leq v-1$, such that $e_i \neq \infty$, we have

$$\left(\sum_{e_i \neq \infty} f_0(E) + \sum_{e_i \neq \infty} (f_1(E)E^{e_{i+1}-e_i} + \dots + f_{v-1}(E)E^{e_{i+v-1}-e_i}) \right) s = 0.$$

By corollary 7, the first sum contains q^{n-1} terms, which is odd, thus

$$\sum_{e_i \neq \infty} f_0(E) \equiv f_0(E) \pmod{2}.$$

By theorem 8 and since q^{n-2} is odd, for each $k \in \{1, 2, \dots, v-1\}$,

$$\sum_{e_i \neq \infty} f_k(E) E^{e_i+k-e_i} \equiv f_k(E) (E^0 + E + \dots + E^{q-2}) \pmod{2}$$

where terms of the form $E^{\infty-e_i}$ are ignored because $E^{\infty-e_i}s = 0$. Combining the above observations, we have

$$f_0(E)s = (f_1(E) + f_2(E) + \dots + f_{v-1}(E))(E^0 + E + \dots + E^{q-2})s. \blacksquare$$

THEOREM 10. $M(E) = m(E^v)$ and so $L(S(\Sigma)) = vL(s(\Sigma))$.

PROOF. By lemma 5, it is enough to show that $f_0(E)s = 0$, and by lemma 9 this is equivalent to showing $(f_1(E) + \dots + f_{v-1}(E))(E^0 + E + \dots + E^{q-2})s = 0$.

If Σ consists of an even number of hyperplanes, then $wt(s) \equiv 0 \pmod{2}$; hence

$$(E^0 + E + \dots + E^{q-2})s = (wt(s) \pmod{2}) = 0$$

and

$$f_0(E)s = 0.$$

If the number of hyperplanes in Σ is odd, then $wt(s) \equiv 1 \pmod{2}$. By lemma 9,

$$\begin{aligned} (E+1)f_0(E)s &= (f_1(E) + f_2(E) + \dots + f_{v-1}(E))(E+1)(wt(s) \pmod{2}) \\ &= (f_1(E) + f_2(E) + \dots + f_{v-1}(E))0 \\ &= 0; \end{aligned}$$

that is, $m(E)|(E+1)f_0(E)$.

By corollary 3, $(E+1)^2|m(E)$ and so $(E+1)|f_0(E)$ and $f_0(E)$ has an even number of terms. Similarly, theorem 2 states that $(E+1)|M(E)$, so $M(E)$ has an even number of terms, and this number is given by the sum of the number of terms in $f_0(E)$ and the number of terms in $f_1(E) + f_2(E) + \dots + f_{v-1}(E)$ (before mod 2 cancellation). At any rate, this implies that $f_1(E) + f_2(E) + \dots + f_{v-1}(E)$ has an even number of terms and $f_0(E)s = (f_1(E) + f_2(E) + \dots + f_{v-1}(E))1 = 0$. \blacksquare

6. SPECIAL CASES OF BINARY SEQUENCES.

In this section we consider four particular choices of Σ , and analyze the linear span of each sequence.

If Σ consists of all nonzero hyperplanes, that is, $\Sigma = \Pi^* = \{H_a : a \neq 0\}$, then the sequence $S(\Pi^*)$ has period v and corresponds to the anti-incidence vector of a hyperplane in an $(n-1)$ -dimensional projective space $PG(n-1, q)$. In this case $s = (1)$ and has $L(s) = 1$, so theorem 10 states that the linear span of $S(\Pi^*)$ is v . It is easy to see that the linear spans of a binary sequence X and its complement \bar{X} differ by at most one. Since $S(\Pi^*)$ has period v and $L(\bar{S}(\Pi^*))$ must divide v , $L(\bar{S}(\Pi^*)) = L(S(\Pi^*)) = v$.

The complement of $S(\Pi^*)$ corresponds to the incidence vector of projective hyperplane in $PG(n-1, q)$. Projective codes with the incidence matrix of points and hyperplanes as parity check rules have been studied by coding theorists, and the rank of this incidence matrix over $GF(p)$, where $q = p^r$, has been obtained.

THEOREM 11 [7], [8], [9]. *For $q = p^r$, the $GF(p)$ rank of the incidence matrix of points and hyperplanes in $PG(n-1, q)$ is $1 + \binom{n-1+p-1}{p-1}^r$.*

Each row of the incidence matrix is a shift of the sequence $\bar{S}(\Pi^*)$. It is not hard to see that the rank of this incidence matrix over $GF(2)$ is precisely the linear span over $GF(2)$ of the sequence $\bar{S}(\Pi^*)$. In the case when $q = 2^r$, the linear span (over $GF(2)$) of a projective hyperplane sequence can be obtained from theorem 11. Combining these observations we have the following theorem.

THEOREM 12. *The linear span of a projective hyperplane sequence of $PG(n-1, q)$ is given by $(q^n - 1)/(q - 1)$ if q is odd and $1 + n^r$ if $q = 2^r$.*

On the other hand, if Σ consists of only a single affine hyperplane H in Π^* , then the sequence $S(H)$ corresponds to the incidence vector of an affine hyperplane and has period $q^n - 1$. The sequence $s(H)$ corresponds to a sequence of all 0's except one 1 and has full linear span $q - 1$. Thus, by theorem 10, an affine hyperplane sequence of an affine space of odd order has full linear span. For affine spaces of even order results on the incidence matrix of points and hyperplanes of $EG(n, q)$ apply.

THEOREM 13 [7]. *For $q = p^r$, the $GF(p)$ rank of the incidence matrix of points and hyperplanes in $EG(n, q)$ is $\binom{n+p-1}{p-1}^r$.*

Combining these facts, we have the following theorem.

THEOREM 14. *The linear span of an affine hyperplane sequence of $EG(n, q)$ is given by $(q^n - 1)$ if q is odd and $(n + 1)^r$ if $q = 2^r$.*

If Σ consists of half of the hyperplanes in Π^* , say, $\Sigma = \{H_{\beta^i} : i = 0, 1, \dots, (q - 3)/2\}$, then the sequence $s(\Sigma)$ is a binary sequence with $(q - 1)/2$ ones followed by $(q - 1)/2$ zeros, which has linear span $(q + 1)/2$. Thus, by theorem 10, the sequence $s(\Sigma)$ has period $q^n - 1$ and linear span $v(q + 1)/2$.

More generally, we can consider choices of Σ with the first half of the sequence $s(\Sigma)$ the complement of the second half. The next lemma gives an upper bound on the linear span of $s(\Sigma)$.

LEMMA 15. *If Σ is chosen so that the first half of the sequence $s(\Sigma)$ is the complement of the second half of the sequence, then*

$$L(s(\Sigma)) \leq (q + 1)/2.$$

PROOF. Let $d = (q - 1)/2$ and consider the sequence $h = (E^d + 1)s$. Since $s_i = s_{i+d}$ for all $i \in \{0, 1, \dots, (q - 3)/2\}$, the sequence h consists of all 1's and $(E + 1)h = 0$. Thus,

$$m(E)|(E^d + 1)(E + 1)$$

and

$$L(s) \leq 1 + (q - 1)/2 = (q + 1)/2. \blacksquare$$

Finally, if q is an odd prime ($q = p$) and Σ consists of all the “odd hyperplanes,” that is, $\Sigma = \{H_a : a \equiv 1 \pmod{2}\}$, the sequence $S(\Sigma)$, called the *parity sequence* of order n , has period $p^n - 1$ and linear span that depends on the linear span of the parity sequence $s(\Sigma)$ of order 1. For all $i \in \{0, 1, \dots, p - 2\}$, $\beta^{i+(p-1)/2} = p - \beta^i$, and since p is odd, β^i and $\beta^{i+(p-1)/2}$ have different parities. Thus the parity sequence of order 1 has the property stated in lemma 15, and $L(s(\Sigma)) \leq (p + 1)/2$.

For all but 14 primes less than 500, $L(s(\Sigma)) = (p + 1)/2$, that is, usually $L(S(\Sigma)) = v(p + 1)/2 = (p^n - 1)/2 + (p^n - 1)/(p - 1)$. For instance the parity sequence in example 2 has linear span $(31)(3) = 93$. The 14 primes with $L(s(\Sigma)) < (p + 1)/2$ are listed in table 1, together with the linear spans of the corresponding parity sequences of order 1. The determination of a closed form expression of the linear span of the parity sequences of order 1 is an interesting (and probably difficult) open problem.

TABLE 1
PRIMES < 500 WITH $L(s) < (p+1)/2$

p	$L(s)$	$(p+1)/2 - L(S)$
29	12	3
113	54	3
163	80	2
197	96	3
239	117	3
277	135	4
311	146	10
337	163	6
349	171	4
373	182	5
397	195	4
421	207	4
463	229	3
491	240	6

7. CONCLUSION.

We have presented general results on the linear spans of a class of binary sequences that are obtained from q -ary m -sequences (q odd) by mapping the elements of $GF(q)$ to 0 and 1. The linear span and minimal polynomial for these sequences are determined by considering a binary sequence of much shorter period $q - 1$. The results imply that the binary sequences under consideration have linear spans that are comparable to their periods, which can be made very long.

ACKNOWLEDGEMENT

We would like to thank E.L. Key for introducing this problem area and for helpful and stimulating discussions.

REFERENCES

1. S.W. GOLOMB, "Shift Register Sequences," Aegean Park Press, Laguna Hills, 1982.
2. J.L. MASSEY, Shift-Register Synthesis and BCH Decoding, *IEEE Trans. Info Theory* **IT-15** (1969), 122-127.
3. E.L. KEY, An Analysis of the Structure and Complexity of Nonlinear Binary Sequence Generators, *IEEE Trans. Info Theory* **IT-22** (1976), 732-736.
4. R. A. RUPPEL, "New Approaches to Stream Ciphers," Ph.D. Thesis, Swiss Federal Institute of Technology, 1984.
5. N. ZIELER, Linear Recurring Sequences, *J. Soc. Indust. Appl. Math.* **7** (1959), 31-48.
6. R. A. GAMES, Crosscorrelation of M -sequences and GMW-sequences with the same primitive polynomial, *Discrete Applied Math.* **12** (1985), 139-146.
7. J.M. GOETHALS AND P. DELSARTE, On a Class of Majority-logic Decodable Cyclic Codes, *IEEE Trans. Info Theory* **IT-14** (1968), 182-188.
8. F.J. MACWILLIAMS AND H.B. MANN, On the p -rank of the Design Matrix of a Difference Set, *Info. Control* **12** (1968), 474-488.
9. K.J.C. SMITH, On the p -rank of the Incidence Matrix of Points and Hyperplanes in a Finite Projective Geometry, *J. Combinatorial Theory* **7** (1969), 122-129.

Some Constructions and Bounds for Authentication Codes

D. R. Stinson

Department of Computer Science
University of Manitoba

1. Introduction

We shall use the model of authentication theory as described by Simmons in [S1], [S2], and [S3]. In this model, there are three participants: a transmitter, a receiver, and an opponent. The *transmitter* wants to communicate some information to the *receiver*, whereas the *opponent* wants to deceive the receiver. The opponent can either impersonate the receiver, making him accept a fraudulent message as authentic; or, modify a message which has been sent by the transmitter.

More formally, we have a set of source states S , a set of messages M , and a set of encoding rules E . A *source state* $s \in S$ is the information that the transmitter wishes to communicate to the receiver. The transmitter and receiver will have secretly chosen an *encoding rule* $e \in E$ beforehand. An encoding rule e will be used to determine the *message* $e(s)$ to be sent to communicate any source state s . It is possible that more than one message can be used to determine a particular source state (this is called *splitting*). However, in order for the receiver to be able to uniquely determine the source state from the message sent, there can be at most one source state which is encoded by any given message $m \in M$.

We assume that the opponent will play either *impersonation* or *substitution*. When the opponent plays impersonation, he sends a message to the receiver, attempting to have the receiver accept the message as authentic. When the opponent plays substitution, he waits until a message m has been sent, and then replaces m with another message m' so that the receiver is misled as to the state of the source.

There will be a probability distribution on the set of source states S . Given the probability distribution on S , the receiver and transmitter will determine a probability distribution on E , called an *encoding strategy*. If splitting occurs, then they will also determine a *splitting strategy* to determine $m \in M$, given $s \in S$ and $e \in E$. The transmitter / receiver will choose the encoding and splitting strategies to minimize the chance that the opponent can deceive them.

This defines two possible games, which we refer to as the impersonation game and the substitution game. Each game has a *value*, which is the possibility that the opponent can deceive the transmitter / receiver, given that they are using the optimal encoding and splitting strategies. We denote the values of these games by v_I (for impersonation) and v_S (for substitution).

Many of the bounds on the values of the games v_I and v_S depend on entropies of the various probability distributions. For a probability distribution on a set X , we define the *entropy* of X , $H(X)$, as follows:

$$H(X) = - \sum_{x \in X} p(x) \cdot \log p(x).$$

As well, the conditional entropy $H(X | Y)$ is defined to be

$$H(X | Y) = - \sum_{y \in Y} \sum_{x \in X} p(y) \cdot p(x | y) \cdot \log p(x | y).$$

An authentication code is said to be *Cartesian* if any message uniquely determines the source state, independent of the particular encoding rule being used. In terms of entropy, this is expressed by the equation $H(S | M) = 0$. Note that in a Cartesian authentication code, there can be no secrecy.

In this paper, we primarily consider authentication systems without splitting. We shall use the following notation. Denote the number of source states by k , and let $S = \{s_i: 1 \leq i \leq k\}$. Denote the number of messages by v , and let $M = \{m_j: 1 \leq j \leq v\}$. Denote by b the number of encoding rules, and write any encoding rule $e \in E$ as $e = (e_i: 1 \leq i \leq k)$, where e_i is the message used to communicate source state s_i , for $1 \leq i \leq k$. Then, the authentication system can be represented by the $b \times k$ matrix A , where row e of A consists of the entries e_1, \dots, e_k . Given an encoding rule $e \in E$, we define $M(e) = \{e_i: 1 \leq i \leq k\}$, where $e = (e_i: 1 \leq i \leq k)$. Also, for any encoding rule e , define $f_e(m) = s$ if and only if $e_s = m$ (if message m does not occur in encoding rule e , then $f_e(m)$ is undefined).

2. Bounds on the values of the impersonation and substitution games

Theorem (Simmons [S2, Theorem 1]) In an authentication system without splitting, $v_I \geq k / v$.

Theorem (Simmons [S2, Theorem 0]) In any authentication system, $v_I \geq 2^{H(MES)} - H(E) - H(M) = 2^{H(M | ES)} + H(S) - H(M)$. In an authentication system without splitting, $H(M | ES) = 0$, so $v_I \geq 2^{H(S) - H(M)}$.

Theorem (Simmons, Brickell [B1, Theorem 3]) $v_S \geq 2^{-H(E | M)} = 2^{H(M) - H(E) - H(S) + H(M | ES)}$. In an authentication system without splitting, $H(M | ES) = 0$, so $v_S \geq 2^{H(M) - H(E) - H(S)}$.

Given any encoding rule e' , and given any $m, m' \in M(e')$, define

$$\delta(e', m, m') = \sum_{\{e \in E: m, m' \in M(e)\}} p(e) \cdot p(S = f_e(m)) / (p(e') \cdot p(S = f_{e'}(m))).$$

Then, let $\delta = \min\{\delta(e', m, m'): m, m' \in M(e'), m \neq m'\}$.

Theorem In an authentication system without splitting, $v_S \geq \delta \cdot 2^{-H(E | M)}$.

Given any message m , define $r_m = |\{e \in E: m \in M(e)\}|$.

Theorem In an authentication system without splitting, $v_S \geq \delta / r$, where $r = \max\{r_m: m \in M\}$.

Given any encoding rule e' , and given any $m, m' \in M(e')$, define

$$\gamma(e', m, m') = \sum_{\{e \in E: m, m' \in M(e)\}} p(e) \cdot p(S = f_e(m)) / p(e').$$

Then, let $\gamma = \min\{\gamma(e', m, m'): m, m' \in M(e'), m \neq m'\}$.

Theorem In an authentication system without splitting, $v_S \geq \gamma \cdot 2^{H(M) - H(E)}$.

Theorem In an authentication system without splitting, $v_S \geq (k - 1) / (v - 1)$.

3. Constructions for authentication systems

Our interest is in constructing authentication systems which meet one or more of these bounds with equality. We are interested in the existence of authentication codes with a specified number of source states, and specified upper bounds on the number of encoding rules, messages, v_I , and v_S . Therefore, we define an $AC(k, v, b, \alpha, \beta)$ to be an authentication code without splitting, having k source states, at most v messages, at most b encoding rules, and where $v_I \leq \alpha$ and $v_S \leq \beta$. Then, we define

$$\epsilon(k, \alpha, \beta) = \min\{b: \text{there exists an } AC(k, v, b, \alpha, \beta)\},$$

and

$$v(k, \alpha, \beta) = \min\{v: \text{there exists an } AC(k, v, b, \alpha, \beta)\}$$

That is, we are attempting to minimize the number of encoding rules (or messages) required in an authentication code for k source states, with upper bounds α and β on the impersonation and substitution games, respectively.

First, observe that we have an easy lower bound on $v(k, \alpha, \beta)$.

Theorem $v(k, \alpha, \beta) \geq \max\{k / \alpha, 1 + (k - 1) / \beta\}$.

Next, we mention a lower bound on $\varepsilon(k, \alpha, \beta)$ due to Brickell ([B1, Theorem 4]).

Theorem $\varepsilon(k, \alpha, \beta) \geq 1 / (\alpha \cdot \beta)$.

This bound can be strengthened, using the quantity δ defined earlier.

Theorem If an AC(k, v, b, α, β) exists, then $b \geq \delta / (\alpha \cdot \beta)$.

Proof: We have $\alpha \geq v_I \geq 2^{H(S)} - H(M)$ and $v_S \geq \delta \cdot 2^{-H(E|M)} = \delta \cdot 2^{H(M) - H(E) - H(S)}$. Hence, we have $\alpha \cdot \beta \geq \delta \cdot 2^{-H(E)}$. Since $H(E) \leq \log b$, the result follows.

In the remainder of this paper, we shall be describing constructions for authentication codes, which will enable us to put upper bounds on ε and v . For our first construction, we require the following definition. A *transversal design* TD($k, \lambda; n$) is a triple (X, G, A) , which satisfies the following properties:

- 1) X is a set of $k \cdot n$ elements called *points*
- 2) G is a partition of X into k subsets of n points, called *groups*
- 3) A is a set of $\lambda \cdot n^2$ subsets of X (called *blocks*) such that a group and a block contain at most one common point
- 4) every pair of points from distinct groups occurs in exactly λ blocks.

We usually denote a TD($k, 1; n$) by TD(k, n). It is well-known that a TD(k, n) is equivalent to $k - 2$ mutually orthogonal Latin squares of order n .

Theorem (Brickell [B1, Theorems 5 and 6]) If there is a transversal design TD(k, n) then there is a Cartesian authentication system with $v_S = 2^{-H(E|M)} = 1/n$, $v_I = 2^{H(S) - H(M)} = 1/n$, $|S| = k$, $|M| = k \cdot n$, and $|E| = n^2$, with no splitting. Conversely, the existence of such an authentication system implies the existence of a transversal design TD(k, n). Hence, if there exists a TD(k, n), then there is an AC($k, k \cdot n, n^2, 1/n, 1/n$), and we have the upper bounds $\varepsilon(k, 1/n, 1/n) \leq n^2$ and $v(k, 1/n, 1/n) \leq k \cdot n$.

