

On Recent Results for MD2, MD4 and MD5	1
1. Introduction 2. Hash Function Properties.....	1
3. Hash Function Structure	2
4. The Status of MD2	3
5. The Status of MD4	3
6. The Status of MD5	3
7. Some Alternative Hash Functions.....	4
8. Summary and Conclusions	5
References.....	6
Proper Initialization for the BSAFE Random Number	7
Abstract	7
Timing Attacks on Cryptosystems	9
Summary.....	9
Impact on RSA Data Security Products	9
Countermeasures	10
Asymmetric Encryption: Evolution and Enhancements ..	11
PayWord and MicroMint.....	17
The HMAC Construction	21
RSA for Paranoids	26
Conventional RSA.....	26
About RSA laboratories	27
Unbalanced RSA	28
Other optimizations	28
Collisions in MD4	29
How Alf did it	30
More Developments with Keyed Hash Functions	31
A linear Protocol Failure for RSA	31
The Secure Use of RSA	32
The RSA Cryptosystem	32
Aims of the Attacker	32
Recovering the Private Key	33
Wiener's Attack	34
Using Probabilistic Primes	34
Decrypting Messages	34
Small Messages	34
Chosen Ciphertext Attacks	34

Low Exponent Attacks	35
Forging Signatures.....	35
RSA Block Formats	36
Conclusions	37
References.....	37
PKCS #1 Block Formats	37
FREQUENTLY ASKED QUESTIONS	39
How Do Digital Time-Stamp Support Digital Signatures?	39
References	40
The 1996 RSA Data Security	41
Elliptic Curve Cryptosystems	42
Alfred Menezes	42
Discrete-log cryptosystems	42
Editor's Note	43
Subscription Information.....	43
Elliptic Curve Cryptosystems	44
Why consider other groups?	44
Elliptic curves	44
References	45
STANDARDS UPDATE	45
Elliptic Curves in Draft IEEE Standard	45
S/MIME Standardized	45
The Future of Integer Factorization	46
Andrew M. Odlyzko.....	46
Introduction.....	46
<i>Factorization records and historical estimates</i>	46
<i>Projections of computing power available in the future</i>	46
<i>Algorithmic improvements</i>	48
<i>Factorization using current algorithms</i>	48
<i>Acknowledgements</i>	49
<i>References</i>	49
<i>Conclusions</i>	49
Appendices	50
Appendix A: <i>Historical estimates of the difficulty of factoring</i>	50
Appendix C: <i>Computing power of historical factorizations</i>	51
Appendix E: <i>Moore's law</i>	51
Appendix D: <i>Current computing power</i>	51
Appendix F: <i>Number of computers</i>	51
Appendix H: <i>Comparison of running times of algorithms</i>	52
Appendix G: <i>Running time of algorithms</i>	52
Appendix J: <i>Attacks on DES</i>	53
<i>On the Security of the RC5 Encryption Algorithm</i>	54
References	55
ALGORITHM SUPDATE	55
MD5 Performance for IP Security Questioned	55

<i>Further Comments on Keyed MD5</i>	56
A N N O U N C E M E N T S	57
<i>RSA Laboratories Technical Reports</i>	57
<i>The 1996 RSA Data Security Conference 1996 17-19</i>	57
The Impending Demise of RSA?	58
GilksBrassard.....	58
Welcome to CryptoBytes	59
Subscription Information.....	59
The Impending Demise of RSA?	60
Message Authentication with MD5.....	62
Recommendations	63
Starting over.....	65
References.....	65
The RC5 Encryption Algorithm*	66
Ronald L Rivest	66
Introduction.....	66
Overview of the Algorithm	66
Encryption and Decryption	66
Key Expansion 3	67
Speed	67
Security	67
References	68
N E W S A N D I N F O R M A T I O N	68
X9FI Considers Triple-DES Standard.....	68
RSA Laboratories Publishes PKCS #1	68
A N N O U N C E M E N T S	69
1995 RSA Laboratories Seminar Series	69
RSA Laboratories Technical Reports	69
Suggestions for Random Number Generation in	70
Introduction	70
Random vs. Pseudo-Random Numbers	70
The Seed	71
Example	72
Further Reading	73
Conclusion	73

**RSA
Laboratories'**

Bulletin

News and advice from RSA Laboratories

On Recent Results for MD2, MD4 and MD5

M.J.B. Robshaw
RSA Laboratories

Abstract. Recent *cryptanalytic* results on the properties of three popular hash functions have raised questions about *their security*. This note *summarizes* these results, gives our assessment of *their implications* and offers our recommendations for product planners and developers who may be using these algorithms.

1. Introduction

A *hash function* (or more accurately a cryptographic hash function or message-digest algorithm) operates on an input string of arbitrary length and generates an output string of fixed length. This output is commonly called a hash *value* or a message digest. While much of the motivation for the design of a hash function comes from its usefulness in optimizing the process of digitally signing some document, hash functions can be used for a wide range of purposes.

MD2 [13], MD4 [20] and MD5 [21] are hash functions that were developed by Ron Rivest at MIT for RSA Data Security. A description of these hash functions can be found in RSA Laboratories Technical Report TR-101 [22]. The widespread popularity of the MD family of hash functions is a testament to their innovative and successful design. Indeed MD4 in particular has been used as the basis for the design of many other hash functions (including MD5, SHA-1, RIPEMD) and MD5 is one of the most widely used hash functions in the world today.

Over the years, various results in the cryptanalysis of MD2, MD4 and MD5 have become available and

this note is intended to summarize these results and their impact. First however we will consider the properties of hash functions. An important issue, that we will repeatedly stress, is that not all applications of a hash function rely for their security on the same property. Consequently, identifying which property of a hash function is appealed to within an implementation is very important and sometimes leads to surprising results.

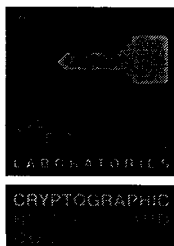
2. Hash Function Properties

Hash functions are designed with a variety of properties in mind and three are commonly singled out in the literature [18].

First, given the hash value output by some hash function, it should be infeasible to find an input or *preimage* that will produce the given output. A slight variation on this gives a second condition which is that even when given an input and output pair for some hash function, it should remain infeasible to find a second distinct preimage that would generate the same output. This is commonly referred to as finding a second *preimage* and a hash function for which it is difficult to find either a preimage or a second preimage is sometimes called a one-way hash function [18].

A third condition that is sometimes required is that it be infeasible to find two inputs to the hash function that will produce the same output. This is commonly referred to as finding a collision for the hash function. Since there are an arbitrary number of possible input strings but only a fixed number of outputs, collisions must exist for a hash function — our objective is to ensure that it is computationally infeasible to find such examples. The term *collision*

Matt Robshaw is a senior research scientist at RSA Laboratories. He can be contacted at matt@rsa.com



resistant hash function [18] is sometimes used to describe a hash function that possesses all three of the properties described here and it is what most people have in mind when talking about hash functions in general.

Note that while a collision-resistant hash function has many useful properties, the property that is actually appealed to within some application can vary dramatically.

For instance, a hash function is often used to reduce some message to a short string which is then signed with a digital signature algorithm. There is a well-known attack on this procedure [26] that depends on the so-called birthday paradox. To prevent this attack we require the property of collision-resistance and as a consequence hash functions that produce an output of at least 128 bits in length. It is worth mentioning that this style of attack is more applicable when the digital signature is computed directly on the digest of the message being signed. Newly proposed probabilistic signature schemes by Bellare and Rogaway [3] have some very attractive properties and it appears that the requirements we place on a hash function in such schemes might be less demanding¹.

Interestingly, digital signatures previously signed using a hash function that is no longer collision resistant are likely to remain safe from compromise. In attacking existing signatures, the task of the cryptanalyst is essentially that of finding a second preimage since the first preimage (the message signed) and the associated hash value are already fixed. This is a much harder problem than that of generally finding a collision and it might well be the case that techniques that generate collisions for a hash function offer no advantage in finding a second preimage. Thus, existing signatures might well remain free from risk of compromise even when collision-resistance is lost.

When a hash function is used for properties other than being collision-resistant, the situation with regards to the suitability of that hash function after the discovery of collisions might remain essentially unchanged. Perhaps one of the properties most

commonly appealed to in a hash function is that of providing a random looking output. A function with this property has many applications and while work on collisions for the hash function clearly has implications for any assumptions about its ideal behavior, it is not clear that they will necessarily offer a substantial practical reason to move away from that hash function as a matter of urgency. Another property that might well be largely unaffected by work on collisions is that of being one-way. When a hash function is being used purely as a one-way function, perhaps to store a table of hashes of computer passwords, then work on collisions might well have little relevance.

In short there are many situations where hash functions are used for properties other than being collision-resistant and we observe that properties that are relevant to cryptographic security in one environment are not necessarily relevant in another.

3. Hash Function Structure

Most hash functions have a similar iterative structure which is based around what is termed a *compression function* [4, 14]. In short, the computation of the hash value for some message depends on what is called a *chaining variable*. At the start of hashing, this chaining variable has a fixed initial value which is specified as part of the algorithm. The compression function is then used to update the value of this chaining variable in a suitably complex way under the action and influence of part of the message being hashed. This process continues recursively, with the chaining variable being updated under the action of different parts of the message, until all the message (and any additional padding specified by the algorithm) has been used. The final value of the chaining variable is then output as the hash value corresponding to that message.

The hash functions MD4 and MD5 are quite similar in design and, as we have already mentioned, they were the inspiration behind the design of several more recent hash functions. MD2 was an earlier hash function with a different structure but it is still widely used if only because it is suitable for 8-bit environments (unlike MD4 and MD5 which are aimed at 32-bit architectures). In the following sections, we will review the status of the three hash functions MD2, MD4 and MD5. This note might be viewed as an update to the RSA Technical Report TR-101 [22] which provides an overview of the same hash functions prior to the very latest developments.

¹ In particular, certain *simple variants* of the schemes in [3] might still securely allow the use of a hash function even though that hash function might not be *fully* collision-resistant. (These variants would only differ from the detailed specifications in [3] in the order in which the random number r and the message M are concatenated prior to hashing.) Further work may well reveal other advantages in varying the length, or the means of generation, of the value r

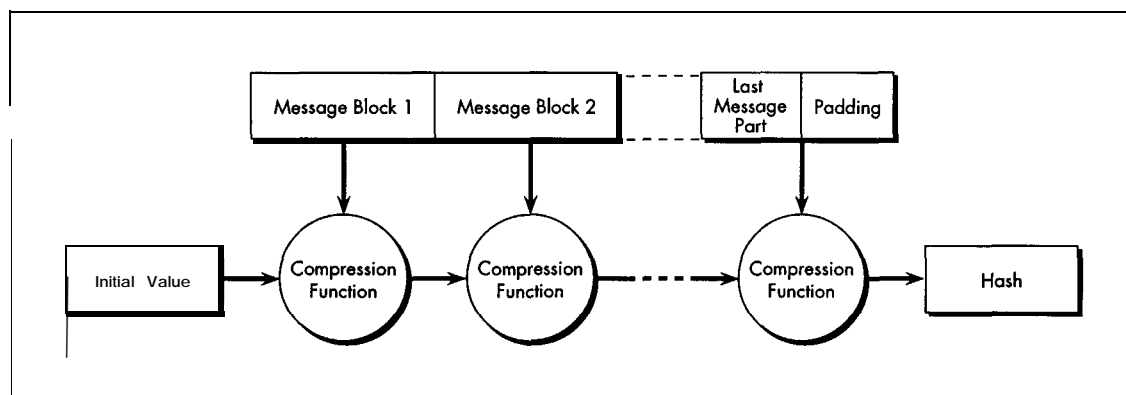


Figure 1.
The use of a compression function in an iterative hash function. The hash functions MD2, MD4, MD5 and SHA-1 essentially follow this design.

4. The Status of MD2

When hashing a message with MD2 there are three different phases. The first is a padding phase whereby the message is padded to form a string that has a length in bytes which is divisible by 16. The second phase is the computation of a 16-byte checksum C which is appended to the end of the message; the checksum is a function of the message and it is computed under the action of a non-linear substitution table based on the digits of π . The resultant string is then divided into 16-byte blocks. The final block, therefore, is the 16-byte checksum. The third phase consists of repeated application of the compression function to compute new values for the chaining variable as a function of both the current value and each message block in turn. The initial value of the chaining variable is fixed as part of the algorithm specifications and the last value for the chaining variable becomes the 16-byte (128-bit) MD2 hash value.

Considerable progress has been made in the cryptanalysis of MD2 [23] and it has been shown that it is possible to find collisions for the compression function of MD2. More precisely, it is possible to find two 16-byte strings such that compression, when starting with particular values of the chaining variable (including the correct initial value), will yield the same output from the compression function. Such work would lead directly to collisions in MD2 were it not for the fact that the final computation of the MD2 hash value depends on the value of the checksum C and this is likely to be different for the two messages.

Currently, it appears to be difficult to make allowances for the existence of the checksum during cryptanalysis. Without the checksum MD2 would be broken but with it, this attempt at cryptanalysis has been thwarted at the very last hurdle. This is perhaps

rather too close for comfort and caution requires that MD2 be no longer recommended for new applications where collision-resistance is required. Questions about the continuing suitability of MD2 for existing applications remain open. Certainly MD2 has not been broken and exist-

ing signatures are not at risk since it is still difficult to find preimages (input messages which yield a given target value with MD2). In addition, future signing will only become vulnerable if full collisions for MD2 can be found. Despite this, while MD2 might still be used in the short term, our recommendation would be to upgrade applications away from MD2 whenever it is practical.

5. The Status of MD4

MD4 was an early design for a particularly fast hash function. Successful cryptanalysis of reduced-round versions of MD4 [15, 5] soon demonstrated, however, that the design of MD4 represented an uncomfortable compromise between security and speed. As a consequence, the more conservatively designed MD5 has always been recommended for use instead of MD4.

Work by Dobbertin [7, 8] has vindicated this early concern about MD4 and it has been shown that collisions for MD4 can be found in about a minute on a typical PC. In addition, a variant of MD4 called **extended-MD4** [20], which offers a 256-bit hash value instead of the usual 128-bit MD4 output, has also been seriously undermined by the work of Dobbertin.

While much of this cryptanalytic work is concerned exclusively with finding collisions, considerable insight into the behavior of MD4 has also been gained, particularly when combined with other independent cryptanalytic work. Both MD4 and extended-MD4 should not be used.

6. The Status of MD5

Attacking MD5 is a much more involved proposition than attacking MD4 since it is a far more complicated algorithm to analyze. The first important advance in the cryptanalysis of MD5 was the discovery of what are termed *pseudo-collisions* for the

compression function of MD5 [6]. A pseudo-collision for the compression function is exemplified by fixing the value of some message block and finding two distinct values for the chaining variable that provide the same output. While the existence of pseudo-collisions is significant on an analytical level, it is of less practical importance. Recall that only a single chaining variable is used during hashing and so the behavior of two related chaining variables is not directly relevant. Instead, it would be more significant if we could identify the value of a single chaining variable for which two different message blocks produce the same output from the compression function. Such an occurrence would have obvious implications for the collision-resistant property we often desire of a hash function. If the value of the chaining variable involved were not the same as the initial value (as provided in the algorithm specifications) then such an occurrence would be termed a *collision for the compression function*. If, however, we could identify two message blocks which provide a collision when the pre-specified initial value is used, then we would have full collisions for the hash function.

At Eurocrypt '96 it was announced that collisions for the compression function of MD5 had been found [9]. In a modification to the techniques used so devastatingly on MD4, Dobbertin demonstrated that collisions for the compression function of MD5 could be found in around 10 hours on a PC. Whereas the pseudo-collisions discovered by den Boer and Bosselaers could not be extended to full collisions for MD5, no such comfort can be drawn with regard to the recent work of Dobbertin. This, of course, should not be viewed as a statement that such an extension is trivial. With the current attack, the cryptanalyst has some freedom in deriving the collision for the compression function, but once this collision has been achieved it seems to be difficult for the cryptanalyst to account for the fact that the initial value to the chaining variable is pre-specified as part of the algorithm. Clearly, to become an attack on the full MD5, some means of directing the value of the input chaining variable to the one specified in the algorithm is required.

While more, possibly very complex, analytical work is required in designing an attack for MD5, there is no telling how much time or effort this might require. Nevertheless, it would be a mistake to rely too much on the fact that current techniques only provide collisions for the compression function of MD5.

Given the surprising speed with which techniques on MD4 were extended to MD5 we feel that it is only prudent to draw a cautious conclusion and to expect that collisions for the entire hash function might soon be found.

Recalling our comments in Section 2, however, we note that there may well be situations where this cryptanalytic work on MD5 has little impact. Note that existing signatures that were generated using MD5 are likely to remain safe from compromise since it seems that current techniques used to cryptanalyze MD5 do not offer any advantage in finding a second preimage. Existing signatures should not be considered as being at risk of compromise at this point. Likewise the random-looking appearance of the output from MD5 and the property of being one-way are not considered to be seriously in question. Finally, one major recent proposal for the use of MD5 has been in what is sometimes referred to as the keyed-MD5 approach to message authentication. In particular, one proposal termed HMAC [1, 2] produces a message authentication code for some message with the help of a hash function, and for implementation efficiency, the use of MD5 in particular has been proposed. The design and properties of HMAC are such that Dobbertin's current techniques that might find collisions for either the compression function or the full hash function of MD5 seem to have little immediate impact on the security of HMAC when MD5 is used [10].

7. Some Alternative Hash Functions

As alternative hash functions, SHA-1, RIPEMD-128 and RIPEMD-160 can still be recommended as being secure for any application. While all three hash functions bear similarities to MD4 and MD5 in their design, the techniques of Dobbertin do not readily extend to these hash functions. Indeed, one of the design criteria for RIPEMD-128 and -160 was that this be the case.

SHA-1 is a revision [17] of the Secure Hash Algorithm (SHA) which first appeared as part of the Secure Hash Standard, FIPS 180 [16]. While the fault in the original SHA and the reasons for the particular change made in SHA-1 are not known, it can be anticipated that SHA-1 is a good hash algorithm to use.

The forerunner to both RIPEMD-128 and RIPEMD-160 was RIPEMD [19], a 128-bit hash function developed within the framework of the European

Union project RIPE (Race Integrity Primitives Evaluation). Its design was based very closely around the principles adopted by Rivest in the design of MD4, and was the subject of considerable analysis, chiefly during the writing of the RIPE report. Despite this however, the techniques recently developed by Dobbertin were first used in attacks on reduced-round versions of RIPEMD [11]. While not extending to the full hash function, the situation with regards to RIPEMD is analogous to that which existed for several years with MD4 when only attacks against a reduced-round version of MD4 were known.

As a consequence, two variants of RIPEMD were proposed [12]. RIPEMD-128 (providing a 128-bit hash value) was intended as a drop-in replacement for RIPEMD and offers additional protection against the cryptanalytic techniques of Dobbertin. However, the longer hash value provided by RIPEMD-160 (with a 160-bit hash value) is likely to be appealing in the longer term [24]. With regards to performance, while RIPEMD-128 appears to be somewhat faster than SHA-1, RIPEMD-160 is slightly slower [12]. All three hash functions are slower than MD5.

8. Summary and Conclusions

In summary, we list the more significant cryptanalytic results on MD2, MD4 and MD5 and describe some of their immediate consequences.

MD2 Collisions have been demonstrated for a modified version of MD2 [23].

Existing signatures formed *using* MD2 are not at risk, but MD2 can no longer be recommended for future *applications that* will depend on the hash function being collision-resistant. MD2 remains *suitable* for use as a one-way hash function.

MD4 A variety of cryptanalytic results cast doubt on the complexity of the MD4 design [15, 5, 25]. Collisions have been demonstrated for MD4 [7, 8].

MD4 should not be used. (This merely restates a recommendation which *has been present* in the literature for some time, in *fact*, its use has not been recommended since the introduction of MD5.)

MD5 Both pseudo-collisions [6] and collisions [9, 10] for the compression function of MD5 have been demonstrated, though collisions for the full MD5 have not yet been achieved.

Existing signatures *formed* using MD5 are not *at risk* and *while* MD5 is *still* suitable for a variety of *applications* (namely those which rely on the one-way property of MD5 and on the random appearance of *the output*) *as* a precaution it *should* not be used for future applications that require the *hash function* to be *collision-resistant*.

In this bulletin we have considered various cryptanalytic developments in the analysis of the compression functions of MD2 and MD5 and for the entire MD4 hash function.

Some of this new cryptanalytic evidence confirms suspicions voiced several years ago by Ron Rivest and makes clear that MD4 should not be used. With regards to existing applications which use MD2 and MD5, collisions for these hash functions have not yet been discovered but this advance should be expected. As a consequence applications which rely on the collision-resistance of a hash function should be upgraded away from MD2 and MD5 when practical and convenient. They can probably be safely “swapped out” in coordination with the vendor’s normal product release cycle. RSA Laboratories currently recommends that in general, the hash function SHA-1 [17] be used instead but RIPEMD-160 would also be a good alternative. (It is interesting that both hash functions borrow heavily from the initial structural design of MD4.)

Occasionally performance requirements or the existence of fielded applications might make a move to SHA-1 undesirable when contrasted with the possibility of using MD5. In such cases it should be confirmed that either collision-resistance is not required within the application or that existing collision attacks do not apply in that particular environment.

It is important to note that these recent cryptanalytic efforts on both MD2 and MD5 are almost exclusively relevant to finding collisions. The most significant impact of the discovery of hash function collisions is on the suitability of that hash function as a part of the process of digitally signing some document. Even then, some signature schemes might well be more tolerant of the presence of collisions than others and so careful analysis is recommended to assess exactly what properties of the hash function are being relied upon in an application. Note that we expect the discovery of collisions to have little or no impact on the other properties that we usually asso-

ciate with a hash function, such as pseudo-randomness or one-wayness.

BSAFE customers should note that while the continued suitability of MD5 for applications requiring collision-resistance is in question, the use of MD5 in other roles, for example as part of a mechanism for random number generation, is still considered safe. Indeed, MD2 and MD5 may well remain suitable for a wide range of applications.

References

- [1] M. Bellare, R. Canetti and H. Krawczyk. The HMAC construction. *CryptoBytes*, 2(1): 12-15, 1996.
- [2] M. Bellare, R. Canetti and H. Krawczyk. Keying hash functions for message authentication. In *Advances in Cryptology — Crypto '96*, pages 1-15, Springer-Verlag, 1996.
- [3] M. Bellare and P. Rogaway. The exact security of digital signatures — how to sign with RSA and Rabin. In *Advances in Cryptology — Eurocrypt '96*, pages 399-416, Springer-Verlag, 1996.
- [4] I. Damgård. A design principle for hash functions. In *Advances in Cryptology — Crypto '89*, pages 416-427, Springer-Verlag, 1990.
- [5] B. den Boer and A. Bosselaers. An attack on the last two rounds of MD4. In *Advances in Cryptology — Crypto '91*, pages 194-203, Springer-Verlag, 1992.
- [6] B. den Boer and A. Bosselaers. Collisions for the compression function of MD5. In *Advances in Cryptology — Eurocrypt '93*, pages 293-304, Springer-Verlag, 1994.
- [7] H. Dobbertin. Alf swindles Ann. *CryptoBytes*, 1(3): 5, 1995.
- [8] H. Dobbertin. Cryptanalysis of MD4. In *Proceedings of the 3rd Workshop on Fast Software Encryption*, Cambridge, U.K., pages 53-70, Lecture Notes in Computer Science 1039, Springer-Verlag, 1996.
- [9] H. Dobbertin. *Cryptanalysis of MD5* Compress. Presented at the rump session of Eurocrypt '96, May 14, 1996.
- [10] H. Dobbertin. The Status of MD5 after a Recent Attack. *CryptoBytes*, 2(2): 1-6, 1996
- [11] H. Dobbertin. RIPEMD with two round compress function is not collision-free. *Journal of Cryptology*. To appear.
- [12] H. Dobbertin, A. Bosselaers, and B. Preneel. RIPEMD-160: A strengthened version of RIPEMD. Final version available via ftp at ftp.esat.kuleuven.ac.be/pub/COSIC/bosselaers/ripemd/.
- [13] B.S. Kaliski Jr. RFC 1319: The MD2 Message-Digest Algorithm. RSA Data Security, Inc., April 1992.
- [14] R.C. Merkle. One way hash functions and DES. In *Advances in Cryptology — Crypto '89*, pages 428-446, Springer-Verlag, 1990.
- [15] R.C. Merkle. Note on MD4. 1990. Unpublished. (A description can be found in [5].)
- [16] National Institute of Standards and Technology (NIST). FIPS Publication 180: Secure Hash Standard (SHS). May 1993.
- [17] National Institute of Standards and Technology (NIST). FIPS Publication 180-1: Secure Hash Standard. April 1994. Available from <http://csrc.nsc1.nist.gov/fips/>.
- [18] B. Preneel. Analysis and Design of Cryptographic Hash Functions. Ph.D. Thesis, Katholieke Universiteit, Leuven, 1993.
- [19] RIPE Integrity Primitives. Final report of RACE Integrity Primitives Evaluation (R1040), Part III: Recommended Integrity Primitives, Chapter 3: RIPEMD, June 1992, pages 67-109.
- [20] R.L. Rivest. The MD4 message digest algorithm. In *Advances in Cryptology — Crypto '90*, pages 303-311, Springer-Verlag, 1991.
- [21] R.L. Rivest. RFC 1321: The MD5 Message-Digest Algorithm. M.I.T. Laboratory for Computer Science and RSA Data Security, Inc., April 1992.
- [22] M.J.B. Robshaw. MD2, MD4, MD5, SHA and other Hash Functions. RSA Laboratories Technical Report TR-101, version 4.0, July 24, 1995.
- [23] N. Rogier and P. Chauvaud. The compression function of MD2 is not collision free. Presented at Selected Areas in Cryptography '95, Carleton University, Ottawa, Canada. May 18-19, 1995.
- [24] I? van Oorschot and M. Wiener. Parallel collision search with application to hash functions and discrete logarithms. In *Proceedings of 2nd ACM Conference on Computer and Communication Security*, pages 210-218, ACM Press, 1994.
- [25] S. Vaudenay. On the need for multipermutations: Cryptanalysis of MD4 and SAFER. In *Proceedings of the 2nd Workshop on Fast Software Encryption*, Leuven, Belgium, pages 286-297, Lecture Notes in Computer Science 1008, Springer-Verlag, 1995.
- [26] G. Yuval. How to swindle Rabin. *Cryptologia*, July 1979.

For more information on this and other recent security developments, contact RSA Laboratories at one of the addresses below.

RSA Laboratories
100 Marine Parkway, Suite 500
Redwood City, CA 94065 USA
415/595-7703
415/595-4126 (fax)
rsa-labs@rsa.com
<http://www.rsa.com/rsalabs/>

RSA
Laboratories'

Bulletin

News and advice from RSA Laboratories

Proper Initialization for the BSAFE Random Number Generator

Dr. **Robert W. Baldwin**
RSA Data Security, Inc.

Abstract

This bulletin will help you avoid a particularly poor way of initializing the random number generators in the BSAFE 1.x or BSAFE 2.x toolkit. If you have followed the advice in the BSAFE 2.1 Users Manual, which is to initialize the generators with large blocks of seed bytes, then you do not have this security weakness. However, random number generators are often used to produce encryption keys, so the impact of poor initialization could be substantial.

We first explain how you can recognize whether your product has this problem. If it does, we have included a simple fix that can be added to your product's source code. Alternatively, you can upgrade to the BSAFE 3.x toolkit which eliminates the problem without any need for changes in your product's source code.

Recognizing the problem

You need to look at all places that your product calls `B_RandomUpdate`. If the number of seed bytes passed to each invocation of `B_RandomUpdate` is small, then you may have the weakness, even if `B_RandomUpdate` is called many times.

In the sample code, `B_RandomUpdate` is called 20 times, but each call only passes a single seed byte. The algorithm used to update the state of the random number generator (in BSAFE 1.x and BSAFE

2.x) assumes that each call will pass a substantial amount of unpredictability in each block of seed bytes. However, in this case, there is only one byte in each seed block, and it is likely to be a character from the home row of the keyboard, of which there are only 10. That is very little unpredictability per seed block.

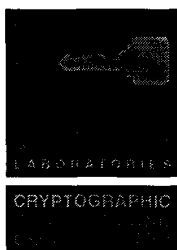
```
DisplayMessage
    ("Please enter 20 random keystrokes");
for (i = 0 ; i < 20 ; i++)
{
    c = GetUserKeystroke();
    if (0 != B_RandomUpdate(randomObj,
        &c, 1, NULL-CONTEXT))
        return (ERROR-XXX);
}
```

The number of states for the generator does not depend on the order in which the seed blocks are processed by `B_RandomUpdate`. After executing the sample code, the random number generator will be in one of $(20+10-1)!/(20!(10-1)!)$ possible states, which is about 23 bits of randomness. This is not even enough randomness to generate a good 40 bit key for an exportable cryptographic product, let alone provide strong domestic security. The implementer of this code may have expected that there would be 10^{**20} possible states, roughly 66 bits of randomness, which is reasonable for systems that use the DES cipher.

Fixing the problem

There are several ways to fix the problem. One would be to gather up all the seed bytes and pass

Bob Baldwin is a senior engineer at RSA Data Security. He can be contacted via baldwin@rsa.com.



them via a single call to `B_RandomUpdate`. This would extract the maximum amount of randomness from all the seed bytes. For example:

```
DisplayMessage
    ("Please enter 20 random keystrokes");
for (i = 0 ; i < 20 ; i++)
{
    seedBuffer[i] = GetUserKeystroke();
}
if (0 != B_RandomUpdate(randomObj,
    seedBuffer, 20, NULL-CONTEXT))
    return (ERROR_XXX) ;
```

An alternative is to include a counter value with each byte of seed. Surprisingly, this fixes the problem with small seeds. For user input, the counter could be replaced with a sample from a clock that is updated at least 10 times per second. The code for this alternative is shown below. This approach extracts nearly as much randomness from the input as the previous approach and it is well suited for applications that periodically update the random object as they executes.

```
/* Update the random object anytime a key is pressed. */
int AddNewRandomness ( )
{
    /* Using a clock (10th of second OK) would be better than */
    /* a counter and would not require static storage. */
    static struct { /* Static, so counter is persistent. */
        long    counter;
        char    keystroke;
    } seedBlock;

    seedBlock.counter++;
    seedBlock.keystroke = GetLastKeystroke();
    if (0 != B_RandomUpdate(randomObj,
        (char *)&seedBlock, sizeof(seedBlock),
        NULL-CONTEXT))
        return (ERROR-XXX) ;
    return (0) ;
}
```

For further information contact technical support by calling (415)595-7705 between **9** A.M. and **5** P.M. Pacific time, or fax at (415)595-1873, or email at tech-support@rsa.com.

For more information on this and other recent developments in cryptography, contact RSA Laboratories at one of the addresses below.

RSA Laboratories

100 Marine Parkway, Suite 500
Redwood City, CA 94065 USA
415/595-7703
415/595-4126 (fax)
rsa-labs@rsa.com
<http://www.rsa.com/rsalabs/>



RS A
Laboratories'

Bulletin

News and advice from RSA Laboratories

Timing Attacks on Cryptosystems

Dr. Burt Kaliski

RSA Laboratories

Summary

A new class of "timing attacks" published recently by Paul Kocher, an independent cryptography consultant (see J. Markoff, "Secure digital transactions just got a little less secure," New York Times, December 11, 1995), poses a potential risk to a variety of cryptosystems, including RSA, DSA, Diffie-Hellman and RC5.

The concept of timing attacks has been known for years, but Kocher's results are new and significant in that they can recover complete key information given only the running time of an operation. Previous attacks could only recover partial key information or required timing information on the individual steps within a cryptographic operation.

Of greatest concern are algorithms with a key-dependent correlation between input and running time, including RSA, DSA, Diffie-Hellman, and RC5. Algorithms without such correlations are unaffected. In general, DES, DESX, triple-DES, RC2, RC4, and standard message-digest algorithms (for which the "key information" might be the input message) are not affected, although in special cases, according to Kocher, there may be some minor leakage of key information.

