Cryptography in Context

Cryptography in Context

Gerard Tel

Contents

Preface

This reader contains 16 papers, written by students of Utrecht University as part of the Cryptography course of Fall 2007. The course treats the theory of Cryptography, underlying many solutions in computer and IT security. The challenge for the students is, to connect this theory to actual security products in their project study.

The papers were presented during a symposium on January 31, 2008. Photo's of this event can be found on `gerardtel.nl` (select the year 2008 and the series *5. Cryptography in Context*). For the purpose of the symposium, the presentations were divided into four themes, namely ***Broken Dreams*** (Chapters 1 through 4), about systems of which the security was broken, ***Safe Digital Society*** (Chapters 5 through 8), about security used in a nation-wide scale to support the economical or democratic infrastructure, ***Deployed Systems*** (Chapters 9 through 12), about security products used in a smaller scale, and ***How to Compute*** (Chapters 13 through 16), about cryptographic algorithms.

The session chairs during the symposium were Thomas van Dijk, Johan Kwisthout, Eelko Penninkx, and Marinus Veldhorst. I hope the reader will be a nice souvenir for the speakers and writers, and that it will provide for all others an interesting overview of some computer security topics. Two of the papers are especially recommended: the ones by Wouter Penard and Tim van Werkhoven, and by Merel Rietbergen received the highest grade possible: a 10 mark.

Gerard Tel (course teacher),
Utrecht, February 2008.

Chapter 1

On the Secure Hash Algorithm family

Written by *Wouter Penard and Tim van Werkhoven.*

1.1 Introduction

This chapter is on the Secure Hash Algorithm family, better known as the SHA hash functions. We will first introduce secure hash algorithms as part of digital signature schemes and derive properties a hash function is required to have from this context. SHA was originally designed for this purpose. Next we give a short overview of the history of the SHA family, showing when new members of the family were introduced and note a couple important milestones in the academic analysis of the functions.

In the second section we will review how the SHA algorithms work, focusing especially on the SHA-1 hash function. After dissecting the SHA-1 hash function itself, we will relate and compare it to the other hash functions in the SHA family, SHA-0 and SHA-2.

Besides analyzing the specific SHA functions, we will also treat the generic structure of hash functions, as these are relevant for the security of a hash function. We will cover the different aspects in some detail and illustrate how naive implementation of hash functions can be hazardous, some attacks can be mounted against any hash function if they are not implemented correctly.

The main part of this chapter will focus on the weaknesses found in the SHA-0 hash function by different researchers. We will explain in more detail how the SHA-0 function was initially broken by treating the so-called differential collision search against this function. And in this way aim to provide the reader with some insight how attacks against the SHA functions are build up. After explaining how the idea works, we will

explain why this attack is less successful against the SHA-0 successor, SHA-1. Finally we will look at SHA-2 and summarize the cryptanalysis results for the SHA family.

The last part will deal with the implications of broken hash functions, and how and in which scenarios these are especially dangerous. Although the weaknesses found may not seem all too severe, these attacks can indeed have serious consequences, as we will see in the final section.

1.1.1 Purpose of Cryptographic Hash Algorithms

A hash function is a function which takes an arbitrary length input and produces a fixed length 'fingerprint' string. Common usage of such a function is as index into a hashtable. Cryptographic hash functions have several additional properties which makes them suitable to use as a means to check the integrity of a message and as part of digital signature schemes. Such a scheme consists of a secret key k_s, a public key k_p and two functions $\text{Sign}(M, k_s)$, which produces signature S, and $\text{Verify}(M, S, k_p)$, which returns a boolean indicating if the given S is a valid signature for message M. Such a signature should prove the *authenticity* and *integrity* of the message; Such that we can be sure it really was the sender of the message *who* sent it, and that it really was *this* message he sent. Finally we can use the signature to prove that the sender sent it and no one else could have done so: *non-repudiation*

A function requirement is that $\text{Verify}(M, \text{Sign}(M, k_s), k_p) = \textit{true}$ for a key pair (k_s, k_p). On the other hand it should be impossible to create a forged signature. It is possible to distinguish between two types of forgeries, *Existential* and *Universal* forgeries. In the first case the attacker creates a valid M, S pair, given the public key k_p. Note that the attacker cannot influence the computed message, and thus, in general, the produced M will be a random string. To create a universal forgery the attacker computes a valid signature S given M and k_p.

A public-private key cryptosystem, like for example RSA [RSA78], can be used to place such a signature. Here secret key (n, d) is used to sign the message and the public key (n, e) is used to verify the signature. In order to create a universal forgery it would be required to effectively compute the private component of the RSA key system, which is assumed to be infeasible. However finding an existential forgery is trivial, for a random S we can find a corresponding message by calculating $M = S^e \,\%\, n$. Another drawback is that RSA can only sign messages with limited length, a straightforward, but bad solution would be to split the message in blocks and sign each block individually. However now it is possible for an attacker to rearrange the individual blocks, which would result in a new message with a valid signature. Finally RSA is a relatively slow algorithms.

Cryptographic hash functions can be used to resolve these problems. Such a hash function, H, takes an arbitrary length message as input and produces a fixed length message digest D. Now we compute the message digest for a message and sign this digest instead of the actual message. This hash function has to have a number of properties to ensure the properties of authenticity, integrity and non-repudiation for the signature scheme as a whole. In section 1.3.1 we will discuss these properties and how these are compromised by the attacks on the algorithms.

Casus 1.1: Storing Passwords

When storing passwords, keeping them in plaintext is a bad idea. If the computer gets stolen, so are all the passwords. If the password is hashed however, the thief only has the hash, which he cannot use to login elsewhere as hash functions are hard to invert.
Recently, rainbow tables have been showing up. These tables are like Helmann tables in the sense that they provide a time-memory tradeoff. The table is created for certain passwords (i.e. lowercase passwords up to eight bytes). Now if a cracker has the hash, he can look it up in the hashtable and if he's lucky he will find the password, even when it's hashed!
To protect against these attacks, one can add a *salt* to the password before hashing, which is a random string of a few bytes. Consider a password of 64 bits, with 2^{64} possibilities. One can construct a rainbow table for these passwords and once you have a hash, you can calculate the password belonging to it. If you add a random string of 64 bits to the password and hash this, a rainbow table attack becomes infeasible because of the enormous growth in extra possibilities. Note that this salt does not need to be protected, it can be stored in plain text along with the password. This is because the salts differ per password, and building a new table for every password makes rainbow tables useless.

The SHA algorithms are, besides its use in digital signature, fundamental part of various protocols and applications, including TLS and SSL, PGP and SSH.

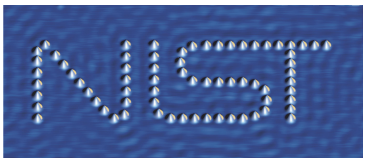
1.1.2 Short History

The Secure Hash Algorithm (SHA) was developed by the NIST in association with the NSA and first published in May 1993 as the Secure Hash Standard. The first revision to this algorithm was published in 1995 due to a unpublished flaw found, and was called SHA-1. The first version, later dubbed SHA-0, was withdrawn by the NSA. The SHA hash function is similar to the MD4 hash function, but adds some complexity to the algorithm and the block size used was changed. SHA was originally intended as part of the Digital Signature Standard (DSS), a scheme used for signing data and needed a hash function to do so.

In 1998, two French researchers first found a collision on SHA-0 with complexity 2^{69} [1], as opposed to the brute force complexity of 2^{80} [CJ98]. This result was improved in drastically improved by Wang e.a.which could find a collision with complexity 2^{39} [WYY05b], which is within practical reach. It took only five years for the initial SHA

[1] As the reader might recall, complexity is defined as the number of hash operations needed for a specific attack.