We can prove a generalization of this result, using transversal designs with $\lambda \geq 1$.

Construction 1 If there is a transversal design TD($k, \lambda; n$) then there is a Cartesian authentication system with $v_S = \lambda \cdot 2^{-H(E|M)} = 1/n$, $v_I = 2^{H(S) - H(M)} = 1/n$, $|S| = k$, $|M| = k \cdot n$, and $|E| = \lambda \cdot n^2$, with no splitting. Conversely, the existence of such an authentication system implies the existence of a transversal design TD($k, \lambda; n$). Hence, if there exists a TD($k, \lambda; n$), then there is an AC($k, k \cdot n, \lambda \cdot n^2, 1/n, 1/n$), $\varepsilon(k, 1/n, 1/n) \leq \lambda \cdot n^2$, and $v(k, 1/n, 1/n) \leq k \cdot n$.

Suppose our desire is to construct an authentication code $AC(k, k \cdot n, b, 1/n, 1/n)$. We can construct such a code if a $TD(k, \lambda; n)$ exists for $b = \lambda \cdot n^2$. (Note that this satisfies the bound $b \geq \delta / (\alpha \cdot \beta)$ with equality, where $\alpha = \beta = 1/n$ and $\delta = \lambda$.) Thus, given k and n , we are interested in the smallest λ such that a $TD(k, \lambda; n)$ exists. First, we observe that there is a simple numerical bound on k in terms of λ and n .

Theorem (Hanani [H1]). If a $TD(k, \lambda; n)$ exists, then $k \leq (\lambda \cdot n^2 - 1) / (n - 1)$.

Consequently, if we use a $TD(k, \lambda; n)$, then we have a lower bound on b , namely

$$b = \lambda \cdot n^2 \geq kn - k + 1.$$

We present an infinite example of transversal designs which meet this bound with equality.

Theorem For all prime powers $n \geq 2$, and for any $d \geq 1$, there is an $AC(k, k \cdot n, n^d, 1/n, 1/n)$, where $k = (n^d - 1) / (n - 1)$; hence $\epsilon((n^d - 1) / (n - 1), 1/n, 1/n) \leq n^d$ and $v((n^d - 1) / (n - 1), 1/n, 1/n) \leq k \cdot n$.

Proof: In [H1], Hanani shows that for any prime power n , and for any $d \geq 1$, there is a $TD((n^d - 1) / (n - 1), n^{d-2}; n)$.

Corollary For any $\alpha > 0$, $\epsilon(k, \alpha, \alpha)$ is $O(k/\alpha^2)$ and $v(k, \alpha, \alpha)$ is $O(k/\alpha)$.

Proof: Let $n = 2^j$, where $2^j \geq 1/\alpha \geq 2^{j-1}$. Then n is $O(1/\alpha)$. Now, choose d so that $n^d \geq k(n - 1) + 1 > n^{d-1}$. Since $k \leq (n^d - 1) / (n - 1)$, we have $\epsilon(k, \alpha, \alpha) \leq n^d$. But, $n^d \leq k(n^2 - n) + n = O(k \cdot n^2)$. Since n is $O(1/\alpha)$, therefore $\epsilon(k, \alpha, \alpha)$ is $O(k/\alpha^2)$. Also, $k \cdot n$ is $O(k/\alpha)$.

As another example of the use of transversal designs with $\lambda > 1$, let's consider codes with parameters $AC(k, v, b, 1/6, 1/6)$. For $k = 4$, we cannot construct such a code from a $TD(4, 6)$, since this TD does not exist (this is the famous 36 officers problem of Euler, i.e. a pair of orthogonal Latin squares of order 6). In [B1], Brickell constructs an example of an $AC(4, 30, 36, 1/6, 1/6)$ with splitting. However, we can employ a $TD(7, 2, 6)$, which is constructed in [H1, p. 49], to obtain an $AC(7, 42, 72, 1/6, 1/6)$.

More generally, we have the following class of authentication codes with 7 source states.

Theorem For all $n \geq 2$, there is an $AC(7, 7 \cdot n, 2n^2, 1/n, 1/n)$; hence $\epsilon(7, 1/n, 1/n) \leq 2n^2$.

Proof: For these n , there is a $\text{TD}(7, 2; n)$ (see [H1]).

The authentication codes obtained from Construction 1 are Cartesian. Hence, the opponent, on seeing a message being sent, knows the source state. Therefore, no secrecy is possible in such an authentication system. We also want to be able to construct good authentication codes with secrecy. Ideally, we would like to have $H(S | M) = H(S)$; i.e. the message gives absolutely no clue as to the state of the source. If this happens, then we say that the authentication code is *perfectly* non-Cartesian.

Our main construction for perfectly non-Cartesian authentication codes uses group-divisible designs, which are a generalization of transversal designs. A *group-divisible design* $\text{GD}(k, \lambda, n; v)$ is a triple (X, G, A) , which satisfies the following four properties:

- 1) X is a set of v elements called *points*
- 2) G is a partition of X into v/n subsets of n points, called *groups*
- 3) A is a set of subsets of X (called *blocks*), each of size k , such that a group and a block contain at most one common point
- 4) every pair of points from distinct groups occurs in exactly λ blocks.

Note that a $\text{TD}(k, \lambda; n)$ is equivalent to a $\text{GD}(k, \lambda, n; k \cdot n)$. Also, a (v, b, r, k, λ) -BIBD (balanced incomplete block design) is equivalent to a $\text{GD}(k, \lambda, 1; v)$.

We have the following construction.

Construction 2 Suppose there exists a $\text{GD}(k, \lambda, n; v)$. Then there is a perfectly non-Cartesian $\text{AC}(k, v, \lambda \cdot v \cdot (v - n) / (k - 1), k / v, (k - 1) / (v - n))$.

Proof: Let (X, G, A) be a $\text{GD}(k, \lambda, n; v)$. By simple counting, each point occurs in $r = \lambda \cdot (v - n) / (k - 1)$ blocks, and the total number of blocks is $\lambda \cdot v \cdot (v - n) / (k \cdot (k - 1))$. What we do is construct k encoding rules from every block of the group-divisible design: for each block $A = \{x_1, \dots, x_k\}$ of the group-divisible design, and for each i , $0 \leq i \leq k - 1$, we define an encoding rule $e(A, i) = (e_j : 1 \leq j \leq k)$, where $e_j = x_{(j+i) \bmod k}$.

There are $\lambda \cdot v \cdot (v - n) / (k - 1)$ encoding rules in the resulting authentication code. We shall use each encoding rule with probability $(k - 1) / (\lambda \cdot v \cdot (v - n))$. It is not difficult to verify that $v_1 = k / v$ and $v_S = (k - 1) / (v - n)$.

Finally, the authentication code is perfectly non-Cartesian since $p(s | m) = p(s)$ for every $s \in S$ and every $m \in M$.

It is interesting to note that this code has $H(M) = \log v$, $H(E) = \log(\lambda \cdot v \cdot (v - n) / (k - 1))$, and $v_S = \gamma 2^{H(M)} - H(E)$, where $\gamma = \lambda$.

Corollary Suppose there exists a (v, b, r, k, λ) -BIBD. Then there is a perfectly non-Cartesian AC($k, v, k \cdot b, k / v, (k - 1) / (v - 1)$).

Proof: This is the case where every group of the group-divisible design has size 1. Note that here we have $v_S = (k - 1) / (v - 1)$.

Corollary Suppose there is a TD($k, \lambda; n$). Then there is a perfectly non-Cartesian AC($k, n \cdot k, \lambda \cdot k \cdot n^2, 1 / n, 1 / n$).

Consequently, $\epsilon(k, \alpha, \alpha)$ is $O(k^2 / \alpha^2)$ and $v(k, \alpha, \alpha)$ is $O(k^2 / \alpha)$, even if we restrict ourselves to perfectly non-Cartesian codes.

These two constructions for authentication codes both have two very nice properties which we have not yet emphasized. First, the encoding strategy in each case is *uniform*: each encoding rule is used with equal probability $1 / b$. Second, this encoding strategy yields the stated game values for *any* source distribution.

The final topic we consider is the construction of authentication codes for *uniform* source distributions ($p(s) = 1 / k$ for any source state s). As before we consider only codes without splitting. The best we could hope for is to attain the bounds $v_I = k / v$ and $v_S = (k - 1) / (v - 1)$. So, we shall study AC($k, v, b, k / v, (k - 1) / (v - 1)$); such authentication codes will be called *optimal*.

We have the following characterization of authentication codes which are optimal with respect to the uniform probability distribution on the source states.

Lemma An authentication system is optimal with respect to the uniform probability distribution on the source states if and only if the following properties are satisfied:

- i) for every $m \in M$, $\sum_{\{e \in E: e \in m\}} p(e) = k / v$.
- ii) for every $m \neq m'$, $\sum_{\{e \in E: m, m' \in e\}} p(e) = (k^2 - k) / (v^2 - v)$.

In many authentication codes, the optimal encoding strategy is to choose every encoding rule with probability $1 / b$. If we assume that this encoding strategy is in fact optimal, then the properties above are of a purely combinatorial nature. We have the following

Theorem An authentication system is optimal with respect to a uniform encoding strategy and a uniform probability distribution on the source states if and only if the following properties are satisfied:

- i) for every $m \in M$, $|\{e \in E: m \in e\}| = k \cdot b / v$.
- ii) for every $m \neq m'$, $|\{e \in E: m, m' \in e\}| = b \cdot (k^2 - k) / (v^2 - v)$.

This says that the rows of E , considered as unordered sets, form a balanced incomplete block design with parameters (v, b, r, k, λ) , where $r = k \cdot b / v$ and $\lambda = b \cdot (k^2 - k) / (v^2 - v)$. So, we can produce optimal authentication codes from BIBDs when the source states are equiprobable.

Using known families of BIBDs, we can obtain many authentication codes for uniform source distributions. For example, using projective geometries, we have the following.

Theorem For any prime power n , and any integer $d \geq 2$, there is an optimal authentication code for the uniform source distribution on $n + 1$ source states, for $v = (n^{d+1} - 1) / (n - 1)$ and $\lambda = 1$.

References

- B1. Ernest F. Brickell, A Few Results in Message Authentication, Congressus Numerantium 43 (1984), 141-154.
- H1. H. Hanani, On Transversal Designs, Math. Centre Tracts 55 (1974), 42-52.
- S1. Gustavus J. Simmons, A Game Theory Model of Digital Message Authentication, Congressus Numerantium 34 (1982), 413-424.
- S2. Gustavus J. Simmons, Message Authentication: A Game on Hypergraphs, Congressus Numerantium 45 (1984), 161-192.
- S3. Gustavus J. Simmons, Authentication Theory / Coding Theory, in "Advances in Cryptology: Proceedings of CRYPTO 84", Lecture Notes in Computer Science, vol. 196, 411-432, Springer Verlag, Berlin, 1985.

Towards a Theory of Software Protection

(Extended Abstract)

Oded Goldreich

Computer Science Department
Technion, Haifa 32000, Israel

ABSTRACT

Software protection is one of the most important issues concerning computer practice. The problem is to sell programs that can be executed by the buyer, yet cannot be duplicated and/or distributed by him to other users. There exist many heuristics and ad-hoc methods for protection, but the problem as a whole did not receive the theoretical treatment it deserves.

In this paper, we make the first steps towards a theoretic treatment of software protection: First, we distill and formulate the key problem of *learning about a program from its execution*. Second, we present an *efficient* way of executing programs (i.e. a interpreter) such that it is infeasible to learn anything about the program by monitoring its executions. A scheme that protects against duplication follows.

How can one efficiently execute programs without allowing an adversary, monitoring the execution, to learn anything about the program? Current cryptographic techniques can be applied to keep the contents of the memory unknown throughout the execution, but are *not applicable* to the problem of hiding the access pattern. Hiding the access pattern *efficiently* is the essence of our solution. We show how to implement (on-line and in an “oblivious manner”) to fetch instructions to a memory of size m by making less than $t \cdot m^\epsilon$ actual accesses, for every fixed $\epsilon > 0$.

1. INTRODUCTION

Software protection is one of the most important issues concerning computer practice. The problem is to sell programs that can be executed by the buyer, yet cannot be duplicated and/or distributed by him to other users. A lot of engineering effort is put into trying to provide “software protection”, but this effort seems to lack theoretical foundations. In particular, there is no crisp definition of what the problems are and what should be considered as a satisfactory solution. In this paper, we make the first steps towards a theoretic

Work done while author was in the Laboratory for Computer Science, MIT.

Partially supported by a Weizmann Postdoctoral Fellowship, an IBM Postdoctoral Fellowship, and NSF Grant DCR-8509905.

treatment of software protection, by distilling a key problem and solving it efficiently.

Before going any further, we distinguish between two intuitive notions: the problem of *protection against duplication* and the problem of *protection against distribution*. Loosely speaking, the first problem consists of ensuring that there is no efficient method for creating executable copies of the software; while the second problem consists of ensuring that, in case duplication succeeds, the illegal duplicator should be unable to prove in court that he has designed the program. In this paper we concentrate on the first problem, which clearly implies a solution to the second one.

We claim that *protection against duplication* must use some hardware measures: mere software (which is not physically protected) can always be duplicated. On the other extreme, the trivial solution is to rely only on hardware. That is, to sell physically-protected special-purpose computers for each task. This "solution" has to be rejected as infeasible and too expensive. We conclude that a real solution to protecting software from duplication should combine feasible software and hardware measures.

It has been suggested [Be, K] to protect software against duplication by selling a *physically shielded* CPU together with an *encrypted* program. The CPU will contain the corresponding decryption key, and will be installed in a computer system. The CPU will execute the program using the memory, I/O devices and other components of the computer. As customary, the CPU itself will contain only a small amount of storage space. We stress that only the CPU will be physically shielded and that all other components of the computer, including the memory in which the encrypted program and data are stored, will not be shielded.

The above setting is on the right track. It only uses a small amount of physical protection (shielding), and its implementation is feasible in current technology. However, the above setting does not constitute a full solution since it was not specified exactly how the CPU is to execute the program using the memory. A naive specification states that the computer operates as an ordinary Random Access Machine, except for the extra encryption and decryption performed by the CPU. This naive specification is not good enough, since certain properties of the program as its loop structure will not be kept secret from an observer. It is true that *straightforward duplication* of the program is not possible since one part of the program (i.e. the key) is in the shielded CPU which is unduplicatable. But protection against duplication should mean more than foiling straightforward attempts. In particular it should mean that the user is unable to learn enough about the program so that he can latter reconstruct it by himself. We thus view the above setting (i.e. a small shielded CPU and an encrypted program) as the start point for the study of software protection, rather than as a satisfactory solution. In fact, we will use this setting as the framework for our investigations, which are concerned with the following key question:

What can a user learn about the program he bought?

1.1 What Can Be Learnt by Executing a Program

We recall that the program consists of an encrypted code and a shielded CPU capable of "executing" the code (on an external memory device which may be monitored by the user). The user can run the program on inputs of its choice and watch the sequence of memory accesses during such executions. Furthermore, he

can even interfere in the execution by changing the contents of the memory locations. In any case, the pattern of memory accesses certainly carries knowledge about the program. In many cases, one can easily infer from the access pattern essential properties of the program such as its loop structure. In some cases, this may suffice in order to reconstruct the program.

Our goal is to make it infeasible for the adversary to improve his ability of reconstructing the program by experimenting with it. If it is initially "easy" to reconstruct the program then we require nothing, but in case this task is initially "hard" then experimenting with the program should not help. We meet our goal by requiring that the adversary can not learn anything about the encrypted program, except for its input/output relation and its running time. Certainly, if an adversary can learn nothing (except I/O relation and running-time) from his experiments then he can not improve his ability of reconstructing the program. Thus, the notion of a CPU which defeats experiments (i.e. prevents learning about a program from its executions) is the key to preventing software duplication. Intuitively, a CPU defeats experiments if it is infeasible to distinguish the sequences of memory accesses of any two programs run by it. The technical difficulty in the definition is the need to decouple the specified behaviour of the programs (i.e. input/output relation and running time) from the sequences of memory accesses made during their executions.

Definition (sketch): We say that a *CPU defeats experiments* if no probabilistic polynomial-time adversary can, on input an encrypted program, distinguish the two cases:

- 1) The adversary is *experimenting with the genuine CPU*, which is trying to execute the encrypted program through the external memory.
- 2) The adversary is *experimenting with a fake CPU*. The interactions of the fake CPU with the memory are almost identical to those that the genuine CPU would have had with the memory when executing a dummy program (e.g. *while TRUE do skip;).* The execution of the dummy program is timed-out by the number of steps of the real program. When timed-out, the fake CPU writes to the memory the same output that the genuine CPU would have written on the "real" program (and the same input).

Constructing an efficient CPU which defeats experiments

The problem of constructing a CPU which defeats experiments is not an easy one. Essentially there are two issues: The first issue is to hide from the adversary the values stored and retrieved from memory, and to prevent the adversary's attempts to change these values and/or to launch an attack on the encryption function. This is done using traditional cryptographic techniques (e.g. probabilistic encryption [GM] and message authentication [GGM]) in an innovative manner. The second issue is to hide (from the adversary) the sequence of instructions and variables accessed during the execution (hereafter referred as *hiding the access pattern*).

Hiding the memory access pattern is a completely new problem and traditional cryptographic techniques are not applicable to it. A trivial but unacceptably wasteful solution consists of scanning through the entire memory each time a variable needs to be accessed. In this paper, we provide an efficient solution to the problem of hiding the access pattern. This solution is the basis of our construction of an efficient CPU which defeats experiments.

Main Theorem: Let m denote the size of the external memory, and assume that one-way permutations exist. Then there exist a way to execute programs (through the memory) without leaking any knowledge about them, such that t instructions of the original program require only $t \cdot m^\varepsilon$ memory accesses, $\varepsilon > 0$.

(The actual expression is $t \cdot 2^{\sqrt{2\log_2 m \cdot \log \log_2 m}}$.)

1.2 The Hidden Access Game

The Main Theorem is proved by reducing the problem of executing programs without leaking knowledge about them, to a “hidden access game”. The reduction uncouples the traditional cryptographic issues of encryption and authentication from the new issue of hiding an access sequence. The access game consists of a main player (called the *magician*), m marked balls, and $2m$ boxes each capable of storing a single ball. Initially the m balls are placed in the first m boxes, such that ball i is in the i th box. The magician can hold only a single ball in his hands at any time. There are two additional players called the *instructor* and the *adversary*. The game proceeds in rounds as follows. In each round, the instructor secretly specifies to the magician a ball (say ball i), and the magician “answers” by conducting a sequence of actions such that at the sequence’s end the magician holds ball i in his hands. The magician’s actions consists of inserting his hand into a box for a moment, during which he either drops a ball or takes a ball or does nothing. The adversary can only sees into which box the magician has inserted his hand, but cannot see whether the magician dropped a ball, took a ball or did nothing. (It goes without saying that the adversary cannot see through the box.) The instructor is not collaborating with either magician or adversary. Can the magician follow the game without allowing the adversary to learn anything about the instruction sequence? More precisely, we require that the sequence of visible actions yields no information about the sequence of instructions.