For one of the new attacks to succeed, it must be possible to measure the running time of crypto-

graphic operations. Thus operations in an interactive protocol such as SSL, or in a cryptographic module in the attacker's possession, such as a smart card, are at the greatest risk. This is true even if there is some "noise" in the measurement, such as transmission delays, since the attacker can factor out the noise by averaging enough measurements. Operations performed privately and without external feedback, on the other hand, such as off-line digital signatures or file encryption, are unaffected.

Impact on RSA Data Security Products

The timing attacks potentially affect the implementations of the RSA and DSA algorithms in BSAFE and RSAREF, which, like many other implementations, are optimized for performance, and hence take an amount of time that can potentially be correlated with the input. (TIPEM and other products based on BSAFE are similarly affected.)

The attacks do not affect the implementation of Diffie-Hellman key agreement in BSAFE and RSAREF, as the Diffie-Hellman exponent is not a fixed key, but instead varies from one call to another. They also do not affect the implementation of RC5 in BSAFE on standard platforms as the RC5 rotations take a constant amount of time on those platforms and so do not give opportunity for correlation. However, RC5 may be affected on other platforms.

Other algorithms in BSAFE and RSAREF, including DES, DESX, triple-DES, RC2, RC4, and the message-digest algorithms, as noted above, are in general not affected.

Burt Kaliski is chief scientist at RSA Laboratories. He can be contacted at burt@rsa.com



Countermeasures

A simple way to prevent timing attacks, regardless of algorithm, is to ensure that all operations with a given algorithm take the same amount of time by "quantizing" the operations into a fixed time period. This approach is highly dependent on the environment, and may degrade performance, but it requires no modification to the algorithm implementations. "Quantizing" code was recently offered by Matt Blaze of AT&T Bell Laboratories at <ftp://research.att.com/dist/mab/quantize.sbar>.

For RC5, it is sufficient to ensure that the rotations in RC5 take a constant amount of time, which is the case on all platforms supported by BSAFE 3.0.

For RSA, one can prevent the attacks by introducing what is called "blinding" into the cryptographic operations, without changing the underlying implementation. This approach, suggested by Ron Rivest, is being incorporated into BSAFE 3.0 for RSA. (Similar algorithmic could be incorporated for countermeasures for DSA and Diffie-Hellman.) Specifically, the RSA private-key operation $y := x^d \bmod n$, where x is the input, y is the output, n is the RSA modulus and d is the private exponent, is implemented as follows:

1. Generate a secret random number r between 0 and $n-1$.
2. Compute $x' := x r^e \bmod n$, where e is the public exponent.
3. Compute $y' := (x')^d \bmod n$ with the ordinary RSA implementation.
4. Compute $y := y' r^{-1} \bmod n$. It is easy to see that the result is as expected by noting that $r^e \equiv r^{-1} \pmod{n}$.

Since step 3 involves a random, secret x' , its running time cannot be correlated with the input x , hence the term "blinding." Consequently, a timing attack cannot obtain any information about the private key.

There is still the theoretical concern that information about the random value r may be revealed from the time for steps 2 and 4, but this does not seem to be a practical issue provided the value of r varies from one RSA operation to the next.

A practical way to generate the r value in step 1, suggested by Steve Burnett, is to compute a one-way function on the input x and the RSA private key; this makes the r value vary from one input value to another, and also keeps the r value unknown. This is being done in BSAFE 3.0 following techniques based on MD5 similar those of Krawczyk et al. for message authentication (see "Keyed-MD5 for message authentication" submitted as an Internet-Draft in November 1995, and M. Bellare, R. Canetti, and H. Krawczyk, "Keyed hash functions and message authentication," to be presented at the 1996 RSA Data Security Conference). In particular, the seed is computed as

$$\text{seed} := \text{MD5}(p \parallel \text{pad}_p \parallel \text{MD5}(q \parallel \text{pad}_q \parallel c))$$

where p and q are the RSA primes, least significant byte first, pad_p and pad_q are strings of zero bytes sufficient to extend p and q to lengths that are multiples of MD5's block size of 64 bytes, and \parallel denotes concatenation. The random r value is generated from the seed with BSAFE's MD5Random pseudorandom generator.

For more information on this and other recent developments in cryptography, contact RSA Laboratories at one of the addresses below.

RSA Laboratories

100 Marine Parkway, Suite 500
Redwood City, CA 94065 USA
415/595-7703
415/595-4126 (fax)
rsa-labs@rsa.com
<http://www.rsa.com/rsalabs/>

RSA LABORATORIES' CryptoBytes

The technical newsletter of RSA Laboratories, a division of RSA Data Security, Inc.

Contents

1

Asymmetric
Encryption: Evolution
and Enhancements

2

Editor's Note

7

PayWord and
MicroMint:
Two Simple
Micropayment
Schemes

12

Message
Authentication Using
Hash Functions:
the HMAC Construction

16

Announcements

Asymmetric Encryption: Evolution and Enhancements

Don B. Johnson and Stephen M. Matyas
IBM Cryptography Center of Competence, MS P330
522 South Road
Poughkeepsie, NY 12601 USA

When public key cryptography was invented, one of its uses was identified as the secure transport of secret symmetric keys. The objectives of such a key transport mechanism keep evolving as attacks are identified, hidden assumptions are revealed, proofs of security are given, and additional capability is needed. The process continues in this article.

We trace the evolution of some asymmetric key transport mechanisms, starting with the method in PKCS #1 [10]. We then discuss, in historical order, two masking techniques developed by IBM cryptographers, and the method currently under study in ANSI draft standard X9.44 RSA Key Transport. We then give ideas that may be useful when using elliptic curve cryptography, where the size of the block is typically much less than that used with other algorithms, for example, RSA.

We will use the following terminology:

Formatted block — a block of data passed as input to the methods. It contains a secret symmetric key

and other data to provide evidence of correct recovery and to thwart certain attacks.

Masked block — a block that results when the formatted block is masked to hide patterns.

Encrypted block — the block that results after the formatted or masked block has been asymmetrically encrypted.

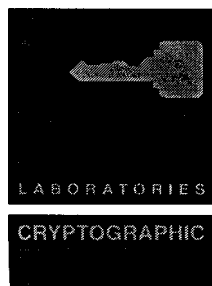
For readers interested in some of the security issues involved in using RSA, an earlier CryptoBytes article entitled The Secure Use of RSA [9] contains much useful information.

PKCS #1

The Public Key Cryptography Standard #1 was designed by the cryptographers at RSA Data Security, Inc. [10]. PKCS #1 describes a method to RSA encrypt a secret symmetric key. The formatted block is passed directly to the RSA encrypt process. It uses the following method (with rationale):

1. A leading 0x00 is in the block to be RSA encrypted, ensuring the encryption block is less than the RSA modulus.
2. A block type encoded octet of 0x02 follows the leading 0x00, indicating the block is to be encrypted using a public key.
3. At least eight non-zero pseudorandom padding octets (bytes) are appended to the right after the block type octet. The padding octets should be generated independently for each RSA encryption, especially if the same key is being encrypted. This thwarts Hastad's attack [6] and allows use of a low value (e.g., 3) for the public

(continued on page 3)



Editor's Note

We encourage any readers with comments, opposite opinions, suggestions or proposals for future issues to contact the CryptoBytes editor.

Readers of the Autumn 1995 issue of CryptoBytes will recall that we included two articles related to RSA encryption. In one, we concentrated on the secure use of conventional RSA and in the other we looked at a new proposal by Adi Shamir termed Unbalanced RSA.


Our lead article in this issue is also related to RSA encryption, but in reality has wider implications for public-key technology in general. The use of what was introduced as Optimal Asymmetric Encryption Padding (OAEP) has been receiving a lot of attention recently and it seems an appropriate time to look at this topic in more depth. We lead this issue with an article by Don Johnson and Stephen Matyas on the evolution of padding schemes for use with asymmetric encryption. Many readers will already be aware that standardization efforts are currently underway on the specification of such techniques (for instance as part of the current Secure Electronic Transactions (SET) standard effort) and we hope that this article will provide readers with some of the background to these discussions. Future developments will, of course, be reported in upcoming issues of CryptoBytes.

For new technologies we turn our attention to what have been called micropayment schemes and to two that were recently devised by Ron Rivest and Adi Shamir. Micropayment techniques are aimed specifically at accommodating commerce over the Internet and World-Wide Web where small-scale payments are often required at an exceptionally low operational cost. In this article, the significant features of PayWord and MicroMint are described as well as the philosophy behind some of the security issues involved.

Our third article is a presentation of HMAC by Mihir Bellare, Ran Canetti and Hugo Krawczyk. In the use of keyed-MD5 for message authentication, HMAC appears to be a solution that is rapidly gaining acceptance and a report on this construction, particularly since it forms a continuation of previous CryptoBytes articles, will be of interest to a great many readers.

This issue of CryptoBytes is the first of what will become the second volume. In the year since the newsletter was launched we have had a lot of positive response to both the aims of the newsletter and to

the articles we have been carrying. We are delighted to announce that CryptoBytes will now be available free of charge and that all issues will be directly accessible via the World-Wide Web.

The future success of CryptoBytes depends on input from all sectors of the cryptographic community, and as usual we would very much like to thank the writers who have contributed to this first issue of the second volume. We encourage any readers with comments, opposite opinions, suggestions or proposals for future issues to contact the CryptoBytes editor at RSA Laboratories or by E-mail to bytes-ed@rsa.com. 

Newsletter Availability and Contact Information

CryptoBytes is a free publication and all issues, both current and past, are available via the World-Wide Web at <http://www.rsa.com/rsalabs/cryptobytes/>.

For each issue a limited number of copies are printed. They are distributed at major conferences and through direct mailing. While available, additional copies of the newsletter can be requested by contacting RSA Laboratories though a nominal fee to cover handling costs might be charged for individual requests.

RSA Laboratories can be contacted at:

RSA Laboratories
100 Marine Parkway, Suite 500
Redwood City, CA 94065
415/595-7703
415/595-4126 (fax)
rsa-labs@rsa.com

About RSA Laboratories

RSA Laboratories is the research and development division of RSA Data Security, Inc., the company founded by the inventors of the RSA public-key cryptosystem. RSA Laboratories reviews, designs and implements secure and efficient cryptosystems of all kinds. Its clients include government agencies, telecommunications companies, computer manufacturers, software developers, cable TV broadcasters, interactive video manufacturers, and satellite broadcast companies, among others.

Asymmetric Encryption
Continued from page 1

- key exponent, which is important for performance reasons.
4. An octet of 0x00 is then appended on the right of the above to act as a delimiter between the non-zero pseudorandom octets and the key.
 5. The secret symmetric key is then appended on the right of the above.

A simplified diagram of the PKCS #1 method follows:

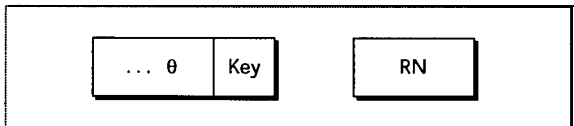


Figure 1. Simplified PKCS #1 method. Key is the secret symmetric key and RN is a random number. The actual specification of PKCS #1 differs from this simplified representation. See the text for details.

After RSA decryption, an error is returned if any of the following conditions exist:

1. The block type octet is not 0x02.
2. There are not at least eight padding octets. A padding octet cannot be all binary zeros.
3. The block cannot be parsed. Besides the above reasons, this might happen if a zero octet delimiter is not found.

There is significant security provided by the PKCS #1 method. We are unaware of any realistic attacks on this method in practise. Placing the non-zero block type octet near the left end of the block ensures that the RSA modulo reduction will occur, even for low public key exponents, like 3. There are at least 64 bits of pseudorandomness to thwart an exhaustion attack and Hastad's attack. There are a minimum of 16 bits of redundancy (block type octet and zero delimiter octet) to provide evidence that the formatted block has been recovered correctly.

However, a concern could be raised that some of the bits in the formatted block are known. Recent results by Don Coppersmith [4] show that for an RSA public exponent of 3 and an n-bit modulus, if an attacker knows all but $n/3$ of the plaintext and all the ciphertext, he can recover all the plaintext; if an attacker knows that two messages agree in all but $n/3$ bits and are encrypted with the same public key, he can recover the plaintext.

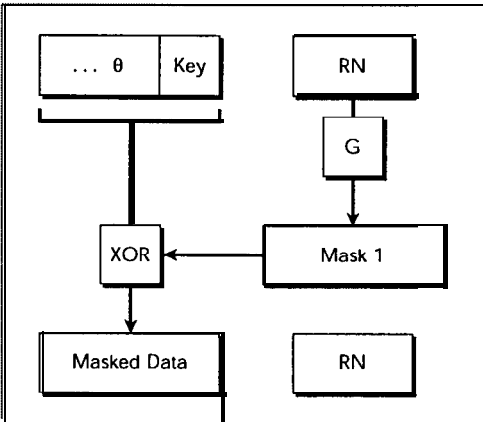
Readers interested in the development of PKCS#1 are referred to the editors note at the conclusion of this article.

IBM Transaction Security System

At about the same time as the PKCS was being published, IBM developed a public key extension to their Transaction Security System (TSS) to support digital signatures and key transport. Cryptographers at IBM (Johnson, Le, Martin, Matyas, and Wilkins) developed a method to mask a formatted block before its being RSA encrypted [7, 8]. A simplified diagram of the JLMW method is given in Figure 2.

Its objectives, as given in the paper, are as follows:

1. Ensure the method is strong for any arbitrary public key value, large or small.
2. Ensure that the method is strong regardless of whether the same secret symmetric key value is encrypted by many different public keys.
3. Ensure there are least at 32 bits of redundant data that may be used to authenticate the symmetric key recovery process.
4. Remove any structure from the data that is encrypted by the public key.
5. Ensure the symmetric key can always be encrypted as one block by the public key.



Goal 1 is a common goal of most asymmetric systems, as often some keys will have better performance than others. Goals 2, 3, and 5 are also goals of the PKCS #1 design. Goal 3 was addressed in the PKCS #1 method by using 16 bits and in the JLMW method by using 32 bits.

Goal 4 was not addressed in PKCS #1 and is the most unusual. In fact, goals 3 and 4 appear to conflict. Goal 3 says there is redundancy (structure) to check and goal 4 says there does not appear to be any structure. The solution is to realize that while there may need to be significant structure, there should not APPEAR to be any structure. This implies a masking operation is performed on the structured data so that it appears

At about the same time as the PKCS was being published, IBM developed a public key extension to their Transaction Security System to support digital signatures and key transport.

Figure 2. Simplified JLMW method. RN is a random number and G is a masking function that produces a mask from a supplied random input.

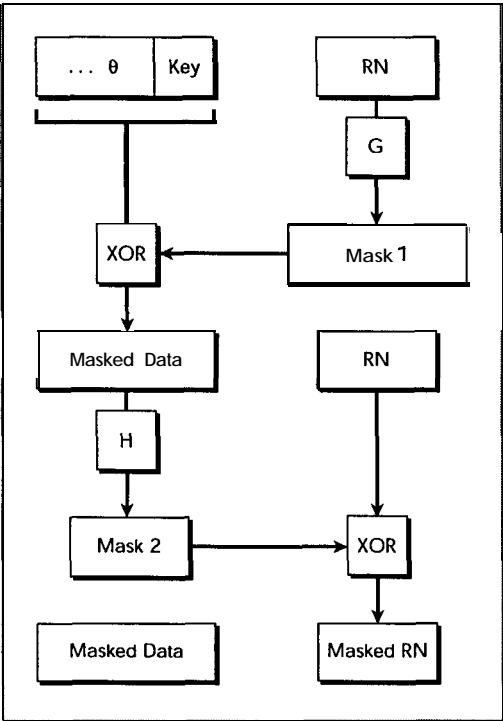
While there is significant structure in the formatted block, the masked block appears random.

random. The idea is to provide a random number in the formatted block, that would be input to a pseudorandom generator and used to generate a pseudorandom string that could be exclusive-ORed with the formatted block, thereby masking any structure. A different random number is generated each time a formatted block is to be RSA encrypted. (This ensures that a formatted block is not encrypted twice using different RSA public keys.) In effect, RSA encryption is performed on masked data consisting of a random appearing string concatenated to a random number. While there is significant structure in the formatted block, the masked block appears random.

Besides the basic masking method, specifying control information was identified as an option.

Optimal Asymmetric Encryption

Other cryptographers at IBM (Bellare and Rogaway) saw a way to improve the JLMW construction [3]. Fig. 3 gives a simplified diagram of their construction.



The question they asked themselves was: Suppose an adversary could recover some of the bits in the encrypted block more easily than recovering the entire encrypted block. If this were possible, then this knowledge might enable one to more easily deduce some or all of the remaining bits: that is, it might give an advantage in recovering other bits.

A result given by Alexi, Chor, Goldreich and Schnorr suggests that finding the low order bits of an RSA encrypted block are as hard as finding the entire encrypted block [1].

But what if an adversary attacked bits not covered by this result, or what if another asymmetric algorithm was used, one possibly without this property?

Although they did not find an attack, they apparently had the following concern about the JLMW

masking method: If an adversary could discover the random number component of the masked block, then an adversary could reconstruct the masking string. Knowing the masking string and the structure in the formatted block, it might be possible for an attacker to deduce other parts of the formatted block. Their insight into the construction of a good masking function was to realize that one could also protect the random number by masking it with a hash value generated on the masked block (less the random number). If this was done in the right way, then an adversary would have to recover the entire masked block in order to invert the masking operation and recover the formatted block containing the secret symmetric key. They were then able to provide a proof of the security of their method.¹

To provide for non-malleability, 128 binary zeroes are used in the formatted record.² This also provides evidence of correct recovery.

Enhanced Optimal Asymmetric Encryption

The enhancement over the OAE method provided in the proposed ANSI X9.44 draft standard is essentially to add a one-way hash of information associated with the key, such as higher-level security protocol data [2]. As this also provides for non-malleability, the 128 bits of binary zeroes in the formatted block, called for by Bellare and Rogaway, may be omitted.

While still providing non-malleability, this provides a strong coupling mechanism of the secret symmetric key to arbitrary information associated with the key. For example, such information may include:

- 1. Method version identification.
- 2. Identification of the H and G masking functions.
- 3. Type of symmetric key algorithm.
- 4. Intended usage of the symmetric key.
- 5. Cryptoperiod for the key.
- 6. Identifiers for the creator and/or recipient of the key.
- 7. Key creation timestamp.

¹ DES may be viewed as a ladder with 16 rungs, each rung corresponding to a round of DES. This is also called a Feistel network or ladder. On the odd numbered rounds, a one-way function (under the control of the key) operates on the rightmost 32 bits to produce a mask that is exclusive-ORed with the leftmost 32 bits. On the even numbered rounds the one-way function operates on the (masked) leftmost 32 bits to produce a mask that is exclusive-

Figure 3. Simplified Bellare-Rogaway OAE method. G and H are masking functions.

- 8. A nonce used in a request/response protocol to ensure freshness.

As there are many possible ways to use this capability, it is important that it be open-ended. As new protocols are devised and the need to strongly couple currently unforeseen information to the secret key becomes identified, this method can grow to meet potential future requirements. A simplified diagram of this mechanism is shown in Figure 4.

Reverse Signature Concept

One way to perceive the enhancement of coupling a hash of information associated with a key to the key itself is as a kind of "reverse signature." In a normal digital signature, only the owner of the private key can generate a signature, but anyone can have access to the public key to verify the signature. In a "reverse signature," anyone can generate such a signature, but only the owner of the private key can verify it.

Using asymmetric key cryptography, there are two ways to cryptographically couple information to a key: by using a digital signature and by using a reverse signature.

However, there are several distinct advantages to the reverse signature concept:

- 1. The implementation of key recovery is simplified, needing only the associated private key. It does not need a public key to verify the recovery or to provide coupling to associated information about the key.
- 2. All digital signatures may be calculated on the encrypted symmetric key block. None need to be calculated on the unencrypted symmetric key block. This ensures all parties can verify any digital signatures as needed, without requiring recovery of the symmetric key. This can be important

in a generalized environment with many certification authorities in varying relationships.

- 3. In an interactive environment, one may be able to avoid the use of digital signatures. In this scenario, the initiator RSA-encrypts a secret nonce using the public key of the secret key generator. The secret key generator is the only entity that can recover the secret nonce, so only the initiator and the secret key generator know it. The secret key generator generates the secret key and includes the secret nonce in the RSA encrypted reply. The initiator then supplies the secret nonce to the secret key recovery process. The initiator is assured that the secret key is from the key generator. The key generator is assured that only the initiator is able to recover the secret key.

- 4. As the information associated with the key is supplied to the key recovery process, it is expected to be easy to demonstrate that this method provides secrecy only for a symmetric key of limited size. This can be important due to product export considerations.

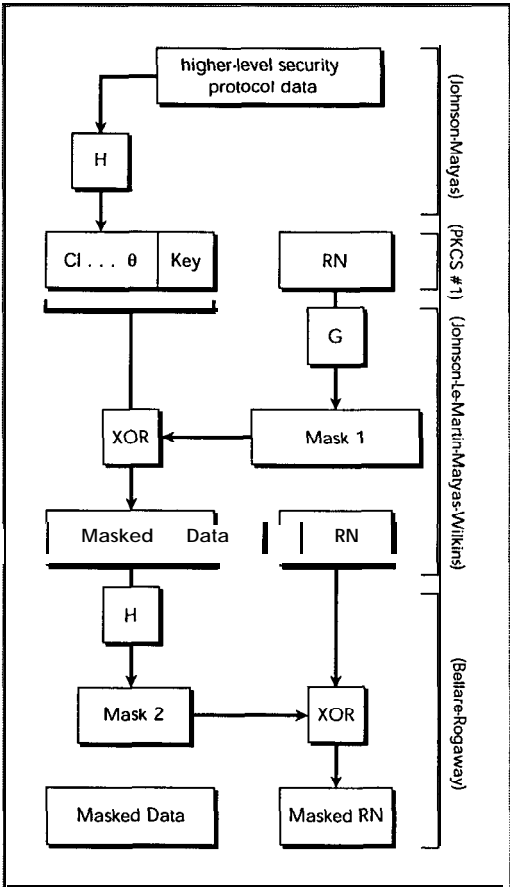


Figure 4.
Simplified
Johnson-Matyas
X9.44 draft
method

Ored with the rightmost 32 bits. Another way to perceive the improvement that the OAE method provided over the JIMW method is to realize that the JIMW method is analogous to a single round of DES encryption, while the OAE masking method is analogous to two rounds of DES encryption. That is, with the JIMW method, the left part (A) was masked using the right part (B); with the ER method, the left part (A) was masked using the right part (B) and the right part (B) was masked using the masked left part (masked A).

2 Non-malleability was defined in a paper by Dolev, Dwork, and Naor [5]. Informally, a cryptosystem is non-malleable if it is no easier to determine some plaintext A in some relation to some unknown plaintext B if one knows the ciphertext of B than if one does not. An example given in the paper uses two bidders encrypting their bids. It should not be possible for one bidder to see the encrypted bid of the other and somehow be able to offer a bid that was slightly lower, even if he would not know what the resulting bid actually was.

Further Enhancements

The recent emergence of practical cryptosystems based on elliptic curves has resulted in other potential refinements. These are mainly because elliptic curve cryptographic algorithms using a blocksize of about 155 bits are thought to be strong; however, this means that the abundance of space in a block that allows the ability to store lots of data like a symmetric key, a reverse signature, and a random number when using larger blocksize is severely limited. The following are some ideas to help address this and are under consideration by the IEEE P1363 workgroup:

1. The Feistel ladder can be balanced, rather than unbalanced. That is, the dividing line between the two sides for the masking process operating on the unformatted block could be put right in the middle. By symmetry considerations, a balanced construct is preferable from a security viewpoint as it ensures optimal distribution of the random secret data throughout the block. There is no "funnel" in which secrecy may be reduced. It may also be easier to implement. Also, by freeing the dividing line between the two parts from the possibly arbitrary lengths of certain fields, a reduction in overall length may be able to be achieved.
2. The random number is used for 2 purposes, to ensure uniqueness and to help thwart dictionary attacks. However, use of a random number has a concern in that the uniqueness is probabilistic, not guaranteed. Also the random number must be long enough so that the chance of a duplicate is sufficiently small. If the symmetric key being sent is already large enough to thwart dictionary attacks, then the random number may be able to be replaced with a counter or some other way to ensure uniqueness of the block. Not only will this result in a smaller size than the use of a random number, uniqueness can be guaranteed.
3. One may consider doing more than two rounds of masking to achieve some desired property.

Conclusion

In this article we have presented the increases in security and capability that have occurred when using asymmetric encryption. The advantageous attributes of these ideas are such that we believe they will be able to be used in a wide variety of cryptographic solutions. ■

Acknowledgments

We thank Mihir Bellare for helping us with the ANSI X9.44 proposal to ensure it incorporated ideas in "Optimal Asymmetric Encryption." We thank Phillip Rogaway for confirming that using a hash function on public information ~~combined~~ with the secret key supports security enhancements over the basic OAE method.

References

- [1] W. Aled, B.Z. Chor, O. Goldreich, and C.P. Schnorr, RSA and Rabin Functions: Certain Parts are as Hard as the Whole, *SIAM Journal on Computing*, 17(2), 194-209, April 1988.
- [2] Draft ANSI X9.44 RSA Key Transport standard, October 1994.
- [3] M. Bellare and P. Rogaway, Optimal Asymmetric Encryption - How to Encrypt with RSA, in A. De Santis, ed., *Advances in Cryptology - Eurocrypt '94 Proceedings*, volume 950 of *Lecture Notes in Computer Science*, Springer-Verlag, New York, 1994.
- [4] D. Coppersmith, Finding a Small Root of a Univariate Modular Equation, *IBM Research Report RC 20223*, October 11, 1995; revised November 8, 1995.
- [5] D. Dolev, C. Dwork, M. Naor, Non-Malleable Cryptography, *Proceedings of the 23rd ACM Symposium on Theory of Computing*, 1991.
- [6] J. Hastad, Solving simultaneous modular equations, *SIAM Journal on Computing*, 17(2), 336-341, April 1988.
- [7] Common Cryptographic Architecture: Cryptographic Application Programming Interface - Public Key Algorithm, IBM publication SC40-1676, April 1993.
- [8] D. Johnson, A. Le, W. Martin, S. Matyas, and J. Wilkins, Hybrid key distribution scheme giving key recovery, *IBM Technical Disclosure Bulletin*, 37(2A), 5-16, February 1994.
- [9] B.S. Kaliski and M.J.B. Robshaw, The Secure Use of RSA, *RSA Laboratories' CryptoBytes*, 1:3, 7-13, Autumn 1995.
- [10] RSA Laboratories, Public Key Cryptography Standard #1: RSA Encryption Standard Version 1.5, November 1993.

Editor's Note: We consider this article by Johnson and Matyas to be particularly timely. Not only are a variety of efforts already underway for standardizing on a version of the optimal asymmetric encryption methods, but RSA Laboratories is also currently preparing to revise PKCS #1 - our public-key cryptography standard for RSA encryption and signatures. Progress on all these efforts will be reported in forthcoming issues of *CryptoBytes* and in other RSA Laboratories' publications.

PayWord and MicroMint (extended abstract)

Ronald L. Rivest
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139 USA

Adi Shamir
Applied Math Department
The Weizmann Institute of Science
Rehovot 76100, Israel

Many electronic payment schemes have been proposed in recent years to address the problem of secure transactions over open networks. Micropayment schemes are a special kind of payment schemes for applications in which each payment is very small. To support micropayments, exceptional efficiency is required; otherwise, the cost of the mechanism will exceed the value of the payments.

PayWord and MicroMint are two simple micropayment schemes. In both schemes, the players are brokers, users and vendors. Brokers authorize users to make micropayments to vendors and redeem the payments collected by the vendors (See Figure 1). Broker-user and broker-vendor relationships are long-term, while user-vendor relationships are transient.

Our micropayment schemes might be considered lightweight when compared with full payment schemes in the sense that our security goal is to keep honest people honest (a similar situation exists with newspaper vending machines): exceptionally small-scale fraud and abuse cannot be totally prevented. In return, however, we try to achieve the following efficiency goals:

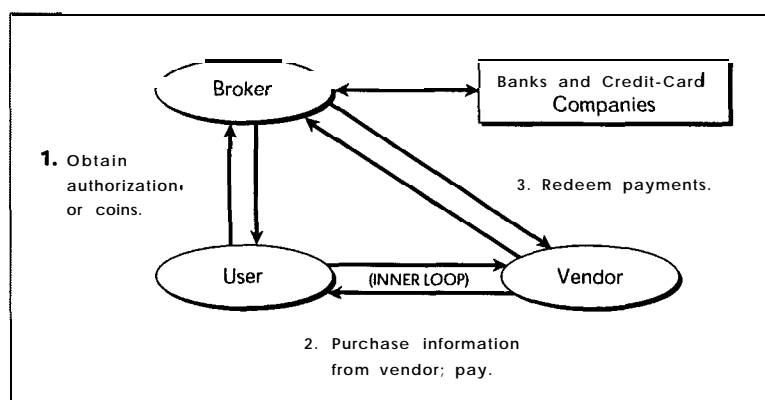
- Minimize the number of public-key operations by using hash operations whenever possible. (Roughly, hash functions are about 100 times faster than RSA signature verification and about 10,000 faster than RSA signature generation.)
- Minimize communication (particularly on-line

communication) with brokers. Since there may be only a few nationwide brokers, it is important that their computational burden be both reasonable and "off-line."

- Make the communication per transaction efficient, especially for repeated small purchases.

PayWord is a credit-based scheme and is optimized for sequences of micropayments. It is also secure and flexible enough to support larger variable-value payments as well. MicroMint introduces a new paradigm for electronic coins and is designed to eliminate public-key operations altogether. It has lower security but higher efficiency.

...our security goal is to keep honest people honest...



PayWord

Roughly speaking, PayWord consists of the following components:

1. A user establishes an account with a broker, who issues the user a digitally-signed certificate. This authorizes the user to make payword chains, which are chains of hash values.
2. Before contacting a vendor, the user creates a vendor-specific payword chain of n links.
3. The user authenticates the complete chain to the vendor with a single public-key signature, and then successively reveals each payword in the chain to make each micropayment.
4. The vendor redeems the paywords received from the user with the original broker.

Payword Chains

In the construction of a payword chain, we will use a one-way hash function, such as MD5 [6]. The hash function h has the property that, given a hash value y it is hard to find an input x , such that $y=h(x)$.

Figure 1.
Broker/Vendor/
User Relationship

Ron Rivest is associate director of MIT's Laboratory for Computer Science. He can be contacted at rivest@theory.lcs.mit.edu. Adi Shamir is professor at the Applied Math Department of the Weizmann Institute of Science, Israel, and can be contacted at shamir@wisdom.weizmann.ac.il. Rivest and Shamir are co-inventors of the RSA cryptosystem.

This article is edited from the full paper [7] by Yiqun Lisa Yin. She is a research scientist at RSA Laboratories, and can be reached at yiqun@rsa.com.

A user creates a payoff chain $\{w_0, w_1, w_2, \dots, w_n\}$ by picking the final payoff w_n at random and computing payoffs according to $w_i = h(w_{i+1})$. The first payoff w_0 is called the "root" of the chain. Given w_0 it is hard for anyone other than the user to figure out the rest of the payoffs in the chain. By revealing w_0 , however, the user is "committed" to the payoff chain she just created.

User-Broker relationship

A user begins a relationship with a broker by requesting an account and a PayWordCertificate. The user first gives the broker over a secure authenticated channel: her credit-card number, her public key PK_u and her "delivery address" (e.g., IP-address). The broker then issues the user a digitally signed certificate which authorizes her to make payoff chains until a given expiration date, and authorizes the delivery of goods only to the specified delivery address.

The user's certificate C_u thus has the following form:

$$C_u = \{broker, user, user's\ delivery\ address, PK_u, expiration\ date, other\ info\}_{SK_B}$$

where $\{ \}_{SK_B}$ denotes that the contents of $\{ \}$ are signed with the broker's private key SK_B . The PayWord certificate C_u is a statement by the broker to any vendor that authentic payoffs produced by the user and used before the expiration date will be redeemed.