Casus 1.2: The National Institute of Standards and Technology



The American NIST publishes recommendations on how to use cryptography in practice. For example, the hashes the NIST currently recommend are SHA-1 and the SHA-2 family. Besides hashes, they also recommend which encryption functions to use, i.e. which are considered safe at the moment, but are also involved with techniques behind (pseudo) random number generators and policies on choosing passwords. Besides cryptography, the NIST is also active in other fields. The NIST logo you see in this box for example, has been written with cobalt atoms in the nano structures department of the institute.

function to be broken, and after another seven years, the best attack possible is only half of the (logarithmic) complexity of the original hash function. Fortunately, the NSA already foresaw this in 1995 and released SHA-1.

Cryptanalysis on SHA-1 proved to be much more difficult, as the full 80-round algorithm was broken only in 2005, and this attack still has a complexity of 2^{63}[WYY05a]. This restricts the attack only to the theoretical domain, as a complexity of 2^{63} is still unfeasible on present-day computers. Nonetheless, this collisional attack requires less than the 2^{80} computations needed for a brute-force attack on SHA-1. We will explain why it took 7 years longer to break SHA-1 in section 1.4.2.

In addition to the SHA-1 hash, the NIST also published a set of more complex hash functions for which the output ranges from 224 bit to 512 bit. These hash algorithms, called SHA-224, SHA-256, SHA-384 and SHA-512 (sometimes referred to as SHA-2) are more complex because of the added non-linear functions to the compression function. As of January 2008, there are no attacks known better than a brute force attack. Nonetheless, since the design still shows significant similarity with the SHA-1 hash algorithms, it is not unlikely that these will be found in the (near) future. Because of this, an open competition for the SHA-3 hash function was announced on November 2, 2007[2]. The new standard is scheduled to be published in 2012. Fortunately, the SHA-2 hash functions produce longer hashes, making a feasible attack more difficult. Consider for example the SHA-512 hash function, producing 512 bit hashes and thus having an approximate complexity against collisional attacks of 2^{256}. Even if the logarithmic complexity would be halved (from 256 to 128), this would still be out of reach for practical purposes for the coming decade or so.

1.2 Description of the SHA Algorithms

In this section we will describe how the SHA-1 algorithm works and relate it to both its predecessor, SHA-0 and its successor SHA-2. In each of the algorithms we identify

[2] `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf`

two phases: first message expansion followed by a state update transformation which is iterated for a number, 80 in SHA-1, of rounds. In the next sections we make use of the following operators: $\ll$ and $\gg$, the left and right shift operator, and $\lll$ and $\ggg$ the bitwise left- and right rotate operator.

1.2.1 SHA-1

SHA-1 takes as input a message with maximum length $2^{64}-1$ bits and returns a 160-bit message digest. The input is processed in parts of 512 bit each, and is padded using the following scheme. First a 1 is appended and then padded with 0's until bit 448, finally the length of the message is inserted in the last 64-bits of the message, the most significant bits padded with 0's. The reason that first a 1 is appended is that otherwise collisions occur between a sample messages and the same message with some zeros at the end.

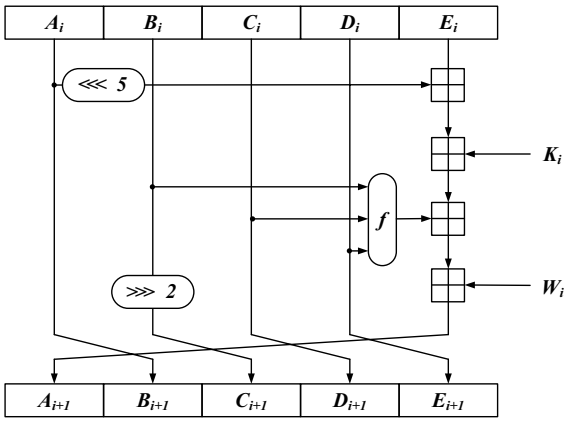


Figure 1.3: Schematic overview of a SHA-0/SHA-1 round. At the top are the five 32-bit registers A–E for round i, and at the bottom are the values for these registers for the next round. In between this is the mixing of these registers with constants K_i and a part of the expanded message, W_i. The square boxes denote addition and the different functions f are given above.

The intermediate results of each block are stored in five 32-bit registers denoted with $h_0, \ldots, h_5$. These five registers are initialized with the following hexadecimal values:

$H_0 =$ `0x67452301`
$H_1 =$ `0xEFCDAB89`
$H_2 =$ `0x98BADCFE`
$H_3 =$ `0x10325476`
$H_4 =$ `0xC3D2E1F0`

Next, the algorithm uses four constants K0, K1, K2 and K3, with values:

K0 = `0x5A827999`
K1 = `0x6ED9EBA1`
K2 = `0x8F1BBCDC`
K3 = `0xCA62C1D6`

SHA-1(M):
(* Let M be the message to be hashed *)
for each 512-bit block B in M **do**
$W = f_{exp}(B)$;
$a = H_0$; $b = H_1$; $c = H_2$; $d = H_3$; $e = H_4$;
for $i = 0$ **to** 79 **do**
if $0 \leq i \leq 19$ **then**
$T = a \lll 5 + f_{if}(b, c, d) + e + W_i + \text{K0}$;
else if $20 \leq i \leq 39$ **then**
$T = a \lll 5 + f_{xor}(b, c, d) + e + W_i + \text{K1}$;
else if $40 \leq i \leq 59$ **then**
$T = a \lll 5 + f_{maj}(b, c, d) + e + W_i + \text{K2}$;
else if $60 \leq i \leq 79$ **then**
$T = a \lll 5 + f_{xor}(b, c, d) + e + W_i + \text{K3}$
$e = d$; $d = c$; $c = b \lll 30$; $b = a$; $a = T$;
$H_0 = a + H_0$; $H_1 = b + H_1$; $H_2 = c + H_2$; $H_3 = d + H_3$; $H_4 = e + H_4$;
return concat(H_0, H_1, H_2, H_3, H_4);

Algorithm 1.4: The SHA-1 algorithm.

Finally we define four functions: f_{exp}, f_{if}, f_{maj} and f_{xor}. The first takes the 512-bit message as argument, each of the other functions take three 32-bit words (b, c, d) as argument.

The f_{exp} function expands the initial 512-bit input message M, consisting of 16 32-bit words M_i with $0 \leq i \leq 15$ to 80 32-bit words W_i with $0 \leq i \leq 79$.

$$W_i = \begin{cases} M_i, & \text{if } 0 \leq i \leq 15 \\ W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16} \lll 1, & \text{if } 16 \leq i \leq 79 \end{cases}$$

The other functions are defined as:

$$\begin{aligned} f_{if}(b, c, d) &= b \wedge c \oplus \neg b \wedge d \\ f_{maj}(b, c, d) &= b \wedge c \oplus b \wedge d \oplus c \wedge d \\ f_{xor}(b, c, d) &= b \oplus c \oplus d \end{aligned}$$

The SHA-1 round is graphically depicted in Figure 1.3.

1.2.2 SHA-0

The design proposal for SHA-0 was withdrawn by the NSA shortly after publishing it. The only difference between SHA-0 and SHA-1 is in the message expansion phase.

$$W_i = \begin{cases} M_i, & \text{if } 0 \leq i \leq 15 \\ W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}, & \text{if } 16 \leq i \leq 79 \end{cases} \tag{1.1}$$

Note the absence of the leftrotate operation in the SHA-0 expansion.