There is a wasteful solution corresponding to the simple solution of the software protection problem: on every instruction the magician inserts his hand to all boxes in a predetermined order. Our proof of the above Theorem offers a better solution: in order to follow t instructions the magician needs to make only $t \cdot 2^{\sqrt{2\log_2 m \cdot \log \log_2 m}}$ actions (hand insertions).

Remark: The access game studied in this paper can be viewed as the Random Access Machine analogue of the *oblivious Turing Machine* problem studied by Pippenger and Fischer [PF]. The difference is that their solution heavily relies on the fact in their setting the instruction pattern is local (i.e. after asking for ball i , the instructor can only ask for either ball $i-1$ or ball $i+1$).

ORGANIZATION

In Section 2 we establish a formal framework and present a definition of the phrase “a CPU executes programs without leaking knowledge about them”. In Section 3 we sketch a reduction of the the problem of implementing such executions to the problem of implementing a magician in the above access game. *The reader who is merely interested in the access game is encourage to skip these sections and proceed directly to Sections 4 and 5.* Section 4 consists of the first non-trivial solution to the access game: a solution involving an

overhead factor of \sqrt{m} . In Section 5, a recursive solution involving an overhead of $2^{\sqrt{2\log m \cdot \log \log m}}$ - is presented. In Section 6, we present a $\Omega(\log m)$ lower bound on the overhead in a solution to the access game. We conclude with some remarks and open problems.

2. OUR DEFINITION OF SOFTWARE PROTECTION

Loosely speaking, our definition of protected software is that the adversary having the CPU and the encrypted program can “learn” nothing “substantial” about the program except for its input/output relation and running time. In order to present a formal definition we need first to define the interaction between the CPU, memory, adversary and to parameterize the encryption. We next turn to define transformations on programs (compilers) and define “learning substantially” as the ability to distinguish the original programs by monitoring the executions of their compiled mappings. Compilers which map programs in a manner that defeats any attempt to learn something substantial are then defined as protecting software. The reader may note that in this section we present the transformations on programs as compilers while in the introduction they were presented as interpreters. This difference is clearly not essential.

2.1. Interactive Machines, CPU, Memory, Programs, and Encryption

We start by defining the memory and the CPU as two interacting machines. The definition matches the standard notion of a RAM (e.g. [AHU]) in case the memory and CPU are interacting with each other. The only detail worth emphasis is that the CPU can only use space linear in its input parameter.

Definition 1 (Probabilistic Interactive Machines - sketch): A *probabilistic interactive machine (PIM)* consists of a read-only input tape, a write-only output tape, a *work tape* and a finite control. In addition to the above the PIM may receive and send messages through a special communication channel.

Definition 2 (Linear PIM): A *linear PIM* is a PIM that on input x accesses only the first $O(|x|)$ cells of its work tape.

Definition 3 (Memory): The *memory* is a (linear) PIM operating as hereby specified. On input a string y partitioned (by special marks) into m blocks, the memory copies the input to its work tape, and from this point on considers the i th block of y as its i -th cell. Subsequently, the memory is message driven. When reading a new message of the form (σ, i, z) the memory acts as follows. If $\sigma=S$ and $1 \leq i \leq m$ then the memory *sends* a message consisting of the current contents of its i -th cell. If $\sigma=P$ and $1 \leq i \leq m$ then the memory *puts* z as the new contents of its i -th cell (if z is too long -- it is truncated). If $\sigma=T$ the the memory outputs the contents of its work tape, and stops. In case none of the above holds, the memory remains idle.

Remark: For the sake of simplicity, we have assumes at this point that the programs conduct all their computation in the space occupied initially by the input. In practice, the actual input will be padded by blanks to allocate sufficient work space for the execution. The padded input will then serve as input to the program. An alternative approach, in which the memory size grows during the execution to meet workspace needs, will

be explored in the full version of this paper.

Definition 4 (CPU - sketch): The *CPU* is a linear PIM which operates as hereafter specified. The input to the CPU is ignored, and its only purpose is to trigger the execution of the CPU, and to specify the permitted “length” of the CPU’s work tape. The CPU starts its execution by sending a (fetch) message of the form $(S, 1, \cdot)$. Subsequently it operates in “rounds”. In each round it reads a new arriving message (into its work tape), applies a polynomial-time computation to its work tape (an “elementary operation in the terminology of the RAM model [AHU]), and concludes by sending a message (consisting of the contents of a portion of its work tape). (After sending a message of the form (T, \cdot, \cdot) -- the CPU halts.)

Definition 5 (programs, data, and computations): The input to the memory (y) is partitioned (by a special symbol) into two parts called the *program* (denoted here as π) and the *data* (denoted x). The output of the memory (on input $y = (\pi, x)$), after interacting with the CPU, is denoted $\pi(x)$ and called *the result of π ’s computation on input x* .

Definition 6 (Probabilistic Encryption and its Security [GM] - sketch): A *probabilistic encryption scheme* is a triplet of probabilistic polynomial-time algorithms denoted G, E, D . On input n (in unary) algorithm G outputs a (legal) key K of length n . On input a key K and a message M , algorithm E randomly selects an encryption denoted $E_K(M)$, such that $D_K(E_K(M)) = M$. Loosely speaking, we say that the encryption scheme is *secure* if on input n (in unary), and the messages M_1 and M_2 , their probabilistic encryptions $E_K(M_1)$ and $E_K(M_2)$ (where $K = G(n)$) are polynomially-indistinguishable (even when given access to a black box implementing E_K).

Remark: We do not assume here that the encryption scheme is public-key.

2.2. Cryptographic CPU, Specification Oracle, and Compilers

Definition 7 (Cryptographic CPU - sketch): The *Cryptographic CPU (CCPU)* operates essentially as a CPU except for the following details:

- 1) The input is considered as a cryptographic key K of length n (and is not ignored).
- 2) The CCPU can effect (as an “elementary operation”) E_K and E_K^{-1} on any string of length n .

Remark: The time and space complexities of effecting E_K and E_K^{-1} are ignored in the above definition. In considering an implementation of a CCPU, the time complexity of effecting E_K enters as a multiplicative factor, while the space complexity enters as an additive term. Both complexities depend only on the length of K , and thus are independent of the length of the data (to the program run by the CCPU). In the factoring-based implementation, the time complexity is $O(n^3)$ while the space complexity is $O(n)$.

Definition 8 (A specification oracle): A *specification oracle for a program π* , is an oracle that on query x returns $(\pi(x), t_{\pi}(x), s_{\pi}(x))$, where $\pi(x)$ is the output of π on input x , $t_{\pi}(x)$ is the running-time of π on input x , and $s_{\pi}(x)$ is the storage-requirement of π on input x .

Remark: For the sake of simplicity, we assume in the rest of this extended abstract that both $t_{\pi}(x)$ and $s_{\pi}(x)$ depend only on the length of x . Furthermore, we will assume that these functions are easily computable.

Thus, the only interesting thing in the oracle's answer is $\pi(x)$.

Our objective is to claim that no adversary can learn anything about π , when given input $E_K(\pi)$ and interacting with the CCPU. This is false when π is executed in the straightforward manner. What we do is map π into a "functionally equivalent" program π' and execute π' . We will require that no adversary given the encryption of π' (and interacting with the CCPU) can learn anything about π .

Definition 9 (Compiler): A compiler C is a probabilistic polynomial time algorithm that on input an integer n (in unary) and a program π outputs an n -bit cryptographic key $K = G(n)$ and an encrypted program $E_K(\pi')$, such that for every x , $\pi(x) = \pi'(x)$. We denote π' by $C(\pi)$.

2.3. Software Protection and its Cost

Now we are ready to state our definition of software protection. Loosely speaking, a compiler is said to protect software if whatever can be efficiently computed on input an (encrypted) compiled program (when interacting with a CCPU (having the corresponding key)) -- can be efficiently computed given access only to the specification oracle for the program.

Notation (sketch): By 1^n we mean the unary representation of n . When writing $D_1(x) \equiv_P D_2(f(x))$, we mean that the probability distributions generated by the algorithms D_1 and D_2 on "random" x 's are polynomially indistinguishable [GM, Y]. Let A and B be interacting machines, then $A_{B(y)}(x)$ denote the probability distribution output by A on input x , when A is interacting with B which gets y as a private input (i.e. A does not get y). Let M be an oracle-machine, and π be a program, then $M^{\pi}(x)$ denotes the output distribution of M on input x and access to a specification oracle for π .

Definition I (Software Protection -- sketch): Let P denote the CCPU. The compiler C *protects software* if for every probabilistic polynomial-time interacting machine A , there exists a probabilistic polynomial-time oracle-machine M , such that for all programs π the following holds

$$A_{P(K)}(E_K(C(\pi))) \equiv_P M^{\pi}(1^{|K|}),$$

where K is chosen randomly among all cryptographic keys of length $|K|$.

Definition II (cost of software protection): Let π be a program, and $t_{\pi}(x)$ be as in Definition 8. Let C be a compiler. Let f_C^{π} be a function from integers to reals, such that $f_C^{\pi}(m)$ is the maximum, taken over all m -bit strings x , of $t_{C(\pi)}(x)/t_{\pi}(x)$. Let $f_C(m)$ be a function such that for every π and for sufficiently large m , $f_C^{\pi}(m) \leq f_C(m)$. Then the *overhead created by the compiler C* is at most $f_C(m)$.

3. REDUCTION TO AN ACCESS GAME

The access game, described in the introduction, can be formulated as a randomized procedure that on input a sequence α of elements out of $\{1, 2, \dots, m\}$ outputs two sequences, a *visible* sequence β and a *secret* sequence γ . The sequence β contains elements out of $\{1, 2, \dots, 2m\}$, while the sequence γ consists of elements out of $\{T, D, N\}$. (The reader may think of α as being the instruction sequence, of the procedure as being the

magician, of the visable sequence as the sequence of cells into which the magician has inserted his hand, and of the secret sequence as of what he did when inserting his hand. T stands for "take a ball", D for "drop", and N for "do nothing".) The visable sequence β gives no information about the sequence α (i.e. the conditional probability that the input is α given that the visable output is β equals the a-priori probability that the input is α). The execution of β with γ gives a sequence δ which contains α . (δ is the sequence of balls held in the magician's hand, when he inserts his hands to cells β and acts in them according to γ .) Furthermore, the procedure should satisfy the above conditions when working on-line: every new element of α should cause the procedure to output new portions of β and γ such that they "contain" the element.

In addition, we require that the randomized procedure is efficient in the following sense:

- 1) The next output symbol is computed in time polynomial in m ;
- 2) The space used is logarithmic in m , provided that the procedure has access to a random oracle;
- 3) The procedure can compute at each moment, the number of times each ball was taken out of a cell.

Proposition (The Reduction): Suppose that there exist one-way permutations, and there is a procedure satisfying the above conditions such that for every input of length t it outputs sequences of length $t \cdot f(m)$. Then there exists a compiler that protects software with overhead at most $f(m)$.

The proof employs the following "traditional" cryptographic techniques - probabilistic encryption [GM], pseudorandom function [GGM], and (provably secure) message authentication [GGM].

- 1) Probabilistic encryption is used in order to make it infeasible to tell anything about the contents of a memory location. The existence of one-way permutations implies the existence of probabilistic encryption schemes [GM, Y]. More efficient schemes exist under the intractability of factoring [ACGS, BG].
- 2) The pseudorandom functions replace the random oracle used by the procedure. It is crucial that they can be implemented using "small" space. Pseudorandom functions exist if one-way permutations exist [BM, Y, GGM].
- 3) Message authentication is used in order to prevent the adversary launching a chosen ciphertext attack on the encryption scheme. Another use of authentication is to prevent the adversary from switching the contents of memory location, or to replace the contents by a previous contents of the same location. (It is thus crucial that the procedure satisfies the additional condition (3).) Message authentication is implemented using pseudorandom functions [GGM].

4. THE "SQUARE ROOT" SOLUTION

We will describe the solution, using the intuitive "magician" formalism of the introduction. Recall that there are m balls marked 1 through m , which initially reside in the first m cells such that ball i is in the i -th cell. Altogether there are $2m$ cells, and suppose that $m \geq 2\sqrt{m}$.

The solution described below allows the magician to follow a sequence of t instructions, by committing at most $t \cdot \sqrt{m}$ actions. We describe a solution in which the magician is allowed to hold up to 2 balls at any point in time.

Following is an outline of the magician's procedure:

- 0) Initially, for $1 \leq i \leq m$, the i th cell contains ball number i . All other cells are empty.

while TRUE *do*:

- 1) Randomly permute the contents of the first $m + \sqrt{m}$ cells. That is, select a permutation π over the integers 1 through $m + \sqrt{m}$ and relocate the contents of cell i in cell $\pi(i)$.
- 2) Execute \sqrt{m} instructions as follows. During the execution of these instructions, maintain the balls (accessed by these instructions) in cells number $m + \sqrt{m} + 1$ through $m + 2\sqrt{m}$. The instruction "get ball i " is executed as follows. First scan through the special \sqrt{m} cells and check whether ball i is in one of these cells. If the i th ball is not found there then we retrieve it from cell $\pi(i)$; else we access the next empty cell (i.e. one of the cells $m + 1$ through $m + \sqrt{m}$ which was not accessed before).
- 3) Return balls to their initial locations.

Before getting to the implementation details of the above steps, we provide some hints to as why no information about the instruction sequence is revealed by the sequence of viable actions. Step (1) is syntactically independent of the instruction sequence. The accesses executed in step (2) are of two types: scanning through all cells from the $m + \sqrt{m} + 1$ -th to the $m + 2\sqrt{m}$ -th, and accessing a new random cell between 1 and $m + \sqrt{m}$. No information about the instruction sequence is leak by this! The access pattern of Step (3) is identical to a combination of the second type of accesses made in Step (2), and the accesses of step (1).

4.1 How to randomly permute the contents of the memory

We first show how to implement a random permutation by using a random oracle and sorting, and next show how to implement sorting using a random oracle.

Choosing and "storing" a random permutation

We show how to choose and store a random permutation over $\{1, 2, \dots, t\}$, using $O(\log t)$ storage and a random oracle. The idea is to use the oracle in order to tag the elements with random distinct (with high probability) integers. The permutation is obtained by sorting the elements by their tags⁽¹⁾. (It suffices to have the tags being drawn at random from the set $\{1, 2, \dots, t^{\log t}\}$.) Let $f : \{1, 2, \dots, t\} \rightarrow \{1, 2, \dots, t^{\log t}\}$ be a random function trivially constructed by the random oracle. Then $\pi(i) = k$ if and only if $f(i)$ is the k -th smallest element in $\{f(j) : 1 \leq j \leq t\}$.

1) Remark: Luby and Rackoff [LR] showed that three iterations of the DES can be used to construct a pseudorandom permutation out of three random functions. However, this pseudorandom permutation is not good enough for our purposes since it can be distinguished from a random permutation with probability $\Theta(q^2/t)$, where q is the number of permutation evaluations.

Arranging the balls by the chosen permutation

Now we face the problem of sorting the t elements (by their tags) in a manner which leaks no information about the permutation. The crucial condition is that the magician which executes the sorting can store only a fix number of balls (say 2) at a time. The idea is to “implement” Batcher’s Sorting Network [Bat], which allows to sort t elements by $t \lceil \log_2 t \rceil^2$ comparisons. Each comparison is “implemented” by accessing both the corresponding cells, retrieving their contents, and then putting the contents back in the desired order. The sequence of accesses generated for this purpose is fixed and independent of the permutation to be implemented. Note that the magician can easily compute at each point which comparison he needs to implement next. This is due to the simple structure of Batcher’s network, which is uniform with respect to logarithmic space⁽²⁾.

Computing the permutation in succeeding steps (2) and (3)

The way the permutation π is defined does not allow an immediate method of computing $\pi(i)$ on input i . This computation will be required in the subsequent executions of steps (2) and (3). We will “compute” $\pi(i)$ by conducting a binary search on the $f(\cdot)$ ’s, using the fact that (after step (1)) the $f(\cdot)$ ’s are “stored”, in sorted order, in the cells. Note that $\pi(i)$ is computed in order to access the $\pi(i)$ -th cell, and therefore the accesses done in the binary search do not add any information (since they are determined by $\pi(i)$).

4.2 How to simulate a single access

Now it is straightforward to give the details of Step (2). Throughout step (2), $count$ maintains the number of single accesses simulated in the current run. $count$ is initially 0 and is incremented until it reaches \sqrt{m} . On instruction “take ball i ” the magician proceeds as follows:

- 2a Scans through locations $m + \sqrt{m} + 1$ to $m + 2\sqrt{m}$. If the ball i is in either of these cells then fetch it, and sets j such that ball i was taken from the $m + \sqrt{m} + j$ -th cell. If neither of these cells contains ball i then set $j = count$.
- 2b If $j \neq count$ then the magician accesses the $\pi(m + count)$ -th cell (which is empty!); else the magician accesses the $\pi(i)$ -th cell, and retrieve its contents (i.e. ball i).
- 2c Scans through locations $m + \sqrt{m} + 1$ to $m + 2\sqrt{m}$ again, and put ball i in the $m + \sqrt{m} + j$ -th cell. Increments $count$ by 1.

4.3 How to rearrange the balls

Rearranging the balls is done in two substeps: first we undo the effect of the execution of Step (2) and next the effect of Step (1). Following is a description of the first substep. The second substep can be incorporated in the next execution of step (1).

For $j=1$ to \sqrt{m} the magician proceeds as follows:

Accesses the $m + \sqrt{m} + j$ -th cell. If it contains a ball, say ball i , then the magician accesses the $\pi(i)$ -th cell and puts ball i there. If the $m + \sqrt{m} + j$ -th cell is empty then the magician accesses its $\pi(m + j)$ -th cell (but puts nothing there).

⁽²⁾ The simplicity of Batcher sorting network is the main reason we prefer it upon the asymptotically superior Ajtai-Komlos-Szemerédi sorting network [AKS].

4.4 Analysis

The reader may easily verify that the sequence of accesses of the magician indeed yields no information about the sequence of instructions. It is left to calculate the overhead of the simulation (i.e. the ratio of accesses over instructions). The permutation applied after every \sqrt{m} instructions causes an overhead of $O(m \cdot \log^2 m)$, which amounts to an amortized overhead of $O(\sqrt{m} \cdot \log^2 m)$ actions per instruction. In addition, each of the instructions causes $O(\sqrt{m})$ actions to be taken in step (2). Other actions taken in step (3) are negligible in number. The total overhead thus amounts to $O(\sqrt{m} \cdot \log^2 m)$ actions per instruction⁽³⁾. We get

Theorem 1: There exist a magician procedure with $O(\sqrt{m} \cdot \log_2 m)$ overhead.

Furthermore, this procedure is efficient in the sense of Section 3.

5. THE RECURSIVE SOLUTION

The recursive solution presented in this section is based on a generalization of the solution presented in Section 4. One can view the solution of Section 4 as consisting of two parts: the random shuffling and reshuffling of the cells contents every \sqrt{m} original accesses (steps (1) and (3)), and the simulation of the instructions through their randomized locations (step (2)). Substeps (2a) and (2c) actually simulates a “powerful” magician which can hold up to \sqrt{m} balls in its hands at any time: The magician looks whether he holds already the required ball. If the answer is negative then the magician fetches the ball, else he reaches for a “new” empty cell. Holding up to \sqrt{m} balls was simulated in the obvious manner by scanning through extra \sqrt{m} cells.