User-Vendor relationship

User-vendor relationships are transient. A user might visit a web site, purchase ten pages, and then move on elsewhere. When the user is about to contact a new vendor, she computes a fresh payoff chain $\{w_0, w_1, w_2, \dots, w_n\}$. Here n is chosen at the user's convenience; it could be ten or ten thousand. She then computes her commitment for that chain:

$$M = \{vendor, C_u, w_0, current\ date, other\ info\}_{SK_u}$$

The commitment M , which is signed with the user's private key SK_u , authorizes the broker to pay the vendor for any of the payoffs w_1, w_2, \dots, w_n redeemed on the current date. Paywords are vendor-specific and user-specific; they are of no value to another vendor. Upon receiving the commitment M , the vendor verifies the user's signature on M and the broker's signature on C_u (contained within M), and checks the

expiration dates. After the commitment, payment from a user to a vendor consists of a payoff and its index: (w_i, i) . The payment is not signed by the user.

The user spends her paywords in order: w_1 first, then w_2 , and so on. If each payoff is worth one cent and each web page costs one cent, then she discloses w_i to the vendor when she orders her i -th web page from the vendor that day. This leads to the PayWord payment policy: For each commitment a vendor is paid 1 cents, where (w_i, i) is the committing payment received with the largest index. This means that the vendor needs to store only one payment from each user: the one with the highest index. The broker can confirm the value to be paid for w_1 by determining how many applications of h are required to map w_1 into w_0 .

Vendor-Broker relationship

A vendor needn't have a prior relationship with a broker, but does need to obtain the public key of the broker in an authenticated manner, so he can authenticate certificates signed by the broker. He also needs to establish a way for the broker to pay him redeemed paywords.

At the end of each day (or other suitable period), the vendor sends the broker a redemption message giving, for each of the broker's users who have paid the vendor that day, the commitment C_u and the last payment $P = (w_i, i)$. The broker needs to first verify each commitment received by checking the user's signatures (since he can recognize his own certificates), and then verify each payment (w_i, i) by computing the hash function i times. The broker will normally honor all valid redemption requests.

Security

In a typical scenario for PayWord, each payoff represents a very small amount of money, such as one cent. After the commitment, the paywords do not need to be signed by the user, since the paywords are self-authenticating using the commitment. Such a feature may allow each payoff to represent a larger amount of money (e.g., software selling at \$19.99). Hence, the level of security and flexibility of PayWord makes it possible to support certain "macro-payment" as well.

In PayWord, the contents of a payment does not specify what item it is payment for. A vendor may

User-vendor relationships are transient. A user might visit a web site, purchase ten pages, and then move on elsewhere.



$$h(x_1) = h(x_2) = \dots = h(x_k) = y$$

for some n -bit string y . Note that each coin is a bit-string whose validity can be easily checked by anyone, but which is hard to produce. This is similar to the requirements for a public-key signature, but the complexity of the signature makes it an overkill for a transaction whose value is only one cent.

The process of minting coins

The process of computing $h(x)=y$ is analogous to tossing a ball x at random into one of 2^n bins; the bin which ball x ends up in is the one with index y . A coin is thus a set of k balls that have been tossed into the same bin. Getting k balls into the same bin requires tossing a substantial number of balls altogether, since balls can not be "aimed". To mint coins, the broker will create 2^n bins, toss approximately $k2^n$ balls, and create one coin from each bin that now contains at least k balls. In this way each ball has a chance of roughly $1/2$ of being part of a coin.

A small problem in this basic picture, however, is that computation is much cheaper than storage. The number of balls that can be tossed into bins in a long computation far exceeds the number of balls that can be memorized on a reasonable number of hard disks. We thus propose to make most balls unusable as part of a coin, in a manner that depends on the hash value y . To do so, we say that a ball x is "good" if the high-order t bits of the hash value y have a value z specified by the broker. More precisely, let $n = t + u$. If the high-order t bits of y are equal to z , then y is called "good," and the low-order u bits of y determine the index of the bin in which the (good) ball x is tossed.

A proper choice of t enables us to balance the computational and storage requirements of the broker. This slows down the generation process by a factor of 2^t , without slowing down the verification process. The broker thus tosses approximately $k2^n$ balls, memorizes about $k2^u$ good balls that he tosses into 2^u bins, and generates from them about $(1/2) \approx 2^u$ valid coins.

A typical scenario

Here is a sketch of how a typical broker might choose the parameters (k, u, t, n) and make investment in hardware. We suppose that the broker wishes to have a net profit of \$1 million per month,

and he charges a 10% brokerage fee to vendors for redemption. Thus, the broker needs to sell one billion coins (approximately 2^{30}) per month to collect his \$1 million fee. (This requires a customer base of 500,000 if an average user buys 2,500 coins for \$25 per month.)

The broker first chooses $k = 4$; a coin will be a good 4-way hash collision. He then chooses $u = 31$ (i.e., creates 2^{31} bins), since this will allow him to create about $(1/2) \approx 2^{31} = 2^{30}$ coins.

The broker chooses his hash function $h(x)$ as the encryption of some fixed value v with key x under DES, which can be implemented by so-called field-programmable gate array chips. Each chip costs about \$400 and computes 2^{25} values per second. Buying 2^8 of these chips costs the broker \$100,000 and allows him to compute about 2^{54} values per month. Since $k = 4$, the broker chooses $n = 52$ and $t = 21$. Storing all good pairs $(x, h(x))$ requires less than 2^{37} bytes, and costs less than \$40,000 using standard magnetic hard disk technology. Hence, the total cost for the broker's hardware is less than \$150,000, which is less than 15% of the first month's profit.

Selling coins, making payments, and redemption
At the beginning of each month, the broker either reveals a new hash function h or changes the value z for that month and sells coins to users. Such sales can be on a debit basis or a credit basis, since the broker can recognize coins when they are returned to him for redemption. In a typical purchase, a user might buy \$25.00 worth of coins (2500 coins), and charge the purchase to his credit card. The broker keeps a record of which coins each user bought. Unused coins are returned to the broker at the end of each month.

Each time a user purchases a web page, he gives the vendor a previously unspent coin (x_1, x_2, \dots, x_k) . The vendor verifies that it is indeed a good k -way collision by computing k hash values. The vendor returns the coins he has collected to the broker at the end of each day. If the coin has not been previously returned, the broker pays the vendor one cent (minus any brokerage fee) for each coin. If a coin is received more than once, the broker does not pay more than once. Which vendor gets paid can be decided arbitrarily or randomly by the broker.

Note that each coin is a bit-string whose validity can be easily checked by anyone, but which is hard to produce,

Security analysis

We believe that users and vendors will have little motivation to cheat in order to gain only a few cents; even if they do, the consequences are of no great concern. Our security mechanisms are thus primarily designed to discourage large-scale attacks, such as massive forgery or persistent double-spending.

Forgery: Can an adversary forge MicroMint coins? (Economically?) First, the computational difficulty of minting coins makes small-scale forgery not really a concern. (For the parameter choice given above, a forger would need to spend 80 years on a standard workstation just to generate the first coin). Second, large-scale forgers can be detected and countered. In particular, coins "expire" monthly, and the new hash function for each month is revealed only at the beginning of that month. (The broker works during May to make coins good for June; forger only learns the hash function at the beginning of June and so starts out way behind.) Additional protection against forgery may be obtained by restricting coins to satisfy "hidden predicates" which are only announced if forgery is detected by the broker. For example, legitimate coins may all satisfy condition that the low-order bit of x_1 is equal to some complicated function of other bits. Forger's coins will typically not pass this additional "verification condition".

Double-spending: What if a user "double-spends" his MicroMint coins? There is no "anonymity" in MicroMint: the broker keeps track of whom each coin was sold to and notes when it is returned by vendor. Small-scale double-spending is not a concern. A user whose coins are consistently double-spent will be caught and black-listed; he will not be sold any more MicroMint coins.

Vendor fraud: What if a vendor gives copies of coins received to an accomplice? Vendors who consistently redeem coins that are also redeemed by other vendors will be black-listed and refused further redemption service by the broker. Users might cooperate with the broker to identify bad vendors by identifying where coins were first spent.

Theft of coins. If theft of coins is judged to be a problem during initial distribution to users or during redemption by vendors, it is easy to transmit coins

in an encrypted form. Since user/broker and vendor/broker relationships are relatively stable, long-term encryption keys (e.g., DES keys) can be arranged.

To prevent theft of coins as they are being transferred from user to vendor, we can make coins user-specific so that they are of no value to other users. To produce such coins, we generalize the notion of a "collision" to more complicated combinatorial structures which are related to the identity of a user and can be easily checked during verification. The user-specific coins can be further modified so that they are vendor-specific as well, making a stolen coin even less desirable. Details of these extensions are in the full version of the paper [7].

Conclusions

In this article we have presented two new micropayment schemes which are exceptionally economical in terms of the number of public-key operations employed. Furthermore, both schemes are off-line from the point of view of the broker. The area of micropayments has attracted considerable attention recently, and several researchers have proposed related schemes (Millicent [4], NetBill [2], NetCard [1], Pedersen's tick payments [5], and a micropayment scheme based on iKP [3], etc). More details about our schemes and the relationships between the various proposals are described and analyzed in the full version of this extended abstract [7].

References

- [1] R. Anderson, H. Maniavas, and C. Sutherland. NetCard - A practical electronic cash systems. 1996. Available from author: Ross.Anderson@cl.cam.ac.uk.
- [2] B. Cox, J.D. Tygar, and M. Sirbu. NetBill security and transaction protocol. <http://www.ini.cmu.edu/netbill/home.html>.
- [3] R. Hauser, M. Steiner, and M. Waidner. Micro-payments based on iKP. January 1996. <http://www.zurich.ibm.com:80/Technology/Security/extern/ecommerce/iKP.html>.
- [4] M. Manasse. The Millicent protocols for electronic commerce. 1995. <http://www.research.digital.com/SFC/millicent>.
- [5] T. Pedersen. Electronic payments of small amounts. Technical Report DAIMI PB-495, Aarhus University, Computer Science Department, August 1995.
- [6] R. Rivest. RFC 1321: The MD5 Message Digest Algorithm. RSA Data Security, Inc., April 1992.
- [7] R. Rivest and A. Shamir. PayWord and MicroMint - Two simple micropayment schemes. April 1996. <http://theory.lcs.mit.edu/~rivest>.

...users and vendors will have little motivation to cheat in order to gain only a few cents...

Our security mechanisms are thus primarily designed to discourage large-scale attacks, such as massive forgery or persistent double-spending.

The HMAC Construction

Mihir Bellare

Department of Computer Science & Engineering
Mail Code 0114, University of California at San Diego
9500 Gilman Drive, La Jolla, CA 92093 USA

Ran Canetti

Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139 USA

Hugo Krawczyk

IBM T.J. Watson Research Center, PO Box 704
Yorktown Heights, New York 10598 USA

There has recently been a lot of interest in the subject of authenticating information using cryptographic hash functions like MD5 and SHA-1, particularly for Internet security protocols. We report on our HMAC construction [1] which seems to be gaining acceptance as a solution.

Introduction

Two parties communicating across an insecure channel need a method by which any attempt to modify the information sent by one to the other, or fake its origin, is detected. Most commonly such a mechanism is based on a shared key between the parties, and in this setting is usually called a MAC, or Message Authentication Code. (Other terms include Integrity Check Value or Cryptographic Checksum). The sender appends to the data D an authentication tag computed as a function of the data and the shared key. At reception, the receiver recomputes the authentication tag on the received message using the shared key, and accepts the data as valid only if this value matches the tag attached to the received message.

The most common approach is to construct MACs from block ciphers like DES. Of such constructions the most popular is the CBC MAC. (Its security is analyzed in [4,12]). More recently, however, people have suggested that MACs might be constructed from cryptographic hash functions like MD5 and

SHA-1. There are several good reasons to attempt this: In software these hash functions are significantly faster than DES; library code is widely and freely available; and there are no export restrictions on hash functions.

Thus people seem agreed that hash function based constructions of MACs are worth having. The more difficult question is how best to do it. Hash functions were not originally designed for message authentication. (One of many difficulties is that they are not even keyed primitives, i.e., do not accommodate naturally the notion of a secret key). Several constructions were proposed prior to HMAC, but they lacked a convincing security analysis.

The HMAC construction is intended to fill this gap. It has a performance which is essentially that of the underlying hash function. It uses the hash function in a black box way so that it can be implemented with available code, and also replacement of the hash function is easy should need of such a replacement arise due to security or performance reasons. Its main advantage, however, is that it can be proven secure provided the underlying hash function has some reasonable cryptographic strengths. The security features can be summarized like this: if HMAC fails to be a secure MAC, it means there are sufficient weaknesses in the underlying hash function that it needs to be dropped not only from this particular usage but also from a wide range of other popular usages to which it is now subject.

Several articles in the literature survey existing constructions, their properties, and some of their weaknesses, so we will not try to do this again here. In particular the reader is referred to Tsudik [17], who provides one of the earliest works on the subject; Kaliski and Robshaw who, in the first CryptoBytes [8], compare various possible constructions; updates appearing in succeeding issues of CryptoBytes; and Preneel and van Oorschot [12,13], who present a detailed description of the effect of birthday attacks on "iterated constructions" and also a new construction called MDx-MAC.

We now move on to discuss the HMAC construction, status, and rationale. For a complete description, implementation guidelines, and detailed analysis we refer the reader to [1,9].

Mihir Bellare is an assistant professor at UCSD. He can be contacted at mihir@cs.ucsd.edu. Ran Canetti is a post-doctoral fellow at MIT and can be contacted at canetti@theory.lcs.mit.edu. Hugo Krawczyk is a member of the Network Security Group at IBM. He can be contacted at hugo@watson.ibm.com.

..people seem agreed that hash function based constructions of MACs are worth having. The more difficult question is how best to do it.

HMAC

Let H be the hash function. For simplicity of description we may assume H to be MD5 [15] or SHA-1 [16]; however the construction and analysis can be applied to other functions as well (see below). H takes inputs of any length and produces an l -bit output ($l=128$ for MD5 and $l=160$ for SHA-1). Let Text denote the data to which the MAC function is to be applied and let K be the message authentication secret key shared by the two parties. (It should not be larger than 64 bytes, the size of a hashing block, and, if shorter, zeros are appended to bring its length to exactly 64 bytes.) We further define two fixed and different 64 byte strings ipad and opad as follows (the "i" and "o" are mnemonics for inner and outer):

ipad = the byte 0x36 repeated 64 times
 opad = the byte 0x5C repeated 64 times.

The function HMAC takes the key K and Text , and produces:

$$\text{HMAC}_K(\text{Text}) = H(K \oplus \text{opad}, H(K \oplus \text{ipad}, \text{Text}))$$

Namely,

1. Append zeros to the end of K to create a 64 byte string
2. XOR (bitwise exclusive-OR) the 64 byte string computed in step (1) with ipad
3. Append the data stream Text to the 64 byte string resulting from step (2)
4. Apply H to the stream generated in step (3)
5. XOR (bitwise exclusive-OR) the 64 byte string computed in step (4) with opad
6. Append the H result from step (5) to the 64 byte string resulting from step (5)
7. Apply H to the stream generated in step (6) and output the result

The recommended length of the key is at least 1bits. A longer key does not add significantly to the security of the function, although it may be advisable if the randomness of the key is considered weak.

HMAC optionally allows truncation of the final output say to 80 bits.

As a result we get a simple and efficient construction. The overall cost for authenticating a stream

Text is close to that of hashing that stream, especially as Text gets large. Furthermore, the hashing of the padded keys can be precomputed for even improved efficiency.

Note HMAC uses the hash function H as a black box. No modifications to the code for H are required to implement HMAC. This makes it easy to use library code for H , and also makes it easy to replace a particular hash function, such as MD5, with another, such as SHA-1, should the need to do this arise.

HMAC was recently chosen as the mandatory-to-implement authentication transform for the Internet security protocols being designed by the IPSEC working group of the IETF (it replaces as a mandatory transform the one described in [10]). For this purpose HMAC is described in the Internet Draft [9], and in an upcoming RFC. Other Internet protocols are adopting HMAC as well (e.g., s-http [14], SSL [7]).

The rationale

We now briefly explain some of the rationale used in [1] to justify the HMAC construction.

As we indicated above, hash functions were not originally designed to be used for message authentication. In particular they are not keyed primitives, and it is not clear how best to "key" them. Thus, one ought to be quite careful in using hash functions to build MACs.

The standard approach to security evaluation is to look for attacks on a candidate MAC construction. When practical attacks can be found, their effect is certainly conclusive: the construction must be dropped. The difficulty is when attacks are not yet found. Should one adopt the construction? Not clear, because attacks might be found in the future.

The maxim that guided the HMAC construction was that an absence of attacks today does not imply security for the future. A better way must be found to justify the security of a construction before adopting it.

You can't make good wine from bad grapes: if no strengths are assumed of the hash function, we can't hope to justify any construction based on it. Accordingly it is appropriate to make some assumptions on the strength of the hash function.

HMAC uses the hash function H as a black box. No modifications to the code for H are required to implement HMAC.

You can't make good wine from bad grapes: if no strengths are assumed of the hash function, we can't hope to justify any construction based on it.

A well justified MAC construction, in our view, is one under which the security of the MAC can be related as closely as possible to the (assumed) security properties of the underlying hash function.

A well justified MAC construction, in our view, is one under which the security of the MAC can be related as closely as possible to the (assumed) security properties of the underlying hash function.

The assumptions on the security of the hash function should not be too strong, since after all not enough confidence has been gathered in current candidates (like MD5 or SHA-1). In fact, the weaker the assumed security properties of the hash function, the stronger the resultant MAC construction is.

We make assumptions that reflect the more standard existing usages of the hash function. The properties we require are mainly collision-freeness and some limited "unpredictability." What is shown is that if the hash function has these properties the MAC is secure; the only way the MAC could fail is if the hash function fails.

In fact the assumptions we make are in many ways weaker than standard ones. In particular we require only a weak form of collision-resistance. Thus it is possible that H is broken as a hash function (for example collisions are found) and yet HMAC based on H survives.

A closer look

Security of the MAC means security against forgery. The MAC is considered broken if an attacker, not having the key K , can find some text Text together with its correct MAC value $\text{HMAC}_K(\text{Text})$. The attacker is assumed able to gather some number of example pairs of texts and their valid MACs by observing the traffic between the sender and the receiver. Indeed the adversary is even allowed a chosen message attack under which she can influence the choice of messages for which the sender computes MACs. Following [4,3] we quantify security in terms of the probability of successful forgery under such attacks.

The analysis of [1] applies to hash functions of the iterated type, a class that includes MD5 and SHA-1, and consists of hash functions built by iterating applications of a compression function f according to the procedure of Merkle [11] and Damgård [5]. (In this construction an l -bit initial variable IV is fixed, and the output of H on text x is computed by breaking x into 512-bit blocks and hashing in stages using

f , in a simple way that the reader can find described in many places, e.g. [8].)

Roughly what [1] says is that an attacker who can forge the HMAC function can, with the same effort (time and collected information), break the underlying hash function in one of the following ways:

- 1. The attacker finds collisions in the hash function even when the IV is random and secret, or
- 2. The attacker is able to compute an output of the compression function even with an IV that is random, secret and unknown to the attacker. (That is, the attacker is successful in forging with respect to the application of the compression function secretly keyed and viewed as a MAC on fixed length messages.)

The feasibility of any of these attacks would contradict some of our basic assumptions about the cryptographic strength of these hash functions. Success in the first of the above attacks means success in finding collisions, the prevention of which is the main design goal of cryptographic hash functions, and thus can be assured hard to do. But in fact, even more is true: success in the first attack above is even harder than finding collisions in the hash function, because finding collisions when the IV is secret (as is the case here) is far more difficult than finding collisions in the plain (fixed IV) hash function. This is because the former requires interaction with the legitimate user of the function (in order to generate pairs of input/outputs from the function), and disallows the parallelism of traditional birthday attacks. Thus, even if the hash function is not collision-free in the traditional sense, our schemes could be secure.

Some "randomness" of hash functions is assumed in their usage for key generation and as pseudo-random generators. (For example the designers of SHA-1 suggested that SHA-1 be used for this purpose [6].) Randomness of the function is also used as a design methodology towards achieving collision-resistance. The success of the second attack above would imply that these randomness properties of the hash functions are poor.

The analyses in [1] used to establish the above are exact (no asymptotics involved), consider generic

rather than particular attacks, and establish a tight relationship between the securities.

Resistance to known attacks

As shown in [12,2], birthday attacks, that are the basis to finding collisions in cryptographic hash functions, can be applied to attack also keyed MAC schemes based on iterated functions (including also CBC-MAC, and other schemes). These attacks apply to most (or all) of the proposed hash-based constructions of MACs. In particular, they constitute the best known forgery attacks against the HMAC construction.

Consideration of these attacks is important since they strongly improve on naive exhaustive search attacks. However, their practical relevance against these functions is negligible given the typical hash lengths like 128 or 160. Indeed, these attacks require the collection of the MAC value (for a given key) on about $2^{1/2}$ messages (where l is the length of the hash output). For values of $l \geq 128$ the attack becomes totally infeasible. In contrast to the birthday attack on key-less hash functions, the new attacks require interaction with the key owner to produce the MAC values on a huge number of messages, and then allow for no parallelization. For example, when using MD5 such an attack would require the authentication of 2^{64} blocks (or 2^{73} bits) of data using the same key. On a 1 Gbit/sec communication link, one would need 250,000 years to process all the data required by such an attack. This is in sharp contrast to birthday attacks on key-less hash functions which allow for far more efficient and close-to-realistic attacks [18].

References

[1] M. Bellare, R. Canetti and H. Krawczyk. Keying hash functions for message authentication. *Advances in Cryptology – Crypto 96 Proceedings*, Lecture Notes in Computer Science, N. Koblitz ed., Springer-Verlag, 1996.

[2] M. Bellare, R. Canetti and H. Krawczyk. Pseudorandom functions revisited: The cascade construction. Manuscript, April 1996.

[3] M. Bellare, R. Guérin and P. Rogaway. XOR MACs: New methods for message authentication using finite pseudorandom functions. *Advances in Cryptology – Crypto 95 Proceedings*, Lecture Notes in Computer Science Vol. 963, D. Coppersmith ed., Springer-Verlag, 1995.

[4] M. Bellare, J. Kilian and P. Rogaway. The security of cipher block chaining. *Advances in Cryptology – Crypto 94 Proceedings*, Lecture Notes in Computer Science Vol. 839, Y. Desmedt ed., Springer-Verlag, 1994.

[5] I. Damgård. A design principle for hash functions. *Advances in Cryptology – Crypto 89 Proceedings*, Lecture Notes in Computer Science Vol. 435, G. Brassard ed., Springer-Verlag, 1989.

[6] National Institute for Standards and Technology. Digital Signature Standard (DSS). *Federal Register*, Vol. 56, No. 169, August, 1991.

[7] A.O. Freier, P. Karlton, and P.C. Kocher. The SSL Protocol – Version 3.0. Internet draft draft-freier-ssl-version3-01.txt, March 1996.

[8] B. Kaliski and M. Robshaw. Message Authentication with MD5. *RSA Labs' CryptoBytes*, Vol. 1 No. 1, Spring 1995.

[9] H. Krawczyk, M. Bellare and R. Canetti. HMAC-MD5: Keyed-MD5 for Message Authentication. Internet draft draft-ietf-ipsec-hmac-md5.txt.00, March 1996.

[10] P. Metzger and W. Simpson. IP Authentication using Keyed MD5. IETF Network Working Group, RFC 1828, August 1995.

[11] R. Merkle. One way hash functions and DES. *Advances in Cryptology – Crypto 89 Proceedings*, Lecture Notes in Computer Science Vol. 435, G. Brassard ed., Springer-Verlag, 1989. (Based on unpublished paper from 1979 and his Ph.D thesis, Stanford, 1979).

[12] B. Preneel and P. van Oorschot. MD-x MAC and building fast MACs from hash functions. *Advances in Cryptology – Crypto 95 Proceedings*, Lecture Notes in Computer Science Vol. 963, D. Coppersmith ed., Springer-Verlag, 1995.

[13] B. Preneel and P. van Oorschot. On the security of two MAC algorithms. *Advances in Cryptology – Eurocrypt 96 Proceedings*, Lecture Notes in Computer Science, U. Maurer ed., Springer-Verlag, 1996.

[14] E. Rescorla and A. Schiffman. The Secure HyperText Transfer Protocol. Internet draft draft-ietf-ws-http-01.txt, February 1996.

[15] R. Rivest. The MD5 message-digest algorithm. IETF Network Working Group, RFC 1321, April 1992.

[16] FIPS 180-1, Secure Hash Standard. Federal Information Processing Standard (FIPS), Publication 180-1, National Institute of Standards and Technology, US Department of Commerce, Washington D.C., April 1995.

[17] G. Tsudik. Message authentication with one-way hash functions. *Proceedings of Infocom '92*.

[18] P. van Oorschot and M. Wiener. Parallel Collision Search with Applications to Hash Functions and Discrete Logarithms. *Proceedings of the 2nd ACM Conf., Computer and Communications Security*, Fairfax, VA, November 1994.

... birthday attacks . constitute the best known forgery attacks against the HMAC construction. (. .) However, for reasonable hash functions, their practical relevance is negligible.

A N N O U N C E M E N T S

The 1996 RSA Laboratories Seminar Series

RSA Laboratories is pleased to announce preliminary details of the 1996 RSA Laboratories Seminar Series which will be held on August 15 and 16 in Palo Alto, California. (The annual Crypto conference will be held the following week in Santa Barbara, California.)

The aim of the Seminar Series, as in previous years, is to provide a bridge between academics who devise and analyze cryptographic technology and practitioners who are involved in actually implementing and integrating this technology into products.

This year we intend to use the Seminar Series to provide a snapshot of the state of cryptographic research and its relation to the practical world. Invited speakers will address issues in a variety of cryptographic fields and provide an assessment of what the future might hold.

Sessions will be informal and relaxed and attendees, who might perhaps see a different side to cryptographic technology, will be encouraged to ask questions and comment on the direction of current research.

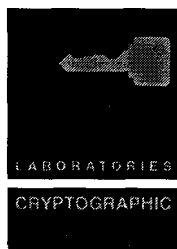
It is not often clear how much interaction takes place across what can sometimes be a significant division between academic specialists and the practitioners of cryptographic techniques. Our aim is to provide a forum within which both parties involved in the development of cryptography can meet to exchange ideas and views on both current knowledge and the potential for future advances.

More complete details about the 1996 RSA Laboratories Seminar Series will be available soon and made available on the World-Wide Web at <http://www.rsa.com/rsalabs/>. In the meantime more information can be obtained by contacting RSA Laboratories by any of the means given on the inside front cover. ■■

in this issue:

- Asymmetric Encryption: Evolution and Enhancements
- PayWord and MicroMint
- The HMAC Construction

For contact and distribution information, see page 2 of this newsletter.



100 MARINE PARKWAY
REDWOOD CITY
CA. 94065-1031
TEL 415/595-7703
FAX 415/595-4126
rsa-labs@rsa.com

PRESORT
FIRST CLASS
U.S. POSTAGE
PAID
MMS, INC

RSA LABORATORIES' CryptoBytes

The technical newsletter of RSA Laboratories, a division of RSA Data Security, Inc.

Contents

1

RSA for Paranoids

2

Editor's Note

4

Algorithms Update

7

The Secure Use of RSA

14

Frequently Asked
Questions

16

Announcements

RSA for Paranoids

Adi Shamir

Applied Math Department
The Weizmann Institute of Science
Rehovot 76100, Israel

One of the most important decisions in practical implementations of the RSA cryptosystem is the choice of modulus size. It is clear that the standard size of 512 bits no longer provides adequate protection, and should be substantially increased. However, the time complexity of modular exponentiation grows rapidly with the size of the modulus, and thus it is difficult to choose a size which combines efficient operation with long term security. In this note we describe a new variant of the RSA cryptosystem called "unbalanced RSA", which makes it possible to increase the modulus size from 500 bits to 5,000 bits without any speed penalty.

Conventional RSA

The security of the RSA cryptosystem depends (but is not provably equivalent to) the difficulty of factoring the modulus n , which is the product of two equal size primes p and q . Until recently, the minimum recommended size of n was 512 bits. However, recent advances in factoring algorithms make it necessary to increase this size. Since the time complexity of RSA computations is already substantial, and grows cubically with the size of the modulus, the new size should by necessity be a compromise between efficiency and security. The choice is particularly difficult for paranoid organizations

whose encrypted messages should remain secret for several decades, since it is almost impossible to predict the progress of factoring algorithms over such a long period of time. The only reasonable course of action is to use huge margins of safety, but this will make the RSA operations extremely slow.

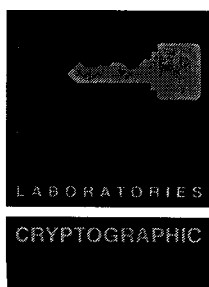
All the known factoring algorithms can be divided into two broad types: algorithms whose running time depends on the size of the factors, and algorithms whose running time depends only on the size of the factored number n . The oldest factoring algorithms typically searched for the smallest factor p of n , and were thus of the first type. However, modern algorithms tend to use indirect approaches which require the same time to find a single digit or a fifty digit prime factor of n .

The fastest factoring algorithm of the first type is currently the elliptic curve method. Its asymptotic running time is $\exp(O((\ln(p))^{1/2} \cdot (\ln \ln(p))^{1/2}))$, but its basic operations are very slow. The largest factor ever found in practice with this algorithm was about 145 bits long (A. Lenstra, private communication), and it is very unlikely that this algorithm will be able to find the 256 bit factors of 512 bit RSA keys in the next few years.

Factoring algorithms of the second type are much faster, since they can use a wider array of mathematical techniques. The best algorithm of this type is currently the general number field sieve. It has an asymptotic complexity of $\exp(O((\ln(n))^{1/3} \cdot (\ln \ln(n))^{2/3}))$, and is believed to be capable of factoring 512 bit moduli in 10,000 to 15,000 MIPS-years (A. Lenstra, private communication). This is

(continued on page 3)

Adi Shamir is professor at the Applied Math Department of the Weizmann Institute of Science, Israel. He is a co-inventor of the RSA cryptosystem and can be contacted at shamir@wisdom.weizmann.ac.il.



Editor's Note

In this, the autumn edition of *CryptoBytes*, we have decided to concentrate on issues related to the use of the RSA cryptosystem.

The main article in this issue is entitled The Secure Use of RSA. Adapted from a seminar presented by Ron Rivest at the 1995 RSA Laboratories Seminar Series, this article considers the myths and realities of using the RSA cryptosystem. From low exponent attacks to padding rules, this article provides the ground rules for avoiding a variety of potential attacks on RSA when used either to provide message encryption or to digitally sign documents.

We also have a variety of reports on several recent research results.


At most conferences, time is set aside for researchers to present their latest results outside of the main program, and consequently, outside of the conference proceedings. This means that it is often hard to track down the essential details of what was said. As initial steps to make information on rump session items more widely available, we present reports on three items that were presented at recent rump sessions.

A novel idea by Adi Shamir, leading to a rather unconventional implementation of RSA, has some very intriguing properties. In this issue of *CryptoBytes* we include an article by Adi Shamir which provides more details on ideas that were originally presented at the rump session of Eurocrypt '95 earlier this year.

We also have news of two items from the rump session of the Crypto '95 conference. We report on a new "protocol failure" when RSA is used with exponent three and also on a new attack on the envelope method of using a hash function for message authentication. (In one way or another, message authentication with hash functions has featured in all issues of *CryptoBytes* so far!) Also, as part of our ongoing features, we provide an Algorithms Update on some very exciting new work in the analysis of hash functions by Hans Dobbertin.

Finally, in this issue, we introduce a new column: Frequently Asked Questions. Extracted from the ongoing enhancements to RSA Laboratories' Answers to Frequently Asked Questions, this column will feature answers to new questions on today's cryptography that

are of particular interest to our readers. In this issue, the question "How do digital time-stamps support digital signatures?" is answered by Stuart Haber, Burt Kaliski and Scott Stometta.

The future success of *CryptoBytes* depends on input from all sectors of the cryptographic community and as usual we would very much like to thank the writers who have contributed to this third issue. We encourage any readers with comments, opposite opinions, suggestions, proposals for questions in our Frequently Asked Questions column or proposals for items in future issues to contact the *CryptoBytes* editor by any of the methods given below. 