1.2.3 SHA-2

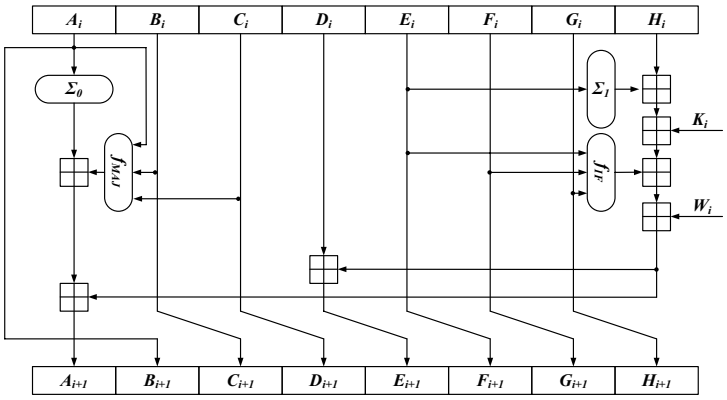


Figure 1.5: Schematic overview of a SHA-2 round. In comparison with SHA-1, there are additional functions to mix the data. Also note that instead of five there are now eight registers each holding 32-bit words (SHA-224/256) or 64-bit words (SHA-384/512). 64 of these rounds on top of each other form SHA-256, 80 rounds form SHA-512.

SHA-2 is a common name for four additional hash functions also referred to as SHA-224, SHA-256, SHA-384 and SHA-512. Their suffix originates from the bit length of the message digest they produce. The versions with length 224 and 384 are obtained by truncating the result from SHA-256 and SHA-512 respectively. SHA-256 uses a block size of 512 bits, and iterates 64 rounds, while SHA-512 uses a 1024 bit block size and has 80 rounds. Furthermore, SHA-512 uses an internal word size of 64 bits instead of the 32 bit used by all other SHA variants. The SHA-2 algorithms follow the same structure of message expansion and iterated state update transformation as SHA-1, but both message expansion and state update transformation are much more complex. We will discuss SHA-256 in more detail to indicate differences between SHA-1 and SHA-2.

SHA-256 uses sixty-four constants $K_0, .., K_{63}$ of 32 bits each and eight registers to store intermediate results H_0, .., H_7. The values of these constants can be found in [NIS02]. The function definitions for SHA-256 are given by

$$W_i = \begin{cases} M_i, & \text{if } 0 \leq i \leq 15 \\ \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16}, & \text{if } 16 \leq i \leq 63 \end{cases} \tag{1.2}$$

with

$$\begin{aligned} \Sigma_0(x) &= x \ggg 2 \oplus x \ggg 13 \oplus x \ggg 22, \\ \Sigma_1(x) &= x \ggg 6 \oplus x \ggg 11 \oplus x \ggg 25, \\ \sigma_0(x) &= x \ggg 7 \oplus x \ggg 18 \oplus x \gg 3, \\ \sigma_1(x) &= x \ggg 17 \oplus x \ggg 19 \oplus x \gg 20. \end{aligned}$$

SHA-256(M):
(* Let M be the message to be hashed *)
for each 512-bit block B in M **do**
$W = f_{exp}(B)$;
$a = H_0$; $b = H_1$; $c = H_2$; $d = H_3$; $e = H_4$; $f = H_5$; $g = H_6$; $h = H_7$;
for $i = 0$ **to** 63 **do**
$T_1 = h + \Sigma_1(e) + f_{if}(e, f, g) + K_i + W_i$;
$T_2 = \Sigma_0(a) + f_{maj}(a, b, c)$;
$h = g$; $g = f$; $f = e$; $e = d + T_1$; $d = c$; $c = b$; $b = a$; $a = T_1 + T_2$;
$H_0 = a + H_0$; $H_1 = b + H_1$; $H_2 = c + H_2$; $H_3 = d + H_3$;
$H_4 = e + H_4$; $H_5 = e + H_5$; $H_6 = e + H_6$; $H_7 = e + H_7$;
return concat(H_0, H_1, H_2, H_3, H_4, H_5, H_6, H_7);

Algorithm 1.6: The SHA-256 algorithm.

The other functions used in the algorithm are given by

$$\begin{aligned} f_{if}(b, c, d) &= b \wedge c \oplus \neg b \wedge d, \\ f_{maj}(b, c, d) &= b \wedge c \oplus b \wedge d \oplus c \wedge d. \end{aligned}$$

A round of the stronger SHA-2 hash function is shown in Figure 1.5.

1.3 Generic Attacks

1.3.1 Brute Force Attacks

When considering attacks on hash functions, there are several different attacks possible, with varying difficulty. We define several properties of a hash function below (see also [Mir05]).

Definition 1.1 *Hash function H is* one-way *if, for random key k and an n-bit string w, it is hard for the attacker presented with k, w to find x so that $H_k(x) = w$.*

Definition 1.2 *Hash function H is* second-preimage resistant *if it is hard for the attacker presented with a random key k and random string x to find $y = x$ so that $H_k(x) = H_k(y)$.*

Definition 1.3 *Hash function H is* collision resistant *if it is hard for the attacker presented with a random key k to find x and $y = x$ such that $H_k(x) = H_k(y)$.*

Note that the difference between a second-preimage and a preimage attack is that in the first case the message is known (and thus the hash) while in the latter case only the hash is known. It is generally believed that knowing the input message does not make it easier to compute a second message with the same hash.

It is clear that the last definition implies the second definition, if it is hard to find a collision between two chosen strings x and y, it is even harder to find an x given a y with the same hash. It is in general more difficult to find a relation between one-wayness of a hash function and the collisional resistance. If a function is one-way, it does not mean it is difficult to find a collision. For example take a function which takes a string of arbitrary length and returns the last 10 characters of this string. Clearly, from these ten characters, it is impossible to reconstruct the input string (if this string was longer than 10 characters), but it is also easy to see that collisions can be generated without any trouble.

Now if we want to attack a hash function on the second definition, the (second-) preimages attack, the best method that works on any hash function (i.e. a generic attack) is an exhaustive search. Given a hash function H_k, i.e. given w, k find x such that $H_k(x) = w$ (with k l-bit and w n-bit), would on average take 2^{n-1} operations (yielding a 50% chance of finding a preimage for w). If we are dealing with a random H and treat it as a black box, a preimage attack is as difficult as a second-preimage attack. This means that knowing that y with $H_k(y) = w$ does not give an advantage for finding x such that $H_k(x) = H_k(y) = w$.

On the other hand, if we are looking for a collision for a hash function H, things are a lot easier. Because of the 'birthday paradox' the complexity of such a problem is only about $2^{n/2}$. Given 23 random people in a room, the chance that there are two people with the same birthday is a little over 50%, much higher than intuition would suggest, hence the name. A simple explanation is that the chance two people do not share their birthdays is 364/365. Now given p people the chance that two people share a birthday is given by

$$P = 1 - \prod_{k=1}^{p-1} \left(1 - \frac{k}{365}\right) \quad \text{for } p < 366 \tag{1.3}$$

and for $p = 23$ this yields a chance of 50.7%. Of course for $p \geq 366$ the chance is exactly 1.

The birthday paradox can be used on cryptographic hashes because the output of a hash function can be considered (pseudo-)random. Therefore if the hash is N bits long, there are 2^N hashes, but after trying only a fraction of that, we have a high chance for a collision. If we generalize 1.3 to range 2^N instead of 365, and count trials with t, we get the following expression

$$P(t; 2^N) = 1 - \prod_{k=1}^{t-1} \left(1 - \frac{k}{2^N}\right) \quad \text{for } t < 2^N \tag{1.4}$$

Now equating 1.4 to 0.5 results in the following expression

$$\begin{aligned} t &\approx \frac{1}{2} + \sqrt{\frac{1}{4} + 2 \cdot 2^N \cdot \ln 2} \\ &\approx 1.172^{N/2} \end{aligned} \tag{1.5}$$

i.e. the result is that we only have to try about $2^{N/2}$ hashes before finding a collision. This means that finding a collision is much simpler than a preimage attack. Again using the birthday illustration, finding two people with the same birthday is relatively easy (finding a collision), but finding someone else who shares *your* birthday (preimage attack) is much harder.

1.3.2 Structure of Hash Functions

To understand the impact of the different attacks possible, we will elaborate a bit on the structure of hash functions. Most hash functions consist of two parts, one part being the *compression function*, which takes a key (k) and some fixed-length input message (x) and outputs a hash. The second part is called the *domain extender* and links several compression functions together in such a way that the total hash function can operate on an input string of arbitrary length.