When trying to generalize the solution, we want to decrease the amortize cost of the random shuffling. Thus we will consider a more powerful magician capable of holding up to $f(m) > \sqrt{m}$ balls (say $f(m) = m^{3/4}$). The amortized cost of steps (1) and (3) is thus $m \cdot (\log_2 m)^2 / f(m)$. The key question is: *how are we going to simulate the magician with the $f(m)$ -size hand?*

We will think of the magician’s hand as a heap containing up to $f(m)$ elements. We need to support $f(m)$ find operations and up to $f(m)$ element insertions to this heap. This translates to $f(m) \cdot \log_2 f(m)$ access operations to the data structure. We can view the situation as a new simulation, this time on $O(f(m))$ cells and for $O(f(m) \cdot \log f(m))$ instructions. In other words, we need to issue $\log_2 f(m)$ new instructions into the $f(m)$ -size hand per each instruction into the original cells. We thus get.

$$\text{overhead}(m) = \frac{m \cdot (\log_2 m)^2}{f(m)} + O(\log f(m)) \cdot \text{overhead}(f(m))$$

Solving the recurrence, we get $\text{overhead}(m) = O(2^{\sqrt{2\log_2 m \cdot \log_2 \log_2 m}})$. Thus

3) Actually, the above choice of parameters is not optimal. Repermuting the balls after every $O(\sqrt{m} \cdot \log m)$ instructions, yields an overhead of $O(\sqrt{m} \cdot \log m)$ actions per instruction.

Theorem 2: There exist a magician procedure with $O(2^{\sqrt{2\log m \cdot \log \log m}})$ overhead.
 Furthermore, this procedure is efficient in the sense of Section 3.

6. A LOWER BOUND

A simple combinatorial argument shows that any oblivious simulation of arbitrary RAMs should have an average $\Omega(\log m)$ overhead.

Theorem 3: Every successful magician procedure must make at least $m + (t-1)\log_3 m$ actions in order to implement t instructions.

The proof uses very little of the structure of the problem, and therefore we do not believe that the lower bound obtained is tight.

7. CONCLUSIONS AND OPEN PROBLEMS

We have reduced software protection to a “hidden access game”. The reduction was carried out on the instruction level. However, an identical reduction can be carried out on any level of programming modularity; e.g. cash memory accesses, paging mechanisms etc.

The hidden access game has also other applications, as allowing many users to run secretly private programs on a public computer, foiling flow analysis in distributed communication networks etc.

Formulating the problem of preventing software duplication has lead us to consider the question of what can be learnt about a program by watching its executions. It seems that formulating the problem of preventing software distribution (i.e. fingerprinting software) will lead to different questions. It will be very interesting to try and come up with a theoretical framework and definitions for “fingerprinted software”.

A more technical problem is to provide better solutions to the hidden access game, and in turn to present compilers which protect software at lower cost. We believe that this should be possible. On the other hand, proving better lower bounds on the overhead of good magicians will be also interesting. At this point the gap between the known upper and lower bounds, on the overhead, is quite large: $O(2^{\sqrt{2\log m \cdot \log \log m}})$ versus $\Omega(\log m)$.

ACKNOWLEDGEMENTS

It is my pleasure to thank friends and colleagues for their contributions to this work and its presentation. In particular I wish to thank Baruch Awerbuch, Benny Chor, Shimon Even, Shafi Goldwasser, Silvio Micali, Ron Rivest, and Yacov Yacobi. Special thanks to Hugo Krawczyk for carefully reading an earlier version of the manuscript, pointing out some errors, and suggesting several improvements.

REFERENCES

- [AHU] Aho, A.V., J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley Publ. Co., 1974.
- [AKS] Ajtai, M., J. Komlos, and E. Szemeredi, "An $O(n \cdot \log n)$ Sorting Network", *Proc. 15th STOC*, 1983, pp. 1-9.
- [ACGS] Alexi, W., B. Chor, O. Goldreich, and C.P. Schnorr, "RSA and Rabin Functions: Certain Parts Are As Hard As The Whole", to appear in *SIAM Jour. on Computing*. Extended Abstract in *Proc. 25th FOCS*, 1984.
- [Bat] Batcher, K., "Sorting Networks and their Applications", *AFIPS Spring Joint Computer Conference*, 32, 1968, pp. 307-314.
- [Be] Best, R., "Microprocessor for Executing Encrypted Programs", US Patent 4,168,396. Issued September 1979.
- [Blu] Blum, M., unpublished manuscript, 1983.
- [BG] Blum, M., and S. Goldwasser, "An Efficient Probabilistic Public-Key Encryption Scheme which Hides All Partial Information", *Advances in Cryptology: Proceedings of CRYPTO 84*, Springer Verlag, Lecture Notes in Computer Science (196), 1985, pp. 289-299.
- [BM] Blum, M., and Micali, S., "How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits", *SIAM Jour. on Computing*, Vol. 13, 1984, pp. 850-864.
- [GGM] Goldreich, O., S. Goldwasser, and S. Micali, "How to Construct Random Functions", *Proc. of 25th Symp. on Foundation of Computer Science*, 1984, pp. 464-479. To appear in *Jour. of ACM*.
- [GM] Goldwasser S., and S. Micali, "Probabilistic Encryption", *Jour. of Computer and System Science*, Vol. 28, No. 2, 1984, pp. 270-299.
- [K] Kent, S.T., "Protecting Externally Supplied Software in Small Computers", Ph.D. Thesis, MIT/LCS/TR-255, 1980.
- [LR] Luby, M., and C. Rackoff, "Pseudo-random Permutation Generators and Cryptographic Composition", *Proc. of 18th STOC*, 1986, pp. 356-363.
- [PF] Pippenger, N., and M.J. Fischer, "Relation Among Complexity Measures", *Jour. of ACM*, Vol. 26, No. 2, 1979, pp. 361-381.
- [Y] Yao, A.C., "Theory and Applications of Trapdoor Functions", *Proc. of the 23rd IEEE Symp. on Foundation of Computer Science*, 1982, pp. 80-91.

APPENDIX

Definition (polynomial indistinguishability [GM, Y]): Let $\{\Pi_1^n\}$ and $\{\Pi_2^n\}$ be two probability ensembles; that is for every integer n and i , Π_i^n is a probability distribution on strings of length $\leq \text{poly}(n)$. The ensembles $\{\Pi_1^n\}$ and $\{\Pi_2^n\}$ are *polynomial indistinguishable* if the following holds:

For every probabilistic polynomial-time algorithm A , every constant c , and sufficiently large n , the probability that A outputs 1 on n and a string selected according to Π_1^n equals up to n^{-c} the probability that A outputs 1 on a string selected in Π_2^n .

Definition 5 (Probabilistic Encryption and its Security [GM]): A *probabilistic encryption scheme* is a triplet of probabilistic polynomial-time algorithms denoted G, E, D such that:

- 1) On input n (in unary representation) algorithm G outputs a key K of length n .
- 2) On input a key K and a message M , algorithm E outputs an encryption denoted $E_K(M)$.
- 3) On input a key K and an encrypted message $E_K(M)$, algorithm D always outputs M . In case D is given an illegal key-encryption pair it may behave arbitrarily.

The following security definition implies that both G and E can map an input to many (more than polynomially many) possible outputs. Let M_1 and M_2 be two messages and A be a probabilistic polynomial-time machine which operates as follows. Algorithm A receives as input two encryptions $E_K(M_1)$ and $E_K(M_2)$ in arbitrary order. In addition, algorithm A is given access to a black box implementing E_K (i.e. when sending q to the black box, A receives back as an answer a random encryption $E_K(q)$). Let $\Pi_{A,1}^n$ denote the probability that A outputs 1 when K is a key randomly chosen by G on input n , and the encryption of M_1 was placed to the left of the encryption of M_2 on the input tape. Similarly, $\Pi_{A,2}^n$ denotes the probability of output 1 when M_2 's encryption was placed to the left of M_1 's. An encryption scheme is *secure* if for every two messages M_1 and M_2 and every probabilistic polynomial-time algorithm A , $\{\Pi_{A,1}^n\}$ and $\{\Pi_{A,2}^n\}$ are polynomially-indistinguishable.

Remark: The original definition of encryption security [GM] is for Public-Key Encryptions and thus access to a black box implementing E_K is redundant. Above we gave a more general definition that suits both Private-Key and Public-Key encryptions.

Two Observations on Probabilistic Primality Testing

Pierre Beauchemin Gilles Brassard¹ Claude Crépeau² Claude Goutier

Université de Montréal
C.P. 6128, Succ "A"
Montréal (Québec)
Canada H3C 3J7

1. Introduction

In this note, we make two loosely related observations on Rabin's probabilistic primality test. The first remark gives a rather strange and provocative reason as to *why is Rabin's test so good*. It turns out that a single iteration fails with a non-negligible probability on a composite number of the form $4j+3$ *only if* this number happens to be easy to split. The second observation is much more fundamental because it is not restricted to primality testing: it has profound consequences for the entire field of probabilistic algorithms. There we ask the question: *how good is Rabin's algorithm?* Whenever one wishes to produce a uniformly distributed random probabilistic prime with a given bound on the error probability, it turns out that the size of the desired prime must be taken into account.

2. A Brief Survey of Primality Testing

How difficult is it to distinguish prime numbers from composite numbers? This is perhaps the single most important problem in computational number theory. We do not attempt here an exhaustive review of its long history. Let us only mention some of the most outstanding modern steps. It has been known for several years that the problem of recognizing prime numbers belongs to **P** under the Extended Riemann's Hypothesis [Mi] and that it belongs to Co-RP [R1, SS] and NP [P] without any assumptions. It can also be solved in *almost* polynomial time by a deterministic algorithm that runs for a number of steps in $O(m^{O(\log \log n)})$, where m is the size of the number to be tested [APR]. More recently, it was found to lie in RP [GK, AH], and therefore in ZPP [G] as well. In other words, this problem can be solved in probabilistic polynomial time by a Las Vegas [B] algorithm: whenever an answer is obtained, that answer is correct.

From a theoretical point of view, the problem of primality testing is therefore solved (although it remains of interest to figure out whether or not it belongs to **P** without assumptions). However, the polynomial that gives the running time of [GK] is of the twelfth degree and [AH] does not improve on this, which makes these algorithms of little practical use. For very large numbers (several hundreds of decimal digits), this leaves us with Rabin's probabilistic test [R1] as the best approach.

1. Supported in part by NSERC grant A4107

2. Supported in part by an NSERC postgraduate scholarship; current address: M.I.T.

Let $\text{prob}[\text{Rabin}(n) = \text{verdict}]$ denote the probability that one iteration of this algorithm on input n returns *verdict*, where *verdict* can either be “prime” or “composite”. The basic theorem about Rabin’s test is that

$$\text{prob}[\text{Rabin}(n) = \text{“prime”} \mid n \text{ is indeed prime}] = 1$$

whereas

$$\text{prob}[\text{Rabin}(n) = \text{“prime”} \mid n \text{ is in fact composite}] \leq \frac{1}{4}.$$

One is therefore certain that n is composite whenever any single run of *Rabin*(n) returns “composite”. On the other hand, one can never be sure that n is a prime no matter how many runs of *Rabin*(n) have returned “prime”. This test is usually run in a loop as follows :

```
function RepeatRabin( $n, k$ )
  {  $n$  is an odd integer to be tested for primality;
     $k$  is a safety parameter discussed below }
  var  $i$  : integer;  $done$  : Boolean
   $i \leftarrow 0$ 
  repeat
     $i \leftarrow i + 1$ 
     $done \leftarrow (\text{Rabin}(n) = \text{“composite”})$ 
    if  $done$  then return “composite” { for sure }
    else return “prime” { probably (?) }.
```

There is a trade-off in the choice of the parameter k above : the bigger it is, the more confident we are in the advent of a “prime” answer but the more time it takes to build up this confidence. This paper addresses two aspects of the question : *just how confident in a number’s primality can we be after running this test?*

3. Why is Rabin’s Test so Good?

This section only applies when n is of the form $4j+3$. In this case, Rabin’s test (which is then equivalent to Solovay-Strassen’s [SS]) becomes quite simple. Let

$$\mathbb{Z}_n^* = \{x \mid 1 \leq x < n \text{ and } \gcd(x, n) = 1\}$$

$$\text{and } R_n = \{a \in \mathbb{Z}_n^* \mid a^{(n-1)/2} \equiv \pm 1 \pmod{n}\}.$$

The basic theorem states that $R_n = \mathbb{Z}_n^*$ whenever n is a prime, whereas $\#R_n \leq \#\mathbb{Z}_n^*/4$ otherwise, still assuming that $n \equiv 3 \pmod{4}$. Notice that both 1 and $n-1$ always belong to R_n . This theorem is used naturally as follows :

```
function Rabin( $n$ )
  { we assume that  $n$  is of the form  $4j+3$  }
   $a \leftarrow$  random integer uniformly selected in  $2..n-2$ 
  if  $a \in R_n$  then return “prime”
  else return “composite” .
```

Whenever n is composite, the error probability of this procedure is clearly given by $(\#R_n - 2)/(n-3)$, so that elements of R_n others than 1 and $n-1$ are known as *false witnesses* for n . From the basic theorem, we know that this error probability is always smaller than 25%. However, it is well known to be often *much* smaller. Monier gives an exact (but rather scaring) formula for this probability [Mo]; see also [Kr]. As a corollary of Monier’s formula, the error probability never exceeds $(\phi(n)/2^{r-1} - 2)/(n-3)$, where r is the number of distinct prime factors of n and $\phi(n) = \#\mathbb{Z}_n^*$.

denotes Euler's function [HW]. Despite this tightening of the bound on the error probability (at least when n has more than three distinct prime factors), it turns out that the latter is usually *still* much smaller. In other words, Rabin's test performs in practice much better than one might naively expect.

For instance, 42,799 ($= 127 \times 337$) admits only 880 false witnesses, compared to $\phi(42,799)/4 = 10,584$. Even better, Rabin's test *never* fails on integers of the form $3 \times 5 \times 7 \times 11 \times \dots$ such as 15,015: these admit no false witnesses at all. More impressively, it is enough to test deterministically for each $a \in \{2, 5, 7, 13\}$ in order to decide primality without *any* failures up to 25×10^9 (using $\{2, 3, 5, 7\}$ still leaves *one* error in this range [PSW]). Although "high risk" numbers exist, such as $n = 79,003$ ($= 199 \times 397$) or $3,215,031,751$ ($= 151 \times 751 \times 28,351$) with $\#R_n = \#\mathbb{Z}_n^*/4$ and $(\#R_n - 2)/(n - 3) \approx 24.8\%$, these are not the rule (one can nonetheless prove, using Monier's formula, that every composite number of the form $(12m+7)(24m+13)$ is such a high risk number, provided both $12m+7$ and $24m+13$ are prime). We address here the following bizarre question: *why* is Rabin's test so good?

In order to give some sort of answer, we define the following set:

$$H_n = \{ b \in \mathbb{Z}_n^* \mid b \notin R_n \text{ and } (\exists a \in R_n)[a^2 \equiv b^2 \pmod{n}] \}.$$

Assume for the moment that n is not a prime power (i.e. not of the form p^m for some prime p and some integer $m \geq 2$). Theorem 1 states that each element of H_n is a *handle* that allows easy splitting of n (i.e. finding at least one non trivial factor of n) and that there are at least as many such handles as there are false witnesses. Our provocative interpretation states that it is only possible for a *single iteration* of Rabin's test to fail (i.e. declare n prime) with a non-negligible probability if it happens that n is easy to split (and hence obviously composite)! This result extends to every composite n of the form $4j+3$ in an obvious way since prime powers are easy to split. In other words, there exists a simple probabilistic splitting algorithm whose running time is small on every composite number congruent to 3 modulo 4 on which Rabin's test is not extremely effective. More precisely, for any polynomial p , the splitting algorithm succeeds at finding a non trivial factor in expected polynomial time on all those composite integers n such that $\text{prob}[Rabin(n) = \text{"prime"}] \geq 1/p(|n|)$ and $n \equiv 3 \pmod{4}$, where $|n|$ denotes the size of n (in bits or in decimal digits).

What happens when n is of the form $4j+1$? We leave this as an open question. Let us only point out that Rabin's test *could* actually work better on numbers congruent to 3 modulo 4 than on numbers congruent to 1 modulo 4. Indeed, among the 4842 odd composite integers smaller than 25×10^9 that count 2 as a false witness, only 1033 ($\approx 21\%$) are of the form $4j+3$ [PSW].

Theorem 1

- (i) $(\forall b \in H_n)[\gcd(n, 1 + b^{(n-1)/2} \pmod{n}) \text{ is a non trivial divisor of } n]$;
- (ii) $\#H_n \geq \#R_n$.

Proof

Except for the exponents, all calculations in this proof are done modulo n .

- (i) Consider any $b \in H_n$. Let $a \in R_n$ be such that $a^2 = b^2$. Let $x = b^{(n-1)/2}$. We know that $x \neq \pm 1$ because $b \notin R_n$. On the other hand, $x^2 = b^{n-1} = (b^2)^{(n-1)/2} = (a^2)^{(n-1)/2} = (a^{(n-1)/2})^2 = (\pm 1)^2 = 1$. Therefore, x is a non trivial square root of 1, and this is enough to split n by the well-known formula $\gcd(n, 1+x)$ [R2].

- (ii) For each $a \in R_n$, define $B_n(a) = \{ b \in \mathbb{Z}_n^* \mid a^2 \equiv b^2 \pmod{n} \}$. Because n is composite and is not a prime power, $B_n(a)$ contains at least 4 elements [HW]. Consider $b \in B_n(a)$ such that $b \neq \pm a$. We have

$$\begin{aligned} b^{(n+1)/2} &= (b^2)^{(n+1)/4} && (\text{because } n \equiv 3 \pmod{4}) \\ &= (a^2)^{(n+1)/4} = a^{(n+1)/2} \\ &= a \times a^{(n-1)/2} = \pm a && (\text{because } a \in R_n). \end{aligned}$$

Therefore, $b^{(n-1)/2} = b^{(n+1)/2} / b = \pm a / b \neq \pm 1$ because $b \neq \pm a$, hence $b \in H_n$. This shows that to each pair $a, -a$ of elements of R_n corresponds at least two distinct elements in H_n , completing the proof that H_n contains at least as many elements as R_n . Notice also that this reasoning is trivially extended to conclude, as mentioned earlier in this section, that $\#R_n \leq \phi(n) / 2^{r-1}$, where r is the number of distinct prime factors of n , because each quadratic residue admits in this case exactly 2^r distinct square roots [HW]. \square

Notice that part (i) of this proof still holds when $n \equiv 1 \pmod{4}$. Unfortunately, part (ii) fails miserably because R_n is then always empty, due to the fact that each square root of the square of a false witness is also a false witness. This fact may partly explain the observed phenomenon that Rabin's test seems to be less effective on these numbers.

4. How Good is Rabin's Test?

We must first ask the following question: *what* is Rabin's test good for? At least two answers come to mind: to *decide* on the primality of a given integer and to *generate* one or several primes (perhaps of a given size). We shall consider these two settings in turns, starting with the second.