Contact and Subscription Information

CryptoBytes will be available by subscription and individual annual subscription to four issues is U.S.\$90. To subscribe, contact RSA Laboratories at:

RSA Laboratories
100 Marine Parkway, Suite 500
Redwood City, CA 94065
415/595-7703
415/595-4126 (fax)
rsa-labs@rsa.com

We are hoping that electronic subscription to *CryptoBytes*, via the World-Wide Web, will be available soon. All back issues are available via the World-Wide Web at <http://www.rsa.com/rsalabs/cryptobytes/>.

The *CryptoBytes* editor can be contacted by any of the above methods, or by E-mail to bytes-ed@rsa.com.

About RSA laboratories

RSA Laboratories is the research and development division of RSA Data Security, Inc., the company founded by the inventors of the RSA public-key cryptosystem. RSA Laboratories reviews, designs and implements secure and efficient cryptosystems of all kinds. Its clients include government agencies, telecommunications companies, computer manufacturers, software developers, cable TV broadcasters, interactive video manufacturers, and satellite broadcast companies, among others.

At most conferences, time is set aside for researchers to present their latest results outside of the main program.... This means that it is **often** hard to track down the essential details of what was said.

RSA for Paranoids
Continued from page 1

only 2-3 times harder than the factorization of RSA-129 achieved (by a different algorithm) in early 1994, and is likely to be demonstrated within a year.

Since the inception of the RSA cryptosystem, all the record breaking factorizations of RSA keys were based on algorithms of the second type, and it is reasonable to assume that this trend will continue in the foreseeable future. If this is the case, there is no need to increase the sizes of n and its prime factors at the same rate. In this paper we use this observation in order to propose a new variant of the RSA cryptosystem, which we call "unbalanced RSA". Its goal is to provide much higher security at essentially no extra cost.

Unbalanced RSA

As a typical example of the new RSA variant, we describe an "ultimate security" implementation, which is likely to provide long term security even to professional paranoids. In this version we increase the overall size of n by a factor of 10 (to 5,000 bits), and the size of its prime factor p by a factor of 2 (to 500 bits). The size of the other factor q of n is 4,500 bits, and the name of this variant reflects the unequal sizes of the two prime factors of n .

Such a modulus is way out of reach of today's factoring algorithms, and is likely to remain secure even if factoring algorithms of the first type will become much better, and factoring algorithms of the second type will become enormously better. Only a major breakthrough in factoring (such as a practical polynomial time factoring algorithm, which will completely kill the RSA scheme) can possibly threaten it.

The main problem in using such a modulus n is the fact that standard RSA implementations become about 1000 times slower: A typical 500 bit exponentiation which required 1 second on a microprocessor would now require 16 minutes, which is unacceptable.

Since RSA encryption is typically used only in order to exchange session keys for fast secret key cryptosystems, the cleartexts are usually quite short: even three independent keys for triple DES require only 168 bits, and it is unlikely that anyone would use the RSA cryptosystem to exchange secret keys which are larger than p . We can thus assume that the cleartext is in the range $[0, p)$, and that it is padded with random bits to make sure that it is not much shorter than p .

A well known way of speeding up the RSA encryption process $c = m^e \pmod n$ is to use a small encryption exponent e . Since m is only one tenth the size of n , encryption exponents which are smaller than 10 do not provide any wraparound during the modular reduction, and should be avoided. When e is around 20, the size of m^e is about twice the size of n , and thus the wraparound effect is similar to the squaring operation of full size numbers in Rabin's scheme. Note that for such an e , most of the multiplications in the computation of $m^e \pmod n$ are non modular multiplications of relatively small numbers, and only the last squaring is likely to be a full size modular multiplication. The recommended range of e is 20-100, which makes the encryption time with a 5,000 bit modulus comparable to the decryption time with a 500 bit modulus (i.e., less than a second).

The other operation we have to consider is the decryption process $m = c^d \pmod n$ in which c , n and d are 5,000 bit numbers. If we use the Chinese remainder theorem, we can easily compute $m_1 = c^d \pmod p$ via a 500 bit exponentiation, by reducing c modulo p and d modulo $p-1$ at the beginning of the computation. However, we then have to carry out the 4,500 bit exponentiation $m_2 = c^d \pmod q$, which is $9^3=729$ times slower.

The main purpose of this note is to show that there is no need to carry out this expensive computation. By definition, m_1 is equal to $m \pmod p$. However, the cleartext m is known to be smaller than p , and thus $m \pmod p$ is simply m itself. By combining these observations, we conclude that m_1 is the original cleartext m , and thus it is just a waste of time to carry out the computation of m_2 modulo q , which will yield the same result.

Other optimizations

While the unbalanced RSA variant has about the same time complexity as the original variant, its space complexity and public key size are 10 times larger. We now show how to use additional optimization techniques to avoid these extra overheads.

Consider first the issue of RAM size. On personal computers the 10-fold increase in the size of the registers is meaningless, but smart cards contain very small RAMs and their cost is directly proportional to the size of their memory. In addition, many smart card manufacturers have already designed RSA

*In this paper . . .
[we] propose
a new variant
of the RSA
cryptosystem,
which we call
"unbalanced
RSA". Its goal
is to provide
much higher
security at
essentially
no extra cost.*

... the technique described in this paper can also be used to double the speed and halve the key size of standard implementations of the RSA cryptosystem in which p and q have similar sizes.

coprocessors which can handle 512 bit moduli, and would be reluctant to redesign them in order to accommodate much larger moduli. However, in most applications the smart card is talking to a powerful PC or workstation, and we can choose to implement either the encryption or the decryption side of the RSA key exchange on the smart card. It turns out to be easier to let the powerful machine choose and encrypt the session key. The arbitrarily long ciphertext is sent to the smart card in a bit serial mode, and the smart card reduces it on-the-fly modulo p by clocking it through the 512 bit input register of its coprocessor. The resultant value is then exponentiated by the 512 bit coprocessor to obtain the session key. The advantage of this approach is that today's coprocessors would be able to handle variable length moduli (dictated by the changing state of the art of factoring algorithms) without expensive periodic replacements of millions of smart cards in large-scale consumer applications.

Another potential drawback of the new variant is the lo-fold increase in the size of the public key directory. However, the unbalanced construction of the new public keys makes it possible to eliminate this problem. Let G be a publicly known pseudo random bit generator which maps each user identity u (name, email address, etc.) into a unique 5,000 bit pseudo random target value t . Each user picks a random 500-bit prime p , and defines the other prime q as a random prime number in the range $[a, a + 2^{50}]$ where a

is the smallest integer greater than or equal to t/p . Since the prime numbers are relatively dense, such a q almost certainly exists, and $n = pq$ is very close to the target value t (the difference $s = n - t$ is expected to be about 550 bits long). Each user u publishes this 550-bit s as his public key, and anyone can recover the user's 5,000-bit modulus n by computing $G(u) + s$.

It is important to note that this approach does not make n easy to factor: the only difference between our n and standard n is that we start the search for q at a point which is the ratio of a random number and a prime number, rather than at a random number. Since the distribution of prime numbers p is not completely uniform, the choice of a is not completely uniform. We smooth this slight nonuniformity by specifying a fairly large range of size 2^{50} in which q can be chosen, which makes the exact location of the range's endpoints irrelevant. Once n is chosen, it makes no difference whether we specify it by its bits or by its distance s from another number t , and the fact that t is a pseudo random rather than a truly random number cannot change this fact.

Finally, we note that the technique described in this paper can also be used to double the speed and halve the key size of standard implementations of the RSA cryptosystem in which p and q have similar sizes. The optimization does not apply to the RSA signature scheme, since the verifier cannot carry out computations modulo the unknown p . ■

A L G O R I T H M S U P D A T E

Collisions in MD4

Recent work by Hans Dobbertin has discovered that collisions for MD4 can be found within a few minutes on a typical PC. Even more impressive is the fact that collisions can be constructed, in around an hour, so that the text they represent makes sense. For an example, see how Alf swindles Ann opposite.

MD4 was among the first of a new generation of what are termed *dedicated hash functions*, designed from first principles to be hash functions rather than being based around some other primitive such as a block cipher. MD5, RIPEMD, the secure hash algo-

rithm SHA and its more recent revision SHA-1 follow the same design approach that was pioneered by Rivest with MD4. Rivest also designed an extended version of MD4 with a 256-bit rather than 128-bit hash value which was considered to be much more secure than MD4. It appears, however, that these new techniques will also compromise extended-MD4.

While MD4 remained secure until now, it was felt that MD4 made little allowance for potential cryptanalytic developments. Consequently, the use of MD4 has been discouraged for some considerable time, and instead the usual recommendation was to replace MD4 with the more conservatively designed

MD5. Dobbertin's work clearly confirms that if MD4 is still being used anywhere, then it should be immediately replaced.

As for the applicability of Dobbertin's techniques to other hash functions such as MD5, RIPEMD and

SHA- 1, the task facing the cryptanalyst is much more involved. Initial review of those algorithms suggests a limited opportunity for new developments (except on reduced-round versions of RIPEMD which have already succumbed to this type of analysis) and *Crypto-Bytes* will report on any further developments.

Alf Swindles Ann

Hans Dobbertin

German Information Security Agency
P.O. Box 20 10 63,
D-53 133 Bonn, Germany

Alf wanted to sell Ann his house, and Ann was interested. They agreed on a price of \$176,495. Alf asked Ann to sign a contract using a digital signature scheme which is based on some public-key algorithm and the hash function MD4. The contract read as follows:

CONTRACT

At the price of \$176,495 Alf Blowfish sells
his house to Ann Bonafide. . . .

"The first 20 bytes (each of them is represented by an asterisk above) are random. They have been placed before the text for security reasons!" claimed Alf, and Ann signed the contract. Later, however, Alf substituted the contract file by another which read as follows:

CONTRACT

At the price of \$276,495 Alf Blowfish sells
his house to Ann Bonafide. . . .

The contract had been prepared by him such that replacing \$176,495 by \$276,495 does not change the MD4 hash value!

How Alf did it

For the precise definition of the above digital contract note that the first sixteen 32-bit words are:

Professor Hans Dobbertin is particularly interested in the evaluation and design of cryptographic algorithms. He can be contacted at dobbertin@skom.rhein.de.

- $M_0 = 0x9074449b$
 $M_1 = 0x1089fc26$
 $M_2 = 0x8bf37fa2$
 $M_3 = 0x1d630daf$
 $M_4 = 0x63247e24$
 $M_5 = 0x4e4f430a$
 $M_6 = 0x43415254$
 $M_7 = 0x410a0a54$

$M_8 = 0x68742074$
 $M_9 = 0x72702065$
 $M_{10} = 0x20656369$
 $M_{11} = 0x2420666f$
 $M_{12} = 0x2c363731$
 $M_{13} = 0x20353934$
 $M_{14} = 0x20666c41$
 $M_{15} = 0x776f6c42$

The twenty bytes of M_0 - M_4 are the above mentioned "random bytes". The bytes of M_5 , in reverse ordering (according to the definition of MD4) and interpreted as ASCII read as follows:

Oa 43 4f 4e = Line-feed 'CON',

and so on to M_{15} which reads

42 6c 6f 77 = 'Blow'.

The sequence M_i ($i < 16$) has been chosen such that setting $M'_{12} = M_{12} + 1$ and $M'_i = M_i$ for $i < 16, i \neq 12$ gives a collision, i.e.

$$\text{compress}(IV; M) = \text{compress}(IV; M')$$

for the compress function of MD4 and its fixed initial value IV. In [1] the basic method of generating such collisions was described. They can be found in less than one hour on a PC. Interpreting $M_{12} = 0x2c363731$ and $M'_{12} = 0x2c363732$ we get:

$M_{12} = 31\ 37\ 36\ 2c = '176,'$
 $M'_{12} = 32\ 37\ 36\ 2c = '276,'$

In view of the definition of MD4 as the iterative application of compress, we obtain a collision by taking any bit string and appending it to M and M'.

Where MD4 is still in use, it should be replaced!

References

[1] Dobbertin, H.: *Cryptanalysis of MD4*, preprint.

Where MD4 is still in use, it should be replaced!

... the new attack does demonstrate a certification weakness in the envelope method...

More Developments with Keyed Hash Functions

At the rump session of Crypto'95, Bart Preneel in joint work with Paul van Oorschot described a new attack on one method of message authentication using a keyed hash function (see *CryptoBytes* volume 1, number 1).

In what is sometimes called the *envelope method*, a hash function H can be used to provide authentication of a message m under the action of two keys k_1 and k_2 by using the result of $H(k_1 \cdot m \cdot k_2)$ as a message authentication code for the message m . A variety of padding schemes might also be specified as additional security measures.

Previously (see *CryptoBytes* volume 1, number 2), Preneel and van Oorschot have reported that the envelope method of keyed-MD5 allows a MAC forgery attack with "2⁶⁴ known message-MAC pairs and a single chosen text." These known messages have to be the same length and additional reductions in the data requirements are possible if the messages are known to have a particular form.

At the rump session of Crypto '95, Preneel described a new attack on the envelope method which allows recovery of the secret key rather than just a MAC forgery. For the new attack to succeed, the length of the messages being authenticated is carefully chosen by the cryptanalyst to ensure that the action of the second key k_2 is split into two parts, one under the influence of k_a and the other under k_b , where k_2 is equal to the concatenation $k_a \cdot k_b$.


The attack requires two phases, the first of which is in effect the previously mentioned MAC-forgery attack. Using information obtained from this first phase, the cryptanalyst requests the MACs on 2^{| k_a |} pairs of chosen messages (2^{| k_a |+1} messages in total) where | k_a | denotes the number of bits in k_a . These new message pairs are chosen according to different guesses for k_a .

When the guess for k_a is correct, the MACs generated for that pair of messages will be the same and k_a can be correctly identified. The rest of k_2 can be found either by exhaustive search, or by using messages of a second carefully chosen length to split the unknown remainder of k_2 once again.

The increased work and data requirements for the new attack over that offering MAC-forgery depend on the length of k_a since this determines the number of guesses a cryptanalyst is required to try in the second phase. However, since the cryptanalyst can choose this length, the requirements of the attack are effectively dominated by the requirements for the MAC-forgery attack.

Since the MAC-forgery attack is barely practical the same must be said of this new attack which recovers the key. However the new attack does demonstrate a certification weakness in the envelope method since the secret key can be recovered with much less work than the length of the secret key might imply. While the use of additional padding (to prevent the splitting of k_2) would thwart this attack, it seems fair to recommend different mechanisms than the envelope method for providing message authentication with a keyed hash function. *CryptoBytes* will report on future developments.

A linear Protocol Failure for RSA With Exponent Three

At the rump session of Crypto '95, Matthew Franklin and Michael Reiter of AT&T Bell Laboratories presented a new weakness of RSA with low encrypting exponent. This weakness depends on two messages being encrypted with exponent three with respect to the same RSA modulus, and on a (non-trivial) linear relationship being known to exist between the two messages. In such cases the messages can be computed efficiently by an attacker that has access only to the public key, the two ciphertexts, and the linear relation. This differs from the "small exponent" protocol failure, generalized by Hastad (described opposite in The Secure Use of RSA), which requires that messages be encrypted with respect to different *moduli*. It also differs from the "common modulus" protocol failure, observed by Simmons, which requires that a single message be encrypted with different *exponents*. This weakness can be exploited to yield passive attacks on protocols that encrypt related messages. Examples of such protocols are a signature sharing protocol for RSA, proposed by Franklin and Reiter at Eurocrypt '95, and a key distribution protocol, proposed by Tatebayashi, Matsuzaki, and Newman at Crypto '89. This weakness does not affect the encryption of arbitrary unrelated messages. 

The Secure Use of RSA

Burt Kaliski and Matt Robshaw

RSA Laboratories
100 Marine Parkway, Suite 500
Redwood City, CA 940651031

Since its invention over fifteen years ago, the RSA cryptosystem [21] has been submitted to extensive scrutiny and analysis. Its continuing resistance to attack has resulted in its being well trusted and used widely.

One of the most notable features about RSA is its apparent simplicity and considerable elegance. This elegance is due in most part to the algebraic framework in which the algorithm is defined. But this framework can itself be a dangerous environment in which to practice cryptography. Indeed it is straightforward to write down a few mathematical identities that at first sight seem to threaten the integrity of RSA when used either for encryption or for the provision of digital signatures.

As we will see, however, for each of these apparent threats to RSA there is a simple and practical countermeasure. It is all a matter of using the technology correctly. In this article we will show how RSA can be used to its full potential.

The RSA Cryptosystem

RSA is a public-key cryptosystem and digital signature scheme that was invented in the late 1970s at MIT by Ron Rivest, Adi Shamir and Len Adleman [21]. The system depends on modular exponentiation and is defined as follows.

We will denote the public key by (n, e) and the private key by (n, d) . The integer n , which is known as the modulus, is chosen as the product of two primes P and q . The integers e and d are exponents and they are chosen so that $ed \equiv 1 \pmod{\phi(n)}$ where $\phi(n) = (p-1)(q-1)$.

Encryption and decryption of a message m can be described as follows. Using the public key (n, e) the sender of the message computes $c = m^e \pmod n$ and c

is the encrypted message. The legitimate receiver uses the private decryption key (n, d) to compute $c^d \pmod n$. The underlying mathematics guarantees that

$$c^d \equiv (m^e)^d \equiv m^{ed} \equiv m^1 \equiv m \pmod n$$

and the original message can be recovered.

Closely following the original Diffie-Hellman model for a public-key cryptosystem [8], the private and public keys in RSA can be used to provide a digital signature. Here the digital signature is generated using the private key (which is known to only one user) and verification of the signature can be completed with the corresponding public key. The signature s of message m with the private key (n, d) is given by $s = m^d \pmod n$ while public verification of the signature s can be achieved using (n, e) since the comparison

$$m \stackrel{?}{\equiv} s^e \pmod n$$

will hold for a legitimate digital signature and fail otherwise.

Both encryption and the creation of digital signatures are simple and straightforward operations. The mathematical structure underpinning the cryptosystem ensures that the link between these operations and their reverse can be easily established. But other links can be made and as motivation for what follows we will make note of two, easily established identities:

$$\begin{aligned} (m_1 \cdot m_2)^e &\equiv m_1^e \cdot m_2^e \pmod n \text{ and} \\ (m \cdot r^e)^d &\equiv m^d \cdot r \pmod n. \end{aligned}$$

We will see that exploitation of these identities and other properties can lead to attacks on both the generation of RSA signatures and on RSA encryption. In this article we will describe these attacks and some straightforward and practical counter-measures.

Aims of the Attacker

Three consequences of a successful attack on a public-key cryptosystem or digital signature scheme such as RSA might be:

1. recovery of the private key,
2. message decryption, and
3. signature forgery.

... for each of these apparent threats to RSA there is a simple and practical countermeasure. It is all a matter of using the technology correctly.

Burt Kaliski is chief scientist and Matt Robshaw is a research scientist at RSA Laboratories. They can be contacted at burt@rsa.com or matt@rsa.com.

It **is** generally accepted that *RSA numbers composed of primes of about the same size are among the hardest to factor...*

It may well be possible that an attack achieving the second or third result can be mounted without the attacker actually recovering the private key. In this article we consider attacks that might be used to achieve each of these goals in turn, and give rules of thumb which prevent the attacks being successful.

We note that in a practical implementation of RSA there are additional security considerations beyond those covered here. For instance, the interchangeability of signatures and decryption might lead to unexpected security weaknesses if the same key were used for both. Additionally, the source of randomness for key generation and the storage of the private key are vital issues in the secure implementation of RSA. Suffice it to say that no implementation of RSA can be considered fully secure unless all issues, including the system-dependent factors, are taken into account.

Recovering the Private Key

As a first line of attack, an adversary might take the public key (n, e) of some user and attempt to recover the private decrypting (or equivalently the signing) exponent d . Interestingly, one of the links that has been established between the RSA cryptosystem and the difficulty of factoring is based on a number-theoretic result predating RSA [16], which can be used to demonstrate that obtaining a private exponent d from the public exponent e and the modulus n is as hard as actually factoring n . In essence, knowledge of the exponents e and d allows the attacker to factor the modulus n .

With this connection in mind, we first consider the most well known (and currently the best) attack that an adversary might mount on RSA when trying to recover the private key, namely an attempt to factor the RSA modulus n .

Factoring

The problem of factoring and its continuing improvements are the topic of a great many articles and papers and we will not attempt to replicate the information that can be found in three, splendid articles [2,17,20].

It is generally accepted that RSA numbers composed of primes p and q of about the same size are among the hardest to factor for their size. (See, however,

the article by Adi Shamir in this issue of CryptoBytes on using RSA moduli with two primes that are widely differing in size!) For large enough numbers, and certainly for the size of numbers we use in today's implementations of RSA, the general number field sieve (GNFS) is the best "general-purpose" factoring method. The older quadratic *sieve* and its variants are faster below a certain size of modulus (currently around 116 decimal digits in length [2]). Most importantly however, users of RSA can determine the current level of factoring ability and make allowances for a certain amount of future improvement. As a consequence, general-purpose factoring need not be a concern to users of RSA if the primes (and hence the RSA modulus) are chosen to be sufficiently large.

Depending on the form of the primes p and q that are multiplied together to give the user's modulus, it might be argued that "special-purpose" factoring methods such as Pollard's p -f method [18] and super-encryption attacks [23], which we discuss next, might end up being faster than using the GNFS. With these and similar threats in mind, *strong primes* were introduced so that p and q were chosen to satisfy a variety of conditions; for instance p might be chosen so that $p-1$ has a large factor r and $r-1$ has a large factor, etc. These conditions, and similar ones on the form of the numbers $p+1$ and $q+1$ would guarantee that the modulus n resists the special-purpose factoring methods.

The introduction of the *elliptic curve* method (ECM) of factoring changed all this [15]. This factoring method has some probability of success regardless of the actual form of the prime. As a consequence, the dominant property for the security of the primes we use is size. In protecting against the ECM factoring technique one protects, with a very high probability, against all special-purpose factoring techniques. In short, large primes are more important than strong primes.

One early proposed attack against RSA is called *superencryption*. Simmons and Norris [23] observed that after a sufficient number of repeated *encryptions*, the original message would eventually be recovered. This would lead to an attack on RSA if the number of encryptions required were small. But this is not the case if the primes are large and chosen at random and so while an interesting observation, *super-encryption* is not a practical attack. Likewise, the

... strong primes were **introduced** so that [...] the modulus n resists the special-purpose factoring methods. The introduction of the elliptic curve method of factoring **all** this.

observation that p and q being very close together in numerical value allows the efficient factorization of n [25], is not relevant to practical security provided p and q are sufficiently large and chosen at random.

Wiener's Attack

There is an interesting attack due to Wiener [24] which allows an attacker to recover the private exponent in an ingenious way. Wiener shows that a continued fraction approximation using the publicly available parameters e and n can provide sufficient information to recover d . This attack will be efficient and practical when the private exponent d is small relative to the RSA modulus; that is when $d < n^{1/4}$.

This attack is a concern if the *private* exponent d is deliberately chosen to be small, perhaps in an attempt to improve the efficiency of decryption or signing. In general use however, it is unlikely that a small private exponent will be generated if the public exponent e is chosen first; when e is small, d is always large enough to resist this particular attack and if e is chosen at random, then with an overwhelming probability, d will be large enough to resist Wiener's attack. If the private exponent d is chosen, it should not be too small.

Using Probabilistic Primes

One frequent question is how the security of RSA might be affected if one of the primes used to compute the modulus is, in fact, not a prime number after all.

First, the factors of n will be smaller than expected, and so it may be easier to factor the modulus with special-purpose factoring methods. Second, except in the unlikely event that we have mistakenly generated a so-called "Carmichael number," decryption and verification will yield incorrect results for most messages and signatures, an occurrence which could be used to reveal the factors of the modulus.

Though a theoretical concern, the possibility of generating a non-prime is not really a practical issue. With modem prime generation methods, the probability that an output is not prime can be made arbitrarily small or even eliminated entirely.

Decrypting Messages

While recovering the private exponent d seems **to be** difficult provided we generate the key pair appropri-

ately, is there a way to compromise the security of RSA encryption without recovering the private exponent?

We highlight three concerns, and describe how to deal with each in turn. Among the theoretically interesting results that we do not have space to cover here are those concerned with the protection offered by RSA encryption for various bits of message information. The interested reader can find more information about this topic in the work of Alexi et al. [1].

Small Messages

Clearly RSA encryption is not effective on small messages when the public exponent e is small. In particular, when $c = m^e < n$, m can be recovered from c by ordinary root extraction. In fact this attack can be extended somewhat even if there has been some modular reduction by guessing how much reduction has taken place. Thus this attack extended to $c > n$ by trial-and-error might still be faster than exhaustive search form.

But the precaution is obvious. Either the public exponent e should be large or the messages should always be large. It is this second suggestion that is the most useful since a small public exponent is often preferred. However we have to be careful to ensure that the large message we use is not merely some multiple of a known value such as a large power of two (as would be the case if the message were padded on the right with zeroes). As we will see, this would allow an attacker to mount some sophisticated attacks. So when the public exponent e is small the messages being encrypted should always be large in numerical value and not a multiple of some known value. This can all be achieved by padding the message appropriately prior to encryption.

Chosen Ciphertext Attacks

One of the identities we highlighted at the beginning of this article was:

$$(m * r^e)^d \equiv m^d * r \pmod{n}.$$

An attacker might exploit this fact in the following way. Having intercepted some ciphertext c , the attacker chooses some random number r and computes $r^e \pmod{n}$. By sending $c * r^e \pmod{n}$ to the legitimate receiver, $c^d * r \pmod{n}$ is recovered which will, in all

*Though a theoretical concern, **the** possibility of generating a non-prime is not really a practical issue.*

Either the public exponent e should be large or the messages should always be large.

...an attacker should not be able to obtain the raw RSA decryption of an arbitrary value.

The use of random padding ... will destroy any known relation between messages.

likelihood, appear to be random. If the attacker were to obtain this decrypted string then the intended ciphertext could be obtained on multiplication by $r^{-1} \bmod n$.

More far-reaching than obtaining the correct decryption of a particular message, is a generalization of this technique by Desmedt and Odlyzko [7]. Since a chosen ciphertext attack on RSA encryption is equivalent in effect to a chosen-text attack on the RSA signature scheme, we shall also see this attack in the next section.

The attacker asks the user of RSA to decrypt carefully chosen ciphertexts and obtains a pool of data consisting of the decryptions of small primes and certain other values. Then, when sufficient information has been accumulated, the attacker can decrypt a particular encrypted message by multiplying the ciphertext by a random r^e and factoring the result into small primes and other values in the pool. (The attacker can try another r if unsuccessful.) The decryption of the ciphertext will be the product of the factors' decryptions and $r^{-1} \bmod n$.

While this attack can be more efficient than factoring the modulus, certain precautions can be taken to ensure that it is impractical. However there is already one lesson we can learn, and that is that an attacker should not be able to obtain the raw RSA decryption of an arbitrary value.

Just how practical is a chosen ciphertext attack? Consider an environment where a subscriber to some conditional-access service has access to the decryption equipment but not to the actual keys. In such a case the subscriber might well be free to interrogate the decryption unit at will, with ciphertexts of the subscriber's own choosing. It would clearly be a security flaw if the corresponding decryptions were then directly accessible to the attacker.

Low Exponent Attacks

One class of attacks, perhaps more than any other, has been the cause of confusion about the correct and safe use of RSA. These attacks are due to Hastad, and while the scope and validity of the attacks is not in question, some have taken the existence of these attacks as evidence for avoiding low public exponents in an implementation of RSA.

Hastad [12] showed that if an attacker is able to intercept the encryptions of a single message m generated using several different RSA keys with a common public exponent e , then it might be possible to recover m . As a special case, given l different encryptions $m^e \bmod n_1, \dots, m^e \bmod n_l$ an attacker can solve for m with the Chinese Remainder Theorem [14].

More generally, given t related encryptions

$$(a_1 m + b_1)^e \bmod n_1, \dots, (a_t m + b_t)^e \bmod n_t$$

where the a_i 's and b_i 's are known and $t > e(e+1)/2$, an attacker can solve for m with lattice reduction techniques. Note that this attack is a concern if the messages are related in a known way. The use of sufficient pseudorandom padding prior to encryption will make such an attack impossible to mount in practice, and we will describe one method for doing this in the accompanying box. Messages that are related in a known way should not be encrypted with many RSA keys.

Some very recent work by Franklin and Reiter [10] (see Algorithms Update in this issue of *CryptoBytes*) has highlighted a potential problem when related messages are encrypted under the same RSA key with a low exponent. This work should not be confused with that of Hastad but the problem they ingeniously exploit relies on the fact that the various components required for their attack are once again related in a known way. The use of random padding, as we have already seen, will destroy any known relation between messages. Related messages should not be encrypted with the same RSA key.

Forging Signatures

The problem facing an attacker who is attempting to forge an RSA digital signature is to construct a valid signature for a new message while observing the signatures of other messages that might be known or chosen.

Among the attacks we will consider here are a simple chosen message attack and existential forgery as well as the signature version of the attack on encryption due to Desmedt and Odlyzko. We do not consider issues such as which hash functions are more suitable for use with the RSA signing operation. There are a great many possible designs for hash functions

to choose from [19] and while those based on modular arithmetic might offer certain implementation advantages (since they might rely on much the same operations as are already required for signing), Coppersmith has demonstrated attacks on one such proposal [5]. Extreme care should be taken before considering a hash function based on modular arithmetic for use with RSA signatures.

Chosen and Known Message Attacks

Just as there was a chosen ciphertext attack on encryption, there is the following chosen message attack on raw RSA signatures. As with decryption, once given the signature of $m' = mr^e \bmod n$, where r is random, an attacker can obtain the signature of m since $m^d \equiv (m')^d r^{-1} \pmod n$. Again, the lesson to learn is simple; it should not be possible to obtain a raw RSA signature on an arbitrary value.

Using the mathematical identities at the beginning of the article, it is always possible to compute new message-signature pairs (m, s) of the form

$$m = r^e \prod_{i=1}^t m_i^{a_i} \bmod n$$
$$s = r \prod_{i=1}^t s_i^{a_i} \bmod n$$

where $(m_1, s_1), \dots, (m_t, s_t)$ are previous message-signature pairs, and r and a_1, \dots, a_t are arbitrary. Since the resulting message m is not known in advance, this is an “existential” forgery; the signature exists but the message may or may not be useful. However by selecting messages m_i to be signed, an attacker can derive a signature of a given message m in a chosen message attack. Again, this can be avoided by padding in a way that destroys the algebraic connections between messages.

While a good padding scheme will destroy the algebraic properties which allow these attacks, it is worth pointing out that de Jonge and Chaum [6] have extended such basic attacks to demonstrate vulnerabilities in an early proposal for a simple padding scheme.

Also note that while the algebraic properties we exploit appear to have a negative impact on the use of RSA, we should point out that this “attack” can be used constructively to provide what is called blinding [4] in anonymous payment systems!

Desmedt and Odlyzko’s Attack and a Variant

The attack of Desmedt and Odlyzko on encryption applies equally to signatures. Perhaps it is more practical when applied to signature forgery rather than ciphertext decryption, since it might be easier to demand and receive the signature on a variety of messages instead of demanding decryptions for a range of ciphertexts. A variant of this attack can be particularly effective when the messages being signed are small (for instance if they are generated as the output from some message-digest algorithm) unless care is taken.

Essentially, the attack of Desmedt and Odlyzko (for both encryption and signatures) relies on factoring the message into small primes and values near \sqrt{n} . An interesting adaptation of this attack (see notes in Section 8.1 of [22]) applies to the case when the numerical input to the signing algorithm is always relatively small. In this adaptation, the attacker factors the particular message into small primes (this may or may not succeed); the signature on the message equals the product of the factors’ signatures. The attacker obtains the primes’ signatures from appropriate combinations of signatures on other messages whose factors are small. As opposed to when the input is as large as the modulus, values near \sqrt{n} are not needed. The probability of success will therefore depend on the size of the message, not on the size of the RSA modulus.

Messages to be signed should thus be as large as the RSA modulus and, for similar reasons as for encryption, not a multiple of some known value.

RSA Block Formats

Throughout this article we have considered one potential attack after another, in each case highlighting its prevention. Is there a standard way to adopt these safeguards? Is there a way that implementations of RSA can communicate with each other and be assured of a basic level of security?