The compression function usually works on blocks on data, much like DES or AES do. The compression function takes a piece of data n bits long and runs several rounds of mixing on it. Before the data is hashed, it is padded so that the total length is an integer multiple of the block size, as explained in section 1.2.1. Usually, the mixing is done with the data itself as well as with some key of length k. The output is then a hash of these two input vectors and hopefully, the input vectors are not recoverable from the output hash. If the compression function is not invertible and calculating collisions is not easier than the birthday attack, then the compression function is strong and usually the resulting hash function is too.

Before the data block is used in the (non-linear) functions, it is sometimes expanded. In SHA, for example, the 512 bit input block (16 32-bit words) is expanded to 80 32-bit words (2560 bits). Then in each of the 80 rounds, a different part of the expanded message is used. This makes some kinds of attacks more difficult on the hash function, as we shall see in 1.4.1. In fact, the only difference between SHA-0 and SHA-1 is the message expansion, which makes the difference between attacks with a complexity of 2^{39} for SHA-0 versus 2^{69} for SHA-1.

In many hash functions, including SHA, the key used in the first round is some initialization vector (IV). After the first round, the output of the hash function is used as key for the next rounds. This construction is part of the Merkle-Damgård scheme, explained below.

A domain extender that is used a lot, and also in the SHA family, is the Merkle-Damgård domain extender, which works as follows. The compression function is used on an initialisation vector used as key and the first block of the input message. The output hash is then used as key for the next iteration where the second block of the data is used as input. In this way, a compression function can be used on a message of any length. Figure 1.7 shows the Merkle-Damgård scheme in action.

1.3.3 Domain Extender Attacks

There are however several attacks possible against this domain extender, regardless of the hash function used. It should be noted that the domain extender does *not*

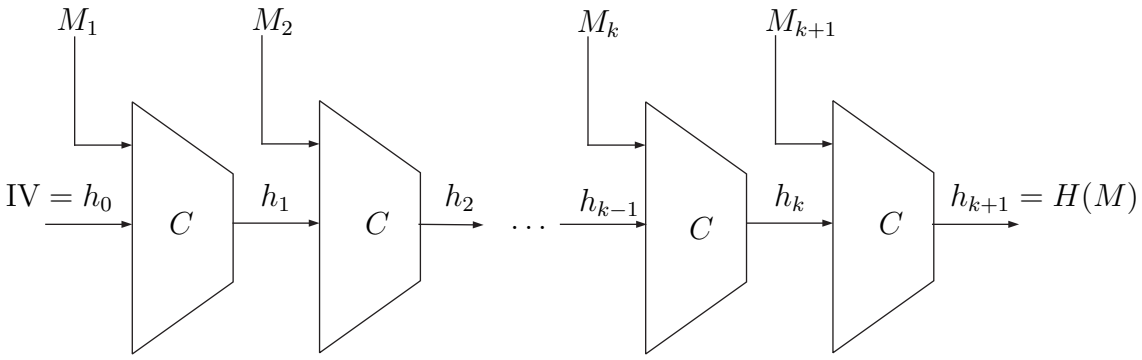


Figure 1.7: The Merkle-Damgård domain extender, see text for details.

add security to the hash scheme. A requirement of a secure hash scheme is that the compression function is safe (i.e. non-invertible and strong collision resistant), only then is the combination of compression function with domain extender secure.

MAC Weaknesses

An example of the way the Merkle-Damgård scheme can be attacked is the following. Consider the signing of a message. If we have a hash function H which works like $H_k(M) = t$ with k the key, M the message and t the signature. If we now would sign our message, and we would protect the key, one would assume that signing M would be be secure in this way.

However, if the hash function uses the domain extender explained above (like SHA-1), this is not always true. Let k be 80 bits long, and M 256 bits. Consider now the scenario where the adversary captures M and the 512 bit signature t. The tag captured is equal to

$$t = \text{SHA-1}(k||M) = C(k||M||\underbrace{100\ldots0}_{112\text{ bits}}||\underbrace{0\ldots101010000}_{64\text{ bits}}), \tag{1.6}$$

with $C()$ the compression function used in SHA-1 and $||$ the concatenation operator. If we now want to forge a message with the same signature t, we can easily construct a new message M' which is also 'signed' with k, without knowing it. Consider the message $M' = k||M||100\ldots101010000||T$, with T arbitrary text we can choose to our liking. Now when we hash this using SHA-1, this results in:

$$\begin{aligned} t' &= \text{SHA-1}(k||M||100\ldots101010000||T||\text{padding}) \\ &= C(C(k||M||100\ldots0||0\ldots101010000), T||\text{padding}) \qquad (1.7) \\ &= C(t, T||\text{padding}) \qquad (1.8) \end{aligned}$$

This hashing calls the compression function twice because the message length exceeds the block length, and therefore the domain extender is employed. However, if we look at the first iteration of the compression function, it is exactly equal to t! Since this is known we can feed this to the compression function in the second round of the domain extender as key, and then hash our own message T with that, and we have a message signed with key k. Clearly, this method of signing messages is completely broken.

Casus 1.8: Colliding Executables

A practical example of using the poisoned block attack is the program *evilize* which is available on Peter Selinger's homepage at `http://www.mathstat.dal.ca/\~selinger/md5collision/`. The program uses the attack on MD5 found by Xiaoyun Wang and Hongbo Yu in March 2005. The program 'evilize' takes a C source code as input, with a `main_evil()` and a `main_good()` routine, and links this against a specially crafted library. The program then re-arranges the data such that there is an if-statement at a special position (exactly in one MD5 block). An additional program then calculates colliding MD5 hashes which are used in conjunction with the if-statement to direct the program into the good or evil subroutine. The original program to calculate MD5 collisions can be found here: `http://www.stachliu.com/md5coll.c`.

```
C:\TEMP> md5sum hello.exe
cdc47d670159eef60916ca03a9d4a007
C:\TEMP> .\hello.exe
Hello, world!

(press enter to quit)
C:\TEMP>
```

```
C:\TEMP> md5sum erase.exe
cdc47d670159eef60916ca03a9d4a007
C:\TEMP> .\erase.exe
This program is evil!!!
Erasing hard drive...1Gb...2Gb... just kidding!
Nothing was erased.

(press enter to quit)
C:\TEMP>
```

Poisoned Block Attack

Another way to attack hash functions is with the so called 'poisoned block attack'. The idea is to find a collision using the relatively fast birthday attack described above. Once such a pair of colliding messages are found, embed these in a bigger file which has some way to hide this data. Let some party sign one of the messages, and use this signature with the other message which the signing party has never seen.

To better explain this attack, consider a colliding pair of messages N and N'. Now consider the two messages T and T' which are constructed as follows:

$$\begin{aligned} T &= M_1||M_2||\ldots||N||\ldots||M_n \\ T' &= M_1||M_2||\ldots||N'||\ldots||M_n, \end{aligned}$$

with M_i a block of data. If we now let a trusted party sign T, this signature will also work for T', since N and N' collide. Now this may not seem very useful, but if we use the conditional possibilities of for example the Postscript format, we can influence the message shown depending on whether we embed N or N':

$$(R1)\ (R2)\ \texttt{eq}\ \{\text{instruction set 1}\}\ \{\text{instruction set 2}\}\ \texttt{ifelse}. \tag{1.9}$$

If $R1$ and $R2$ are equal, Postscript executes the first instruction set, and the second otherwise. Now if we use the poisoned blocks we can choose between these instruction sets, and thus influence the output of the Postscript document. This attack is not only theoretical, but has been carried out in practice by Magnus Daum and Stefan Lucks in 2005. They constructed two files with the same MD5 hash [DL05]. In 2006, Stevens, Lenstra and de Weger constructed two colliding X.509 certificates [SLW06].