4.1. How to Generate Random Primes of a Given Size

The generation of large primes drawn with a uniform distribution from the set of all primes of a given size is of crucial importance in cryptography [RSA]. Although it is possible to generate such primes with certainty using the algorithms of [APR, AH], their running time is currently too high to be used in practice. It is also possible to efficiently generate large certified primes by a variation on Pratt's non-deterministic algorithm [P] (generate the NP certificate and the resulting prime hand in hand) or by more sophisticated techniques [CQ], but the resulting distribution would not be uniform. Again, the most attractive solution in practice is to use Rabin's test as follows:

```
function GenPrime(l, k)
  { l is the size of the prime to be produced;
    k is a safety parameter discussed below }
  repeat
    n ← randomly selected l digit odd integer
  until RepeatRabin(n, k) = "prime"
  return n .
```

The resulting output is a *probabilistic prime* in the sense that we can never be assured that it is indeed prime. We can nonetheless increase our confidence in the number's primality by increasing the safety parameter k . (What a shame that Rabin's algorithm can certify those cryptographically useless composite numbers whereas it can only give probabilistic information on the useful primes! — which is precisely why [GK, AH]'s algorithms are of such (as yet theoretical) interest.)

In order to use *GenPrime* for cryptographic purposes, it is important that its probability of returning a composite integer be estimated. The popular belief is that

$$\text{prob}[\text{GenPrime}(l, k) \text{ is composite}] \leq 4^{-k}$$

because each of the k rounds of *RepeatRabin* has a probability smaller than $\frac{1}{4}$ of failing on any given composite number. If we repeatedly use *GenPrime* to produce m distinct "primes", we therefore expect on the average that less than $m \times 4^{-k}$ of them will turn out to be composite. For instance, Knuth writes: "if we certified a billion different primes with such a procedure⁴, the expected number of mistakes would be less than $\frac{1}{1000000}$ " [Kn, page 379].

This assertion may be true⁵, but the reason is wrong. Indeed, it could *only* be true because $\text{prob}[\text{Rabin}(n) = \text{"prime"}]$ is so much smaller than $\frac{1}{4}$ on most composite numbers. Should the error probability be *exactly* $\frac{1}{4}$ on every composite odd integer, the number of expected errors would be significantly larger than $10^9 \times 4^{-25} \approx 10^{-6}$.

From now on, let us assume we use a hypothetical test (that we shall continue to call *Rabin*) such that

$$\text{prob}[\text{Rabin}(n) = \text{"prime"} \mid n \text{ is indeed prime}] = 1$$

as before, whereas

$$\text{prob}[\text{Rabin}(n) = \text{"prime"} \mid n \text{ is in fact composite}] = \frac{1}{4} \text{ (exactly)}.$$

It turns out that the error probability of each instantiation of *GenPrime*(l, k) depends on the size of the desired prime. As one can easily compute from Lemma 1 and Theorem 2 below, the expected number of errors exceeds $l \times 10^{-6}$ when $k=25$, provided $35 \leq l \leq 2 \times 10^{13}$. In particular, if one wanted one billion 1000-digit primes, the expected number of mistakes would exceed 0.001. To be more provocative, if one had a need for one billion one-billion-digit primes, running Rabin's test a mere 25 times per "prime" would result in an expected 1022 composite numbers among them. Worse still, each call to *GenPrime*($10^{15}, 25$) has a roughly 50% chance of producing a composite number!

In a nutshell, the reason for this confusion is that $\text{prob}[X \mid Y] \neq \text{prob}[Y \mid X]$ in general. In particular, if X stand for " n is composite" and Y for "*RepeatRabin*(n, k) returned "prime""", then it is true that $\text{prob}[Y \mid X] \leq 4^{-k}$, but this does *not* allow us to conclude that $\text{prob}[X \mid Y] \leq 4^{-k}$ as well. In order to get an estimate on $\text{prob}[X \mid Y]$, which is the cryptographically relevant probability, it is necessary to have an *a priori* probability that n is prime *before* even the first call to *Rabin*(n) is performed. Fortunately, the prime number theorem [HW] comes to the rescue, which is where the size of the desired prime comes into play: the *a priori* probability that a randomly selected odd l -digit integer be prime is roughly $2/l\alpha$, where $\alpha \approx 2.3$ stands for the natural logarithm of 10. More precisely,

Lemma 1

If n is uniformly, randomly selected among the odd l -digit integers, then

$$\text{prob}[n \text{ is prime}] \approx \frac{2}{(l-1) \times \alpha} \left[1 - \frac{10}{9l} \right] = 2/l\alpha .$$

4. Knuth does not *explicitly* say how he would use Rabin's test to certify those billion primes, except that he would run it "25-times-in-a-row" on each of them. It is our interpretation that he meant something along the lines of *RepeatRabin*(•, 25). Of course, Knuth's assertion is vacuously true if taken literally: if the integers thus certified are indeed "one billion primes", no mistakes are possible at all!

Proof

Immediate from the prime number theorem [HW], which says that the number of primes not exceeding n is asymptotic to $n/\ln n$. Notice that this approximation is fairly accurate even for reasonably small values of n . For instance, there are 50,847,478 primes smaller than 10^9 , whereas $n/\ln n$ would give about 48,254,942. \square

Theorem 2

Let p be the probability that a uniformly, randomly selected odd l -digit number is prime. The probability that $\text{GenPrime}(l, k)$ returns a composite number is given by

$$\frac{1}{1 + \frac{p}{1-p} 4^k}$$

This is about $(l\alpha/2) \times 4^{-k}$ provided l is substantially smaller than 4^k and about $1/2$ when $l = 2 \times 4^k / \alpha$.

Proof

Let X and Y be as before. Clearly, $\text{prob}[X] = 1-p$ and $\text{prob}[Y|X] = 4^{-k}$ (with our simplification to the effect that Rabin's test fails with probability *exactly* $1/4$ on composite numbers). We are interested in $\text{prob}[X|Y]$. We thus use the formula

$$\text{prob}[X|Y] = \frac{\text{prob}[X] \times \text{prob}[Y|X]}{\text{prob}[Y]},$$

which yields the theorem after routine algebraic manipulation because

$$\begin{aligned} \text{prob}[Y] &= \text{prob}[X] \times \text{prob}[Y|X] + \text{prob}[\text{not } X] \times \text{prob}[Y|\text{not } X] \\ &= (1-p) \times 4^{-k} + p \times 1. \quad \square \end{aligned}$$

Intuitively, the confidence we get in the number's primality from running Rabin's test several times must be weighted by the *a priori* overwhelming probability that it is composite if randomly chosen among the odd integers of a substantial size. For instance, if the size is $2 \times 4^k / \alpha$, only about 1 in 4^k odd integers is prime. If a random odd integer of this size passes k rounds of Rabin's test, it is just as likely that this occurred because we were lucky enough to hit a prime or unlucky enough to observe such behaviour on a composite number!

4.2. How to Decide on the Primality of a Given Integer

Suppose some odd integer n is given to you. You are to decide whether you think it is prime or not. You therefore run Rabin's test for some number k of rounds, and it never finds n to be composite. What can you tell from this?

One obviously wrong answer is: "this number is prime with probability $1-4^{-k}$ ". This makes no sense because any *given* integer is either prime or not.

The classic answer is: "I believe this number to be prime, and my error probability is at most 4^{-k} (in the sense that I expect to be wrong at most once every 4^k such statements if you quizz me long enough)". This is wrong as well because *no* estimate on the error probability of "I believe this

5. We *think* the assertion is true, but we have not yet actually carried out the calculation necessary to prove it.

number to be prime" can be made without an *a priori* estimate on the probability that the number is prime. If you know that it was chosen randomly and uniformly among the odd integers of some given size l , section 4.1 tells you that you can still achieve an error probability below 4^{-k} , but at the cost of running Rabin's test for an additional (roughly) $\log_4 l$ rounds. However, if you do *not* know where the number comes from, you are at a *complete* loss.

Still, there is one thing you can say: "I believe this number to be prime, and if I am wrong I have observed a natural phenomenon whose probability of occurrence was bounded by 4^{-k} ". This statement is certainly weak, but we cannot think of anything stronger one can infer in general from running Rabin's test any number of times.

As mentioned in the introduction, this observation is not restricted to primality testing. Whenever one runs *any* probabilistic algorithm that is not Las Vegas, care must be taken as to how to interpret the outcome. More implications of this issue are discussed in [BB].

Acknowledgements

The first observation sprung from an idea attributed to Silvio Micali by Charles H. Bennett (private communication) to the effect that any algorithm for extracting square roots modulo a prime can be turned into a probabilistic algorithm to decide primality. The main reason why our result applies directly only to numbers of the form $4j+3$ is that no efficient *deterministic* algorithm is known to extract square roots modulo a prime of the form $4j+1$.

No doubt the second observation has been made several times over by independant people, but we have not found any records of this. We wish to thank all those who showed interest for this work at the CRYPTO 86 meeting, in particular: Yvo Desmedt, Oded Goldreich, Gus Simmons and Moti Yung. Our thanks also go to Whitfield Diffie for granting us a full 12 minutes and 42 seconds to present these results.

References

- [AH] Adleman, L. and M.-D. Huang, "Recognizing primes in random polynomial time", *presented at CRYPTO 86*, 1986.
- [APR] Adleman, L., C. Pomerance and R. Rumely, "On distinguishing prime numbers from composite numbers", *Annals of Mathematics*, vol. 117, pp. 173-206, 1983.
- [B] Babai, L., "Monte Carlo algorithms in graph isomorphism testing", *Rapport de Recherches du Département de Mathématiques et de Statistiques*, Université de Montréal, D.M.S. #79-10, 1979.
- [BB] Brassard, G. and P. Bratley, *Introduction to Algorithmics*, Prentice-Hall, Englewood Cliffs, New Jersey, to appear.
- [CQ] Couvreur, C. and J. J. Quisquater, "An introduction to fast generation of large prime numbers", *Philips Journal of Research*, vol. 37, nos. 5/6, pp. 231-264, 1982.
- [G] Gill, J., "Computational complexity of probabilistic Turing machines", *SIAM Journal on Computing*, vol. 6, no. 4, pp. 675-695, 1977.
- [GK] Goldwasser, S. and J. Kilian, "A provably correct and probably fast primality test", *Proceedings of the 18th Annual ACM Symposium on the Theory of Computing*, 1986.
- [HW] Hardy, G. H. and E. M. Wright, *An Introduction to the Theory of Numbers*, Fifth edition, Oxford Science Publications, 1979.
- [Kn] Knuth, D. E., *The Art of Computer Programming*, volume 2 : *Seminumerical Algorithms*, Second edition, Addison-Wesley, Reading, Massachusetts, 1981.

- [Kr] Kranakis, E., *Primality and Cryptography*, Wiley-Teubner Series in Computer Science, 1986.
- [Mi] Miller, G. L., "Riemann's hypothesis and tests for primality", *Journal of Computer and System Sciences*, vol. 13, pp. 300-317, 1976.
- [Mo] Monier, L., "Evaluation and comparison of two efficient probabilistic primality testing algorithms", *Theoretical Computer Science*, vol. 11, pp. 97-108, 1980.
- [PSW] Pomerance, C., J. L. Selfridge and S. Wagstaff, Jr., "The pseudoprimes to $25 \cdot 10^9$ ", *Mathematics of Computation*, vol. 35, no. 151, pp. 1003-1026, July 1980.
- [P] Pratt, V., "Every prime has a succinct certificate", *SIAM Journal on Computing*, pp. 214-220, 1975.
- [R1] Rabin, M. O., "Probabilistic algorithms", in *Algorithms and Their Complexity: Recent Results and New Directions*, J. F. Traub (editor), Academic Press, New York, New York, pp. 21-39, 1976.
- [R2] Rabin, M. O., "Digitalized signatures and public-key functions as intractable as factorization", *MIT/LCS/TR-212*, 1979.
- [RSA] Rivest, R. L., A. Shamir and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *Communications of the ACM*, vol. 21, no. 2, pp. 120-126, 1978.
- [SS] Solovay, R. and V. Strassen, "A fast Monte Carlo test for primality", *SIAM Journal on Computing*, vol. 6, pp. 84-85, 1977.

PUBLIC KEY REGISTRATION

Stephen M. Matyas

Cryptography Competency Center
International Business Machines Corporation
9500 Godwin Drive
Manassas, Virginia 22110

ABSTRACT

A procedure is described for securely initializing cryptographic variables in a large number of network terminals. Each terminal has a cryptographic facility which performs all necessary cryptographic functions. A key distribution center is established, and a public and secret key pair is generated for the key distribution center. Each terminal in the network is provided with a terminal identification known to the key distribution center. The terminal identification and the public key of the key distribution center are stored in the cryptographic facility of each terminal. A terminal initializer is designated for each terminal, and the terminal initializer is notified of two expiration times for the purpose of registering the terminal's cryptovariable with the key distribution center. The cryptovariable is generated by the terminal using its cryptographic facility. Prior to the first expiration time, a registration request is prepared and transmitted to the key distribution center. The registration request includes the terminal identification and the cryptovariable. When the key distribution center receives this request, the cryptovariable is temporarily registered and that fact is acknowledged to the requesting terminal. After the expiration of the second time, the registration is complete. Provisions are also made for invalidating a terminal identification if more than one registration is attempted for a given terminal identification or an intended registration was not made in time.

BACKGROUND

Historically, contributions in the area of public key distribution can be briefly traced, although a thorough treatment of the subject would naturally include the works of many others. The first known proposal for public key distribution involved placing the public key of each user in a public directory (i.e., key distribution center) along with the user's name and address [1]. Anyone wishing to communicate with a particular user would first contact the public

directory and request a copy of that user's public key. A second proposal simply called for a pair of devices wishing to communicate to exchange their public keys via the communication channel in advance of the communication session [2].

An improvement to the first method was made by distributing public keys from a key distribution center (KDC) wherein the messages containing the keying information were "signed" using the secret key of a public/secret key pair belonging to the KDC [3]. In advance, the public key of the KDC is distributed to each communicant in the system and this key is then used to validate the received signature and message containing the keying information. A similar approach was suggested wherein each communicant registers his public key and identifier with the KDC and, in turn, receives from the KDC a public key certificate, which is a message containing his public key and identifier that has been signed using the secret key of the KDC [4]. In advance, the public key of the KDC is distributed to each communicant in the system, and this key is then used to validate received certificates. To communicate, individuals need only exchange and validate each other's public key certificate. Both approaches provide a path with integrity to distribute public keys previously registered with the KDC.

Racal-Milgo proposed a method of dynamic public key distribution which incorporates an anti-spoofing procedure [5]. Briefly, two parties who wish to communicate with a public key algorithm each generate a public and secret key pair. The respective public keys are exchanged via the communication channel. Upon receipt, each communicant calculates a prescribed function of the received public key. The communicants then contact each other via telephone and exchange the calculated values, which can then be verified by the originating communicants. For the check to work, the telephone channel itself must have integrity or the callers must recognize each other's voice.

A similar anti-spoofing technique which pre-dates the Racal-Milgo technique was proposed by Bell Telephone Laboratories [6]. With Bell's technique, the key validation information is exchanged via letter using the postal system rather than by using voice communications. Otherwise, the concept is the same. For the check to work, the postal system handling the mail must have integrity, otherwise the anti-spoofing check itself could be spoofed.

From the foregoing, it is apparent that while advances in public key distribution techniques have been made, the problem of initially securely registering public keys with a key distribution center has not been appropriately dealt with.

PUBLIC KEY REGISTRATION PROCEDURE

It is the object of the public key registration procedure to initialize with security and integrity a large number of devices in an information handling system with cryptographic variables without requiring couriers to transport these cryptographic variables. For convenience, the devices in the network will be terminals. The procedure is general in that it permits the registration of both secret and nonsecret variables, although of primary interest is the registration of public keys of terminals. The public key registration process may also be thought of as part of the larger process of terminal initialization.

Initialization of the terminal is performed by a designated representative called the terminal initializer. In all cases, the terminal initializer is a person who acts responsibly to carry out the steps of the terminal initialization procedure. The terminal initialization procedure comprises the steps of causing the terminal to generate and register one or more cryptovariables with a designated key distribution center (KDC) and promptly reporting to the KDC any encountered problems. Typically, the terminal initializer will be an employee of the organization at the location where the terminal is physically installed, such as a terminal user, terminal owner, manager, or member of the local site security. In situations where a third party key distribution center is employed, the terminal initializer may be a locally appointed agent of the KDC. The terminal initializer has no responsibility for transporting keys, public or private, or for installing secret keys by entering them directly into a cryptographic device. Therefore, the terminal initializer is not a courier, and does not perform the functions of a courier.

Each terminal in the network is provided with a cryptographic facility (CF) consisting of hardware and software components that perform the necessary cryptographic functions to support the required cryptographic operations. A subset of these functions support the terminal initialization procedure. Overall cryptographic security,

including that of the terminal initialization procedure, rests on an assumption of integrity of the CF, including stored keys and programs and associated supporting software, which is guaranteed by the design and by other physical security measures instituted. Prior to the terminal initialization procedure, the KDC generates a public key and secret key pair (PKkdc, SKkdc), which are the keys that operate with the public key algorithm. A unique nonsecret terminal identifier (TID) and the public key of the key distribution center (PKkdc) are assumed to have been installed in the CF of the terminal. The TID and PKkdc could be installed, for example, in microcode as part of the manufacturing process of the terminal. Alternatively, they could be installed at a central location and the terminals with the installed TID and PKkdc shipped to the final destination, or they could be installed by the terminal initializer, i.e., locally after the terminal has been installed.

For each terminal which is to be initialized, as previously mentioned, the KDC designates a terminal initializer who is responsible for carrying out the necessary terminal initialization procedure at the device. Each terminal initializer is provided with a set of instructions outlining the terminal initialization procedure. The security of the procedure rests on the assumption that the terminal initializer will comply with the issued instructions and understands that failure to comply with these instructions may result in an adversary successfully registering a key with the key distribution center. The KDC also provides to the terminal initializer with two expiration dates, ordinarily separated by several days, which delimit periods of time in which certain prescribed steps within the terminal initialization procedure must be completed. The security of the procedure rests on the assumption that the terminal initializer receives notification of the two expiration dates and the terminal initialization instructions at some time well in advance of the expiration dates so that the steps of the procedure can be performed within the prescribed allotted time.

According to the terminal initialization procedure, prior to the first expiration date, a cryptovariable can be temporarily registered at the KDC under the designated TID provided that the TID has not been invalidated and no other prior cryptovariable has been temporarily registered for the TID. In the discussion that follows, the cryptovariable registered is a public key, and therefore this process

is called "public key registration." If a public key has already been registered under a given TID, an attempt to register a different public key under that same TID prior to the first expiration date will result in the TID being invalidated. After the first expiration date, the public key registration process is disabled at the KDC for that TID.

Prior to the second expiration date, the KDC permits a TID to be invalidated without "proof" of the identity of the requestor. This process is called "ID invalidation without proof of identity." After the second expiration date, the process of ID invalidation without proof of identity is disabled for that TID, and the temporary status of the registration is considered changed to that of a permanent registration.

After the second expiration date, the KDC permits a TID to be invalidated only after the requestor has been identified and authenticated and his or her authorization to invalidate a particular TID has been verified. This process is called "ID invalidation with proof of identity."