There are a variety of practical methods for addressing the different recommendations on encryption and signatures. For instance, PKCS #1 [22] (see accompanying box) is a simple, ad-hoc design for protecting RSA encryption and signatures on message digests; it is easy to implement and addresses each of the attacks described above in a heuristic

...if should not be possible to obtain a raw RSA signature on an arbitrary value.

Messages to be signed should be as large as the RSA modulus...

While many of these attacks would be a serious problem ... we have seen that there is always a **practical** and efficient counter measure.

manner. The amount of padding in each block is at least 11 bytes.

ISO/IEC 9796 [13] provides a mathematical design for protecting signatures and also gives message recovery: it is intended for signatures on arbitrary data as well as message digests, and the data or message digest can be recovered from the signature. ISO/IEC 9796 is somewhat more complex than PKCS #1, and about half of each block is padding, but the mathematical motivation is well justified [11].

The construction of Bellare and Rogaway [3], also known as Optimal Asymmetric Encryption Padding, provides a cryptographic design for protecting encryption. It involves hash functions and pseudorandom generators and is the most complex of the three approaches. However, it has the benefit that its security can be directly related to the quality of the hash function or pseudorandom generator. The amount of padding is 16 bytes or more, depending on the level of security.

The whole issue of defining block formats is, in itself, an interesting and delicate problem, and deserves more attention than we can give in this article. Instead we refer the reader to the relevant references for additional information.

Conclusions

In this article, we have provided an overview of the major attacks that have resulted from more than 15 years of cryptanalysis. While many of these attacks would be a serious problem if mounted on a raw form of RSA, we have seen that there is always a practical and efficient countermeasure.

References

[1] W. Alexi, B. Chor, O. Goldreich, and C.P. Schnorr. RSA and Rabin functions: certain parts are as hard as the whole. *SIAM Journal on Computing*, 17(2): 194-209, April 1988.

[2] D. Atkins, M. Graff, A.K. Lenstra, and PC. Leyland. The magic words are squeamish ossifrage. In J. Pieprzyk and R. Safavi-Naini, editors, *Advances in Cryptology—Asiacrypt '94*, pages 263-277, Springer-Verlag, 1995.

[3] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In A. de Santis, editor, *Advances in Cryptology—Eurocrypt '94*, pages 92-111, Springer-Verlag, 1995.

[4] D. Chaum. Security without identification: transaction systems to make big brother obsolete. *Comm. ACM* 28: JO, October, 1985.

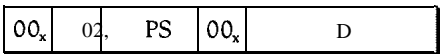
[5] D. Coppersmith. Analysis of ISO/CCITT Document X.509 Annex D. Internal Memo, IBM T.J. Watson Center, June 11, 1989.

[6] W. de Jonge and D. Chaum. Attacks on some RSA signatures. In H.C. Williams, editor, *Advances in Cryptology—*

PKCS #1 Block Formats

PKCS #1, the RSA signature and encryption standard developed by RSA Laboratories and a number of RSA Data Security's customers in the early 1990's, defines two sets of rules for padding a message block prior to encryption or signature. The padding rules are slightly different in the two cases, because of the different style of attacks that they are intended to protect against.

For encryption, the block format is defined as follows. The length of the entire block is the same as the length of the RSA modulus which is being used. It has the following form (from most significant to least significant byte):



The values 00, and 02, are byte values in hexadecimal notation, PS is a pseudorandom string of nonzero bytes, and D is the data to be encrypted. The pseudorandom string must be at least eight bytes long. The byte 00_x

ensures that the block is arithmetically less than the RSA modulus. The byte 02, and the padding makes the input to RSA a large integer, thus preventing short message attacks; the pseudorandomness in PS prevents low-exponent attacks. The overall structure of the block prevents the various multiplicative attacks that we observed in the accompanying article.

More specifically, the pseudorandom string increases the difficulty of a variety of attacks—Hastad's, Franklin and Reiter's, and various chosen ciphertext attacks including Desmedt and Odlyzko's—by a factor of at least $(255)^s \approx 2^{64}$, assuming that the pseudorandom string is discarded by the decryption equipment. To complete any of those attacks, the opponent essentially must guess the bits of the pseudorandom string for at least one, and possibly more messages. (For instance, to complete Franklin and Reiter's attack, the opponent must guess the difference between the encryption blocks for two messages, which involves at least 64 unknown bits.)

The pseudorandom string also thwarts the "verifiable plaintext" attack where an opponent re-encrypts guessed plaintext

Crypto '85, pages 18-27, Springer-Verlag, 1986.

[7] Y.G. Desmedt and A.M. Odlyzko. A chosen text attack on the RSA cryptosystem and some discrete logarithm schemes. In H.C. Williams, editor, *Advances in Cryptology -Crypto '85*, pages 516-522, Springer-Verlag, 1986.

[8] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22: 644-654, 1976.

[9] J.-H. Evertse and E. van Heyst. Which new RSA-signatures can be computed from certain given RSA-signatures? *Journal of Cryptology*, 5(1):41-52, 1992.

[10] M. Franklin and M. Reiter. A linear protocol failure for RSA with exponent three. Presented at the Rump Session of Crypto '95, Santa Barbara, CA.

[11] L.C. Guillou, J.-J. Quisquater, M. Walker, P. Landrock, and C. Shaer. Precautions taken against various potential attacks in ISO/IEC DIS 9796. In I.B. Damgård, editor, *Advances in Cryptology-Eurocrypt '90*, pages 465-473, Springer-Verlag, 1991.

[12] J. Hastad. Solving simultaneous modular equations of low degree. *SIAM Journal on Computing*, 17(2):336-341, April 1988.

[13] ISO/IEC. *International Standard 9796: Information Technology, Security Techniques: Digital Signature Scheme Giving Message Recovery*, 1991.

[14] D.E. Knuth. *The Art of Computer Programming, Vol.2: Seminumerical Algorithms*. Second edition, Addison-Wesley, Reading, MA., 1981.

[15] H.W. Lenstra Jr. Factoring integers with elliptic curves.

Ann. of Math., 126:649-673, 1987.

[16] G.L. Miller. Reimann's hypothesis and tests for primality. *J. Comput. System Sci.*, 13:300-317, 1976.

[17] A.M. Odlyzko. The Future of Integer Factorization. *RSA Laboratories' CryptoBytes*, 1:2, pages 5-12, Summer, 1995.

[18] J. Pollard. Theorems on factorization and primality testing. *Proc. Cambridge Philos. Soc.*, 76:521-528, 1974.

[19] B. Preneel. *Analysis and Design of Cryptographic Hash Functions*. Ph.D. Thesis, K.U.Leuven. January, 1993.


[20] R. Rivest. Dr. Ron Rivest on the Difficulty of Factoring. *Ciphertext*: The RSA Newsletter, vol. 1, no. 1, fall 1993, and reprinted, in an updated form, in an appendix on pages 361-364 in S. Garfinkel, *PGP: Pretty Good Privacy*, O'Reilly & Associates, 1995.

[21] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120-126, February 1978.

[22] RSA Laboratories. *PKCS #1: RSA Encryption Standard*, Version 1.5, November 1993.

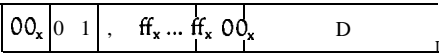
[23] G.J. Simmons and M.J. Norris. Preliminary comments on the MIT public-key cryptosystem. *Cryptologia*, 1(4):406-414, 1977.

[24] M.J. Wiener. Cryptanalysis of short RSA secret exponents. *IEEE Transactions on Information Theory*, 36:553-558, 1990.

[25] H.C. Williams and B. Schmid. Some remarks concerning the MIT public-key cryptosystem. *BIT* 19, pages 525-538, 1979. 

and compares the result to the ciphertext. Since the opponent must guess the pseudorandom string in addition, the difficulty increases by at least a factor of 2^{64} .

For protecting RSA signatures, the signature block format is slightly different and has the following form:




Again 00_x, 01, and ff_x are byte values in hexadecimal notation, and D is the data to be signed, which must be a message digest. There must be at least eight ff_x bytes.

The byte 00, again ensures that the block is arithmetically less than the RSA modulus. The byte 01, and the ff_x padding prevent the variant of Desmedt and Odlyzko's attack on small messages and the overall structure of the block prevents various multiplicative attacks.

Just as for encryption, there is at least a 2^{64} increase in work effort for a variety of attacks. For instance, known and cho-

sen message forgery is 2^{64} harder since an opponent can only obtain "raw" RSA signatures on less than one in every 2^{64} possible messages. The opponent must repeatedly randomize the message to be signed to find one whose block format is correct.

The PKCS #1 block format for signatures is intended only for signatures on the message digests of messages. Some potential attacks arise if the data D is arbitrary, since an opponent can select the value of D so that the resulting signature block has small prime factors, or perhaps is even a natural power. However, when the data D is a message digest, which is effectively a pseudorandom value, these attacks are not a concern; for applications where a message digest is signed with RSA, the format is adequate.

In applications where data is signed directly, a format such as ISO/IEC 9796 designed specifically for signatures with "message recovery" is preferable. 

F R E Q U E N T L Y A S K E D Q U E S T I O N S

How Do Digital Time-Stamps
Support Digital Signatures?

Answered by
Stuart Haber, Burt Kaliski and **Scott Stornetta**

Consider two questions that may be asked by a computer user as he or she views a digital document or on-line record. (1) Who is the author of this record -who wrote it, approved it, or consented to it? (2) When was this record created or last modified?

In both cases, the question is one about exactly this record-exactly this sequence of bits-whether it was first stored on this computer or was created somewhere else and then copied and saved here. An answer to the first question tells who & what: who approved exactly what is in this record? An answer to the second question tells when & what: when exactly did the contents of this record first exist?

Both of the above questions have good solutions. A system for answering the first question is called a digital signature scheme. Such a system was first proposed in [2] and there is a wide variety of accepted designs for an implementation of this kind of system [4,5].

A system for answering the second question is called a digital time-stamping scheme. Such systems were described in [1,3], and an implementation is commercially available from Surety Technologies (<http://www.surety.com/>).

Any system allowing users to answer these questions reliably for all their records must include two different sorts of procedures. First, there must be a certification procedure with which (1) the author of a record can "sign" the record, or (2) any user can fix a record in time. The result of this procedure is a small certifying file, a certificate if you will, that captures the result of this procedure. Second, there must be a verification procedure by which any user can check a record and its accompanying certificate to

make sure it correctly answers (1) who and what? or (2) when and what? about the record in question.

The "certificate" returned by the certification procedure of a digital signature system is usually called a signature; it is a signature for a particular signer (specifying whom) and for a particular record (specifying what). In order to be able to "sign" documents, a user registers with the system by using special software to compute a pair of numbers called keys, a *public* key and a corresponding *private* key. The private key should only be available to the user to whom it belongs, and is used (by the certification or "signing" procedure) in order to sign documents; it is by employing the user's private key that the signature and the record are tied to that particular user. The public key may be available to many users of the system, and is used by the verification procedure. That is, the verification procedure takes a particular record, a particular user's public key, and a putative signature for that record and that user, and uses this information to check whether the would-be signature was correctly computed using that record and the corresponding private key.

Special computational methods are employed for signing documents and for verifying documents and signatures; when these methods are carefully implemented, they have the remarkable property that the knowledge of a user's public key does not enable an attacker or hacker to figure out the user's corresponding private key. Of course, if, either through carelessness or deliberate intent, someone else-a hacker, for example-gains access to the user's private key, then this person will be able to "forge" the legitimate user's signatures on documents of the hacker's choice. At that point, even the value of legitimately signed records can be called into question.

The "certificate" returned by the certification procedure of a digital time-stamping system is a certificate for a particular record (specifying what) at a particular time (specifying when). The procedure works by mathematically linking the bits of the record to a "summary number" that is widely witnessed by and widely available to members of the public-including, of course, users of the system. The computational methods employed ensure that only the record in question can be linked, according to the "instructions" contained in its time-stamp

Digital time-stamps increase the longevity of digitally-signed records.

Stuart Haber is a research scientist in the Security Research Group at Bellcore, Burt Kaliski is chiefscientist at RSA Laboratories, and W. Scott Stornetta is Chairman of Surety Technologies. Surety Technologies was founded by Stornetta and Haber in 1994 as a spin-off from Bellcore, with a mandate to commercialize the digital time-stamping technology developed by Bellcore. The authors can be contacted at stuart@bellcore.com, burt@rsa.com and scotts@surety.com.

certificate, to this widely witnessed summary number; this is how the particular record is tied to a particular moment in time. The verification procedure takes a particular record and a putative time-stamp certificate for that record and a particular time, and uses this information to validate whether that record was indeed certified at the time claimed by checking it against the widely available summary number for that moment.

Two features of a digital time-stamping system are particularly helpful in enhancing the integrity of a digital signature system. First, a time-stamping system cannot be compromised by the disclosure of a key. This is because digital time-stamping systems do not rely on keys, or any other secret information, for that matter. Second, following the technique introduced in [1], digital time-stamp certificates can be renewed so as to remain valid indefinitely.

With these features in mind, consider the following situations.

It sometimes happens that the connection between a person and his or her public signature key must be revoked—for example, if the user's secure access to the private key is accidentally compromised; or when the key belongs to a job or role in an organization that the person no longer holds. Therefore the person-key connection must have time limits, and the signature verification procedure should check that the record was signed at a time when the signer's public key was indeed in effect. And thus when a user signs a record that may be checked some time later—perhaps after the user's key is no longer in effect—the combination of the record and its signature should be certified with a secure digital time-stamping service.


There is another situation in which a user's public key may be revoked. Consider the case of the signer of a particularly important document who later wishes to repudiate his signature. By dishonestly reporting the compromise of his private key, so that all his signatures are called into question, the user is able to disavow the signature he regrets. However, if the document in question was digitally time-stamped together with its signature (and key-revocation reports are time-stamped as well), then the signature cannot easily be disavowed in this way. This is the recommended procedure, therefore, in order to pre-

serve the *non-repudiability* desired of digital signatures for important documents.

The statement that private keys cannot be derived from public keys is an over-simplification of a more complicated situation. In fact, this claim depends on the computational difficulty of certain mathematical problems. As the state of the art advances—both the current state of algorithmic knowledge, as well as the computational speed and memory available in currently available computers—the maintainers of a digital signature system will have to make sure that signers use longer and longer keys. But what is to become of documents that were signed using key lengths that are no longer considered secure? If the signed document is digitally time-stamped, then its integrity can be maintained even after a particular key-length is no longer considered secure.

Of course, digital time-stamp certificates also depend for their security on the difficulty of certain computational tasks concerned with so-called one-way hash functions. (All practical digital-signature systems depend on these functions as well.) Those who maintain a secure digital time-stamping service will have to remain abreast of the state of the art in building and in attacking one-way hash functions. Over time, they will need to upgrade their implementation of these functions, as part of the process of renewal [1]. This will allow time-stamp certificates to remain valid indefinitely.

References

- [1] D. Bayer, S. Haber, and W.S. Stornetta. Improving the efficiency and reliability of digital time-stamping. In R.M. Capocelli, A. De Santis, U. Vaccaro, editors, *Sequences II: Methods in Communication, Security, and Computer Science*, pp. 329-334, Springer-Verlag, New York (1993).
- [2] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22: 644-654, 1976.
- [3] S. Haber and W.S. Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, Vol. 3, No. 2, pp. 99-111 (1991).
- [4] National Institute of Standards and Technology (NIST). FIPS Publication 186: Digital Signature Standard, May 19, 1994.
- [5] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2): 120-126, February 1978. 

Time-stamping strengthens non-repudiation of digital signatures.

A N N O U N C E M E N T S

The 1996 RSA Data Security Conference

The 1996 RSA Data Security Conference will be held January 17-19 in the Fairmont Hotel, San Francisco. First held in 1991, this annual conference is expected to attract more than 1,000 participants and provides an excellent opportunity for business people, academic cryptographers and representatives of government to gather and debate the technology and business issues facing the industry.

Spread over three days, there will be a wide range of seminars, tutorials and presentations. Many of the presentations will be focused on the commercial applications of modern cryptographic technology, with an emphasis on Public Key Cryptosystems, but there will also be simultaneous tracks for developers, cryptographers and analysts from which attendees can pick and choose the talks they wish to attend.

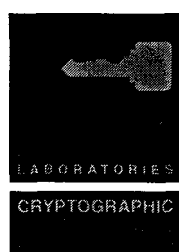
There are plans for several open panel discussions. On the final day, the focus of these discussions will be on electronic commerce and doing business on the Internet. Meanwhile Washington will be the topic for discussion on the second day, with issues ranging from CLIPPER, FORTEZZA and key escrow, to export control. Following the keynote address and company announcements on the first morning, a cryptographer's expert panel will provide an excellent opportunity for attendees to direct their questions directly to some of the leading figures in cryptographic research.

Over the years, the RSA Data Security Conference has proved to be a great opportunity for vendors, developers and specialists to meet, and like previous years it is expected to be filled to capacity very soon. More information can be found on RSA's web page (<http://www.rsa.com/>) or by contacting the conference organizer, Layne Kaplan Events, at 415/340-9300.

in this issue:

- *RSA for Paranoids*
- *Collisions in MD4*
- *The Secure Use of RSA*
- *Digital Time Stamps*

For subscription information, see page 2 of this newsletter.



100 MARINE PARKWAY
REDWOOD CITY
CA. 94065-1031
TEL 415/595-7703
FAX 415/595-4126
rsa-labs@rsa.com

PRESORT
FIRST CLASS
U.S. POSTAGE
PAID
MMS, INC

RSA LABORATORIES' CryptoBytes

The technical newsletter of RSA Laboratories, a division of RSA Data Security, Inc.

Contents

1

*Elliptic Curve
Cryptosystems*

2

Editor's Note

4

Standards Update

5

*The Future of Integer
Factorization*

13

*On the Security of the
RC5 Encryption
Algorithm*

14

Algorithms Update

15

To the Editor

16

Announcements

Elliptic Curve Cryptosystems

Alfred Menezes

120 Math Annex
Auburn University
Auburn, AL 36849 USA

Elliptic curves have been intensively studied in number theory and algebraic geometry for over 100 years and there is an enormous literature on the subject. To quote the mathematician Serge Lang:

*It is possible to write endlessly on elliptic curves.
(This is not a threat.)*

Elliptic curves also figured prominently in the recent proof of Fermat's Last Theorem by Andrew Wiles. Originally pursued for purely aesthetic reasons, elliptic curves have recently been utilized in devising algorithms for factoring integers, primality proving, and in public-key cryptography. Although the high security per bit ratio for elliptic curve cryptosystems has been known for 10 years, it is only recently that high speed implementations have been available.

Discrete-log cryptosystems

In their landmark 1976 paper, Whit Diffie and Martin Hellman invented the notion of public-key cryptography, and introduced the *Diffie-Hellman* key agreement protocol. This proto-

col, which allows two parties Alice and Bob to establish a secret key through an exchange of public messages, works as follows. Let p be a large prime number, and let a be a generator of the multiplicative group \mathbb{Z}_p^* ; in layman's terms this means that the powers $\alpha^0, \alpha^1, \alpha^2, \dots, \alpha^{p-2}$ of a , each reduced mod p , yield all the integers between 1 and $p-1$. The parameters p and α are public knowledge. Alice then generates a random integer a , $0 \leq a \leq p-2$, and transmits $\alpha^a \bmod p$ to Bob. Bob similarly generates a random integer b , $0 \leq b \leq p-2$, and transmits $\alpha^b \bmod p$ to Alice. Both parties can now compute the number $\alpha^{ab} \bmod p$, which serves as their secret key.

The security of the Diffie-Hellman key agreement protocol is based on the apparent intractability of the *discrete logarithm problem* in \mathbb{Z}_p^* : given a prime p , a generator a of \mathbb{Z}_p^* , and an element $\beta \in \mathbb{Z}_p^*$, determine the integer a , $0 \leq a \leq p-2$, such that $\alpha^a = \beta \pmod{p}$. The best algorithm known for this problem is the general number field sieve by Dan Gordon, and has an asymptotic running time of

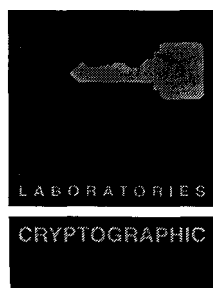
$$\exp((1.923 + o(1))(\log p)^{1/3}(\log \log p)^{2/3}). \quad (1)$$

The security of the RSA public-key cryptosystem is based upon the difficulty of the problem of factoring integers. The number field sieve is the best algorithm known for this problem, and has an asymptotic running time of

$$\exp((1.923 + o(1))(\log n)^{1/3}(\log \log n)^{2/3}) \quad (2)$$

(continued on page 3)

Dr. Alfred Menezes is assistant professor in the "Department of Discrete and Statistical Sciences" at Auburn University, Alabama. His main research interests are public-key cryptography, elliptic curve cryptosystems, and algorithmic number theory. Dr. Menezes also consults on a regular basis for MÖBIUS Encryption Technologies, Mississauga, Ontario, Canada. He can be contacted at menezes@mail.auburn.edu.



Editor's Note

Welcome to the second issue of *CryptoBytes*. During May of this year, we launched this newsletter with the aim of providing a forum for "results and opinions that, while of great cryptographic interest, would not appear at any of the more classical outlets because of their format." The initial response to the general idea of a technical newsletter has been very encouraging, and we are pleased to bring a second issue which further fulfills the aims set out in the first.

In this second issue of *CryptoBytes*, we are presenting two invited articles.


The first, *Elliptic Curve Cryptosystems* by A. Menezes, provides us with an intriguing glimpse at what are potentially very efficient alternatives to cryptosystems based on the discrete logarithm problem. This article is nicely complemented by news that the IEEE P1363 standardization effort is moving quickly towards standards for algorithms based around the use of elliptic curves.

Our second invited article, *The Future of Integer Factorization* by A. Odlyzko, provides a valuable contemporary assessment of factoring ability. The article also provides some thoughts on possible trends for the future and potential developments well into the next century. Because of the obvious relevance of this article to the continuing safe use of the RSA cryptosystem, we conclude the article with the current RSA Laboratories minimum key-size recommendations for users of RSA.

Following on pieces in the last issue of *CryptoBytes* we include an article providing details of an initial analysis of the RC5 encryption algorithm. In addition, readers interested in last issue's article *Message Authentication with MD5* will find a short letter by P. van Oorschot and B. Preneel essential reading.

We also include a short review of recent analysis into the performance of MD5. Since one of the original aims of *CryptoBytes* was to provide a basic, reliable cryptographic news service, we are hoping that this item will be the first of an occasional but ongoing feature, *Algorithms Update*. By providing a suitable forum, we hope that reports on major

algorithm developments, be they cryptographic, cryptanalytic or implementational, can be brought together for the benefit of the cryptographic community.

To include all these items, we have already had to increase the number of pages in *CryptoBytes*. We would very much like to thank all the writers who have contributed to this second issue, and since the future success of *CryptoBytes* depends on input from all sectors of the cryptographic community, we encourage any readers with comments, opposite opinions, suggestions or proposals for future issues to contact the *CryptoBytes* editor at RSA Laboratories or by E-mail to bytes-ed@rsa.com. 

Subscription Information

CryptoBytes is published four times annually; printed copies are available for an annual subscription fee of U.S. \$90. To subscribe, contact RSA Laboratories at:

RSA Laboratories
100 Marine Parkway, Suite 500
Redwood City, CA 94065
415/595-7703
415/595-4126 (fax)
rsa-labs@rsa.com

Back issues in electronic form are available via the World-Wide Web at <http://www.rsa.com/rsalabs/cryptobytes/>.

RSA Laboratories is the research and development division of RSA Data Security, Inc., the company founded by the inventors of the RSA public-key cryptosystem. RSA Laboratories reviews, designs and implements secure and efficient cryptosystems of all kinds. Its clients include government agencies, telecommunications companies, computer manufacturers, software developers, cable TV broadcasters, interactive video manufacturers, and satellite broadcast companies, among others.

*Design and layout for *CryptoBytes* are provided by Coda Graphics, Oakland, CA.*

By providing a suitable forum, we hope that reports on major algorithm developments, be they cryptographic, cryptanalytic or implementational, can be brought together for the benefit of the cryptographic community.



Elliptic Curve Cryptosystems

Continued from page 1

Comparing these two running times, one can conclude that with our current state of knowledge, it is roughly as difficult to compute discrete logarithms modulo a prime p as it is to factor an integer n of the same size.

There are many cryptographic protocols whose security is based on the discrete logarithm problem in \mathbb{Z}_p^* , for example the ElGamal public-key encryption and signature schemes, the Digital Signature Algorithm (DSA), and the Schnorr signature scheme. Originally these protocols were all described in the algebraic setting of the multiplicative group \mathbb{Z}_p^* . However, they can equally well be described in the setting of any finite group, for example the multiplicative group of the finite field \mathbb{F}_{2^m} of characteristic 2.

Why consider other groups?

There are two primary reasons for this. Firstly, other groups may be easier to implement in software or hardware. Secondly, the discrete logarithm problem in the group may be harder than the discrete logarithm problem in \mathbb{Z}_p^* . Consequently, one can use a group G that is smaller than \mathbb{Z}_p^* while maintaining the same level of security. This results in cryptosystems with smaller key-sizes, bandwidth savings, and possibly faster implementation.

Elliptic curves

In 1985, Neal Koblitz and Victor Miller independently proposed using the group of points on an elliptic curve in existing discrete-log cryptosystems. For an introduction to this subject, the reader is referred to the books by Koblitz [2] and Stinson [4]. A more complete treatment is given by Menezes [3]. Without going into all the details, an *elliptic* curve over a finite field F is the set of all solutions (also called points) (x, y) , $x \in F$, $y \in F$, to an equation of a special form. For example, if the finite field is $F = \mathbb{Z}_p$, the integers modulo a prime p , then the equation has the form $y^2 = x^3 + ax + b$, where $a, b \in \mathbb{Z}_p$ and $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$. There is a simple group law for adding two points on the curve to produce a third point. The addition rule simply involves a few operations (addition, subtraction,

multiplication and inversion) in the underlying finite field F , and thus can be efficiently implemented.

The main attraction of elliptic curve cryptosystems arises because the analogue of the discrete logarithm problem in these curves is apparently much harder than the discrete logarithm problem in \mathbb{Z}_p^* and the integer factorization problem. More precisely, let E be an elliptic curve defined over a finite field \mathbb{F}_q , where q is a prime or prime power. Let n denote the number of points on E ; it is always the case that n is roughly equal to q . If n is a prime (or at least divisible by a large prime) then the best method known for computing discrete logarithms in E are the fully exponential algorithms of Shanks and Pollard, which have an expected running time of about \sqrt{n} elliptic curve operations. What should be noted here is that the function \sqrt{n} grows much faster than the running time functions for discrete logs in \mathbb{Z}_p^* (equation (1)) and factoring integers (equation (2)).

As a concrete example, let E be an elliptic curve over the field $\mathbb{F}_{2^{160}}$. As an optimistic estimate, suppose that a machine rated at 1 mips can perform 40,000 elliptic curve operations per second. (This estimate is indeed very optimistic — an ASIC built by MÖBIUS Encryption Technologies for performing elliptic curve operations over the field $\mathbb{F}_{2^{155}}$ has a 40 MHz clockrate and can perform roughly 40,000 elliptic operations per second.) Then the computing power required to compute a single discrete logarithm in E is about 10^{12} MY (mips-years). By contrast, based on the most recent implementation of the number field sieve for factoring by Dodson and Lenstra [1], the computing power required to factor a 1024-bit integer is estimated to be about $3 \cdot 10^{11}$ MY (see Andrew Odlyzko's article in this issue). Thus an elliptic curve cryptosystem over the 160-bit field $\mathbb{F}_{2^{160}}$ offers the same level of security as RSA or DSA with a modulus size of 1024 bits.


The advantage of the elliptic curve system is that arithmetic in the field $\mathbb{F}_{2^{160}}$ is far easier to implement, both in hardware and software, than arithmetic modulo a 1024-bit integer. For example,

There are many cryptographic protocols whose security is based on the discrete logarithm problem in \mathbb{Z}_p^ [...]. However, they can equally well be described in the setting of any finite group.*

[...] elliptic curve cryptosystems offer the most security per bit of any known public-key scheme [...]

the ASIC mentioned above for performing elliptic curve arithmetic. In addition, the encryption and signature schemes based on elliptic curves are an order of magnitude faster than 1024-bit DSA and RSA. Similar speedups are now possible in software due to recent advances in the software implementation of elliptic curve cryptosystems over F_2^m .

In summary, elliptic curve cryptosystems offer the most security per bit of any known public-key scheme. This will tend to increase their attractiveness relative to other cryptosystems as computing power improvements warrant general key size. Elliptic curve arithmetic over F_2^{155} has only 12,000 gates, and would occupy less than 5% of the area typically designated for a smart card processor.

creases. The smaller key sizes result in smaller system parameters, smaller public-key certificates, bandwidth savings, and faster implementations. Elliptic curve systems are particularly beneficial in applications where computational power and integrated circuit space is limited, such as smart cards, PCMCIA cards, and wireless devices. 

References

111 B. Dodson and A. Lenstra, "NFS with four large primes: an explosive experiment", *Advances in Cryptology—CRYPTO'95*, to appear.
121 N. Koblitz, *A Course in Number Theory and Cryptography*, Springer-Verlag, New York, 1994.
131 A. Menezes, *Elliptic Curve Public Key Cryptosystems*, Kluwer Academic Publishers, Boston, 1993.
141 D. Stinson, *Cryptography- Theory and Practice*, CRC Press, Boca Raton, 1995.


S T A N D A R D S U P D A T E

Elliptic Curves in Draft IEEE Standard
Elliptic curve cryptosystems are taking another step forward from theory to practice as part of a draft standard being prepared by IEEE's P1363 working group, *Standard for RSA, Diffie-Hellman and Related Public-Key Cryptography*.

- Four cryptographic mechanisms based on elliptic curves are currently being considered:
- Elliptic Curve Encryption Scheme (ECES), an analog of the ElGamal public-key cryptosystem;
 - Elliptic Curve Signature Scheme (ECSS), an analog of a variant of the ElGamal signature scheme;
 - Elliptic Curve Digital Signature Algorithm (ECDSA), an analog of NIST's Digital Signature Algorithm; and
 - Elliptic Curve Key Establishment Protocol (ECKEP), an extension of Diffie-Hellman key agreement that provides implicit key authentication.

Other algorithms to be included in the standard are RSA, Diffie-Hellman and ElGamal. The standard will also have sections on random number generation and hardware support for public-key cryptography. ISO/IEC JTC 1/WG 2/SC 27 (Security Techniques — Techniques and Mechanisms) is also drafting

a standard for cryptography based on elliptic curves.
For more information on P1363, contact the working group's chair, Burt Kaliski of RSA Laboratories. Meeting notices and draft materials are available by anonymous ftp to ftp.rsa.com in the pub/p1363 directory. The next meeting of P1363 will review the elliptic curve material and is scheduled for August 31-September 1 at the University of California, Santa Barbara, following the CRYPTO '95 conference.

S/MIME Standardized
A group of leading networking and messaging vendors, in conjunction with RSA Data Security, Inc., recently endorsed a specification that will enable encrypted messages to be exchanged between E-mail applications from different vendors. Vendors participating in the announcement included ConnectSoft, Frontier, FTP Software, Qualcomm, Microsoft, Lotus, Wollongong, Banyan, NCD, SecureWare, and VeriSign.
The specification — Secure/Multipurpose Internet Mail Extensions (S/MIME) — is based on the popular Internet MIME standard (RFC 1521) with the secure portion of the message defined by the public key cryptography standards PKCS #7 and PKCS #10.
Developers interested in S/MIME can get more information in "What's New" on RSA's web page (<http://www.rsa.com>). 

Elliptic curve cryptosystems are taking another step forward from theory to practice.

The Future of Integer Factorization

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

Introduction

How large should be the moduli in public key cryptosystems such as RSA and DSS (the US Digital Signature Standard)? The answer depends on the anticipated threats. Even if those are known, there is no way to provide a definitive answer, since progress in integer factorization and discrete logarithm algorithms is not predictable. (Furthermore, there is still the possibility that RSA and DSS could be broken by other methods than factoring or discrete log approaches.) However, since choices of moduli have to be made, it is necessary to make some estimates, and this note attempts to do so, taking into account both the increase in available computing power and future algorithmic developments. The projections made below suggest that RSA and DSS moduli might have to be unpleasantly large. In particular, the 512-bit moduli that have been adopted in a variety of cryptosystems are already unsafe for all applications except those with very modest security requirements.