The implication of the poisoned block attack is obvious. If you can find a collision for a hash function using the Merkle-Damgård scheme, you can embed these blocks in bigger files. When using some higher level formatting language (Postscript or another language with conditionals), this block can be used to determine the flow of the formatting, branching into two different outputs. The bottom line is: don't use broken hash functions.

1.4 Specialized Attacks

Besides generic attacks which work on all hash functions, or all hash functions using some domain extender, there are also attacks which target a single hash function. One can then refine the attack by looking at the specific structure of the studied hash function. These attacks have the disadvantage that they only work on one hash function, but are usually much faster. In 1998, such an attack was launched successfully against SHA-0 and it was broken.

1.4.1 Attacks Against SHA-0

The SHA-0 hash function is 160 bits long, which means that a birthday attack takes about 2^{80} hash operations. In 1998 this hash function was broken by the two French researchers Chabaud and Joux[CJ98]. In this section, their approach is explained.

The method Chabaud and Joux used on SHA-0 can be summarized as follows:

- Linearize the hash function
- Find collisions for the simplified version
- From these collisions, find collisions for the real hash function

The linearization means that the non-linear functions are replaced by linear onces for simplicity. In the Chabaud and Joux analysis, the f_{if}, f_{maj} and ADD functions are replaced by the linear $\oplus$ function. In addition, the expansion function f_{exp} is ignored for the time being. This simplified hash function is called SHI-1. Now the trick is to look at the effect of a disturbance inserted in the expanded message. Recall that the input message block was 512 bits wide, which corresponds to 16 32-bit words. The expansion function expands this to 80 32-bit words. Now call $W^{(i)}$ the ith 32-bit word resulting from the expansion. Let $W_j^{(i)}$ be the jth bit in the ith word, with $0 \leq i \leq 79$ and $0 \leq j \leq 31$. For example $W_0^{(0)}$ would be the first bit in the first word of the expansion process.

Using these definitions, what happens if we negate bit $W_0^{(i)}$? This change will negate bit 1 in $A^{(i+1)}$, i.e. the same bit A, albeit a round later. The next change is in bit 1 of $B^{(i+2)}$ and then bit 31 is changed in $C^{(i+3)}$ (because of the rotation over 30 bits to the left, $\lll 30$). Bit 31 is also changed in $D^{(i+4)}$ and $E^{(i+5)}$. If we want to prevent further changes, we need to change some bits on the expanded input message, W. The bits we

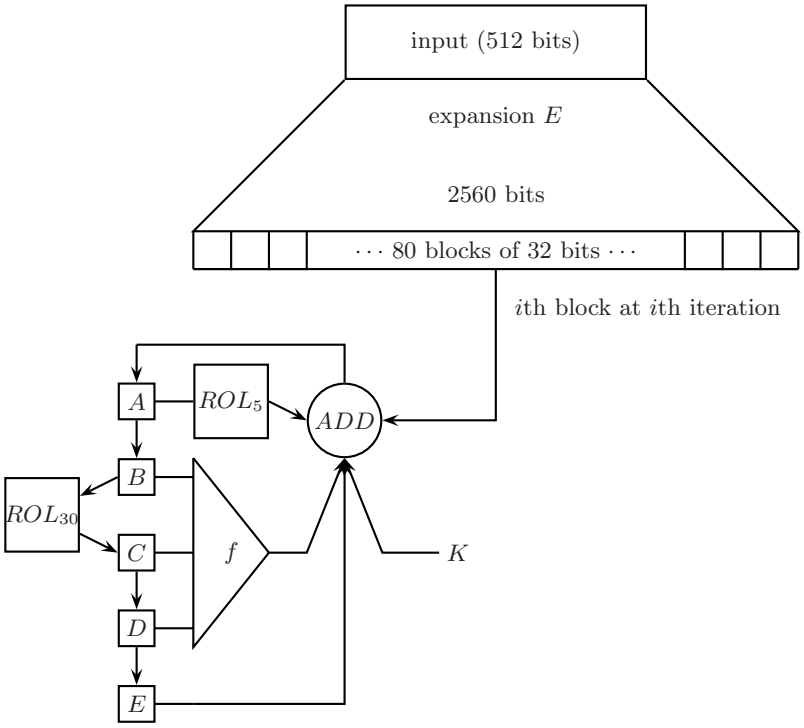


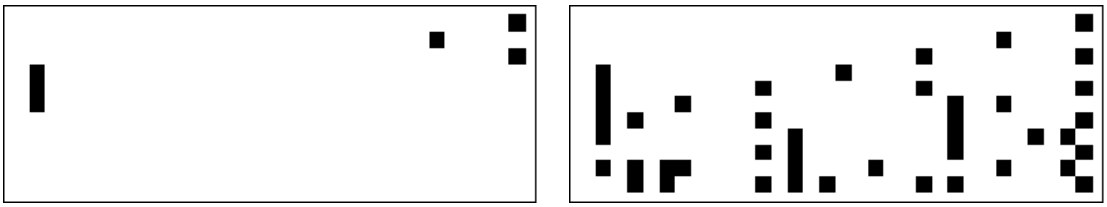
Figure 1.9: One round of the SHA-0 algorithm in a slightly different presentation.

need to negate are then $W_6^{(i+1)}$ to counter the effect of the change in $A^{(i+1)}$ (note the $\lll 5$), $W_1^{(i+2)}$ for $B^{(i+2)}$, $W_3 1^{(i+3)}$ for $C^{(i+3)}$, $W_3 1^{(i+4)}$ for $C^{(i+3)}$ and $W_3 1^{(i+5)}$ for $C^{(i+3)}$. If we change these values in W we can prevent further change in the hash function and we have a local collision.

Now for SHI-1, we can construct two colliding W's. First we choose any (random) W, and after that we construct a second W'. Now for every bit negated in W and W', we apply the correction mechanism to W'. After we apply this correction over W', these two expanded messages have the same SHI-1 hash. Because for a change in round i we need to change bits up to round $i+5$, this means that we cannot have differences in the last five rounds, because these cannot be corrected.

Below are some graphics illustrating the above mechanism. We start with a change in the first bit of the first round (located at the top right). After applying the corrections on W, we get the result shown in Fig. 1.10(a) showing the mask applied to W for the first ten rounds. If we do not apply the correction to the input message W, we get the error propagation shown in Fig. 1.10(b). Displayed in that figure is the state of A for the first ten rounds. If one looks closely, the pattern shown in Fig. 1.10(a) is visible, and in addition there are the results of not correcting the error, which obviously get out of hand.

Now we know how to construct colliding W's for SHI-1 using a mask, but this is actually the result of the expansion process, and thus not directly under control. Only the first 16 32-bit words can be influenced directly, and the rest is generated in accordance with formula (1.1). Fortunately, this function is linear in the sense that the



(a) (b)

Figure 1.10: (a) The mask for W necessary for the correction of the negation of bit 1 in round 1. (b) The result of the error in register A after negating bit 1 in round 1 in W if no correction is applied for SHI-1.

bits do not mix. Thus we want to go back from a mask which we can apply over W to a (much shorter) mask which we can apply over M. After applying this shorter mask to a message M, we have colliding messages for SHI-1.

As said above, the functions are linear and this thus means that we can exhaustively search all possibilities, there are only 2^{16} for each bit position, so in total there are $32 \cdot 2^{16} = 2^{21}$ expansion functions we have to carry out, which is feasible. After iterating over all possible input messages, we find masks to apply over M such that a collision is found, this completes breaking the SHI-1 hash function.

Of course the SHI-1 hash function is a very weak hash and is not used at all. Therefore the next step is to insert the non-linear functions again and see what effect this has on the error propagation.