After the second expiration date and upon request, the KDC will issue a PK certificate for any TID provided that the TID is valid and a public key has been registered for that TID. A PK certificate consists of a TID, public key, certificate expiration date, possibly other data, and digital signature produced on the foregoing data using the secret key of the KDC. One recommended method for calculating a signature is to first calculate an intermediate value or function of the message using a strong one-way cryptographic function. This intermediate value is then decrypted with the secret key SKkdc to produce the signature. If the TID is invalid or no public key has been registered, an appropriate response message is prepared on which a digital signature is calculated using the secret key of the KDC and the message and signature are returned to the requesting terminal.

Under normal operating conditions, the terminal initialization procedure proceeds as follows. Well in advance of the first expiration date, a public key and secret key pair are generated at the terminal using an available key generation procedure. A public key registration request message containing the TID and public key of the terminal is sent to the KDC. Under normal conditions no adversary

will have interfered with the process, and therefore no public key will yet be temporarily registered under the designated TID. Therefore, the KDC temporarily registers the public key under the specified TID, prepares an appropriate response message containing the TID and public key on which a digital signature is calculated using the secret key of the KDC in the manner previously described, and the message and signature are returned to the requesting terminal. After authenticating the received message, the requesting device signals the terminal initializer that the desired public key has been temporarily registered at the KDC under the specified TID. (It is assumed that the hardware and software components involved with terminal initialization have integrity and that the terminal being initialized is the prescribed, genuine terminal.) The procedure for authenticating a signature is similar to the procedure for calculating a signature. The same intermediate one way function of the message, which was used in calculating the signature, is again calculated from the message. The received signature is then encrypted using the public key of the KDC (PKkdc) to recover a clear value of the one way function of the message, and the recovered one way function of the message is compared for equality with the calculated one way function of the message. If the comparison is favorable, the message and signature are accepted; otherwise, if the comparison is unfavorable, the message and signature are rejected.

The protocol now requires a delay, and the terminal initializer must wait for the passage of the second expiration time in order that the KDC may assure that the temporarily registered public key is genuine; i.e., that it originated from the authorized, appointed terminal initializer. After the second expiration time, a terminal-initializer-initiated message containing the TID is sent to the KDC requesting "ID Verification" for that TID. Under normal conditions no adversary will have interfered with the process and therefore the specified TID will be valid and the previously temporarily registered public key will still be registered. But due to the expiration of the second time, the registration is now considered permanent. Therefore, the KDC prepares and returns a message to the requesting terminal specifying the registered public key for that TID. A digital signature is prepared on this message using SKkdc which allows the requesting terminal to authenticate the received message using the installed PKkdc in the manner previously described. This signals satisfactory completion of the terminal

initialization procedure and provides the necessary proof that the desired public key has been successfully initialized at the KDC. Alternatively, the KDC could return a public key certificate to the requesting terminal, and this would also serve as proof to the terminal that the public key had been registered.

Once an authenticated response has been received from the KDC stating that a public key has been temporarily registered or that the TID has been invalidated, the worst that could happen is that an adversary could cause a genuine temporarily registered public key to be erased by invalidating the TID prior to the second expiration time. Hence, for practical purposes, a safe state is reached, and it is therefore possible with no loss in security to allow a protocol variation wherein the terminal-initializer-initiated message sent to the KDC requesting "ID Verification" following the second expiration time can be replaced by a similar terminal-user-initiated message. This protocol variation has the advantage that ordinarily the terminal initializer can complete the terminal initialization procedure with only one terminal visit, prior to T1. The terminal user, who is notified by the terminal initializer of the terminal initialization status and the value of T2, completes the protocol after the second expiration time. Of course, the protocol variation is the same as the original protocol when the terminal initializer and the terminal user are the same person.

In a network where it is convenient for the KDC to send messages to the terminals, such as in a store-and-forward electronic mail distribution system, yet another variation on the protocol is possible. The step following the second expiration time wherein a terminal-initializer-initiated or terminal-user-initiated message is sent to the KDC requesting "ID Verification" is replaced by a step wherein the KDC automatically prepares and sends a response to the original requesting terminal. This response is just the same as that which would have been sent in the response to a request for "ID Verification" except here the response is triggered by reaching the second expiration time rather than upon receiving a request message. Otherwise, the protocol is the same. If no response is received at the terminal within a reasonable period of time after the second expiration time, the terminal initializer or the terminal user, depending on which protocol is used, reports this discrepancy to the KDC.

An expanded discussion of the above initialization procedure can be found in IBM TR 21.1000 [7].

REFERENCES

1. Diffie, W. and Hellman, M. E., "New Directions in Cryptography," IEEE Transactions on Information Theory, Vol. IT-22, No. 6, 644-654 (1976).
2. Rivest, R.L., Shamir, A., and Adleman, L., "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," Communication of the ACM, Vol. 21, No. 2, 120-126 (1978).
3. Needham, R.M. and Schroeder, M.D., "Using encryption for authentication in large networks of computers," Communication of the ACM, Vol. 21, 993-999 (December, 1978).
4. Kohnfelder, L. M., "Using Certificates for Key Distribution in a Public-Key Cryptosystem," Massachusetts Institute of Technology, unpublished paper (May 19, 1978).
5. Abbruscato, C.R., "Public Key Security," Telecommunications, Vol. 18, No. 12, 60-64, 72 (December, 1984).
6. Myers, F.H., "Data Link Encryption System," NTC Conference Record, National Telecommunications Conference, Washington, D.C., New York, NY, 1979, Vol. 3, pp. 43.5.1-43.5.8
7. Matyas, S.M., Initialization of Cryptographic Variables in a Network with a Large Number of Terminals, IBM Technical Report, TR 21.1000, Kingston, N.Y., August, 1986.

Is there an ultimate use of cryptography?

(Extended Abstract)

Yvo Desmedt

Katholieke Universiteit Leuven, ESAT, Belgium†

current address:

Université de Montréal, Dépt. IRO, Case postale 6128, succursale A,
Montréal (Québec), H3C 3J7 Canada

ABSTRACT

Cryptography can increase the security of computers and modern telecommunication systems. Software viruses and hardware trapdoors are aspects of computer security. Based on a combination of these two aspects, an attack on computer security is presented. The complexity of finding such an attack is discussed. A new open problem is: can cryptography prevent such an attack.

1. Introduction

The problem of authenticity plays a very important role in the security of modern telecommunication and computer systems. Cryptography can be used to control access to chips used in these systems and to verify the authenticity of the commands and/or messages and sender. Today such chips are being introduced, e.g. Keyproms. Cryptography can evidently also protect the privacy of the information going around in these systems. In other words, the use of cryptography on each (VLSI) chip in the system increases the computer security, e.g., by protecting against eavesdropping and/or active eavesdropping on the bus of the computer.

In this paper we show that the above solution is not enough to obtain secure computer systems. Let us first introduce the notion of software virus and briefly discuss the hardware aspects of computer security.

A software virus is a subprogram that contains some undesired commands (e.g. time-bomb) and has the capability to copy itself into executables. As a consequence one can completely control a computer (or several computers) using such a software virus. For example one can give somebody a computer game containing such a software virus. The virus copies itself into all executables of the owner of the computer game, while he is playing. If the owner shares some of its executables with other users, the virus affects all executables of the others, and so finally affects the operating system.

Let us briefly discuss some of the hardware aspects of computer security. PCB (printed circuit boards) are sometimes changed during maintenance and it is not excluded that the new PCB is intentionally slightly different from the previous. For example if a PCB controls the access to files, the new PCB may allow a trapdoor access.

Before explaining our attack, we introduce (in Section 2) hardware trapdoors and similar hardware frauds (in chips), which we will call hardware viruses. The name of this term will be clear at the end of this paper. Our attack is presented in Section 3. The attack and hardware viruses would

† This research started while the author was aangesteld navorser NFWO at the Katholieke Universiteit Leuven

not be important if they would be easy to detect. In Section 4 we prove that the computational complexity of proving that no hardware virus exists in a chip is at least NP-hard. In Section 5 the dangers of such hardware viruses are briefly discussed, and more advanced attacks are overviewed in Section 6. Section 7 is related to a new open problem in cryptography.

2. A hardware virus

For a chip it is possible that non-specified commands exist. These non-specified commands may be (or were) used for test purposes of the chip, or will be used in new versions of the chip. However it is also possible that these non-specified commands allow the chip to act completely differently than specified, e.g., block (halt) itself, or destroy itself, or affect all following commands and/or inputs/outputs and performance of the chip. It is possible that these non-specified commands are only known by the VLSI designer, who put them in for fun, or other reasons. Let us now briefly discuss some variations to implement these fraudulent non-specified commands. The non-specified commands can be given using the pins used for commands, or can be introduced through the input. In the last case a 64 bit pattern is given at the input (e.g. using other non-command pins). This 64 bit pattern is recognized by the chip as being the start sequence of non-specified commands. Evidently in a VLSI chip which also performs encryption, this pattern can be signed, and/or hardware trapdoors can be used to bypass the encryption part. If a (VLSI) chip contains such fraudulent non-specified commands, we will call it affected by a *hardware virus*. We remark that the effect of the hardware virus can be much worse than that of a simple trapdoor access to the chip. Another variant is a hardware virus which is activated at random by the chip itself, for example if during the calculations in the chip some pattern of 46 bits appears, the chip starts to do crazy things. The length of this pattern can be chosen by taking into consideration: the speed of the chip and the "desired" frequency (or probability) of the fraudulent effect of the virus. In this case no external command is in fact necessary. The chip performs however differently than specified. We will also say in this case that the chip is affected by a *hardware virus*. Instead of the virus being activated at a random moment, the time can be used to trigger the effect of the virus.

It is evident that such a virus will be easy to detect if the complexity of the VLSI chip is not high (e.g. a lot of repetition). In other words in such cases the designer will probably not introduce a hardware virus. Remark that the harder it is to reverse engineer a chip, the more difficult it becomes to find the hardware virus. The complexity of finding a hardware virus is discussed in Section 4.

A solution against hardware viruses is to make only chips which are straightforward to reverse engineer. However this allows other companies to copy the bright ideas (e.g. new faster algorithms) and use them for their own purposes in different applications. So this solution has important drawbacks. Another approach to limit the probability that the designer introduces a hardware virus is a clearance procedure to check the designer. This solution never excludes the possibility of fraud. A double check method can be used by having two designers instead of one.

In the next section we explain that the last two methods are not sufficient.

3. Software viruses creating hardware viruses

In previous section it was the designer who intentionally introduced the hardware virus in the chip. In this section we discuss how hardware viruses can be introduced by a CAD (Computer Aided Design) program, without the knowledge of the VLSI designer and/or without the knowledge of the CAD designer.

CAD is more and more used to design complex VLSI chips. CAD being (mainly) software suffers from all its inconveniences. In particular a software virus can exist in a CAD system. An awful case of a software virus living in a CAD program is when the virus contains information to create a hardware virus. Let us briefly explain how this works.

Chips contain easily detectable patterns as transistors. These patterns (and others) are used by the software virus to first check if it is in a CAD program. Once confirmed a flag can be set to avoid rechecking and losing time. Then during the execution of the CAD to make the VLSI chip, the software virus checks the complexity of the chip. So it checks if the same block is used over and over again. If this is not the case and if the software virus concludes that the complexity of the chip is high, it introduces a hardware virus.

The hardware viruses introduced by a software virus through a CAD can be very trivial ones. An example of the effect of the hardware virus is to create some electrical shortcut after a fixed time of use of the chip. More complicated hardware viruses can be introduced, using more advanced software viruses. These software viruses can use artificial intelligence and/or software libraries. This last idea is explained in more detail in Section 6.

We remember that a software virus can be introduced through a game. It is possible that the software virus was injected in the CAD on the computer of the designers of the CAD, or at the computer of the users of it. As a consequence it is *very difficult to find it in the CAD, certainly if the length of CAD programs is taken into consideration*. Antibody programs (detecting and removing software viruses) can be ineffective, if the virus is introduced immediately after the antibody program checked the CAD program.

4. Complexity of finding hardware viruses

A hardware virus is only dangerous if it is very hard to detect it. Let us focus on the problem of proving that no hardware virus exists in a chip. In order to prove that this problem is NP-hard, we restrict ourselves to the chips which are nothing else than a Boolean function. Proving that no hardware virus exists in this case means proving that for all possible inputs, the Boolean function f corresponding with the specs is equal to the Boolean function g corresponding with the chip. If for *all* inputs $f = g$, then there is no hardware virus. This problem is the complement of the satisfiability problem, which is NP-complete. This is trivial to understand by considering the function h , which is defined as the exclusive or of f and g . So the problem is CO-NP-complete. If one drops the restriction on the chips and considers more general cases than Boolean functions (e.g. with feedback) we can say that the problem of proving that no hardware virus is in the chip is at least NP-hard. In order to make the proof correct from a mathematical point of view, we allow chips with enormous number of pins, transistors and so on.

We remark that the above reasoning is very similar to a well known problem in testing of VLSI chips.

5. Dangers of hardware viruses

In order to estimate the dangers and the impact of software viruses making hardware viruses, it is very important to remember that chips are not only used in computers. Chips are used today in: telecommunication systems, instruments, controllers (e.g. controlling the temperature of a process in a chemical plant), consumer electronics, security systems (e.g. detecting burglary), medical electronics, industrial electronics (as in robots), cameras, cars, trains, aviation, military applications, space and so on. It is not difficult to imagine the consequences of hardware viruses in these applications, certainly

if the hardware virus is more complex (see Section 6).

It is evident that such hardware viruses (certainly when introduced without the knowledge of the chip designer, see Section 3) affect not only the user, the consumer, the industry using the chips, but also the whole economy based on chip technology and national security.

It is even worse if one takes into consideration that the modern methods of fault tolerant computing are not adequate to protect against such an attack. Indeed the hardware virus in one chip can trigger other ones, and/or the event can be planned from the moment of the design of the virus.

If license fees have to be paid for chips made by the CAD program, the above attack can be used in a more positive way.

6. More advanced and variant CAD attacks

If artificial intelligence continues to develop, it is not excluded that in the future more advanced attacks will become possible. Indeed the software virus in the CAD could also test what kind of chip is under design. Once it figures out what the purpose of the chip is, it selects an adequate hardware virus from a secret software library. If it is for example a disk controller, the hardware virus can be designed to destroy intentionally information on the disk. In the case of a microprocessor (or other part of a computer) the hardware virus contains information to start on the computer (which is under development) a software virus. The opinion of the author related to the last example is that it will only be realistic if VLSI chips contain many more transistors.

Gate arrays are very frequently used in industry. The design is done by the client (some company). The translation from gates to transistors and optimization of these gates is done by another company. After delivery it is even difficult for the designer to reverse engineer the chip. Here a hardware virus can be introduced at several levels: gate level and transistor level. In general it is possible that the introduction of the virus is done just before processing the chip. There are enough steps in the process of making a chip (and certainly a gate array), where a hardware virus can be introduced by a CAD or other program.

7. Open problem

A few years ago several chips were used to implement a cryptographic algorithm. Recently one chip became sufficient. Now cryptographic algorithms can be a part of a VLSI chip. During this process of miniaturization, cryptography is beginning to be used to check access to chips (see Section 1). In other words enlarging its application domain from protecting the system (e.g. against eavesdropping) to protecting parts of the system (access to chips). An open problem is if cryptography can be used on a sub-chip level in order to make virus-free chips, or prove that the chips are virus-free, without affecting the privacy of the chip design, in other words without making chips easily reverse engineerable. From this point of view we can wonder *if there is an ultimate use of cryptography*.

The author thinks that it will be very difficult for modern cryptography to protect against the described attack, without imposing severe restrictions on the chip design methods. This personal idea finds its grounds in the fact that proving that a chip is virus-free is NP-hard (see Section 4).

8. Conclusions

The problem of hardware viruses introduced by a software virus coming from a computer game through a CAD program affects several aspects of computer security and security in general. It is also an interesting open problem if cryptography can move into the sub-chip level, or if its use is limited to the system and chip level.

Nevertheless the author did not really used such an attack against a CAD program, several arguments are given related to the feasibility of such an attack. It would be worth to construct such a software virus in order to obtain more information about practical problems and related aspects.

Acknowledgement

The author wants to thank Gilles Brassard (Univ. de Montréal, Canada) for the discussions about this paper and more specially Section 4, Frank Hoornaert (IMEC and ESAT, Belgium) and William Rowan (Univ. Cal Berkeley, USA) for suggesting Section 6 and Jean-Jacques Quisquater (Philips Research Brussels, Belgium) for his discussions in particular related to Section 1.

S M A R T C A R D

A HIGHLY RELIABLE AND PORTABLE SECURITY DEVICE

LOUIS C. GUILLOU CCETT/ASP Rue du Clos Courtel BP 59
35530 CESSION SEVIGNE - FRANCE

MICHEL UGON BULL CP8 Rue Eugène Henaff BP 45
78193 TRAPPES CEDEX - FRANCE

ABSTRACTS :

At first glance, the smart card looks like an improvement of the traditional credit card.

But the smart card is a multi-purpose and tamper-free security device. And behind a standardized interface, the built-in electronics may evolve, in memory size and in processing power. This evolution, while resulting from economic considerations, is in tune with an enhancement of both physical and logical security.

Some mechanisms in key-carrier cards are described, thus giving a taste of the state of the art in card operating systems. The underlying reality is an invasion of our lifetime by cryptology and computers. This invasion will have a large influence on security in various fields of applications, not only banking operations, but also data processing, information systems, and communication networks.

INTRODUCTION :

Including magnetic stripes and embossed areas, a banking IC card looks like a gadget plastic credit card. But, embedded in the thickness of its plastics, there are one or more integrated circuits designed to perform both processing and memory functions.

Magnetic stripes and embossed areas, traditional technologies of credit cards, are deprived of any processing power, with only memory functions ; the relevant standards specify every details without any degree of freedom for further evolutions.

An IC card is not defined by IC number, position, or performance. On actual cards, the interchanges with the outside are conducted through electrical contacts ensuring a galvanic continuity between built-in electronics and an external interface device. Other types of IC cards may be developed in order to avoid contacts. But an IC card is and will be defined through a standardized interface taking into account its processing power .

Trade-offs between costs and performances are evidently related to a current state of the art and to current needs of the applications. Technological evolutions deal with power consumption and integration scales, resulting in a tremendous increasement of both memory size and processing power. Behind a standardized interface, built-in electronics may evolve while terminals remain unchanged.

Reductions in both engraving size and power consumption will complicate physical investigation of processors dedicated to smart cards, the famous Self-Programming One-chip Microprocessors, abbreviated as SPOMs : existing SPOMs are today tamper-free, and we don't know any successful violation of their transaction memory.

Additions in processing power (CPU and RAM) and in operating systems (ROM) will complexify the logical security, and allow the use of more complex and more various cryptographic algorithms. While unpublished and proprietary, the algorithm named Telepass2, and used in French bank cards, has been successfully evaluated by a notoriously specialized agency. Current SPOMs are already able to implement very secure algorithms.

In semi-conductor industry, technological trends are in tune with security. This exciting situation is very new! And at ISO level, the general approach of this interface takes into account these potential evolutions.

CONTACT LOCATION AND ASSIGNMENT :

In existing ISO standards, credit card surfaces had been partially reserved. It seems impossible to locate contacts on magnetic stripes or in embossed zones. Marketing considerations have been expressed in the US so as to share the front part of a credit card between the issuing bank and the credit card company. Moreover, Japan has chosen a national magnetic stripe position on card front side, in disagreement with ISO standards. The sum of these constraints explains the difficulties met by ISO in its international quest for an agreement on contact location.