I would like to stress that while the assertion about insecurity of 512-bit moduli is easy to support with solid technical evidence, the projections about the future are much less certain, and should not be treated as firm forecasts, but as possible ways that computational number theory might develop.

Only conventional algorithms and standard integrated circuit technologies will be considered. If either quantum computers (à la Shor) or DNA computers (à la Adleman) become practical, the outlook might change, and even larger moduli might become necessary.

Computing power will be measured in units of MY, or mips-years. By convention, a 1 mips machine is equivalent to the DEC VAX 11/780 in computing power, and so 1 MY is one year on a VAX 11/780. This measure has many defects, and nowadays a wide

variety of other benchmarks are used in preference to mips measures. Still, given the uncertainty in any projection far into the future, this measure seems adequate.

43D will refer to an integer of 43 decimal digits.

Discussion will be restricted to integer factorization. Discrete logarithms are, with the present state of knowledge, slightly more difficult to compute modulo an appropriately chosen prime than it is to factor a “hard” integer of the same size, but the difference is not large [11]. Therefore to be on the safe side in designing cryptosystems, one should assume that all the projections about sizes of integers that it will be possible to factor will also apply to sizes of primes modulo which one can compute discrete logarithms.

Factorization records and historical estimates

There is a long record (see Appendix A) of estimates of sizes of integers that could be factored. They have uniformly turned out to be too low, primarily because of unanticipated algorithmic improvements. Faster than expected growth in available computing resources also played a role, though. Table 1 summarizes the progress that has occurred in the last few decades. Table 2 shows how much the computing power available for integer factorizations has increased.

Table 1 (see Appendix B)

Historical records in integer factorization	
year	record factorizations
1964	20D
1974	45D
1984	71D
1994	129D

Table 2 (see Appendix C)

Computing power used to achieve record factorizations	
year	MY
1974	0.001
1984	0.1
1994	5000

Projections of computing power available in the future
The dramatic increase in computing power used for factorization between 1984 and 1994 resulted largely

There is a long record of estimates of sizes of integers that could be factored. They have uniformly turned out to be too low.

Andrew M. Odlyzko is Head of the Mathematics of Communication and Computer Systems Department at AT&T Bell Laboratories in Murray Hill, New Jersey. His interests include computational complexity, cryptography, number theory, combinatorics, coding theory, analysis, and probability theory. He can be contacted at amo@research.att.com.

The organizers of the RSA129 project were able to obtain about 0.03% of the total computing power of the Internet. They also estimated that without extraordinary effort they could have organized a project with 100 times as much power,

It is also possible for small groups of people to assemble considerable amounts of distributed computing power surreptitiously.

from the introduction of distributed computing, using the idle time on a network of workstations. This trend was started by Bob Silverman. (An earlier instance was Richard Schroepel's attempt to factor F_8 , but that work did not attract much public attention.) It was fully developed by Arjen Lenstra and Mark Manasse. The RSA129 factorization used idle time on around 1600 computers around the world during an 8 month period. Most modem factoring methods do lend themselves to distributed implementations.

In the remainder of this discussion, we will only consider factoring attempts similar to that on RSA129, namely ones that use idle time on networks of computers. Unlike attacks on DES, where effective attacks appear to require a special purpose machine (see Appendix J), these attacks do not require any major commitment of financial or technical resources, and can even be mounted surreptitiously.

Another projection of the future of integer factorization has been made by Ron Rivest [14]. Rivest's expectations for progress in computer technology and algorithms do not differ much from mine, but he considers what can be done with a fixed amount of money using machines bought especially for factoring.

Table 3 (see Appendix D)

1994 computing power	
	mips rating
RSA129 project	10^4
Internet	$3 \cdot 10^7$
the world	$3 \cdot 10^8$

We note that even without an extensive effort, the organizers of the RSA129 project were able to obtain about 0.03% of the total computing power of the Internet. They also estimated [1] that without extraordinary effort they could have organized a project with 100 times as much power, or about 3% of the capacity of the Internet.

We should also note that (and this is relevant for cryptosystem security) there are relatively small organizations that have large amounts of computing power all by themselves. Silicon Graphics, for example, has about 5,000 employees and 10,000 workstations, for total computing power of perhaps 10^5 mips, about 10 times the computing power of the

RSA129 project. Thus it is conceivable that a few individuals, such as systems administrators at a large corporation, could organize a covert effort at factoring that would dwarf that of the RSA129 project. In particular, using the current implementations of the number field sieve, they could easily factor a 512-bit integer in under a year of elapsed time, and could do so now.

It is also possible for small groups of people to assemble considerable amounts of distributed computing power surreptitiously. As an example, a 384-bit RSA key was recently broken by Muffett et al. [10] using about 400 MY. In a few years, we might see teenage system administrators for local real estate agents or laundries breaking 512-bit RSA keys without anyone being aware of the attack.

What about the future? Moore's "Law" says that microprocessor processing speed doubles every 18 months. In 10 years, that means an increase by a factor of about 100. Let us assume that this "law" will hold for the next 10 years (Appendix E), and that a further increase in processing power of between 10 and 100 will be achieved in the following 10 years. We then find that the typical processor in 2004 might be rated at 10^3 mips, and in 2014 at $10^4 - 10^5$ mips.

Since there are already over 10^8 computers in the world, it seems safe to assume there will be at least $2 \cdot 10^9$ by 2004. By the year 2014, we might have $10^{10} - 10^{11}$ (Appendix F). Further, by that time almost all are likely to be networked together. However, it is uncertain what fraction might be available for a factoring experiment. Let us consider two scenarios:

- (a) Widely known collaborative effort to break some challenge cipher. It does not seem out of the question that up to 0.1% of the world's computing power might be made available for a year. (The RSA129 project organizers estimated they could have obtained access to 3% of the computing power of the Internet, but this 3% factor might not scale as the networks grow.) This yields $2 \cdot 10^9$ MY available in 2004, and $10^{11} - 10^{13}$ MY in 2014.
- (b) Surreptitious effort arranged by a handful of people at an organization such as a corporation or university. They might have available to them 10^5

computers in 2004, and up to 10^6 in 2014. Thus they might have 10^8 MY at their disposal in 2004, and 10^{10} - 10^{11} in 2014. (Would they attempt to attack a single cryptographic system, or would they attempt to attack 1000 different systems? This would likely depend on what moduli are used, and on the potential payoff.)

We summarize the projections above in Table 4.

Table 4

Computing power available for integer factorization		
year	covert attack	open project
2004	10^8 MY	$2 \cdot 10^9$ MY
2014	10^{10} - 10^{11} MY	10^{11} - 10^{13} MY

Factorization using current algorithms
Of the methods that are currently known and apply to generally hard integers, the general number field sieve (gnfs) has the best asymptotic running time estimate. It is also practical, and runs faster than previous algorithms on generally hard integers of more than about 115D. Since a 119D integer was factored recently using gnfs with 250 MY (see Appendix C), we can project, using standard methods (see Appendix H), the amount of computing that is likely to be required to factor various integers. The results are shown in Table 5.

Table 5

Computing power required to factor integers with current version of gnfs	
bits of n	MY required
512	$3 \cdot 10^4$
768	$2 \cdot 10^8$
1024	$3 \cdot 10^{11}$
1280	$1 \cdot 10^{14}$
1536	$3 \cdot 10^{16}$
2048	$3 \cdot 10^{20}$

Thus, based on tables 4 and 5, moduli of 1280 bits are likely to be safe for well over 20 years, and even 1024-bit moduli are not likely to be vulnerable, unless they conceal extremely valuable information. However, Table 5 assumes that the current version of gnfs will remain the best algorithm. This would be an extremely imprudent assumption, as history shows that algorithmic improvements are often crucial.

It is important to note that 512-bit integers, which are used in a variety of commercial implementa-

tions of RSA, can already be factored with the available computing power. There is no need for new algorithms or faster or more computers, just the effort to find enough people willing to let their workstations be used in their idle time. The machines at Silicon Graphics alone could factor a single 512-bit integer in about half a year total time (under the assumption that they are idle about two thirds of the time).

Algorithmic improvements

The special number field sieve (snfs) applies to integers such as the Fermat numbers. Based on the recent factorization of a 162D special integer by Boender et al. in about 200 MY, we can estimate how long snfs takes to factor various integers, and the results are presented in Table 6.

Table 6

Computing power required to factor integers with the snfs	
bits of n	MY required
768	$1 \cdot 10^5$
1024	$3 \cdot 10^7$
1280	$3 \cdot 10^9$
1536	$2 \cdot 10^{11}$
2048	$4 \cdot 10^{14}$

In particular, it appears that it might be possible to factor $F_{10} = 2^{1024} + 1$ by the year 2000 (continuing the tradition in which F_7 was factored in 1970, F_8 in 1980, and F_9 in 1990).

Is it reasonable to expect a breakthrough that would enable generally hard integers to be factored in about the same time that the snfs factors integers of comparable size? I feel that it is prudent to expect even larger improvements. It is impossible to predict scientific breakthroughs. However, just as in other disciplines (cf. "Moore's Law"), one can observe a steady progress in integer factoring. Every few years a new algorithm appears that allows for factoring much larger integers, and then there is a steady stream of incremental improvements. As one example, there were wide concerns that the linear algebra step that plays a crucial role in most of the fast integer factorization algorithms might become a serious bottleneck when factoring large integers, since it could not be executed easily in a distributed way. However, new algorithms were developed in the last decade that have allayed these concerns.

Is it reasonable to expect a breakthrough that would enable generally hard integers to be factored in about the same time that the snfs factors integers of comparable size?

For many applications, the conjectured progress in factorization is not a serious threat. [...] However, for some records, even 20-year protection is not sufficient.

To factor a 129D integer with the continued fraction method that was used in the 1970s would have required about $6 \cdot 10^{11}$ times more computing power than to factor a 45D integer, but the factorization took only about $5 \cdot 10^6$ times as much because a much better algorithm was used. Thus here the algorithmic improvement was comparable (on a logarithmic scale) to the hardware one. This is similar to what has been reported in other areas, such as numerical analysis, where again better mathematical ideas have contributed about as much as faster computers.

Let us assume that algorithmic improvements will continue to be comparable to those from increasing computing power. Then we can expect that even a covert attack in 2004, which with present algorithms could only factor about a 768-bit integer, will instead be able to factor a 1024-bit one. For the year 2014, the threshold of what might be achievable rises to 1500 bits or even more.

Conclusions

For many applications, the conjectured progress in factorization is not a serious threat. For example, for digital signatures, time-stamping (à la Haber and Stornetta) provides a way to maintain their validity (although in a somewhat cumbersome way, requiring recourse to a document trail) even as secret moduli are factored. (This assumes, of course, that there are no totally unexpected breakthroughs, so that it is possible to estimate at any given time what are the largest integers that might be factorable in the next year, say.) Also, for many records, the security requirements are not serious, in that loss of secrecy after 10 years or sometimes even 10 days is acceptable. Hardware improvements favor the cipher designer, since a 100-fold speedup in commonly used processors allows for a 10-fold increase in the modulus in DSS (assuming, as seems reasonable, that the auxiliary prime q stays at 160 bits, or is increased to at most 240 bits), for example. However, for some records, even 20-year protection is not sufficient. In those cases extremely large moduli appear to be required for many of the most popular public key systems, such as RSA and the various ElGamal-type algorithms, such as the DSA. For extremely sensitive information, it might sometimes be prudent to use 10,000-bit moduli.

The main reason that the projections for lengths of safe moduli are growing so fast is that the asymptotic running time estimates for the latest factoring algorithms are subexponential. In particular, the growth rate for the running time of the number field sieve is not fast. By comparison, there are problems where the only known algorithms have exponential running times, such as DES and related ciphers, and also many public key schemes on elliptic curves (see Appendix D). It might therefore be prudent to consider even more seriously elliptic curve cryptosystems.

Acknowledgements

I thank Len Adleman, John Brillhart, Dan Bernstein, Marije Huizing, Hugo Krawczyk, Arjen Lenstra, Paul Leyland, Mark Manasse, Carl Pomerance, Ron Rivest, and Michael Wiener for their helpful comments.

References

[1] D. Atkins, M. Graff, A. K. Lenstra, and P. C. Leyland, The magic words are squeamish ossifrage, pp. 263-277 in *Advances in Cryptology- ASIACRYPT '94*, J. Pieprzyk and R. Safavi-Naini, eds., Lecture Notes in Comp. Sci. 917, Springer, 1995.

[2] J. Brillhart and J. L. Selfridge, Some factorizations of $2^n \pm 1$ and related results, *Math. Comp.* 21 (1967), 87-96.

[3] B. Dodson and A. K. Lenstra, NFS with four large primes: An explosive experiment, *Advances in Cryptology - CRYPTO '95*, D. Coppersmith, ed., Lecture Notes in Comp. Sci., Springer, 1995, to appear.

[4] J. J. Dongarra, H. W. Meuer, and E. Strohmaier, TOP500 supercomputer sites, report available electronically at URL <http://www.netlib.org/benchmark/top500.html>.

[5] M. Gardner, Mathematical games, *Sci. Amer.*, August 1977, 120-124.

[6] R. K. Guy, How to factor a number, in "Proc. Fifth Manitoba Conf. Num. Math.," *Congressus Num.* 16 (1976), 49-89.

[7] W. Stanley Jevons, *The Principles of Science*, 1874.

[8] A. K. Lenstra and H. W. Lenstra, Jr., eds., *The Development of the Number Field Sieve*, Lecture Notes in Math. 1554, Springer, 1993.

[9] A. J. Menezes, *Elliptic Curve Public Key Cryptosystems*, Kluwer, 1993.

[10] A. Muffett, P. Leyland, A. Lenstra, and J. Gillogly, The BlackNet 384-bit PGP key has been BROKEN, message posted to sci.crypt and other newsgroups on June 26, 1995.

[11] A. M. Odlyzko, Discrete logarithms and smooth polynomials, pp. 269-278 in *Finite Fields: Theory, Applications and*

Algorithms, G. L. Mullen and P. Shiue, eds., Amer. Math. Soc., 1994.

- [12] C. Pomerance, The number field sieve, pp. 465-480 in *Mathematics of Computation 1943-1993: A Half-Century of Computational Mathematics*, W. Gautschi, ed., Amer. Math. Soc., 1994.
- [13] C. Pomerance, J. W. Smith, and R. Tuler, A pipeline architecture for factoring large integers with the quadratic sieve algorithm, *SIAM J. Comput.* **17** (1988), 387-403.
- [14] R. L. Rivest, Dr. Ron Rivest on the difficulty of factoring, first published in *Ciphertext: The RSA Newsletter*, vol. 1, no. 1, fall 1993, and reprinted, in an updated form, in an appendix on pp. 361-364 in S. Garfinkel, *PGP: Pretty Good Privacy*, O'Reilly & Associates, 1995.
- [15] M. J. Wiener, Efficient DES key search, TR-244, May 1994, School of Computer Science, Carleton University, Ottawa, Canada. Presented at the Rump Session of Crypto '93. Available through anonymous ftp from ripem.msu.edu/pub/crypt/docs/des-key-search.ps.

Appendices

Appendix A: Historical estimates of the difficulty of factoring

The problem of factoring integers has fascinated mathematicians for a long time, and there is a famous quote of Gauss on what a fundamental question this is. A concrete estimate of how hard it might be was provided in 1874 by W. Stanley Jevons, the English economist and logician. He conjectured [7] that nobody but he would ever know the factors of the 10D integer 8616460799. However, he was proved wrong by Bancroft Brown around 1925, and perhaps by others even earlier. In an unpublished manuscript, a copy of which was kindly provided by John Brillhart, Brown explained how he obtained the factorization

$$8616460799 = 96079 \cdot 89681.$$

In 1967, John Brillhart and John Selfridge [2] stated that "... in general nothing but frustration can be expected to come from an attack on a number of 25 or more digits, even with the speeds available in modern computers." By 1970 their estimate was out of date because of a new factoring method that allowed Mike Morrison and John Brillhart to factor an integer of 39D. In 1976, Richard Guy [6] stated "I shall

be surprised if anyone regularly factors numbers of size 10^{80} without special form during the present century." He was also shown to be too cautious in a few years. The choice of a 129D integer for the RSA challenge number in 1977 [5] was apparently motivated by an estimate of Ron Rivest that to factor such an integer would require more than "40 quadrillion years" on a computer much faster than any that exist even today. However, this challenge integer was factored in 1994.

Appendix B: Historical factorization records

In 1964, there was practically no organized activity in factoring integers, and so this entry is based on the remarks in [2].

The largest generally hard integer that had been factored by 1974 was F_7 , the 7-th Fermat number, $2^{128} + 1$, which is 39D. (Today it would not be classified as "generally hard," since the special number field sieve handles numbers of this type much more efficiently than general ones. Also, as another technical point, the integer that was factored was $257 \cdot F_7$, since the use of a multiplier speeded up the algorithm.) F_7 had been factored in 1970 by Mike Morrison and John Brillhart, but their paper describing this work was only published in the 1975 D. H. Lehmer special issue of *Math. Comp.* In those days, integer factorization was not fashionable, and there was not much interest in going after records. Therefore inspection of the published literature is not adequate to assess the state of the art. Experts who were active in this area in the 1970s say that they thought that numbers of 45D were doable with the algorithms and machines available then, and so 45D is the figure used here.

1984: This is the Sandia factorization of Jim Davis, Diane Holdridge, and Gus Simmons.

1994: This is RSA129, the RSA challenge integer of 1977, which was factored by a world-wide collaborative effort organized and led by Derek Atkins, Michael Graff, Arjen Lenstra, and Paul Leyland [1].

If we also consider integers of a special form, then the 162D integer $(12^{151} - 1) / 11$ provides another record. It was factored in 1994 by Henk Boender, Joe Buhler, Scott Huddleston, Marije Huizing, Walter Lioen,

The problem of factoring integers has fascinated mathematicians for a long time

*It seems
imprudent
to assume
Moore's "Law"
will be violated
any time soon.*

*The TV set-top
boxes being
designed today
have more
computing
power than
the Cray-1,
the first
supercomputer.*

Peter Montgomery, Herman te Riele, Robby Robson, Russell Ruby, and Dik Winter.

Appendix C: Computing power of historical factorizations

According to John Brillhart, integer factorizations in the early 1970s usually used up only about an hour of cpu time on the mainframes of those days, which are typically rated at 1-10 mips.

The 1984 Sandia factorization used 9.5 cpu hours on a Cray X-MP, which is on the order of 100 mips.

1994: This is the Atkins et al. estimate [1] for the computation, which used the pmpqs algorithm. Since then a 119D integer has been factored by Scott Contini, Bruce Dodson, Arjen Lenstra, and Peter Montgomery in about 250 MY using gnfs (the general number field sieve), suggesting that today the RSA129 factorization could be carried out in about 1000 MY instead of the 5000 MY that was used. See [3] for details.

Appendix D: Current computing power

The oft-quoted figure of 30M users of the Internet is questionable, as it is obtained by assuming there are 10 users per computer. However, the estimate that about 3M machines of one kind or another are hooked up to the Internet seems much more solid. Since many of them are old, an average 10 mips rating seems reasonable for them.

There are well over 10^8 PCs in the world. Since several tens of millions of them already have the 486 chips, which are usually rated at tens of mips, the estimate of $3 \cdot 10^8$ mips is conservative. On the other hand, most of these machines are not easily usable for factoring, since they are not networked, do not have enough memory, do not have operating systems that allow for background jobs to run easily in idle time, etc. All these factors will change in a few years, but now it would be hard to harness a large fraction of all the PCs in a factoring project.

We might note that all the supercomputers in the world (about 10^3 in total) have a computing power of about $3 \cdot 10^6$ mips, with several sites having between 5% and 10% of that capacity. (This esti-

mate equates 1 megaflop with 1 mips, which is not very accurate, and is based on the June 1995 version of the TOP500 report [4].) IBM mainframes shipped in 1994 (which was a record year in terms of mainframe computing power, although not in dollar volume of sales) amounted to only $2 \cdot 10^5$ mips.

Appendix E: Moore's "Law"

There is some skepticism whether this law will continue to hold much longer. Line widths will eventually be so small that entire new technologies might be needed, architectural improvements (speculative execution, etc.) might run into the exponential complexity blowup, and so on. Even if the bare technologies are not a barrier, economics might present one, since the costs of state of the art fabrication facilities are also escalating. However, such concerns are not new, and were already present a decade ago, and yet progress has continued unhindered. Thus it seems imprudent to assume Moore's "Law" will be violated any time soon. Since state of the art microprocessors are already running close to 500 mips, the projection that by 2004 the typical processor will have a rating of 1000 mips (compared to around 10 mips today) seems safe. Beyond that, there are bigger question marks. Even if raw processor speed continues to increase, memories might be more of a bottleneck. Since most fast factorization algorithms rely on substantial memories to perform sieving operations, they are often already limited by memory system performance more than by the processor. However, this is a problem that is afflicting an increasing range of problems. Therefore there are strong incentives towards dealing with it, and again it seems imprudent to assume it will form an insurmountable barrier.

Appendix F: Number of computers

Since there will be about 10^{10} people on Earth in 2014, a projection of 10^{11} computers implies that there will be over 10 computers per person. This may seem fanciful, but we should remember that there are many embedded microprocessors in everyday appliances such as cars, dishwashers, etc., and they will be getting increasingly powerful. To provide voice recognition capability for the coffee pot will require considerable processing power. (The TV set-top boxes being designed today have more computing power

than the Cray-1, the first supercomputer. The new 32-bit video game machines, of which tens of millions are expected to be sold each year, will have similar power.) We might note, as a forerunner of what will be common, that there were two fax machines among the approximately 1600 processors in the RSA129 project.

There is still a question, even if we assume that there will be many computers around, of whether they will all be networked together, and whether their computing power will be easily accessible. Will there be computing-power brokers, selling time on millions of machines? Will people be willing to tolerate somebody else running jobs on their machines, even in spare time?

Appendix G: Running time of algorithms

In a variant of the standard notation, we define

$$L[n, v, a] = \exp(a \cdot (\log n)^v \cdot (\log \log n)^{O(1)}),$$

where $\log n$ refers to the natural logarithm of n .

The heuristic running time (there are no rigorous proofs, but experience and heuristics support the calculations) of the gnfs to factor an integer n is

$$L[n, 1/3, c_0 + O(1)] \text{ as } n \rightarrow \infty,$$

where $c_0 = (64/9)^{1/3} = 1.9229 \dots$. There is also a variant, due to Don Coppersmith, which does not seem to be practical, that allows the replacement of c_0 by $c_1 = 1.9018 \dots$. See 18,121 for presentations of the gnfs.

The special number field sieve, which factors efficiently integers of the form $a^k \pm b$, for example, where a and b are small, and k large (k can also be small, but then has to fall into certain ranges) has running time

$$L[n, 1/3, c_2 + O(1)] \text{ as } n \rightarrow \infty,$$

where $c_2 = (32/9)^{1/3} = 1.5262 \dots$.

Previous methods, such as variants of the quadratic sieve algorithm, have running times of the form

$$L[n, 1/2, 1 + O(1)] \text{ as } n \rightarrow \infty$$

The continued fraction method, which was the most widely used method in the 1970s, appears (at least for the most common variant) to have running time

$$L[n, 1/2, c_3 + O(1)] \text{ as } n \rightarrow \infty,$$

where $c_3 = 2^{1/2} = 1.4142 \dots$.

Appendix H: Comparison of running times of algorithms

The $O(1)$ terms in the estimates of running times of factorization algorithms are usually not computed explicitly. Instead, to estimate how long an algorithm with asymptotic running time $L[n, v, a + O(1)]$ should take to factor an integer n , one takes the observed running time X on an integer m and computes $X \cdot L[n, v, a] / L[m, v, a]$. This has worked well in practice. This method does ignore various important practical aspects, such as the need for memory and communication capacity as well as cpu cycles, but those have been overcome in the past either through better algorithms or general technological improvements, so it seems reasonable to continue to ignore them.

Appendix I: Exponential algorithms

Public key cryptosystems such as RSA and DSA require use of large moduli because known general algorithms for factoring integers and computing discrete logarithms are subexponential, and so hardware improvements by themselves lead to substantial progress. In addition, there have been steady and rapid improvements in algorithms. On the other hand, there are some problems in which the best algorithms known are exponential, and where there has been no recent algorithmic improvement. We cite here as an example attacks on DSA, the US Digital Signature Algorithm, that do not use the structure of the multiplicative group of integers modulo the large basic prime p . The DSA uses discrete exponentiation modulo a prime p , but the exponents are computed modulo a prime q such that q divides $p - 1$. (This is the Schnorr method that speeds up the algorithm.) Therefore all the integers are inside an abelian group of order q . To break DSA, one can either solve the general discrete log problem modulo p , which can be done using a variant of the number field

There are some problems in which the best algorithms known are exponential, and where there has been no recent algorithmic improvement

To find a single DES key will on average take about 300 times as much as the factorization of RSA 129 required.

sieve, or else work inside the group of order q . The best algorithms for the second attack are those of Dan Shanks and John Pollard, both of which take about \sqrt{q} operations. Since these \sqrt{q} operations involve multiplications mod p , though, even for q of 160 bits and p of 1024 bits or more, it would take at least 10^{14} MY on general purpose computers to implement either the Shanks or the Pollard algorithm. Hence we can conclude that 160 bits for q is likely to be safe for at least 20 years, and 200 bits (which would require at least 10^6 as much computing power to break) for much longer. (As with all the other projections about algorithms, this one could turn out to be faulty if a breakthrough occurs.)

As another example where only exponential attacks are known, we can cite elliptic curve cryptosystems [9], proposed initially by Neal Koblitz and Victor Miller. If the elliptic curve is chosen carefully, only the Shanks and Pollard methods are known for computing the analog of discrete logs on these curves. There is still some reluctance to use elliptic curve cryptosystems, though, since they have not been scrutinized as carefully as integer factorization and ordinary discrete logs.

Appendix J: Attacks on DES

For comparison, we present some estimates of the computing power needed to break DES. We consider known plaintext attacks on cipher codebook mode. We assume that only exhaustive key search will be tried, so that on average 2^{55} keys will have to be tested to find the right one. The best software implementations (written in C) appear to achieve rates of up to about 200 KB/sec on 25 mips machines (such as 50 MHz 80486 PCs), which (since each iteration of DES involves encrypting 8 bytes) corresponds to 25,000 encryptions per second, or about 1,000 encryptions per second on a 1 mips computer. Hence 1 MY allows us to test about $3 \cdot 10^{10}$ encryptions. Therefore to find a single DES key will on average take $1.2 \cdot 10^6$ MY, or about 300 times as much as the factorization of RSA129 required.

DES was made to run fast in hardware, and special purpose machines can provide substantial assistance in breaking it. Michael Wiener [15] has proposed a pipelined parallel computer that could be built for about \$1.5M (both development and construction

cost) and would find a single DES key in about 4 hours. What this shows is that special purpose hardware can be of great help in breaking DES. On the other hand, general integer factorization and discrete logarithm algorithms do not benefit that much from special designs, and it is the threat of the free computing power on the Internet that seems most serious. Special designs for sieve processors for factoring have been proposed (see [13]), but the economics of the electronics industry favors general purpose computers. (Fast parallel modular multiplication units could be of use in the implementation of the Pollard and Shanks exponential-time algorithms, though, or of the subexponential-time elliptic curve method.)

RSA Laboratories Minimum Key Size Recommendations

It is clear from articles like Andrew Odlyzko's in this issue of *CryptoBytes*, that considerable thought is required in choosing the size of a modulus used in an implementation of RSA. Balancing the issues of security against performance is difficult with any cryptosystem, but it is particularly difficult when one wants to make allowances for potential cryptanalytic developments.

Currently, RSA Laboratories make the following recommendations for the size of RSA moduli:

User keys, short-term security	768 bits
Organizational keys, medium-term security	1024 bits
Root keys, long-term security	2048 bits

Since developments in factoring can be very unpredictable, implementations of RSA should ideally allow for variable key sizes where at all possible.

On the Security of the RC5 Encryption Algorithm

Burt Kaliski and Yiqun Lisa Yin

RSA Laboratories
100 Marine Parkway, Suite 500
Redwood City, CA 94065 USA

In this article, we give a brief report on the security of the RC5 encryption algorithm 151 against three different types of attack including exhaustive search, differential cryptanalysis 111, and linear cryptanalysis 131. RC5 is a new block cipher recently designed by Ron Rivest. It has a variable block size, a variable number of rounds, and a variable-length secret key. The secret key is used to fill an expanded key table which is then used in encryption. A detailed description of the RC5 encryption algorithm was provided in the Spring issue of *CryptoBytes* [6].

To attack RC5, we can try to find either the original secret key or the expanded key table. Clearly, if the latter approach is used, the attack is independent of the length of the secret key.

The secret key used in RC5 has a variable length with allowed values from 0 to 2,040 bits and the expanded key table for RC5 with rounds has $2^{5(2r+2)}$ bits (for the 64-bit block size). Hence, if both the length of the secret key and the number of rounds are sufficiently large, RC5 is secure against exhaustive search.

Differential and linear cryptanalysis are two powerful techniques developed in recent years for analyzing the security of block ciphers. For differential cryptanalysis, the basic idea is that two chosen plaintexts P and P' with a certain difference $P' \oplus P = \Delta$ provide two ciphertexts C and C' such that $C' \oplus C = \Gamma$ has a specific value with non-negligible probability; such a “characteristic” (Δ, Γ) is useful in deriving certain bits of the key. For linear cryptanalysis, the basic idea is to find linear approximations (parity relations among certain bits of plaintext, ciphertext, and key) which hold with probability $\frac{1}{2} \pm \epsilon$ (i.e., bias $\neq 0$); such approximations can be used to obtain information about the key.

Burt Kaliski is chief scientist and Yiqun Lisa Yin is a research scientist at RSA Laboratories. They can be contacted at burt@rsa.com and yiqun@rsa.com.

We have developed differential and linear attacks [2] on RC5 that are quite effective when the number of rounds is very small. Both attacks recover every bit of the expanded key table. However, the plaintext requirement is strongly dependent on the number of rounds, and the requirement for RC5 with 64-bit block size is summarized in the following table.

Table 1

rounds	Plaintext Requirements					
	4	5	6	7	9	12
differential cryptanalysis	2^{22}	2^{26}	2^{32}	2^{37}	2^{46}	2^{63}
linear cryptanalysis	2^{37}	2^{47}	2^{57}	$>2^{64}$		

The chosen plaintext requirements for differential cryptanalysis of 64-bit RC5 with the indicated number of rounds are shown in the first row of the table. The known plaintext requirements for linear cryptanalysis are shown in the second row.


One can see that for the 64-bit block size, our differential attack on nine-round RC5 uses 2^{46} chosen plaintexts (about the same as DES [4]); the plaintext requirement becomes increasingly impractical for more rounds. Similarly, our linear attack on five-round RC5 uses 2^{47} known plaintexts (about the same as DES) and the plaintext requirement increases rapidly with additional rounds.

We now briefly describe the idea used in the two attacks on RC5. Since RC5 is iterative, if we can derive the subkey in the last round, we can derive all the subkeys in the expanded key table. In our differential attack, the characteristics for each round have the property that the differences in the pair of inputs do not affect the rotation amounts and they can easily be joined together. Moreover, the characteristics for the last round have a special form, allowing us to recover the subkey in the last round. In our linear attack, the linear approximations relate the least significant bits of the input, the output, and the subkey in each round. When enough plaintext/ciphertext pairs are obtained, the experimental biases of the approximations reveal the subkey in the last round. There is strong evidence that the characteristics and the linear approximations used in our attacks are close to optimal. Thus, Rivest’s suggested use of 12 rounds for RC5 with a 64-bit block size is sufficient to make differential and linear cryptanalysis impractical.

Rivest’s suggested use of 12 rounds for RC5 with a 64-bit block size is sufficient to make differential and linear cryptanalysis impractical.

Our analysis shows that data-dependent rotations are very helpful for preventing differential and linear attacks.