Chabaud and Joux continue to define two more simplified versions of SHA-0, SHI-2 and SHI-3. They use these to study the behavior of the non-linear functions in SHA-0. In SHI-2 they add the non-linear functions f_{if} and f_{maj} compared with SHI-1 and in SHI-3 they re-introduces the non-linear ADD to update the state of variable a, but replace f_{if} and f_{maj} again with f_{xor} (compare with SHA-0/1 in Section 1.2.1). They analyze with what probability the f_{if} and f_{maj} will behave as f_{xor}. The reason is that if these functions behave as f_{xor} functions, the masks found on SHI-1 will also work, giving a collision for the more difficult SHI-2 and SHI-3. The question whether these functions behave as f_{xor} can be stated in a series of constraints, and Chabaud and Joux have found that fulfilling all of these happens with a probability of $1/2^{24}$. The most important constraint is that, due to the property of the f_{if}, no consecutive perturbations may occur before round 16. Recall that f_{if} is defined as $b \wedge c \oplus \neg b \wedge d$, now if a bit in both c and d are changed, then the output of f_{if} always changes when compared with f_{xor}. Since the f_{if} is active from round 0 to 19 a perturbation in both round 16 and 17 is allowed, because these will be propagated to state variables c and d by round 20, when f_{xor} is active.

Analyzing the ADD function we see that so called 'carries' occur, which means that a perturbation in a certain bit position might cause a bit in the subsequent position to change. As explained earlier each switched bit requires 6 perturbation to cancel it and produce a local collision. However since the negated was in the first position of word i (W_i^1) this will lead to three corrections in bits on position 31 (W_{31}^{i+3}, W_{31}^{i+4}, W_{31}^{i+5}) and these cannot lead to carries. They can finally reduce the probability of these carries to 1/4, which leads to an overall probability of $1/2^{44}$ for SHI-3.

The next step is SHA-0. Analyzing the combined behavior of SHI-2 and SHI-3 they found an interesting additional property, namely f_{maj} behave exactly as f_{xor} if the bits in consecutive words are not equal: $W_1^i \neq W_1^{i+1}$. They add this constraint to the perturbations of rounds i and $i+1$ when $36 \leq i \leq 55$ (because perturbations in those rounds will be propagated when f_{maj} is active). After suppressing probabilities implied by perturbations in round 14, 16 and 17 they arrive at their claimed complexity of 2^61.

1.4.2 Attacks Against SHA-1

Because in SHA-0 the bits of the words are not interleaved, every bit on a certain position i within a word will only depend on a fixed set of bits. In fact the 16 bits on any position i in $M_0, \ldots, M_{15}$ define all 80 bits in $W_0, \ldots, W_{79}$ in position i according to relation 1.1. This allowed Chabaud and Joux to exhaustively enumerate this expansion, 2^{16} possibilities, and easily reverse it. Because SHA-1 does interleave the bit their approach will not work straight away for SHA-1. Wang e.a.[WYY05a] replace each bit in the disturbance vector by a 32-bit word for which each entry has to satisfy equation 1.1, the SHA-1 message expansion. This however leads to a massive search space of 2^{512}, they use heuristics to limit the search space and find good disturbance vectors. Further explaining their methods goes beyond the scope of this work, but is clear that the addition of the single rotate operation to the message expansion of SHA-0 solved a major weakness.

1.4.3 Timeline Academic Attacks on SHA Family

Finally in this section we list a time line showing major milestone in the academic analysis of the SHA hash family.

SHA-0:

- 1998 Chabaud e.a.[CJ98] introduce an attack with complexity 2^{61}. The first attack with a complexity lower then the expected complexity of a birthday attack.
- 2004 Chabaud e.a.improve the result to 2^{51}.
- 2004 Wang e.a.[WYY05b] give an attack with complexity 2^{40}
- 2006 Naito e.a.[NSS+06] formulate an attack with complexity 2^{36}

SHA-1:

- 2005 Feb. Wang e.a.[WYY05a] present an attack on SHA-1 with complexity 2^{69}
- 2005 Aug. Wang e.a.improve on their result proposing an attack with complexity 2^{63}

The first real collisions were given by Wang, and it is expected that it will become feasible to calculate collisions for SHA-1 in the (very) near future.

For SHA-2 the best result so far is a collision for SHA-2 reduced to 19 out of its 64 rounds proposed by Mendel e.a.in 2006[MPRR06]. Considering the complexity increases exponential with the number of rounds we can conclude that SHA-2 is not even close to broken at this moment.

1.5 Implications of Attacks

As discussed earlier an attack has aim to find two messages x and y such that their hash result collide, i.e. $H(x) = H(y)$. We can quickly conclude that in order to make a successful attack the attacker needs to control both messages. If one of the messages is fixed, the attacker needs to perform a second-preimage attack, which is, as mentioned in Section 1.3.1, a lot harder and there are no such attacks known on SHA-1 at this moment. Because of this all messages, which are signed before a feasible collision attack became known, are safe. Also, because the attacker needs control over both messages, any party who creates and signs documents himself does not have to worry about collision attacks.

However there is danger in any scenario where another party creates or controls the document which is to be signed; or any scenario where there is dependence on a document created and signed by another party. In the first scenario a malicious party could offer a document M for signing for which he has calculated a document M' with colliding hashes. Once received the signature on document M will be valid for M', and can thus be simply copied. In the second scenario, where the malicious user produces and signs a document M himself, it is possible for the attacker to deny ever having signed M by presenting M' and arguing he is victim of a forgery.

Even though current attacks heavily constrain the difference between messages and often fix large parts of them to meaningless random values we have seen that only one such colliding block can be used to construct a poisoned block attack (see 1.3.3). The above applies to hash functions in general for which collisions are possible, but preimage attacks are not. At the time of writing, collisions for SHA-0 can be found in practice, SHA-1 collisions are underway and are expected this year, while SHA-2 collisions are still practically impossible. Also, because of the longer hash size of SHA-2, the algorithms suffer less from a attack which reduces the work by a factor of, say, 2^{20}. Even after such an attack, colliding the simplest version of SHA-2 is still more difficult than it ever was for SHA-1.

1.6 Conclusion

As we have illustrated, SHA-0 is completely broken. The best attacks are becoming feasible to calculate within a reasonable time and as such, this function is completely insecure. Fortunately this was already foreseen at the time when SHA-1 was published. This hash remedies parts of the problems with SHA-0, but at this time there are already attacks possible that are faster than the birthday attack. This is not very remarkable as the hash functions are almost identical, the only difference being the message expansion. The chance that this hash function will also succumb is therefore not unlikely.

The SHA-2 hash functions on the other hand have as of now not been broken. The best attacks possible only work on a reduced version, which gives hope for the strength of these functions. Again, the fact that this hash function is not broken yet is not

very remarkable, as it is much more complex than SHA-1, something that can easily be seen by comparing Figures 1.5 and 1.3. Even if SHA-2 is broken though, it will be quite some time until these attacks will be feasible. Since a SHA-0 hash is 160 bits, a birthday attack has a complexity around 2^{80}, but since the SHA-2 functions produce much longer hashes, the complexity of birthday attacks against these functions range from 2^{112} to 2^{256}. Even if the latter would be broken with the complexity reduced to 2^{128}, which would be quite a successful attack, it would still be much much stronger than the original SHA-1 hash function.

This leaves the authors to wonder why such relatively short hash functions are still used. SHA-1 is very common and produces 160-bit hashes. Although a complexity of 2^{80} is not yet reachable, it is quite close to what is actually feasible. If the hash length would be doubled, the complexity would raise to 2^{160} which is completely impossible to attack. Even when it would be broken severely, it would still pose no practical problems for years to come. Furthermore, the cost of the additional hash length is small, a naive doubling of the hash length would cost a factor of two more in computing time. If the current rise of 64-bit processors is taken into account, optimized implementations could perhaps even be faster.