ISO has reached a first basic agreement on contacts : number, minimum dimensions, relative position and assignment. On existing smart cards, the outside accesses the built-in electronics through six electrical contacts. With respect to GND (ground) as a reference voltage, the outside must provide VCC (supply voltage), VPP (programming voltage), CLK (clocking signal), and RST (reset signal) with suitable signals in order to exchange information on I/O (input/output). Two spare contacts, named RFU, are reserved for future use (see figure 1). This agreement protects the existing dedicated chips and the existing ways to package electronics in the cards.

Nevertheless, two 8-contact positions are yet described in the DIS (Draft International Standard, now under ballot). One beneath the other, the set of sixteen contacts forms a regular pattern located relatively to a corner, in such a way that mixt contactors are easy to design and to produce. In France, mixt contactors are being inserted in public telephone booths as well as in interface devices to be connected to Minitels. Extensively used in France, the upper position is mechanically more reliable : a card is more resistant to

bending with a chip closer to a corner. But the lower position is more in tune with Japanese constraints, and allows to locate the contacts on the back side. While being the traditional one, the upper position is said to be "transitional" in the ISO documents.

The standardization aims at an international agreement on a complete and unambiguous specification of this interface, not only : physical characteristics, contact location and assignment, but also : electrical signals, answer to reset, exchange protocols, (now also at a DIS stage) and interindustry requirements (now under study). ISO assigned in 1981 these general tasks to subcommittee SC17 "Identification Cards" in technical committee TC 97 "Information Systems".

In 1985, ISO entrusted TC68 "Banking" with two new specific work items on security and data contents related to banking operations. The adoption of these new work items was feeled so important that it produced a rearrangement of TC68 with creation of a new subcommittee SC6 dealing with "Financial transaction cards, related media and operations".

SOME TECHNOLOGICAL ASPECTS :

As a result of a transaction, the card delivers information (stored data, computation results), and/or modifies its content (data storage, event memorization). The built-in electronics always include an electrically **Programmable Read Only Memory** (PROM). Each PROM cell originally in state "1" may be turned to state "0" by an electrical process under control of the built-in electronics. This PROM contains the **transaction memory**, the content of which evolves during card life.

Two major technologies are currently producing PROM components : **bipolar** and **MOS** (metal oxyde semiconductor).

Though quicker, bipolar logic is more power consuming and less easy to integrate than MOS logic. But more important : the bipolar writing process destroys a part of the memory cell, in such a way that bipolar PROMs are optically readable (see figure 2)!!

Nethertheless, at the outbreak of IC cards, bipolar technology has been fairly considered : the reason was the irreversibility of the writing process.

No physical method is known to investigate MOS PROMs so as to directly vizualize a cell content without using the internal buses : the MOS writing process is reversible ! And the existing MOS PROMs' can be erased either by ionizing radiations, such as UV-light or X-rays, or by another electrical process. All existing cards are now using MOS components which are cheaper, more secure, and allow to match microprocessor technology with a PROM memory.

The technological controversy lies now in the comparison between UV-erasable PROM, named EPROM, used as write only memory, and electrically erasable PROM, named EEPROM, and so rewritable. When widely available, EEPROMs are feeled to be more flexible than EPROMs, but with much less capacity for a given die size. Actually, EEPROM technology is less mature and more expensive.

Other parallel technological evolutions are in progress :

- scales of integration move from a range from 4 to 3μ to a range from 2 to 1μ ;
- uprising of CMOS and HC MOS considerably reduces power consumption in the cards.

These reductions in engraving scale and in power consumption will increase considerably the difficulties in investigating the PROM contents. And to close the general considerations, let us remember a generally agreed limitation : to obtain a reliable card, the size of the chip must not exceed 20 mm².

A FAMILY OF IC CARDS :

Under control of the built-in electronics, the content of the transaction memory evolves during card life. Depending on the increasing complexity of this electronics, three types of IC cards are currently in use (see figure 3) :

- **Memory cards** containing only a transaction memory with very simple writing protections.
- **Logical cards** containing a transaction memory and a logic array of gates : this logic array of gates tests a confidential code before giving access.
- **Smart cards** containing a programmed microprocessor controlling itself all the accesses to the transaction memory.

In France, these three types of IC cards are illustrated by :

- Memory cards : 40-unit or 120-unit prepayed "Télécartes" anonymously used in public telephone booths ;
- Logic cards : "Télécommunications" identifying their bearer in order to charge phone calls on a number ;
- Smart cards : bank cards and key-carrier cards which may also be used in public telephone booths !

It is inefficient to search for relations between applications and types of cards : the three types are in use in the telephone system, while bank cards are designed such as to easily extend their use in various services.

- The simpler cards are specific to only one purpose, and it is rather difficult to share a chip production between several applications.
- On the other hand, the smart card is essentially a multi-purpose device. And the chips are programmed by mask during the manufacturing process. There is no difficulty to share a chip production. The development of a new mask is easy, and the same line produces chips, whatever the mask be !

This paper is devoted to smart cards including a specific integrated circuit, named SPOM, a microcomputer which merges on the same chip a PROM memory and a microprocessor controlling itself all the accesses to the PROM memory.

A NEW CHIP : THE SPOM

Including two chips (Fairchild 3870 microprocessor + Intel 2716 EPROM memory), the first smart cards were produced in 1979 after a strong international cooperation between MOTOROLA Inc. and CII HONEYWELL BULL.

This stage of a two-chip card is essential in order to prove the feasibility and to convince potential users to start experiments. These devices played also a prominent part in the development of the various other elements of the systems using smart cards in order to initiate applications.

Economical considerations led to merge PROM and microprocessor on the same chip. And the cooperation between BULL and MOTOROLA continued by the studies of a new microprocessor dedicated for smart cards. Such a microprocessor must be able to execute an internal routine which writes in its transaction memory. A new architecture has been invented to manage registers on the internal buses in such a way that the processor may continue its control while holding the right address and the right content on the buses towards the PROM. Such microprocessors are named : Self Programming One-Chip Microprocessors, abbreviated as "SPOMs" (see figure 4).

Since 1981, SPOM01 is produced by MOTOROLA Inc. in East Kilbride (Scotland, United Kingdom) ; since 1985, SPOM02 is produced by THOMSON EUROTECHNIQUE in le Rousset (Provence, France). Trade-offs between costs and performances are largely indebted to the know-how gathered from the first two-chip cards. Both in nMOS technology, they are about 17 mm² in size. BULL CP8 and PHILIPS are currently manufacturing cards with such SPOMs. Very recently, a prototype (30 mm²) comes from HITACHI, referred as 65901, while SPOM03 and 04 were announced by MOTOROLA and THOMSON.

MEMORY SIZES (in bites)

	C.P.U.	RAM	ROM	EPROM
SPOM 01 MOTOROLA	6805	36	1600	1024
SPOM 02 EUROTECHNIQUE	8048	44	2048	1024
65901	RISC	128	3072	2048 (EEPROM)
SPOM 03	6805	52	2048	2048
SPOM 04	8048	64	3072	4096

In addition to PROM memory, SPOMs contain also ROM written by mask during chip manufacturing process and RAM to store temporary results. Let us notice on SPOM02 that a cell of RAM is roughly ten times larger than one of EPROM, and a cell of EPROM twice larger than one of ROM (see figure 5).

Before cutting the wafers on SPOM02 production lines, a 512-byte internal routine is activated through a seventh test-contact. Each validated component receives various information : locks, codes, and a chip serial number, while nothing is written in rejected components. The test-contact is then systematically destroyed, thus definitively disabling invalid components. As a matter of fact, only the self-testing routine may write the witness indicators tested by the card before execution of any command during a transaction with the card.

ADDITIONAL SECURITY FEATURES :

Absolute physical security does not exist, no more for smart cards than for any other computing device. System designers must consider potential consequences of violations. Secret keys in a system must be as diversified as possible, and in a user card, tied to the chip serial number. A violation then results in an attack against only one user and does not endanger the whole system, thus reducing the potential benefits from fraud. These aspects of logical security are strongly related to cryptology.

In key-carrier cards as well as in bank cards using existing SPOMs, the transaction memory is organized in 32-bit words. During card issue, an issuing block is written in a dedicated 6-word zone. This block stores the secret distribution key on four 32-bit words. Each card issuer owns his secret cryptographic function. He uses it to compute or to recompute a unique secret distribution key from each chip serial number. This key, unique for each card, must be correctly used hereafter to control various operations, such as writing new secret words and delivering new authorizations. Very different from a confidential code, this cryptographic key is used as a parameter in a cryptographic computation prescribed in internal routines and executed by the card itself.

The card issuer may remotely and securely authenticate a card : a random value is sent to each calling card which must answer both its chip serial number and the result of a computation using the distribution key. From the chip serial number, the issuer reconstructs the distribution key and then the computation result, thus authenticating the card.

The card issuer may also identify the card user by including a user confidential code during the authentication process : the code given locally modifies the random value sent by the issuer. In a first solution, only the card knows the code : the card modifies the incoming value by an internal code, and the internal modification must cancel the external one. In a second solution, the issuer modifies the random value before sending it to the user : the external modification must then restore the initial value.

In addition to these functions, when the cryptographic computation in the card reverses an external cryptographic computation performed by the issuer, the card may identify its issuer before remotely executing its directives. At the end of a cryptographic computation, the card tests the result : for example, when the 64-bit result consists of two identical 32-bit fields, the card assumes that only its issuer might induce such a result. So the issuer is now the real master of the silicon, because each SPOM may securely and remotely identify its master.

KEYS, AUTHORIZATIONS, ENTITLEMENTS :

First developed in a television environment in order to access broadcast information such as picture, sound and data, the key-carrier cards are deprived of any banking functionality. The conditional access method may be general, while the scrambling method clearly depends upon the nature and the coding of the service components. The key-carrier cards are usable in access control to a large range of services and resources, including terminals, network gates, databases, computers, and even buildings.

Materializing entitlements, these cards are issued and managed by card issuers who are their real owners. The card issuer must be clearly distinguished from both the bearer and the service provider ; the bearer uses the card to recover control words, but cannot alter the card, nor get a copy of recorded keys ; the service provider checks entitlements by verifying their validity, by storing a debit, by consuming a credit ; the card issuer manages entitlement in his cards by delivering new entitlements, by clearing debits, by giving credits.

The key-carrier cards store blocks of authorization, each one consisting of three fields : an identificator, a status and a secret key. Each operation on an entitlement, as well checking as management, results from a transaction with the card. During this transaction, a command asks for a cryptographic computation with incoming data consisting of three fields : an identificator, a parameter, and a cryptogram.

During an entitlement checking transaction, the identificator in the incoming data must correspond to the identificator of an authorization in the card, the status of which must comply with indications given in the parameter ; then the card reconstructs the control word by a cryptographic computation using the secret key of the authorization. The outgoing data in the command asking for the result is a control word which is either sent back to the controller as a witness, or used locally to descramble subsequent service components.

During an entitlement management transaction, the card uses the distribution key, unique to each card, to execute a cryptographic computation. The card tests for a redundancy in the 64-bit result which must consist of two identical 32-bit fields. The card, having identified the voice of its master, executes the directive instructed by this field.

Depending on the way the card and the system manage the status of the authorizations, there are various entitlements : -fixed and renewable subscriptions, -prepayed special events, -pay-per-views either in a prepayed credit or with a limited debit, and -consumptions of tokens in activable blocks.

FOUR TYPES OF AUTHORIZATIONS :

The status of a basic authorization consists of an initial number, coding a starting date, and a gap, coding a duration. The parameter is an operation number, coding a current date. Before reconstructing a control word, the card verifies that the current date lies in the subscription period. The status of such an authorization is fixed and can only be verified. This first type of authorization is a fixed subscription.

The status of an authorization with controlled sessions consists of an initial number, a gap, a limit and the content of a zone reserved for sessions. Each session is defined by a shift from the initial number and a width. The sessions must lie in the main period and the sum of their widths must not exceed the limit. Before computing a control word, the card verifies that the operation number coded in the parameter lies in a session. Only a successful entitlement management transaction may modify the status of such an authorization by opening a new session (shift and width).

Depending on the way the system manages the numbers, this second type of authorization is used to access prepayed special events. It may also be used like a renewable susbcription : the renewal is obtained by opening a new session. For example, depending on the value

of the session width, when numbers are coding weeks, the subscription is either on a monthly basis with width = four, or on a quarterly basis with width = thirteen.

The status of an authorization with impulse sessions consists also of an initial number, a gap, a limit, and the content of a zone reserved for sessions. Before reconstructing a control word, the card verifies that the operation number coded in the parameter lies in a session. In the absence of suitable session, the card, after an explicit agreement of the user, opens directly a new session (a shift and a width) according to indications given by the parameter as long as the sum of the widths does not exceed the limit.

This third type of authorization is a pay-per-view working either on an additive basis (when the limit is an authorized debit), or on a subtractive basis (when the limit is a prepaid credit), depending on the payment.

The status of an authorization for consumption consists of an initial number, a gap, and the content of a reserved zone divided into blocks of tokens. Only a successful entitlement management transaction may open new blocks. Before reconstructing a control word during an entitlement checking transaction, the card consumes the amount of tokens indicated by the parameter. This fourth type of authorization is based on a consumption of tokens in provisions selectively activable.

THE EXISTING KEY-CARRIER CARDS : KCO AND KC1

The first version of key-carrier cards, KCO was developed as a basic tool in pay-TV systems. The MOTOROLA SPOM with such a mask is available since 1983. And these specifications were used to test the THOMSON SPOM in 1985. KCO, now available on both SPOMs, includes the first three types of authorization : subscription and both sessions.

An unpublished cryptographic algorithm, named Twisted Double Fields and described in about 200 bytes in the mask, reverses another expanding external algorithm used by the card issuer to compute entitlement management messages, and by the service provider to compute entitlement checking messages. In the card, a 61-bit result (for example, a control word) is computed from a 23-bit incoming parameter, a 127-bit incoming cryptogram, and a 127-bit internal secret key.

The second version of key-carrier cards, KC1, developed on the THOMSON SPOM, extends KCO functionalities, taking thus advantage of the additional ROM in SPOM02. KC1 introduces the fourth type of authorization with consumption of tokens. KC1 includes also a crypto-writing mechanism : the redundant result of a cryptographic computation using the distribution key indicates the address and the content of the 32-bit word to be written.

An unpublished cryptographic permutation, named Vidéopass, is described on about 200 bytes in the mask of KC1 which may a-priori execute the algorithm in both directions but locks restrict user cards to one direction. A 64-bit result is computed from an incoming 64-bit cryptogram, an either incoming (a parameter) or internal (a non-secret word) 32-bit argument, and a 96-bit internal secret key.

But, more important yet, these chips are specialised by locks written during card issue. Most of the cards are restricted to one direction of the cryptographic computation, while some cards keep both directions. To reverse a computation performed by a user card, the master card must have stored the same secret key written in the reverse order.

Thus, the cryptographic algorithm is dissymetrised :

- the user cards compute certificates on incoming redundant data, and the master cards can only verify their genuiness by recovering the redundancy in the result, without being able to forge certificates.
- the master cards securely produce cryptograms of control words and entitlement management messages : the user cards reconstruct the control word or executes the directives given by the master, without being able to forge management messages.

CONCLUSION :

This approach to smart cards, and more specifically to key-carrier cards, has deliberately excluded the banking purposes, in order to avoid the credit card syndrom. But in the existing banking smart cards, many key-carrier card concepts have been introduced : distribution key, authentication, identification, cryptowriting,... summarized by the user/master mechanism.

The cryptographic computation uses rather proprietary algorithms. But a new version of key-carrier cards, under development by PHILIPS, is implementing the DES. And future chips, with a more powerful CPU and at least four times the actual amount of RAM, will implement public key algorithms. The best way to protect the secret parameters of a public key signature scheme seems to store them in a user-friend and tamper-free device containing a SPOM, with high security features.

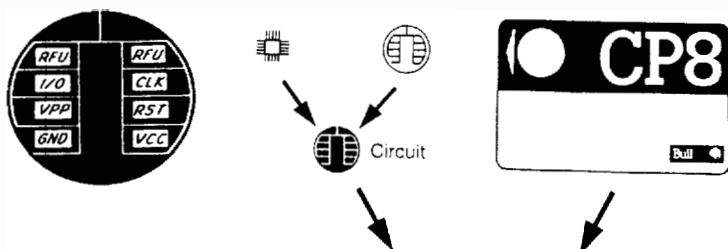


FIGURE 1



FIGURE 2a

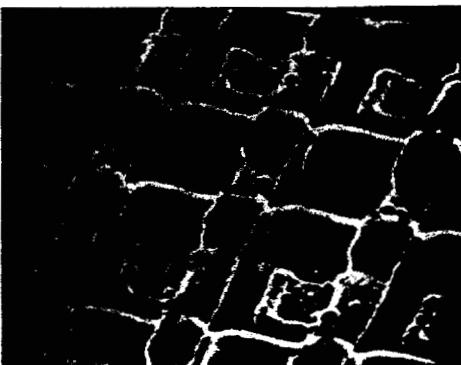


FIGURE 2b

Let us compare the destroyed fuses on the bipolar memory (fig 2a) with the regular pattern of the nMOS memory (fig 2b).

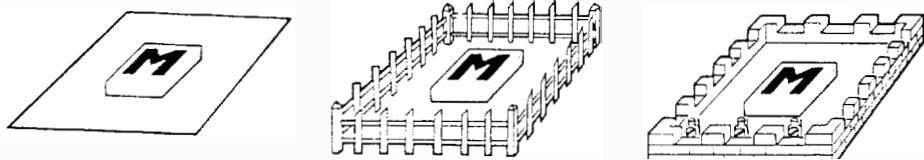


FIGURE 3

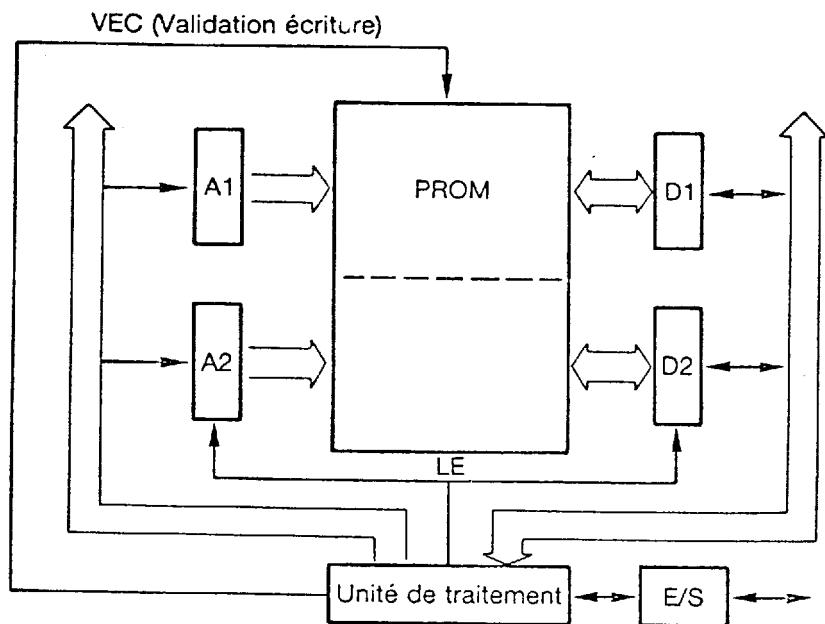


FIGURE 4

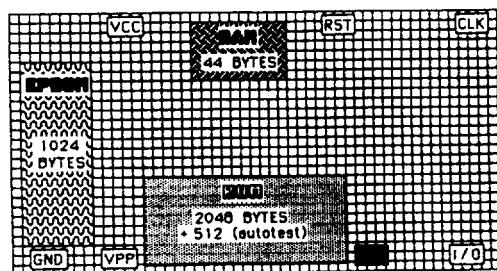


FIGURE 5

THOMAS - A COMPLETE SINGLE CHIP RSA DEVICE

by Dr. Gordon Rankine,
RAANND SYSTEMS Ltd.,
Livingston, EH54 9BJ,
Scotland, Great Britain

Synopsis - This paper examines a novel implementation of a 512-bit modulus exponentiator for applications in RSA key management environments.