In sum, we can conclude that RC5 provides good security against all three types of attacks when both the length of the secret key and the number of rounds are sufficiently large. Unlike DES, RC5 is a parameterized algorithm, giving flexibility in the level of security. As a next step, we will consider other possible forms of analysis and study the key expansion algorithm of RC5.

Finally, we want to point out two distinguishing features of RC5. The first feature is the heavy use of data-dependent rotations. Our analysis shows that data-dependent rotations are very helpful for preventing differential and linear attacks. The second feature is the exceptional simplicity of the encryption algorithm. Such a simple design makes the analysis easier and will help fully determine the security of RC5 in a rather rapid way. 

References

[1] E. Biham and A. Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. Springer-Verlag, 1993.

[2] B.S. Kaliski and Y.L. Yin. On differential and linear cryptanalysis of the RC5 encryption algorithm. To appear at Crypto '95.

[3] M. Matsui. The first experimental cryptanalysis of the Data Encryption Standard. In Y.G. Desmedt, editor, *Advances in Cryptology—Crypto '94*, pages 1-11, Springer-Verlag, 1994.

[4] National Institute of Standards and Technology (NIST). *FIPS Publication 46-2: Data Encryption Standard*. December 30, 1993.

[5] R.L. Rivest. The RC5 encryption algorithm. In *Proceedings of the Workshop on Cryptographic Algorithms*, K.U. Leuven, December 1994. To appear.

[6] R.L. Rivest. The RC5 encryption algorithm. *CryptoBytes*, 1(1), 9-11, Spring 1995.

A L G O R I T H M S U P D A T E

MD5 Performance for IP Security Questioned

A paper to be presented at ACM SigComm '95 questions whether the MD5 message-digest algorithm is fast enough to handle packets on high-speed Internet links.

The paper, "Performance Analysis of MD5" by Joe Touch of ISI, argues that with today's hardware MD5 can achieve at best a speed of 256 million bits/second (Mbps), not enough to keep up with Asynchronous Transfer Mode (ATM) data rates of up to 622 Mbps. Though hardware performance will likely improve over time, so will network requirements, keeping MD5 behind.


MD5 has been proposed as a basis for authenticating messages in version 6 of the Internet Protocol (IPv6), where a message and a secret key are hashed together to form an authentication code that prevents intruders from modifying messages. As such, it could be required to handle packets at network speeds.

A significant reason for the limitation in performance is that MD5 has an iterative design where

message blocks are processed one after another, each one being combined with the result of previous computation through a compression function. Thus the only hardware optimization possible is within the compression function; it is not possible to process multiple blocks at the same time with additional hardware. Moreover, the compression function itself is complex and somewhat hard to parallelize. Many other hash algorithms including NIST's Secure Hash Algorithm (SHA) are subject to the same limitations.

Alternatives recommended by Touch include tree-based hashing; and internal modifications to the algorithm.

MD5 remains sufficient for many applications, such as hashing as part of a digital signature algorithm. Indeed, Touch reports software performance for MD5 ranging from 31 Mbps on a 66MHz Intel '486 to 87 Mbps on a 190 MHz DEC Alpha. On such processors, files even as large as 10 Mbytes can be hashed with MD5 in only a few seconds.

A report by Touch summarizing these results is available as Internet RFC 1810. 

T O T H E E D I T O R

Further Comments on Keyed MD5

This note follows on the excellent overview by Burt Kaliski and Matt Robshaw ("Message authentication with MD5," *CryptoBytes* vol. 1 no.1) in which three schemes are recommended to the IPSEC working group. Citing forthcoming work, it was suggested the best attack (forgery) on these schemes required 2^{64} chosen message texts ("... except when the known messages are all the same length and end with the same suffix").

We have improved this attack in our recent paper^(*) "MDx-MAC and building fast MACs from hash functions," *Proc. Crypto'95*. A generic attack is given requiring 2^{64} known text-MAC pairs and a single chosen text, independent of message lengths or suffixes — only for the second recommended scheme we need the messages to be of the same length. (Perhaps more significantly, the attack applied to CBC-MAC requires only 2^{32} known text-MAC pairs for MAC forgery.) The attack requires an additional 2^{64} chosen text-MAC pairs if only 64 bits of the hash result are retained; this suggests modifying the method to retain only 64 bits, which also saves bandwidth. The number of text-MAC pairs required can be further reduced if the known messages contain a common (not necessarily chosen) sequence of trailing blocks. The attack also applies if messages are fixed-length or prepended by length fields.

Adapting the same attack strategy allows a divide-and-conquer attack if the envelope method is used with distinct front and tail keys, effectively reducing security to the larger of the two. We also provide analysis of the secret prefix and secret suffix methods, and add here that the secret suffix method is subject to an off-line, memoryless, parallelizable attack requiring 2^{64} operations and a single chosen text (P. van Oorschot and M. Wiener, ACM-CCS'94, Fairfax).

Recent partial attacks on MD4, MD5, and the related RIPEMD, including in particular those of S. Vaudenay (Leuven Algorithms Workshop Dec.'94)

and H. Dobbertin (Rump Session, Eurocrypt%), suggest these functions are susceptible to manipulations of their internal structures. This raises concerns about hash-based MACs being susceptible to attacks exploiting properties of the underlying hash. We therefore advise caution in constructing such MACs, and recommend a design more conservative than the envelope method. We agree customized MACs may be preferable, but are reluctant to discard the experience gained over time with MD4 and MD5.

With exquisite timing, our paper already (as submitted Feb.'95) makes a proposal in line with most of the suggestions of Kaliski and Robshaw: MD5-MAC, a customized MAC involving key processing at every compression function step, and built with only minor modifications from MD5 (to minimize the likelihood of introducing new flaws). The same construction yields MACs based on any of MD5, SHA, or RIPEMD.

In addition to being more conservative than the envelope method, only slightly slower (5-20%, depending on processor and implementation), and easily implemented from MD5, the theoretical underpinnings supporting the security of the envelope method, which assume the compression function of MD5 is pseudorandom, appear to similarly apply to MD5-MAC. We caution, however, that we are aware of no results regarding the pseudorandomness of MD5, and note this property may be independent of collision-resistance, the primary property studied to date.

— Bart Preneel, Katholieke Universiteit Leuven
bart.preneel@esat.kuleuven.ac.be
Paul C. van Oorschot, Bell-Northern Research
paulv@bmr.ca

(*) the paper is available by FTP at ftp.esat.kuleuven.ac.be, in the directory pub/COSIC/preneel.

We welcome comments from our readers at any time. In particular we are keen to expand and keep up-to-date issues that have previously appeared in *CryptoBytes*. Comments can be sent via E-mail to bytes-ed@rsa.com.

We therefore advise caution [...], and recommend a design more conservative than the envelope method.

A N N O U N C E M E N T S

RSA Laboratories Technical Reports


The RSA Laboratories Technical Reports are now available via a subscription service. These reports offer detailed summaries of current research on a variety of topics and they bring together information from a wide variety of sometimes obscure sources. Subscription to the Technical Reports will be at one of two levels: individual or corporate. As well as receiving all previously written reports, subscribers will receive new reports as they appear as well as current research notes on items of major significance.

Technical reports that are currently available include recently updated surveys of block ciphers, stream ciphers and the MD family of hash functions. Other reports offer substantial information on such topics as the RSA Factoring Challenge, fair cryptography and the software implementation of RSA. In immediate prepara-

tion are reports on the hardware implementations of RSA, an overview of electronic payment systems and an internal assessment of the security of the new RC5 encryption algorithm.

Contact RSA Laboratories for more information about the RSA Laboratories Technical Report subscription service.

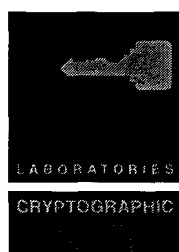
The 1996 RSA Data Security Conference

The 1996 RSA Data Security Conference will be held January 17-19 in the Fairmont Hotel, San Francisco. In addition to daily keynote speeches, the conference will include separate strategic, development and cryptography tracks which the expected 800-1000 attendees will be able to mix and match at will. More information can be found on RSA's web page (<http://www.rsa.com>) or by contacting the conference organizer, Layne Kaplan Events, at 415/340-9300. 

In this issue:

- Elliptic curve cryptosystems
- The future of integer factorization
- On the security of the RC5 encryption algorithm

For subscription information, see page 2 of this newsletter.



100 MARINE PARKWAY
REDWOOD CITY
CA. 94065-1031
TEL 415/595-7703
FAX 415/595-4126
rsa-labs@rsa.com

PRESORT
FIRST CLASS
U.S. POSTAGE
PAID
MMS, INC

RSA LABORATORIES' CryptoBytes

The technical newsletter of RSA Laboratories, a division of RSA Data Security, Inc.

Contents

1

*The Impending
Demise of RSA?*

2

*Welcome to
CryptoBytes*

5

*Message Authentication
with MD5*

9

*The RC5
Encryption Algorithm*

11

News and Information

12

Announcements

The Impending Demise of RSA?

Gilles Brassard

Département d'informatique et de R.O.
Université de Montreal
C.P. 6128, Succursale Centre-Ville
Montreal (Québec)
CANADA H3C 3J7

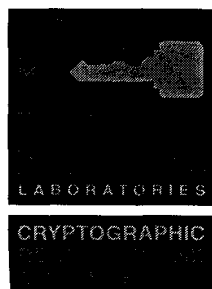
In August 1977, Rivest, Shamir and Adleman issued a ciphertext challenge worth one hundred dollars to *Scientific American* readers when Martin Gardner described their revolutionary RSA cryptographic system in his monthly "Mathematical Games" column. This sounded very safe because it was estimated at the time that the fastest existing computer using the most efficient known algorithm could not earn the award until it had run without interruption for millions of times the age of the Universe. This particular challenge involved the factorization of a 129-figure number r , which appeared to be so out-of-reach at the time that Martin Gardner reported: "Rivest and his associates have no proof that at some future time no one will discover a fast algorithm for factoring composites as large as r [...]. They consider [the] possibility extremely remote." Nevertheless, the \$100 reward was cashed last year and donated to the Free Software Foundation after a mere eight months of intensive computation led by Derek Atkins and Arjen Lenstra. What happened?

Professor Gilles Brassard, Université de Montreal, is interested in all aspects of cryptology but perhaps his best-known contribution is as a co-developer of Quantum Cryptography. He can be contacted at brassard@iro.umontreal.ca. This essay was written while the author was on sabbatical at the University of Wollongong, Australia. Research supported in part by Canada's NSERC and Québec's FCAR.

The increase in raw computing power during those years cannot be discounted, nor can the fact that hundreds of workstations around the world spent all their otherwise idle cycles on the task for the better part of one year. Indeed, the several thousand MIPS-years that were spent on the calculation might not have been available to poor academics back in 1977. But in the Preface of my new textbook *Fundamental of Algorithmics*, soon to be released by Prentice-Hall with Paul Bratley as coauthor, I am quick to point out that far more significant was the discovery of more sophisticated factorization algorithms. When I put on my hat as teacher of algorithmics, I like to use this example to illustrate that more efficient algorithms can produce much more dramatic results than better hardware. To make life more interesting, however, I am wearing quite a different hat as I write this essay!

Even though the "double large prime multiple polynomial variation of the quadratic sieve" algorithm was successful in factoring the 129-figure number relevant to the RSA challenge, the age of the Universe would still not suffice to factor a 500-figure number by this or any other known classical algorithm (such as the number field sieve) even if all the world's computers were put to contribution. Therefore, one should not infer from the fate of the 1977 challenge that RSA has been broken: the lesson is that bigger numbers should be used and that overconfident claims should be avoided. Clearly, even more remarkable advances in algorithmics will be required if RSA is to fail completely or even if it is to fail on keys merely twice the size used in the *Scientific American* challenge. Progress in hardware

(continued on page 3)



Welcome to CryptoBytes

Welcome to the first issue of *CryptoBytes*, the technical newsletter on cryptography from RSA Laboratories.

One of the nicest features of cryptographic research is the speed with which developments occur. This is

a primary reason why so many researchers find working in the field so rewarding. More often than not, however, new results or second-hand accounts of someone's valued opinion circulate for months by word of mouth or by E-mail before appearing in journals or in conference proceedings. In fact, it might be convincingly argued that a great deal of interesting information is never actually

published, never cited, and never properly referred to by other researchers.

A newsletter can alleviate this situation. The aim would be to circulate interesting news as it happens, thereby providing a reliable distribution method for substantial 'bites of Crypto'. In addition, a newsletter might provide a forum for results or opinions that, while of great cryptographic interest, would not appear at any of the more classical outlets because of their format.

Such a newsletter should be of interest to all those involved in cryptography; from those implementing cryptographic techniques and designing cryptographic products to those in the academic development of cryptographic knowledge. Often these groups are viewed as being somewhat exclusive of each other; instead we suggest that there is an important symbiosis. One goal of a newsletter on cryptography must be the transfer of information across these artificial divides. In this way researchers will hear about continuing efforts to implement the fruits of their research efforts and implementers can keep track of the latest cryptographic innovations.

With the first issue of *CryptoBytes* in your hands, we are hoping to achieve some of these goals. We are also hoping to provide a complement to current newsletters such as IEEE's *Cipher*, the IACR newsletter and the TIS *Data Security Letter*, among many others.

Much of the future success of *CryptoBytes* will depend on input from outside of RSA Laboratories. Such input might range from invited articles and researchers providing notification of recent results and developments, through letters and opposite opinions from readers. While RSA Laboratories will coordinate *CryptoBytes*, the intention is for it to become a useful resource for the whole cryptographic community. To help in this process, back issues of *CryptoBytes* will be available free of charge via the World-Wide Web.

We hope that you'll agree that this first issue of *CryptoBytes* is a step towards our goals. We would very much like to thank the writers who have contributed to this first issue, and we welcome any comments, suggestions or proposals for future issues.

-Ron Rivest

Editor's note: Suggestions and contributions for future issues of *CryptoBytes* can be sent to bytes-ed@rsa.com or to RSA Laboratories by any of the methods given below.

Subscription Information

CryptoBytes is published four times annually; printed copies are available for an annual subscription fee of U.S. \$90. To subscribe, contact RSA Laboratories at:

RSA Laboratories
100 Marine Parkway, Suite 500
Redwood City, CA 94065
415/595-7703
415/595-4126 (fax)
rsa-labs@rsa.com

Back issues in electronic form are available via the World-Wide Web at
<http://www.rsa.com/rsalabs/cryptobytes/>.

RSA Laboratories is the research division of RSA Data Security, Inc., the company founded by the inventors of the RSA public-key cryptosystem. RSA Laboratories reviews, designs and implements secure and efficient cryptosystems of all kinds. Its clients include government agencies, telecommunications companies, computer manufacturers, software developers, cable TV broadcasters, interactive video manufacturers, and satellite broadcast companies, among others.



Prof. Ronald Rivest is co-inventor of the RSA public-key cryptosystem, a co-founder of RSA Data Security Inc. and a distinguished associate of RSA Laboratories.

The Impending Demise of RSA?
Continued from page 1

will at best be a minor factor in the eventual success of future attacks against RSA. Right? Wrong!

Quantum computing, an emerging branch of computer science, may well prove the above conventional wisdom false. For the first time, revolutionary new concepts may hold the key to a computer that would go *exponentially* faster than conventional computers, at least for some computational tasks. This means that the speed-up would be increasingly spectacular as the size of the input gets larger, which is precisely the type of claim that had been the prerogative of algorithmic improvements until now. In particular, building on the work of David Deutsch and Richard Jozsa, Ethan Bernstein and Umesh Vazirani, Daniel Simon, and Don Coppersmith, Peter Shor has discovered that quantum computers can factor an n -figure number in a time asymptotically proportional to $n^{2+\epsilon}$ for arbitrarily small ϵ . This means that it would take about the same time to crack an RSA key as to use it legitimately! In other words, quantum computers spell complete disaster on RSA. This has not (yet) forced RSA Laboratories to file for Chapter 11 because there are formidable technological difficulties before the first quantum computer can be built, but the possibility should not be underestimated.

What is a quantum computer? This theoretical notion emerged from the work of Paul Benioff, Richard Feynman and David Deutsch in the first half of the eighties. I cannot say much in this short essay but I shall try to sketch the basic principles. For more detail and references to the work mentioned *here*—such as Peter Shor’s quantum factorization algorithm—I invite you to read my forthcoming paper in *Current Trends in Computer Science*, Jan van Leeuwen (Ed.), Lecture Notes in Computer Science, Volume 1000 (special anniversary volume), Springer-Verlag, 1995. Until this volume has appeared, you may wish to read my earlier account in *Sigact News*, Volume 25, number 4, December 1994, pp. 15 – 21.

Let us begin with a quantum bit, or *qubit* (a word coined by Benjamin Schumacher). In classical digital computing, a bit can take either value 0 or value 1. Nothing in between is allowed. In quantum computing, a qubit can be in *linear superposition* of the two classical states. If we denote the classical states by $|0\rangle$ and $|1\rangle$, then a qubit can be in state $\psi = \alpha|0\rangle + \beta|1\rangle$ for arbitrary complex numbers α and β

subject to $|\alpha|^2 + |\beta|^2 = 1$. The coefficients α and β are called the *amplitudes* of $|0\rangle$ and $|1\rangle$, respectively. A qubit is best visualized as a point on the surface of a unit sphere whose North and South poles correspond to the classical values. If state ψ is *observed* in the sense that the qubit is asked to assume a classical value, it will *collapse* onto $|0\rangle$ with probability $|\alpha|^2$ and onto $|1\rangle$ with complementary probability $|\beta|^2$. So far, it looks as if we have but *reinvented* analogue computation. Things become more interesting when we consider quantum *registers* composed of n qubits. Such registers can be set to an arbitrary quantum superposition of states $\Psi = \sum_{x \in X} \alpha_x |x\rangle$ subject to $\sum_{x \in X} |\alpha_x|^2 = 1$, where X denotes the set of all classical n -bit strings. If this register is asked to assume a classical value, each x in X will be obtained with probability $|\alpha_x|^2$, and the register will collapse onto the observed value.

In principle, it is possible to compute on such registers. If a quantum computer is programmed to compute some function $f : X \rightarrow Y$ and if it is started with superposition $\Psi = \sum_{x \in X} \alpha_x |x\rangle$ in its input register, then it will produce superposition $\Psi' = \sum_{x \in X} \alpha_x |f(x)\rangle$ in its output register *in the time needed to compute f on a single input*. In other words, this provides for *exponentially* many computations to take place simultaneously in a single piece of hardware, a phenomenon known as *quantum parallelism*. We are far from *analogue* computing now. The good news is that we have obtained exponentially many answers for the price of one. The bad news is that Heisenberg’s uncertainty principle forbids us from looking at the output register for fear of spoiling it! More precisely, if we ask the output register to assume a classical value, it will collapse to the value of $f(x)$ for an x randomly chosen in X with probability $|\alpha_x|^2$, and the quantum superposition Ψ' will be destroyed by the measurement. So far, it looks as if quantum computing is not only impractical but useless as well.

What makes quantum computing interesting is the notion of *quantum interference*, which is exactly the principle behind Young’s celebrated double-slit experiment. In a classical probabilistic calculation, it is possible to program the computer to select one of several possible computation paths according to the laws of probability. In any specific instance of the calculation, however, only one of the potential paths is actually taken, and *what-could-have-happened-*

Progress in hardware will at best be a minor factor in the eventual success of future attacks against RSA. Right? Wrong!

Revolutionary new concepts may hold the key to a computer that would go exponentially faster than conventional computers.

It is possible to program a quantum computer so that all potential computation paths are taken simultaneously in quantum superposition.

I like to think that I shall see a special-purpose quantum factorization device in my lifetime.

but-did-not has no influence whatsoever on the actual outcome. In contrast, it is possible to program a quantum computer so that all potential computation paths are taken simultaneously in quantum superposition. What makes this so powerful-and mysterious-is the exploitation of constructive and destructive interference phenomena, which allows for the reinforcement of the probability of obtaining desired results at the expense of the probability of obtaining spurious results. This happens because the amplitude of reaching any given state is the sum of the amplitudes of reaching this state by all possible computation paths. Because amplitudes can be negative (even complex), the amplitude of reaching an undesirable result from one path can be annihilated by the amplitude of reaching it from another path. In the words of Richard Feynman, "somehow or other it appears as if the probabilities would have to go negative." It is not easy to put such interference phenomena to practical use, but Peter Shor's *tourdeforce* proves that it is possible, at least in principle.

The actual implementation of a quantum computer will be very challenging. It may even turn out to remain forever out of reach of human technology. One difficulty is to keep *quantum coherence* in the computer. The problem is that coherent superpositions are very fragile: they spontaneously turn into incoherent statistical mixtures-which are unsuitable for quantum parallelism-because of residual interactions with the environment. Consequently, quantum information disappears gradually from the computer. Rolf Landauer has pointed out the additional problem that quantum computation may be susceptible to spontaneous reversals, which would unpredictably cause the computation to go backwards. Yet another difficulty is that of error correction despite early work by Asher Peres and more recent work by Andre Berthiaume, David Deutsch and Richard Jozsa: in classical digital computing, discrete states are continuously snapped back onto the proper voltages for 0 and 1; no similar process is available for quantum computing even if the program is such that the legitimate states have discrete amplitudes. As a result, the computer may drift away from its intended state.

Nevertheless, Seth Lloyd has promising ideas on how to build "a potentially realizable quantum computer." Furthermore, it was discovered by David DiVincenzo that universal quantum computation can be based on

simple 2-bit quantum gates such as the natural quantum extension of the classical exclusive-or. Experimental physicists such as Serge Haroche and Jean-Michel Raimond are already attempting to build simple quantum gates based on atomic interferometry and microwave cavities capable of trapping single photons for a significant fraction of one second. Even though a full-fledged quantum computer may take a long time to come, I like to think that I shall see a special-purpose quantum factorization device in my lifetime. If this happens, RSA will have to be abandoned. Most other practical public-key systems will be compromised as well because Peter Shor has also devised a quantum algorithm for extracting discrete logarithms.

But do not despair for the fate of cryptography: there will always be quantum cryptography to come to the rescue! Quantum cryptographic systems take advantage of the uncertainty principle, according to which measuring a quantum system in **general** disturbs it and yields incomplete information about its state before the measurement-which is precisely what makes quantum computers difficult to program. When information is encoded with non-orthogonal quantum states, any attempt from an eavesdropper to access the information necessarily entails a probability of spoiling it irreversibly, which can be detected by the legitimate users. Using protocols that I have designed with Charles H. Bennett, building on earlier work of Stephen Wiesner, this phenomenon can be exploited to implement a key distribution system that is secure even against an eavesdropper with unlimited computing power, indeed even against an eavesdropper who has a quantum computer at her disposal! Several prototypes have been built in recent years. In particular, British Telecom announced in September 1994 the successful completion of their fully working apparatus, perfected by Paul Townsend, capable of implementing quantum key distribution over 10 kilometres of ordinary optical fibre. For more information on quantum cryptography, please consult my article in the March 1995 sesquicentennial Anniversary Edition of *Scientific American* on "The Computer in the 21st Century" (reprinted with updates from their October 1992 regular issue).

To paraphrase the Book of Job,
*The quantum taketh away
and the quantum giveth back!*



Message Authentication with MD5

Burt Kaliski and Matt Robshaw

RSA Laboratories
100 Marine Parkway, Suite 500
Redwood City, CA 94065 USA

Message authentication is playing an important role in a variety of applications, especially those related to the Internet protocols and network management, where undetected manipulation of messages can have disastrous effects.

There is no shortage of good message authentication codes, beginning with DES-MAC, as defined in FIPS PUB 113 [7]. However, message authentication codes based on encryption functions such as DES, which were designed for hardware implementation, may be somewhat limited in performance for software, and there is also the question of U.S. export restrictions on encryption functions.

In standards efforts such as the Simple Network Management Protocol [5] and proposals for Internet Protocol security, a more practical solution seemed to be to base the authentication codes not on DES but on hash functions designed for fast software implementation which are widely available without restriction, such as the MD5 message-digest algorithm [9].

But how to do it? Hash functions are intended to resist inversion — finding a message with a given hash value — and collision — finding two messages with the same hash value. Message authentication codes, on the other hand, are intended to resist forgery — computing a message authentication code without knowledge of a secret key. Building a message authentication code on an encryption function thus seems a logical choice (and the security relationship has been recently settled — in work by Mihir Bellare, Joe Kilian and Phillip Rogaway [3]). Building one on a hash function, however, is not as simple, because the hash function doesn't have a key.

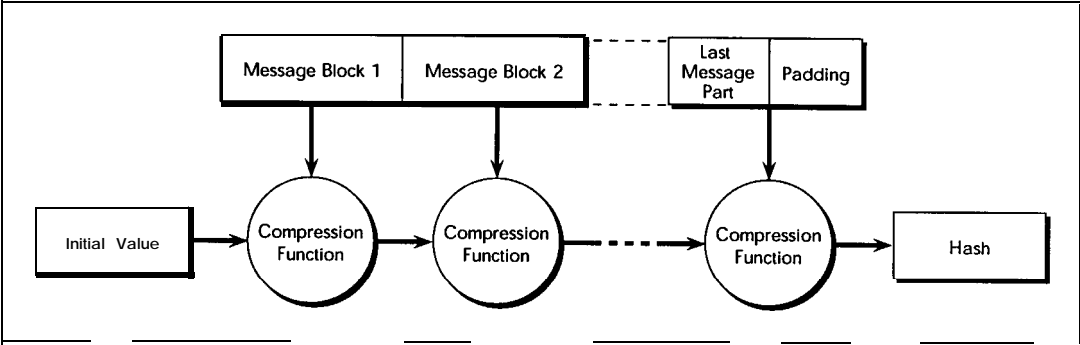
Burt Kaliski is chief scientist and Matt Robshaw is a research scientist at RSA Laboratories. They can be contacted at burt@rsa.com or matt@rsa.com.

A hash function can provide message authentication in a most satisfying manner when combined with a digital signature algorithm, which does have a key. But typical digital signature schemes have some performance overhead, which while acceptable for the periodic setup of communications sessions, is often too large on a message-by-message basis. Thus, the focus is on message authentication based on a shared secret key, which is ideally integrated into the hash function in some manner.

As an illustration of the challenges, consider the “prefix” approach where the message authentication code is computed simply as the hash of the concatenation of the key and the message, where the key comes first and which we denote as MD5 ($k \cdot m$).

MD5 follows the Damgård/Merkle [4,6] iterative structure, where the hash is computed by repeated application of a compression function to successive blocks of the message. (See Figure 1.) For MD5, the compression function takes two inputs — a 128-bit chaining value and a 512-bit message block — and produces as output a new 128-bit chaining value,

A more practical solution seemed to be to base the authentication codes [...] on hash functions.



which is input to the next iteration of the compression function. The message to be hashed is first padded to a multiple of 512 bits, and then divided into a sequence of 512-bit message blocks. Then the compression function is repeatedly applied, starting with an initial chaining value and the first message block, and continuing with each new chaining value and successive message blocks. After the last message block has been processed, the final chaining value is output as the hash of the message.

Figure 1. Damgård/Merkle iterative structure for hash functions.

Because of the iterative design, it is possible, from only the hash of a message, to compute the hash of longer messages that start with the initial message

Figure 2.
A recommended
approach to
message authentication
with MD5.
Here, the keys are
each 128 bits long.
They may be the
same, although
different keys are
preferable.

and include the padding required for the initial message to reach a multiple of 512 bits. Applying this to the prefix approach, it follows that from $MD5(k \cdot m)$, one can compute $MD5(k \cdot m')$ for any m' that starts with $m \cdot p$, where p is the padding on $k \cdot m$. In other words, from the message authentication code of m , one can forge the message authentication code of $m \cdot p \cdot x$ for any x , without even knowing the key k , and without breaking MD5 in any sense. This is called a "message extension" or "padding" attack^[10].

Other hash functions with an iterative design, such as NIST's Secure Hash Algorithm [8], are also vulnerable to the message extension attack, and similar attacks can also be mounted on tree-structured designs.

(Note also that if only part of the hash were output, say only 64 bits, this attack would not be possible; however, this is not a completely satisfying solution

because of other concerns raised below. In SNMP, the message extension attack is not a problem because messages are a fixed length. Another way to avoid the attack is to include an explicit length field at the beginning of the message.)

Because of the message extension attack on the prefix approach, the "suffix" approach, $MD5(m \cdot k)$, would seem to be preferred. But another problem arises: the key may be vulnerable to cryptanalysis, depending on the properties of the compression function. This is because the message authentication code is a function

of known values and the key, assuming the key is passed entirely to the last iteration of the compression function. (The known values are the next-to-last chaining value, which by assumption depends

only on the message; the last part of the message; and the padding.)

An opponent who sees the message authentication codes for many messages thus sees the result of applying the compression function to many different known values and the same key, which may reveal information about the key. While our analysis suggests MD5's compression function is unlikely to reveal information about the key, other hash functions may not fare as well, and so we prefer a more robust design.

The prefix approach is also affected by these issues, but only when the message is very short and there is only a single iteration of the compression function.

Recommendations

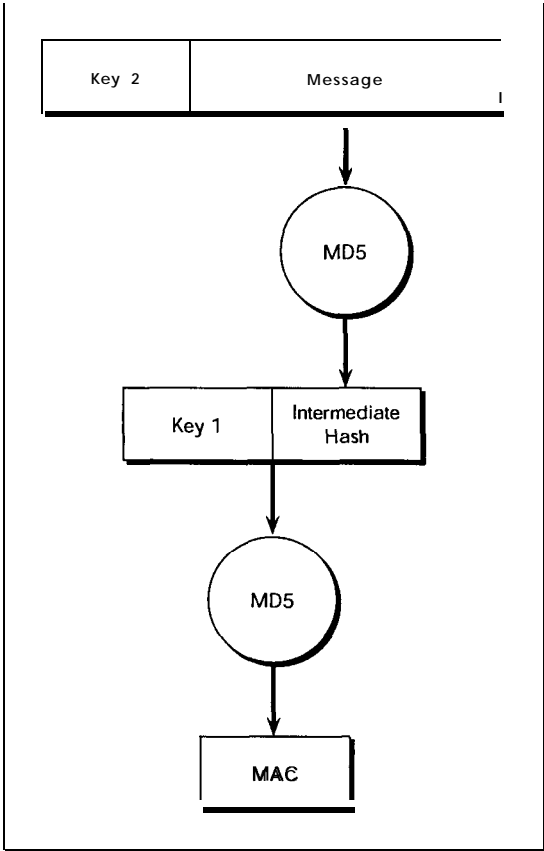
In joint work with Mihir Bellare and Hugo Krawczyk of IBM, we have considered a number of approaches to message authentication with MD5, settling on three which we recommended to the Internet Protocol Security (IPSEC) working group:

- 1. $MD5(k_1 \cdot MD5(k_2 \cdot m))$, where k_1 and k_2 are independent 128-bit keys
- 2. $MD5(k \cdot p \cdot m \cdot k)$, where k is a 128-bit key and p is 384 bits of padding
- 3. $MD5(k \cdot MD5(k \cdot m))$, where k is a 128-bit key

The first and third approaches (see Figure 2) are similar, and solve the message extension attack on the prefix approach by the outer application of MD5, which conceals the chaining value which is needed for the attack. The outer MD5 also solves the concerns of cryptanalysis of the suffix approach, because the message authentication code is a function of the unknown secret key and other varying values, which are unknown. These approaches also approximate certain "provably secure" constructions developed by Bellare, Ran Canetti and Krawczyk [1].

(As a disclaimer, we can imagine hash functions for which this construction still doesn't solve the cryptanalytic problems because information from the inner application leaks to the outer one, but this seems more of a pathological case.)

The third approach may be more vulnerable to attack than the first since there is only one key and so



any information revealed from the outer application of the hash function compromises security, but we know of no such attack on MD5.

Although the third approach has a shorter key size than the first, the first could also be implemented with a 128-bit key, without any apparent loss in security. For instance, the keys k_1 and k_2 could be derived from a single 128-bit key k as $k_1 = \text{MD5}(k \cdot \alpha)$ and $k_2 = \text{MD5}(k \cdot \beta)$, where α and β are distinct constants.