In any case, the story of SHA seems to be a story true for any hash function. MD5, the hash function used before SHA (and which is still used a lot today) also suffered the same fate. First a seemingly unbreakable hash function is released, using all kinds of complex functions to mix the input with some key. Then later some cracks begin to appear in the design and finally the hash function is broken. It seems that widely used hash functions are not much more than a collection of rounds, message expansion and some complex linear functions consisting of AND, OR and XOR gates. The result is something that looks impressive, but has never been proven to be secure.

This problem could be solved by using hash functions based on RSA or ElGamal, which have been proven to be secure (if at least the discrete log is hard, which is very probably is). The security of these hash functions would be beyond doubt and the only worry we would face is that the keys at some point are too short and can be brute forced, but this is of course true for any cryptographic function. The reason that these number theoretical routines are not used is of course that they are way too slow, and the second best people settle for is a seemingly complex function that runs fast on computers.

The story of MD5, which was published, attacked and broken, also seems to hold for SHA-0. The first cracks are beginning to appear in SHA-1 which is not surprising considering the similarity between the two. SHA-2 is still going strong as these functions are much more complex and also use longer hashes, requiring a much more effective attack for collisions to become feasible. In any case, the open contest for SHA-3 recently started and the successor will be announced in 2012. Time for a new cycle of to begin.

Chapter 2

Security and privacy of RFID tags

Written by *Jeroen van Wolffelaar.*

Radio Frequency Identification (RFID) use is increasingly widespread. Because communication is both opaque and contactless, there are several privacy risks. RFID circuits might be read by their corresponding readers without the owner of the tagged item knowing. Communication can also be overheard and/or interfered by interested and/or malicious bystanders, attackers can act as man in the middle, or malicious party can talk to RFID chips directly without the owner realizing.

But even if communication happens with full knowledge of the chip 'owner', without any observers/attackers, the communication will typically still be opaque – the owner of the RFID device does not know what's going on.

2.1 RFID tags

RFID tags are minuscule chips, as small as $50 \times 50 \times 5$ μm, but often closer to $1 \times 1 \times 0.1$ mm[Ten]. They are powered by and communicate using radio waves, ranging from low frequency (30kHz) to Ultra High Frequency (3GHz). The range of operation depends on the frequency and presence and type of antenna, and can be anything from 10cm to hundreds of meters – though typically not more than a meter, and a lot of RFID tags go no further than 10cm in normal circumstances. The antenna can be simply a fiber in paper. RFID chips can be as cheap as 5 eurocent a piece[MR], although most are more expensive, typically no more than a euro though.

Figure 2.1: VeriChip, about twice the size of a grain of rice

The RFID chips can hold arbitrary circuits, Turing-complete chips are feasible, some chips have a 3DES coprocessor such that they can perform current-day strong cryptography. RFID chips are typically equipped with a little bit of EEPROM memory, from 512 bits to a couple dozen kB.

With the increasing usage of RFID, clear communication can become a problem. Therefore, a lot of chips have some form of collision-detection implemented, if a collision is detected and the chip believes it is not the one being talked to, it will refrain from sending.

2.1.1 RFID chip types

RFID chips have varying capabilities, in this section a bit more detail is given about the most common types.

Pure identification tags The simplest form of RFID are the tags, commonly found in warehouses, clothing, etc. The tag supports no other operation than responding with its ID, a single number, for example 128 bits long. Essentially this is a remotely readable bar code, and it implements the same security: none. An example is the VeriChip for human and cattle tracking, see section 2.2.

RFID tags with writable storage More advanced RFID tags have some writable storage. An example is the NXP Mifare Ultralight chip[1]. This chip provides two security features[NXP]:

- 12 4-byte pages individually lockable to a readonly-state.
- 32 "turn on only" bits

RFID chips with security features Of course there are also RFID chips with actual cryptographic authentication, implying some private key being present on the card. A popular type of RFID chip (the Mifare Classic) uses a proprietary protocol, see more about that in section 2.4. Modern passports use the well known and public 3DES cipher (see 2.3).

2.2 Case: Baja Beach Club

This section studies privacy aspects of RFID chips that can only emit an ID, and have no other security

By far the most RFID tags are essentially remote-readable bar codes. A particularly interesting application is the VeriChip[2], a chip for tagging humans. The chip is a simple, no-security ID tag, and can be implanted in humans or animals. At the end of 2007, about 2000 people have a VeriChip implanted in their shoulder[Mor]. Since anyone can read these chips with appropriate equipment, anyone can uniquely identify the tagged people this way, it's like having your primary key printed in large letters on

[1] `http://www.nxp.com/products/identification/mifare/ultralight/`
[2] `http://www.verichipcorp.com`

Casus 2.2: Tin Foil hats

Certain conspiracy theorists believe the government, aliens, or a cooperation thereof are trying to control the minds of them, mere mortal civilians. To counter this threat, so called tin foil hats were developed: a foil of (nowadays) aluminium, wrapped around the head, to protect the brain from manipulation and interference by radio waves.
In 2005, a couple of students of MIT conducted an empirical study to the effectiveness of tin foil hat form factors, comparing the "classical", "fez" and "centurion" styles of hats, see `http://people.csail.mit.edu/rahimi/helmet/` for their story (with pictures and video). Their conclusion:
The helmets amplify frequency bands that coincide with those allocated to the US government between 1.2 GHz and 1.4 GHz. [...] It requires no stretch of the imagination to conclude that the current helmet craze is likely to have been propagated by the Government, possibly with the involvement of the FCC. We hope this report will encourage the paranoid community to develop improved helmet designs to avoid falling prey to these shortcomings.[RRTV]

your forehead. Unlike bar codes, line of sight is not required, and reading is a matter of milliseconds. The possessor of the chip is not informed or asked permission when reading is happening.

Of course, just having some long number does not give away any significant information yet. In all applications with tagging RFID chips, there is an associated database (possibly distributed) correlating the ID with useful information. Privacy concerns of such databases *per se* are outside the scope of this chapter.

Using a no-security tag for authentication is of course possible, but must be regarded *security by twilight* (obscurity would be too much honour). After all, using appropriate hardware, anyone can clone such tags. Care should therefore be taken to not grant sensitive access solely based on such RFID tag. The Baja Beach Club Rotterdam[3] grants access to the VIP lounge, and allows drinks to be deducted from the VIP's balance using their VeriChip implant, voluntarily injected. Identity theft becomes pretty literal this way.

In the case of a VeriChip, the person wearing the chip is aware of him/herself wearing a tag. However, tags on items (clothes, ...) you buy might not be as readily visible.

2.2.1 Summary and notes

- Private data security as with bar codes, magnetic swipe cards
- Caveat: invisibility of radio-waves does not mean invisibility to attackers, higher awareness required on ability to read and clone, even without line-of-sight

[3] `http://www.baja.nl/`

- Effective counter-measure to unauthorized reading: shielding (if done properly, see casus 2.2)

2.3 Case: Biometric passports

This section studies privacy aspects of RFID chips that store a lot of (personal) data on the chip itself

Biometric data, essentially a binary string of often much more than 10.000 bits, cannot easily be stored optically on paper. But not only machine-readable data capacity was seen as a problem, also authenticity is. Many modern passports therefore have an RFID chip in them, containing all the personal data, including photo, and optional biometric data, in digital form. The way these so-called *biometric passports* work is defined by the International Civil Aviation Organization (ICAO)[4], document 9303[Org]. This United Nations agency was founded in 1947 as a result of the 1944 Chicago convention, and has since its founding defined numerous standards for international travel documents. In 1968, the ICAO started to work on defining a standard for machine-readable passports. In 1980, they published the result of that work: the most important personal details were to be written in a standardized typewriter font, called OCR-B, in two lines at the bottom of the primary identification page. The font was, as the name suggests, specifically designed to be easily readable with OCR (Optical Character Recognition), and those two lines are together known as the *Machine Readable Zone (MRZ)*. In 2005 all participating countries agreed to mandate this standard by 2010, for new passports only, 30 years after initial publication.