The device, known by the internal project code THOMAS, is a complete single chip RSA implementation. No other device is necessary to compute the RSA components, other than the control elements associated with the crypto-system.

The approach chosen is examined to establish the benefits from the implementation in comparison with potentially faster but less flexible techniques.

1. BACKGROUND

In 1985, TALLIS Security, a division of British Telecom, approached RAANND SYSTEMS Ltd. to design, develop and manufacture, a high speed stream encryption device with an RSA key management procedure. Subsequently, the Government and Advanced Projects division of British Telecom adopted the idea and outlined an extension of the work to become a standard Telecom product for medium and high security line communications. This resulted in the product now known as LEKTOR. During the development program, the RSA implementation chosen was a hardware-assisted MC6809. It became apparent during the development that the implementation of the exponentiation could be the basis for a chip to implement a high-speed RSA device. Further development resulted in the design of a device which bore little resemblance to the original idea, being correspondingly faster and more silicon-efficient. Thus - THOMAS was an accident.

2. WHY VLSI RSA?

Before identifying strategies for the successful implementation of single chip modulus exponentiation functions, hereafter to be (erroneously) denoted by RSA, the question as to whether an RSA device is necessary or desirable must be considered.

The RSA algorithm is now well-known, developing from original public key cryptographic methods, first published in [1 - Diffie, Hellman, 1976]. The recognised version of the algorithm conventionally known as RSA is attributed to [2 - Rivest, Shamir, Adleman, 1978]. Subsequent to this, variants using the same mathematical basis have been developed. As these still employ the exponentiation, any device fulfilling the requirements of the RSA public key algorithm automatically satisfies the related applications. Accordingly, the term RSA will also embrace variants that satisfy this criterion.

The strength of an RSA system is based on the factorisation problem associated with the product of large primes. Recent advances, including application of a technique known as the Quadratic Sieve [3 -

Caron, Silverman, 1986], have shown that factorisation of 80+ digits is now achievable. These use networks of powerful devices, but still require approximately a month to complete the task, every additional 3 digits approximately doubling the required period. This corresponds to 256 bit (nominal) RSA key pairs. When RSA was first advocated, [2 - Rivest, Shamir, Adleman, 1978], 200 digits were suggested as having the desired security for the foreseeable future, corresponding to nominally 600 bits of RSA key pair. Thus, a solution embodying the range of security required, performs 256 to 512 bit operations.

Thus the problem has been established; performing wide, repeated, multiplications and divisions, to achieve a useful operational application of the method. Where software routines have been implemented, except on powerful mini-computers, or hardware-assisted micro-computers, the times achieved for the computation have been poor. Typically, for 512 bit values of average density, an MC6809 requires 4 minutes, an INT8086 70 seconds, and a MC68000 30 seconds. These performances are adequate for key management applications. Many have been implemented, but are less than satisfactory for fast authenticators, rapid key changes using RSA as a transport mechanism, let alone for RSA stream encypherment.

Furthermore, the requirements to perform the operation require the presence of a complete processor sub-system. This may be acceptable in applications where there is a requirement for substantial computing or processing elsewhere, but is an undesirable addition when the remaining requirements are trivial. Not surprisingly, a demand arose for VLSI implementations to achieve orders of magnitude improvement for applications of the nature outlined.

Many papers and much research and development has been devoted to the production of techniques and devices to achieve such performance. These include [4 - Orton, Roy, Scott, Peppard, Tavares, 1986], [5 - Kochanski, 1985], [6 - Rivest, 1985], [7 - Roy, Tavares, Peppard, 1985], [8 - Orton, Peppard, Tavares, 1986], and [9] - Scott, Tavares, Peppard, 1986], and [10 - Beth, Cook, Gollman, 1986]. However, ignoring material not in the public domain associated with hardware implementations of RSA, there are merely a handful of successful implementations, either known to the cryptographic world, or commercially available, albeit embedded in a commercial product. Even these devices, reflect a very recent success, for reasons outlined below. Apart from applications requiring the highest security, the motivation driving VLSI implementations has been cost and performance. The cost savings are reflected in the difference between the components to achieve a desired performance, and a device; the savings in power; and in real estate hence saving in manufacture and test times. Naturally, the savings are offset by the development costs, which in the past have tended to be very substantial.

Notwithstanding the intense academic research associated with cryptography and the establishment of the DES standard, the techniques devised have not achieved the levels of adoption anticipated. This, in turn, has also reduced the interest in the commercial world to develop such devices. However, the advancement of money transfer in many areas and publicity for the success of hackers has rekindled the interest, generating the few RSA implementations that are now available.

With the advent of high performance Digital Signal Processors, e.g. the TMS320 family, a compact, medium performance, device has become generally available for applications of the nature of RSA. NPL [11 - Clayden, 1985] has developed algorithms for the TMS32010 which now execute a 512 bit operation in nominally 2.5 seconds. This has

subsequently been enhanced, [12 - Barrett, 1984] and [13 - Barrett, 1986], to typically 1.5 seconds. With further DSP devices appearing on the market, with progressively higher performance, this might appear to detract from the need for VLSI purpose-designed devices. However, whilst such devices give high performance results, these devices consume substantial power, and require additional circuitry and devices to implement the operation. Furthermore, whilst intrinsically flexible by the nature of the algorithm implementation, the device physically is inflexible, and may not be implemented in differing forms to suit particular applications.

Summing up, the climate, the technology, and the need for RSA VLSI devices now coincide!

3. MATHEMATICAL FUNCTIONALITY

The device is required to execute the two functions:-

$$\begin{aligned} A &= B * C \bmod N \\ &\quad \text{and} \\ A &= B ^\star C \bmod N \end{aligned}$$

The conventional usage of the exponentiator is associated with RSA key management operations. However, the device also acts as a high-speed multiplier, with or without modulus correction, for general processing requirements.

4. HISTORIC FAILURE

There have been many attempts to produce devices that perform a high-performance exponentiation function. There have been almost as many failures. These failures may be attributed to the following reasons:-

Exceeding available technology,
Exotic implementation mechanisms,
Ambitious requirements.

Each of these reasons tends to overlap certain areas of the other.

Despite the rapid advances in semiconductor technology, only recently have VLSI chips of 100,000 transistors plus become readily available, particularly to commercial organisations, where yield and cost have been fundamental to the application. Accordingly, the technology for useful implementations has only become available within the space of the last two years. (By useful, the effective bit width of such a device or concatenation of devices is presumed to be substantially greater than 256, typically 512, as above.)

Secondly, the repercussions of the necessity of large bit widths produces a desire to find techniques that overcome the square- or cube-law deterioration in performance as the bit widths increase. These techniques inevitably demand greater areas of silicon, increased power, and poorer yields.

Finally, the poor performance of the software solutions with the need for high-speed solutions has tended to project higher speed requirements on the device. Thus, where key management functions have been the goal, the need for very fast implementations is generally unnecessary.

5. THE LESSONS OF HISTORY

With hindsight, it is the case that demands were made for performance that outstripped the available capabilities. With the substantial improvements in technology, performance requirements may be achieved by less elegant techniques, permitting a wider flexibility. Thus, considering the three reasons for past failures, the goals may be redefined as follows:-

Assume current technology,
Use old and tried mechanisms,
Limit performance.

With the ability to use more transistors on a silicon die, use the largest number available commensurate with desired unit cost. Under these circumstances, the yield may be ignored, with amortisation of all losses against the acceptable figure.

The RSA algorithms are well known and the requirements for multiplication and modulus division or reduction are well known. Techniques for the execution of these tasks are available for small numbers of bits or groups that are effective and undemanding. If the resultant device, compromised by such unsophisticated techniques, achieves an adequate performance, accept the limitations.

As an overall strategy, set a lower limit on performance that achieves the desired end.

These decisions are all, of course, self evident. Equally, if these targets had been implemented, there would have been RSA devices of a single chip form available for some time, giving a performance of typically 512 bit full exponentiation with 512 bit data and modulus of 10 seconds, well in advance of the DSP devices. Whilst such performance could not be deemed electric, it has only recently been overtaken by the high-performance Data Signalling Processors (DSP) and with a number of associated components (see above).

These represent a basic set of criteria to produce a device. However, to these may be added a further set of requirements that will be shown to complement the criteria, producing a technology component that is flexible for many implementations.

6. IMPLEMENTATION FUNCTIONALITY

The device, known by the internal project code THOMAS, is a complete single chip RSA implementation. "Completeness" was declared to be the complete absence of any other device to perform the RSA computation. Of necessity, there would be other devices to produce a crypto-system of the desired complexity, but not associated with the mathematical operation. In addition, the following criteria were dictated for such a device.

A restriction on the architecture was decreed to ensure an organisation suitable for adequate testing and simulation before commitment to silicon, and ATE testing at a wafer level before encapsulation to a high degree of reliability. This architecture was required to exhibit a high degree of flexibility, preferably at the silicon compiler stage, to permit a family of related devices to be produced easily, efficiently, and inexpensively. This results in an implementation that supports any (reasonable) internal bit width without encroaching on

the effective bit size of the machine, as perceived by the user or the local processor.

In order to comply with minimal systems, interface characteristics were required to satisfy modern usage, the de facto industry standards.

The packaging options were to be as wide as possible, but offering a choice from very compact (e.g. surface-mount or bonding) to more conventional DIP standards as required.

The design and quality programs were to satisfy both commercial and military standards, if possible.

The implementation be chosen to minimise design time, and risk. This included the simplification of simulation, and the guarantee of a fully operational working device, first time.

7. APPLICATIONS

7.1 KEY MANAGEMENT - LINE SECURITY

RAANND SYSTEMS Ltd., and BRITISH TELECOM, have a series of high performance DES and B-CRYPT stream encryptors for low to high performance line security applications. The use of THOMAS reduces power consumption, real estate, and minimises overheads associated with certification, session key exchanges, and subsequent establishment of further session keys at high frequency. These applications require the 256 to 512 bit effective widths.

7.2 SMARTish CARD

The virtues of authentication and key management apply equally well to areas associated with "SMART cards or intelligent tokens. There are generally restrictions on the number of devices that may be accommodated within a flexible, thin, carrier. However, the security requirements may be even greater, with the portability of the medium. Accordingly, the technology of THOMAS may be used to produce an internal 8-bit or even 4-bit architecture where silicon area is limited, thereby permitting the incorporation of EEPROM, RAM, and processor on the same silicon die.

7.3 STREAM ENCRYPTION

Although typical applications of RSA have not included stream encypherment, such usage offers a highly secure line, even at 512 bit. With the chosen architecture, ready expansion to 1024 bit effective bit width is immediately available. Alternately, the internal bit width may be increased from 64 bits to 128 or other useful values, giving an immediate linear increase in performance.

7.4 CONCATENATION

Where speed requirements or security levels are variable, the same architecture may be used as a slice of the desired word length, but effectively increasing the internal bit width.

8. THOMAS IMPLEMENTATION

THOMAS employs 4 kilobits of RAM / REGISTER storage to hold the following 512 bit (max) parameters:-

Key, Modulus, Data, Result,
and 4 work variables.

The internal structure is based on a 64-bit width, hence the RAM is organised as 64 words of 64 bits.

All ALU functions are based on a 64-bit width operation.

The core of the device employs 5000 standard cells, organised as ALUs, registers, multiplexors, and control logic. This is supported by an integrated RAM array, 64 words x 64 bits. The die is nominally 16mm x 16mm.

All I/O is controlled via 16 control/status and buffer port registers, with automatic internal destination computation performed transparently to the user. As the internal bit width is 64 bits, THOMAS may be configured via the control register to function in multiples of 64 bit slices. Where the data lengths are high, e.g. 512 bits, but the key is small, e.g. 2 to 8 bits, a key length register may be loaded with the significant length of the key to over-ride the default execution time, the algorithm and hence the implementation being wholly symmetric.

Consistent with the desire to minimise external circuitry, an on-board oscillator produces the nominal 20 MHz clock, though external crystal control is permitted. This, in turn, is used to generate lower frequency clocks to drive associated circuitry.

Although the RSA implementation itself has no need for a random number generator, two on-board generators are provided, white noise and a pseudo-random shift register generator, which provides a random output for other uses in a system, via a status register and at package pin.

The operation restores all parameters to the initial state, thereby permitting further data to execute with the same modulus and key, or new data with the same modulus and a newly loaded key.

All I/O is performed on a byte or word-wide basis, user selected or pin configured, with pin configuration of READ / WRITE and ENABLE operation to suit INTEL or MOTOROLA buses.

The implementation produces a cubic relationship for encryption / decryption times. Thus, for a full 512 bit exponentiation with a 512 bit key, the device typically produces the result in 750 milliseconds seconds, whereas a 256 bit x 256 takes 98 milliseconds.

9. THOMAS APPLICATIONS

The device has immediate applications in products that employ RSAkey management. It offers substantial savings in power consumption, volume, and costs. If this were its only application area, this would be substantial. However, the value of the device is the demonstration of a powerful solution, which, by virtue of its internal architecture, lends itself to a series of wider applications.

9.1 HIGH-SPEED RSA

Although THOMAS is an ASIC, using standard celss, simply by increasing expenditure and development time, a full-custom device could have been fabricated. This remains as an opportunity for either wider internal architectures for higher performance, or with the same basic implementation, a smaller silicon area and reduced power dissipation, with a nominal performance improvement.

9.2 INTEGRATION

The architecture chosen is based on an internal 64-bit wide path. Any linear multiple, based on powers of 2, offers scope to reduce the performance and hence decrease throughput linearly. Accordingly, an 8-bit wide pathway increases the execution time by 64/8. Simultaneously, this reduces the core and control logic requirements by approximately 1/6, thereby permitting the introduction of additional components, e.g. EEPROM, ROM, RAM, and a small processor. This offers a high performance single chip key management system and encryptor, ideally suited for SMART-type card applications, battery operated and/or hand-helded devices, or similar applications.

10. CONCLUSION

THOMAS is the first of a family of devices that embody an RSA exponentiation facility for a wide range of applications. The availability of this feature permits the ready incorporation of secure key management in all areas of privacy and high security, with performance as required.

REFERENCES

- [1] - W. Diffie, M. Hellman, "New Directions in Cryptography", I.E.E.E. Trans. Information Theory, vol. IT-22, pp. 644-654, November, 1976.
- [2] - R. L. Rivest, A. Shamir, and L. Adleman, "On Digital Signatures and Public Key Cryptosystems", Communications ACM, vol. 21, pp120-126, February, 1978.
- [3] - T. Caron, R. Silverman, "Parallel Implementation of the Quadratic Sieve", presented at CRYPTO 86, Santa Barbara, CA., August, 1986.
- [4] - G. A. Orton, M. P. Roy, P. A. Scott, L. E. Peppard and S. E. Tavares, "VLSI Implementation of Public Key Encryption Algorithms", presented at CRYPTO 86, Santa Barbara, CA., August 1986.
- [5] - M. Kochanski, "Developing an RSA Chip", presented at CRYPTO 85, Santa Barbara, CA., August 1985.

- [6] - R. L. Rivest, "RSA Chips (Past / Present/Future)", Advances in Cryptology, Proc. of EUROCRYPT 84, pp. 159-165, Springer-Verlag, Berlin, 1985.
- [7] - M. P. Roy, S. E. Tavares, and L. E. Peppard, "A CMOS Bit-slice Implementation of the RSA Public Key Encryption Algorithm", Proc. 1985 Can. Conf. on VLSI, Toronto, Canada, pp. 52-55, November 1985.
- [8] - G. Orton, L. E. Peppard and S. E. Tavares, "A Fast Asynchronous RSA Encryption Chip", I.E.E.E. Custom Integrated Circuits Conference, Rochester, N.Y., May 1986.
- [9] - P. A. Scott, S. E. Tavares and L. E. Peppard, "A Fast LSI Multiplier for $GF(2^{**m})$ ", I.E.E.E. Journal on selected Areas in Communications", vol. SAC-4, pp. 62-66, January 1986.
- [10] - T. Beth, B. M. Cook, D. Gollman, "Architectures for Exponentiation in $GF(2^{**n})$ ", presented at CRYPTO 86, Santa Barbara, CA., August, 1986.
- [11] - D. o. Clayden, "Some Methods of Computing the RSA modular exponential", NPL Technical Memorandum TTCC 20/85, August 1985.
- [12] - P. Barrett, "Communications Authentication and Security using public key encryption", M.Sc. Dissertation, Programming Research Group, Oxford University, Sept. 1984.
- [13] - P. Barrett, "Implementing the RSA Algorithm on a standard Digital Signal Processor (DSP)", presented at CRYPTO 86, Santa Barbara, CA., August, 1986.

Author Index

- Barrett, P. 311
Beauchemin, P. 443
Benaloh, J. Cohen, 213, 251
Beth, T. 302
Blakley, G. R. 266
Brassard, G. 223, 234, 443
Brickell, E. F. 3
Cade, J. J. 64
Cain, M. R. 393
Chan, A. H. 405
Chaum, D. 49, 118, 195, 200
Cook, B. M. 302
Crépeau, C. 223, 234, 239 443
de Jonge, W. 49
Desmedt, Y. 111, 459
Dixon, R. D. 266
Evertse, J.-H. 118, 200
Fiat, A. 186
Games, R. A. 405
Goldreich, O. 104, 171, 426
Gollmann, D. 302
Goutier, C. 443
Guillou, L. C. 464
Gyoery, R. 347
James, N. S. 60
Jueneman, R. R. 327
Kaliski, B. S., Jr. 84
Lidl, R. 60
Matyas, S. M. 451
Micali, S. 171, 381
Moore, J. H. 3, 9
Nam, K.H. 35
Niederreiter, H. 60
Orton, G. A. 277
Peppard, L. E. 277
Peralta, R. 200
Purtill, M. R. 3
Putter, P. S. 393
Quisquater, J.-J. 111
Rackoff, C. 381
Rankine, G. 480
Rao, T. R. N. 35
Robert, J.-M. 234
Roy, M. P. 277
Scott, P. A. 277
Seberry, J. 347
Shamir, A. 186
Simmons, G. J. 9
Sloan, B. 381
Stinson, D. R. 418
Tavares, S. E. 277
Tompa, M. 261
Ugon, M. 464
van de Graaf, J. 200
Wagner, N. R. 393
Wigderson, A. 171
Woll, H. 261