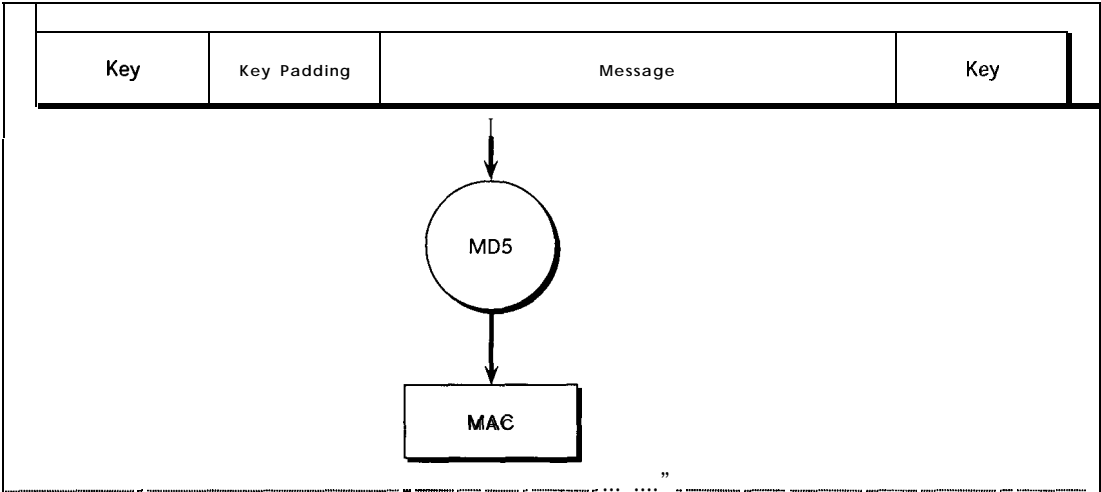
The second approach (see Figure 3) is somewhat like triple encryption, where the first and third keys are the same (the second key is the message). The padding on the key at the beginning ensures that overall, there are at least two iterations of the compression function. Message extension in the prefix approach is solved by the key at the end, and the cryptanalysis of the suffix approach is solved by the key at the beginning. (Without the padding, very short messages might be vulnerable.)

It remains to be seen which, if any, of these three approaches is adopted.

Interestingly, one of the approaches we had been previously promoting is not among the three we recommended to the IPSEC working group, based on our concerns about key exposure. That approach, $\text{MD5}(k \cdot \text{MD5}(m))$, had the advantages that the inner MD5 is applied to the message in the familiar way — as a hash function — and the outer MD5 is applied to a fixed-length value, thereby avoiding message extension. However, since $\text{MD5}(m)$ is known, the door is open for possible cryptanalysis of the outer MD5 to recover the key k . While we once again do not have an attack that recovers the key, we felt as a general design principle that the key should be better concealed.

(Another concern with this approach, observed by some, is that collisions in MD5 — two messages with the same hash — result in collisions in the message

authentication code. We do not consider this an intrinsic problem with this option, since MD5 is designed to resist collisions, at least to a certain level of difficulty. Nevertheless, we have no objection if the design of a message authentication code raises that level even further.)



Yet another approach that we considered was $\text{MD5}(\text{MD5}(k \cdot m))$, which again applies MD5 in a familiar way. However, in terms of “provability” under certain assumptions it is less attractive than the three we recommended. (This does not mean that the approach is insecure, simply that the assumptions required for it to be secure are more complicated.)

As the IBM team has pointed out to us, all of the approaches are vulnerable to a chosen message attack involving about 2^{64} chosen messages. This general attack exploits the iterative structure of the message authentication code and applies to MACs based on encryption functions as well. The basic idea is that if two messages $a_i \cdot b$ and $a_j \cdot b$ have the same MAC, then it is possible that the “collision” occurred before b was processed, so that for any c , $a_i \cdot c$ and $a_j \cdot c$ have the same MAC. Having found two messages $a_i \cdot b$ and $a_j \cdot b$ with the same MAC, the opponent asks for the MAC of $a_i \cdot c$ for some c , thereby obtaining (fraudulently) the MAC of $a_j \cdot c$.

As chosen message attacks go, 2^{64} is quite a large number, and we know of no general way to extend the attack to known messages, except when the known messages are all the same length and end with the same suffix. Full details are given in [1].

Figure 3. Another recommended approach to message authentication with MD5. Here, the key is 128 bits long and the key padding is 384 bits long.

Starting over

So far, our research has focused on adapting an existing hash function to message authentication, which is a practical solution, since MD5 is already trusted, and software for MD5 is widely available. For the long term, designing a message authentication code from scratch is perhaps a better solution.

Mihir Bellare, Roch Guérin and Phillip Rogaway [2] describe techniques for such message authentication that are “provably secure,” under certain assumptions about the underlying functions. Their techniques are also highly parallelizable, a feature that the iterative approach lacks by definition.

Bellare *et al*’s techniques assume the existence of a pseudorandom function, which takes two inputs, a key and a message block, and produces one output. By assumption, if the key input is fixed and unknown, it is difficult to distinguish the pseudorandom function on the message block from a truly random one in any reasonable amount of time. (This is similar to the idea that it is difficult to find collisions for a hash function -although it is possible because they exist, the amount of time required is large.)

The message authentication code is computed by combining, perhaps by bit-wise exclusive-or, the outputs of the pseudorandom function applied to the blocks of the message. To maintain the ordering of the different blocks, each block is tagged with its position in the message. A random block is also included for technical reasons.

Bellare *et al* show that if an opponent can forge message authentication codes, even with the opportunity to request message authentication codes on many different messages, then the opponent can also distinguish the pseudorandom function from a truly random one. Thus, under the assumption that it is difficult to distinguish the pseudorandom function from a truly random one, the message authentication code is secure.

The independent processing of the message blocks leads to the parallelizability of this approach.

It seems that many of the concerns about designing a message authentication code from a hash function

are a consequence of the fact that the key is processed only once, or maybe twice. As a result, the key is isolated, and information about it can be obtained, or other parts of the message can be manipulated independent of the key. By contrast, in message authentication codes based on encryption functions, such as DES-MAC, the key is processed at every step. In Bellare *et al*’s techniques, the key is processed at every step.

We expect that MD5’s compression function or a variant of it may be a suitable pseudorandom function for Bellare *et al*’s techniques, something which further research will determine.

References

- [1] M. Bellare, R. Canetti and H. Krawczyk. *Keying MD5—Message authentication via iterated pseudorandomness*. In preparation.
- [2] Mihir Bellare, Roch Guérin and Phillip Rogaway. *XOR MACs: New methods for message authentication using block ciphers*. Accepted to Crypto '95.
- [3] Mihir Bellare, Joe Kilian and Phillip Rogaway. The security of cipher block chaining. In Yvo G. Desmedt, editor, *Advances in Cryptology- Crypto '94*, volume 839 of *Lecture Notes in Computer Science*, pages 341-358. Springer-Verlag, New York, 1994.
- [4] I.B. Damgård. A design principle for hash functions. In G. Brassard, editor, *Advances in Cryptology: Proceedings of Crypto '89*, volume 435 of *Lecture Notes in Computer Science*, pages 416-427. Springer-Verlag, New York, 1990.
- [5] J. Galvin and K. McCloghrie. *RFC 1446: Security Protocols for version 2 of the Simple Network Management Protocol (SNMPv2)*. Trusted Information Systems and Hughes IAN Systems, April 1993.
- [6] R. Merkle. One way hash functions and DES. In G. Brassard, editor, *Advances in Cryptology: Proceedings of Crypto '89*, volume 435 of *Lecture Notes in Computer Science*, pages 428-446. Springer-Verlag, New York, 1990.
- [7] National Institute of Standards and Technology (formerly National Bureau of Standards). *FZPS PUB 113: Computer Data Authentication*. May 30, 1985.
- [8] National Institute of Standards and Technology. *FZPS PUB 180: Secure Hash Standard (SHS)*. May 11, 1993.
- [9] R. Rivest. *RFC 1321: The MD5 Message-Digest Algorithm*. RSA Data Security, Inc., April 1992.
- [10] Gene Tsudik. Message authentication with one-way hash functions. *ACM Computer Communications Review*, 22(5):29-38, 1992.

The RC5 Encryption Algorithm*

Ronald L. Rivest
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139 USA

Introduction

RC5 is a fast symmetric block cipher suitable for hardware or software implementations. A novel feature of RC5 is the heavy use of *data-dependent rotations*. RC5 has a variable-length secret key, providing flexibility in its security level.

RC5 is a parameterized algorithm, and a particular RC5 algorithm is designated as *RC5- $w/r/b$* . We summarize these parameters below:

- w* The word size, in bits. The standard value is 32 bits; allowable values are 16, 32, and 64. RC5 encrypts two-word blocks: plaintext and ciphertext blocks are each $2w$ bits long.
- r* The number of rounds. Allowable values are 0, 1, . . . 255.
- b* The number of bytes in the secret key *K*. Allowable values of *b* are 0, 1, . . . 255.

RC5 uses an “expanded key table” *S*, derived from the user’s supplied secret key *K*. The size *t* of table *S* depends on the number of rounds: *S* has $t = 2(r+1)$ words.

It is not intended that RC5 be secure for all possible parameter values. On the other hand, choosing the maximum parameter values would be overkill for most applications.

We provide a variety of parameter settings so that users may select an encryption algorithm whose security and speed are optimized for their application, while providing an evolutionary path for adjusting their parameters as necessary in the future. As an example, RC5-32/16/7 is an RC5 algorithm with the number of rounds and the length of key equivalent to

DES. Unlike unparameterized DES, however, an RC5 user can easily upgrade the above choice to an 80-bit key by moving to RC5-32/16/10.

As technology improves, and as the true strength of RC5 algorithms becomes better understood through analysis, the most appropriate parameters can be chosen. We propose RC5-32/12/16 as providing a “nominal” choice of parameters. Further analysis is needed to analyze the security of this choice.

Overview of the Algorithm

RC5 consists of three components: a *key expansion* algorithm, an *encryption* algorithm, and a *decryption* algorithm. These algorithms use the following three primitive operations (and their inverses).

- Two’s complement addition of words, denoted by “+”. This is modulo- 2^w addition.
- Bit-wise exclusive-OR of words, denoted by \oplus .
- A left-rotation (or “left-spin”) of words: the rotation of word *x* left by *y* bits is denoted $x \ll y$. Only the $\lg(w)$ low-order bits of *y* are used to determine the rotation amount, so that *y* is interpreted modulo *w*.

Encryption and Decryption

We assume that the input block is given in two *w*-bit registers *A* and *B*. We also assume that key-expansion has already been performed, so that the array *S*[0 . . . *t*-1] has been computed. Below is the encryption algorithm in pseudo-code. The output is also placed in registers *A* and *B*.

```
A = A + S[0];
B = B + S[1];
FOR i = 1 TO r DO
    A = ((A ⊕ B) <<< B) + S[2*i];
    B = ((B ⊕ A) <<< A) + S[2*i+1];
```

We note the exceptional simplicity of this five-line algorithm. We also note that each RC5 round updates *both* registers *A* and *B*, whereas a “round” in DES updates only half of its registers. An RC5 “half-round” (one of the assignment statements updating *A* or *B* in the body of the loop above) is thus perhaps more analogous to a DES round.

The decryption algorithm can be easily derived from the encryption algorithm.

**RC5 and RSA-RC5 are registered trademarks of RSA Data Security, Inc. Patent pending*

As technology improves, and as the true strength of RC5 algorithms becomes better understood through analysis, the most appropriate parameters can be chosen.

Professor Ronald L. Rivest is associate director of MIT’s Laboratory for Computer Science. He can be contacted at rivest@theory.lcs.mit.edu. A complete paper on RC5 was presented at the Leuven Algorithms Workshop in December 1994. A non-line version of the complete paper can be obtained by ftp or web. FTP: under [pub/rivest/rc5](ftp://pub/rivest/rc5) on [theory.lcs.mit.edu](ftp://theory.lcs.mit.edu); WEB: under <http://theory.lcs.mit.edu/~rivest>. Parts of this article originally appeared in Dr. Dobbs’s Journal. Copyright © 1995 Miller Freeman Inc.

The encryption algorithm is very compact, and can be coded efficiently in assembly language on most processors.

A distinguishing feature of RC5 is its heavy use of data-dependent rotations

Key Expansion
The key-expansion routine expands the users secret key *K* to fill the expanded key array *S*, so that *S* resembles an array of $t = 2(r + 1)$ random binary words determined by *K*. The key expansion algorithm uses two “magic constants” and consists of three simple algorithmic parts.

The key-expansion algorithm uses two word-size binary constants P_w and Q_w . They are defined for arbitrary word size *w* as follows:

$$P_w = \text{Odd}((e-2)2^w)$$
$$Q_w = \text{Odd}((\phi-1)2^w)$$

where
 $e = 2.718281828459...$ (base of natural logarithms)
 $\phi = 1.618033988749...$ (golden ratio),

and where Odd(*x*) is the odd integer nearest to *x* (rounded up if *x* is an even integer, although this won't happen here).

The first algorithmic step of key expansion is to copy the secret key *K*[0...*b*-1] into an array *L*[0...*c*-1] of $c = \lceil b/u \rceil$ words, where $u = w/8$ is the number of bytes/word. This operation is done in a natural manner, using *u* consecutive key bytes of *K* to fill up each successive word in *L*, low-order byte to high-order byte. Any unfilled byte positions of *L* are zeroed.

The second algorithmic step of key expansion is to initialize array *S* to a particular fixed (key-independent) pseudo-random bit pattern, using an arithmetic progression modulo 2^w determined by the “magic constants” P_w and Q_w . Since Q_w is odd, the arithmetic progression has period 2^w .

$$S[0] = P_w;$$
$$\text{FOR } i = 1 \text{ TO } t-1 \text{ DO}$$
$$S[i] = S[i-1] + Q_w;$$

The third algorithmic step of key expansion is to mix in the users secret key in three passes over the arrays *S* and *L*. More precisely, due to the potentially different sizes of *S* and *L*, the larger array will be processed three times, and the other may be handled more times.

$$i = j = 0;$$
$$A = B = 0;$$

```
DO 3*max(t,c) TIMES:
  A = S[i] = (S[i] + A + B) <<< 3;
  B = L[j] = (L[j] + A + B) <<< (A+B);
  i = (i + 1) mod(t);
  j = (j + 1) mod(c);
```

The key-expansion function has a certain amount of “one-wayness”: it is not so easy to determine *K* from *S*.

Speed
The encryption algorithm is very compact, and can be coded efficiently in assembly language on most processors. The table *S* is accessed sequentially, minimizing issues of cache size. The RC5 encryption speeds obtainable are yet to be fully determined. For RC5-32/12/16 on a 90MHz Pentium, a preliminary C++ implementation compiled with the Borland C++ compiler (in 16-bit mode) performs a key-setup in 220 microseconds and performs an encryption in 22 microseconds (equivalent to 360,000 bytes/sec). These timings can presumably be improved by more than an order of magnitude using a 32-bit compiler and/or assembly language—an assembly-language routine for the ‘486 can perform each round in eight instructions.

Security
A distinguishing feature of RC5 is its heavy use of *data-dependent rotations*—the amount of rotation performed is dependent on the input data, and is not predetermined.

The encryption/decryption routines are very simple. While other operations (such as substitution operations) could have been included in the basic round operations, our objective is to focus on the data-dependent rotations as a source of cryptographic strength.

Some of the expanded key table *S* is initially added to the plaintext, and each round ends by adding expanded key from *S* to the intermediate values just computed. This assures that each round acts in a potentially different manner, in terms of the rotation amounts used. The xor operations back and forth between *A* and *B* provide some avalanche properties, causing a single-bit change in an input block to cause multiple-bit changes in following rounds.

The use of variable rotations helps defeat differential cryptanalysis (Biham/Shamir^[1]) and linear crypt-

analysis (Matsui ^[3]), since bits are rotated to “random” positions in each round; Kaliski and Yin analyze the security of RC5 against both types of cryptanalysis ^[2]. For the standard word size $w = 32$, their differential attack can be applied to RC5 with less than 12 rounds and their linear attack can be applied to RC5 with less than six rounds. An assessment of the RC5 encryption algorithm will appear in the Summer issue of *CryptoBytes*; meanwhile, I invite the reader to help determine the strength of RC5.

References

[1] E. Biham and A. Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. Springer-Verlag, New York, 1993.
[2] B. S. Kaliski Jr. and Y. L. Yin. *On differential and linear cryptanalysis of the RC5 encryption algorithm*. Accepted to Crypto '95.
[3] M. Matsui. The first experimental cryptanalysis of the Data Encryption Standard. In Y. G. Desmedt, editor, *Advances in Cryptology—Crypto'94*, volume 839 of *Lecture Notes in Computer Science*, pages 1-11, Springer-Verlag, New York, 1994.



N E W S A N D I N F O R M A T I O N

X9F1 Considers Triple-DES Standard
The ANSI-accredited X9F1 working group has begun work on a standard for bulk data encryption for financial services based on so-called triple-DES, a method of extending the security of the Data Encryption Standard by encrypting three times with DES.

While triple-DES has been a standard mechanism for several years for encrypting keys as part of ANSI X9.17, attention has turned recently to triple-DES for bulk data encryption, in response to the decreasing security of DES's 56-bit key and the shortage of trusted alternatives to DES.

The specifics of the standard are yet to be determined, but two recommendations by cryptography experts are likely to have strong influence: that the three encryptions involve three different keys (X9.17 involves only two, where the first and third encryption is with the same key), and that modes of operation for bulk data encryption, such as cipher block chaining, be built around triple-DES as a primitive.

Modes involving single-DES instead of triple-DES as a primitive, such as encrypting three times with single-DES in cipher block chaining mode, have been shown by Eli Biham in the past year to be potentially no stronger than single-DES against certain attacks. Encrypting with triple-DES in cipher block chaining mode is not vulnerable to those attacks.

And while two-key triple-DES is significantly stronger than single-DES, it has a certain “certificational weakness” observed by Merkle and Hellman in 1980 which was revealed in 1990 as a known-plaintext at-

tack by Wiener and van Oorschot. No such attacks are known for three-key triple-DES.

Balloting of the standard is expected in 1996.

RSA Laboratories Publishes PKCS #1 1
Culminating a year of development, RSA Laboratories has published the latest in its series of Public-Key Cryptography Standards, *PKCS #1 1: Cryptographic Token Interface Standard (Cryptoki)*.

PKCS #11 specifies an application programming interface (API) called Cryptoki to devices which hold cryptographic information and perform cryptographic functions, such as ISO smart cards, PCMCIA cards, and the SmartDisk. Cryptoki isolates applications from the device technology, presenting a common, logical view of the device called a “cryptographic token.”

The interface supports a wide range of cryptographic mechanisms, including RSA, DSA, Diffie-Hellman, DES, triple-DES, RC2, RC4, MD2, MD5, and SHA; tokens are expected to support subsets of these mechanisms according to application profiles.

Cryptoki is at the algorithm-specific, technology-independent layer in the current cryptographic API standardization effort, and is expected to combine nicely with interfaces at the algorithm-independent, security-service-oriented layer such as the Generic Security Services API (GSSAPI).

Copies of PKCS #11 and other PKCS standards can be obtained by anonymous FTP to ftp.rsa.com in the pub/pkcs directory, or by E-mail to pkcs@rsa.com.

Modes involving single-DES instead of triple-DES as primitive [...] have been shown to be potentially no stronger than single-DES against certain attacks.

Cryptoki [...] is expected to combine nicely with interfaces at the algorithm-independent, security-service-oriented layer.

A N N O U N C E M E N T S

1995 RSA Laboratories Seminar Series

RSA Laboratories is pleased to announce details of the 1995 Seminar Series. Now in its third year, the Seminar Series has been expanded again and will now be presented at two US locations.

The Seminar Series is an intensive three-day presentation on all aspects of cryptography. In a slight departure from previous years, the first day of the Seminar Series will be a self-contained overview of the basic ideas and cryptographic techniques that are used today. Building on this introduction, the remainder of the seminar series will provide detailed analysis on many of the algorithms, techniques and theoretical foundations which dominate current cryptographic thinking.

The East Coast Seminar Series will be held at the Columbia Inn in Columbia, MD, from July

19-21; the West Coast Seminar Series will be held at the Hotel Sofitel in Redwood Shores, CA, from August 23-25. Contact RSA Laboratories for more information on how to register.

RSA Laboratories Technical Reports

The RSA Laboratories Technical Reports are now available via a subscription service. These reports offer detailed summaries of current research on a variety of topics and they bring together information from a wide variety of sometimes obscure sources. Subscription to the Technical Reports will be at one of two levels; individual or corporate. As well as receiving all previously written reports, subscribers will receive new reports as they appear as well as current research notes on items of major significance.

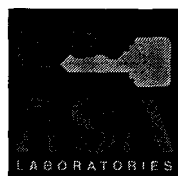
Contact RSA Laboratories for more information about the Technical Report subscription service.



Coming in the Summer 1995 CryptoBytes:

- *Elliptic curve cryptosystems*
- *RSA key size recommendations*
- *RC5 update*

for subscription information, see page 2 of this newsletter.



**CRYPTOGRAPHIC
RESEARCH AND
CONSULTATION**

100 MARINE PARKWAY
REDWOOD CITY
CA. 94065.1031
TEL 415/595-7703
FAX 415/595-4126
rsa-labs@rsa.com

PRESORT
FIRST CLASS
U.S. POSTAGE
PAID
MMS, INC

RSA
Laboratories'

Bulletin

News and advice from RSA Laboratories

Suggestions for Random Number Generation in Software

Tim Matthews
RSA Data Security

Introduction

The generation of random numbers is critical to cryptographic systems. Symmetric ciphers such as DES, RC2, and RC5 all require a randomly selected encryption key. Public-key algorithms — like RSA, Diffie-Hellman, and DSA — begin with randomly generated values when generating prime numbers. At a higher level, SSL and other cryptographic protocols use random challenges in the authentication process to foil replay attacks.

But truly random numbers are difficult to come by in software-only solutions, where electrical noise and sources of hardware randomness are not available (or at least not convenient). This poses a challenge for software developers implementing cryptography. There are methods, however, for generating sufficiently random sequences in software that can provide an adequate level of security. This bulletin offers suggestions on generating random numbers in software, along with a bit of background on random numbers.

Random vs. Pseudo-Random Numbers

What is a truly random number? The definition can get a bit philosophical. Knuth speaks of a sequence of independent random numbers with a specified distribution, each number being obtained by chance and not influenced by the other numbers in the se-

quence. Rolling a die would give such results. But computers are logical and deterministic by nature, and fulfilling Knuth's requirements is not something they were designed to do. So-called random number generators on computers actually produce pseudo-random numbers. Pseudo-random numbers are numbers generated in a deterministic way, which only appear to be random.

Most programming languages include a pseudo-random number generator, or "PRNG." This PRNG may produce a sequence adequate for a computerized version of blackjack, but it is probably not good enough to be used for cryptography. The reason is that someone knowledgeable in cryptanalysis might notice patterns and correlations in the numbers that get generated. Depending on the quality of the PRNG, one of two things may happen. If the PRNG has a short period, and repeats itself after a relatively short number of bits, the number of possibilities the attacker will need to try in order to deduce keys will be significantly reduced. Even worse, if the distribution of ones and zeros has a noticeable pattern, the attacker may be able to predict the sequence of numbers, thus limiting the possible number of resulting keys. An attacker may know that a PRNG will never produce 10 binary ones in a row, for example, and not bother searching for keys that contain that sequence.

The detail of what makes a PRNG cryptographically "good" is a bit beyond the scope of this paper. Briefly stated, a PRNG must have a high degree of unpredictability. Even if nearly every bit of output is known, those that are unknown should remain hard to predict. The "hardness" is in the sense of compu-

Tim Matthews is a cryptographic systems engineer at RSA Data Security. He can be contacted at tim@rsa.com.



tational difficulty — predicting the bits should require an infeasible amount of computation. A true random number generator, like a hardware device, will have maximum unpredictability. A good PRNG will have a high degree of unpredictability, making the output unguessable, which is the goal.

One essential ingredient in producing good random numbers in software, then, is to use a good PRNG. Important to note is that although the PRNG may produce statistically good looking output, it also has to withstand analysis to be considered strong. Since the one included with your compiler or operating system may or may not be, we recommend you don't use it. Instead, use a PRNG that has been verified to have a high degree of randomness. RSA's BSAFE toolkit uses the MD5 message digest function as a random number generator. BSAFE uses a state value that is digested with MD5. The strength of this approach relies on MD5 being a one-way function — from the random output bytes it is difficult to determine the state value, and hence the other output bytes remain secure. Similar generators can be constructed with other hash functions, such as SHA1.

an attacker has a high likelihood of re-creating your sequence of pseudo-random bytes by guessing the exact seeding time. Once he has the pseudo-random bytes, he can re-create your keys. The security issue becomes one of making sure an attacker cannot determine your seed.

You may be wondering why use a random number generator to generate random bytes, if to use it, you need to first generate random bytes. Seeding is a bootstrap operation. Once done, generating subsequent keys will be more efficient. Another important point is that the information collected for the seed does not need to be truly random, but unguessable and unpredictable. Once the seed is fed into MD5, the output becomes pseudo-random. If attackers cannot guess or predict seeds, they will be unable to predict the output.

There are two aspects to a random seed: quantity and quality. They are related. The quality of a random seed refers to the entropy of its bits. Cryptographers use the word entropy a lot, so it is worth knowing. In a system that produces the same output each time, each bit is fixed, so there is no uncertainty, or zero entropy per bit. If every possible sequence of outputs is equally likely (i.e. truly random) then there is total uncertainty, or one bit of entropy per output bit. There are precise mathematical formulas for entropy, but the short summary is the more entropy per bit, the better. Since the quality may vary, it is a good idea to account for this with quantity. Sufficient quantity makes it impractical for an attacker to exhaustively try all likely seed values. Let's start with quality.

System Unique	Variable and Unguessable	External Random
Configuration files	contents of screen	Cursor position with time
Drive configuration	Date and time	Keystroke timing
Environment strings	High resolution clock samples	Microphone input (with microphone connected)
	Last key pressed	Mouse dii timing
	Log file blocks	Mouse movement
	Memory statistics	Vii input
	Network statistics	
	Process statistics	
	Program counter for other processes or threads	

Less Entropy

More Entropy

Table 1
Seed Sources

The Seed

The other component in producing good random numbers is providing a random seed. A good PRNG like BSAFE's will produce a sequence that is sufficiently random for cryptographic operations, with one catch: it needs to be properly initialized, or "seeded." Using a bad seed (or no seed at all) is a common flaw in poorly implemented cryptographic systems. A PRNG will always generate the same output if started with the same seed. If you are using MD5 with the time of day as the seed, for example,

Table 1. shows a list of potential sources for building the initial seed pool. External random events are the best, but harder to get than variable or unique information. Sources that are variable, while not random, are very difficult for an attacker to guess. Quantities that are unique to a system are also hard to guess and usable if mom bytes are needed.

In general, collect as much external random information as possible. Supplement this with sources from the two other columns if more bytes are needed. Using a composite of many items makes the attacker's task more difficult. In an application where several keys will be generated, it may make sense to collect enough seed bytes for multiple keys, even before the first is generated. Be careful of in-



formation that moves across a network that could be intercepted by a dedicated attacker. Mouse movements on X-terminals, for example, may be available to anyone listening on the wire.

Now we get to the issue of quantity. A developer cannot assume that all of the bits collected are truly random, so a useful rule of thumb is to assume that for every byte of data collected at random, there is one bit of entropy. This may either be a bit conservative, or a bit generous, depending on the source. To illustrate this rule of thumb, take the example of user keystrokes, which many consider to be a good source of randomness. Assuming ASCII keystrokes, bit 7 will always be zero. Many of the letters can be predicted: they will probably all be lowercase, and will often alternate between left and right hand. Analysis has shown that there is only one bit per byte of entropy per keystroke.

To guard against this kind of analysis, the idea is to collect one byte of seed for each bit required. This information will be fed into the PRNG to produce the first random output.

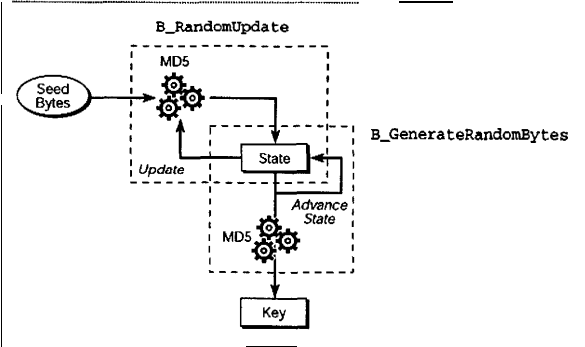
As an example, if the seed will be used to produce a random symmetric encryption key, the number of random bytes in the seed should at least equal the number of effective bits in the key. In the case of DES, this would be 56 random bits culled from a seed pool of 56 bytes. Any less and the number of possible starting keys is reduced from 2^{56} to something smaller, reducing the amount of effort required by an attacker in searching the seed space by brute force. Attacks like this have recently been widely publicized on the Internet and in the press. For public-key algorithms, the goal is to make searching for the seed at least as difficult as the hard mathematical problem at their core. This will discourage attackers from searching for seeds instead of attacking problems like factoring composite numbers and calculating discrete logarithms. A seed of 128 bits (taken from a seed pool of 128 bytes) should be more than enough for the modulus sizes being used today.

One last thing that should be mentioned is updating the seed, or "re-seeding." It makes sense to allow an application to add seed bits as they become available. User events often provide additional sources of randomness, but obviously have not taken place when an application starts. These should be

included as they occur. Re-seeding also frustrates attackers trying to find the seed state using a brute force attack. Since the seed will be change, say, every thirty seconds, the seed state becomes a moving target and makes the brute force attack infeasible. The idea is to take the existing seed and mix it together with the new information as it becomes available.

Example

A brief example is in order. The diagram in Figure 1 illustrates how functions in BSAFE would be used to generate random keying material.



The first step is to supply the pool of random seed bytes. Let's assume that the application needs a random 80-bit RC2 key. Using the rule of thumb that one byte of data yields one bit of randomness, a minimum of 80 bytes will be needed for the pool. This pool would be gathered from the sources listed in Table 1. The `B_RandomUpdate` function in BSAFE takes the seed pool and runs it through the MD5 message digest algorithm to create the state.

The state is then used by the function `B_GenerateRandomBytes`, which runs it through MD5 to produce the key. This is the key that would be used for RC2. As an added measure, BSAFE automatically advances the state after random bytes are generated.

Notice the arrow labeled "Update" within `B_RandomUpdate`. This is where reseeded is done. By calling `B_RandomUpdate` again, the state can be mixed with more seed information. Random information like key timing and mouse movement can be used here, along with changes in the system statis-

Figure 1
Random Seed
Process



tics, clock, and various files. Now the next **RC2** key generated will be based on new seed material as well as the old.

Conclusion

Done properly, random number generation in software can provide the security necessary for most cryptographic systems. Using a good **PRNG** and choosing good seed material are the two critical points.

Developers may wish to create a set of routines to pull random and unique information from the operating system, which can then be used in any applications requiring **cryptography**. It may be desirable to save encrypted seed state for use in subsequent sessions.

Over time, as the need for cryptomphy in software increases, hardware and operating system vendors may provide more tools and hooks for random information. In the meantime, however, the techniques described can be used.

Further Reading

For more information on random numbers and cryptography, take a look at the following:

- Donald Eastlake, Steve Crocker, Jeff Schiller, "Randomness Recommendations for Security," IETF RFC 1750, 1994
- Ian Goldberg and David Wagner, "Randomness and the Netscape Browser," *Dr. Dobbs's Journal*, January 1996
- Donald E. Knuth, *The A# of Computer Programming: Seminumerical Algorithms*, Addison-Wesley, Reading, MA, 1981
- Colin Plumb, "Truly Random Numbs," *Dr. Dobbs's Journal*, November 1994
- RSA Data Security, Inc., *BSAFE User's Manual*, Version 3.0, 1996
- Bruce Schneier, *Applied Cryptography*, John Wiley & Sons, Inc., New York, 1995

For more information on this and other recent developments in cryptography, contact RSA Laboratories at one of the addresses below.

RSA Laboratories
100 Marine Parkway, Suite 500
Redwood City, CA 94065 USA
415/595-7703
415/595-4126 (fax)
rsa-labs@rsa.com
<http://www.rsa.com/rsalabs/>