In 1998 the ICAO began to work on a biometric passport specification. Optical storage is then not adequate anymore, although current day 2D barcodes can store up to a megabyte of data. However, the ICAO decided that the new passport specification was to be using some form of electronic data storage. By September 2001, when the WTC terrorist attack happened, most of the work was already done, but it took until 2005 before the result was published as a standard. The standard defines numerous security measures, aimed at preventing unauthorized changes, unauthorized remote reading, and unauthorized copying.

The key feature is the presence of a signature by a government public/private key pair, certifying that the passport is genuine and has not been tampered with. Of course, the RFID chip can still be cloned, but it's no longer possible to replace the photo or change name, birth date, etc. Using *Active Authentication*, also cloning becomes impossible, by means of a Challenge-Response phase based on a private/public keypair stored on the passport. The rest of this section will focus on measures against unauthorized remote reading.

[4] `http://www.icao.int/`

Remote reading Three researchers from UCL[5], Louvain, Belgium conducted a study on biometric passports as issued in Belgium since 2004. Contradictory to what the responsible minister claimed, they discovered that the first generation biometric passports issued in the first 1.5 years did not possess any security measure at all.

The **second generation** passports do implement *Basic Access Control (BAC)*. The authentication is based on a Challenge-Response using 3DES. The secret key for BAC is a hash of the Machine Readable Zone.

Summarizing ICAO 9303 Part 1 Volume 2 Appendix 5:

Biometric passport Basic Access Control

Passport		Reader
	1	$K_{ENC} = \text{Select}_{112b}(\text{SHA1}(MRZ))$
Select random C_P, K_P; Send C_P	2	Select random C_R, K_R
	3	Send $a = \text{Enc}_{K_{ENC}}(C_R \mid\mid C_P \mid\mid K_R)$
Decrypt a and check received C_P	4	
Send $b = \text{Enc}_{K_{ENC}}(C_P \mid\mid C_R \mid\mid K_P)$	5	Decrypt b and check C_R
Continue encrypted communication using $K_P \oplus K_R$ as session key		

Not mentioned in this protocol description is the use of a distinct second key derived from the MRZ to checksum all exchanged messages, to further secure the protocol. Note that the encryption key (just as the checksumming key) is using only $2 \cdot 56 = 112$ bits from the hash function: the first and last key of the three DES keys used for encryption are the same! Considering the slow operation of the RFID chip (400 tries per second in one example), brute forcing a 112 bit key is still infeasible. Even single DES would be expected to take 2.8 million years. The passport does not function as an oracle (note how the reader is supposed to send the first encrypted message, and the passport will not respond unless its challenge is correctly encrypted), so brute forcing the encryption or hash key on alternative equipment is not possible. Eavesdropping a successful conversation does enable this possibility, though.

Brute forcing The major weakness in the implementation is the selection of what information from the MRZ is actually used to hash and derive keys from. Only the carrier's birth date, the document number, and the document expiry date are used. Considering that the birth date might be readily available from the victim's OV-chipkaart (see 2.4) or estimated based on appearance, only document number and expiry date are left to be determined. Expiry dates are in a five-year interval with uneven distribution, and the document number is in some countries (such as Belgium) linear and strongly correlated to the issue date. Combining all this, brute forcing the security key is a feasible attack, allowing an attacker to read the passport contents unauthorized.

Other vulnerabilities Some information of a passport is given even before the BAC authentication. The chip responds in certain ways to unauthenticated requests, depending on the exact chip design. Also the RFID chip ID is given before authentication. If,

[5] `http://www.uclouvain.be/`

like in the United States, the chip ID is not random but has a fixed prefix, one can deduce the nationality easily and quickly from a passport. In order to reduce the chance of unintended remote reading, the United States have implemented a wire frame in the cover of their passports, such that when closed, the RFID chip is nearly completely shielded by the Faraday cage so formed.

2.3.1 Summary and notes

- Remote reading of personal data is a serious threat to privacy
- Although a sound and secure protocol seems to be defined for the unauthorized reading of passport data, lack of key entropy endangers this security. Therefore, it would be advisable if countries would take care to generate high-entropy passport numbers: adequate length and random allocation. In hindsight, it would have been better if the ICAO standard would have used more of the MRZ for the key material
- Low-tech security measures like the US Faraday cage should be adopted by more countries
- Holders of current biometric passports are well advised to take care of their own passport shielding

2.4 Case: OV-chipkaart

This section studies privacy aspects of smart RFID chips: chips that take part in some complex protocols, implementing security measures to prevent unauthorized manipulation, reading and cloning in a hybrid system with on-chip storage of closed information, such as monetary balance

The "OV-chipkaart" is an all-in-one fare ticket system for the Dutch public transport, set to replace conventional train tickets, the ubiquitous "strippenkaart"[6], and any type of season ticketing. The card is currently in pilot-phase in Rotterdam, and 1 billion euro has been spent so far on development and hardware. The official plan is to deploy the card countrywide in the beginning of 2009, but the recent results by two German researchers with the chip and other, unrelated, technical problems make it very uncertain whether that target date is going to stand.

There are two types of cards in use for the OV-chipkaart: paper disposable tickets, and plastic personal cards.

Casus 2.3: Hacking OV-Chipkaart disposable tickets

In July 2007, two students from the University of Amsterdam set to investigate the security of the disposable OV-chipkaart (Dutch public transport ticketing system)[SS]. These disposable tickets use the Mifare Ultralight RFID chip. They identified a number of implementation and design problems, such as disabling the invalidating process of the card, and allowing them to reset the card after an itinerary. All of these attacks were done with genuine disposable tickets and a simple (100-200 euro) RFID reader. The major implementation problems have been fixed in the meanwhile, but the system remains vulnerable, especially to simulating the card with user-controlled RFID hardware.

2.4.1 Disposable tickets

The paper disposable ticket, for n-day use and/or m-fare use, can be bought at ticket vending machines. It was largely reverse-engineered in July 2007 by two students (see casus 2.3), and consequently fixed: but the choice of RFID chip makes it impossible to counter cloning with custom RFID hardware, as demonstrated in January 2008 on Dutch television. Hacking the card might be a bummer for the public transport operators, but in itself is not harmful to privacy. However, multiple issues exist that have a downside to the card *user*:

- The itinerary log is readable by anyone from the disposable card (the data is not encrypted). This can be easily changed without the need for different hardware
- A relay attack makes it possible to travel on a victim's card without his/her consent or knowledge: an attacker travels with an RFID relaying device that forwards (via GSM, for example) frames to an accomplice who relays the frames from and to a victim's card. This cannot be resolved with the current hardware choices.
- Vandalism is trivial: anyone with proper equipment can write garbage to a disposable ticket and then locking it, thereby permanently invalidating the ticket

2.4.2 Personal OV-chipkaart

The personalized (also available is a limited anonymous variation) plastic credit-card sized regular OV-chipkaart will be the most used card in the Dutch public transport system. It can be loaded with a balance, season-ticket products, or a combination. In addition, it features an auto-load feature: if the balance drops below a user-settable threshold, the associated bank account will be charged a preconfigured amount and the balance will be increased by that amount.

[6] a paper with 15 strips, to be stamped when travelling, the number of strips stamped depends on the distance travelled

The card is using the NXP Mifare Classic 4k[7], which uses a proprietary security protocol: to communicate with the card, a key is required. Authentication is happening with a Challenge-Response protocol, and subsequent communication is encrypted. There are different keys, each having different permissions per page on the card: some keys might only be able to read certain pages, others can also write, or only increment/decrement, or have no access at all.

On the card the balance, season ticket products, the last 10 transactions, and balance threshold are stored, plus the birth date of the holder (only on non-anonymous cards). No other personal data is stored, but Trans Link Systems[8], the consortium developing and deploying the card, stores such data in their own databases.

Although the mode of operation of the chip is documented on the website of NXP[9], details on how the crypto in the card works are unavailable to the public. It is also completely unknown how Trans Link Systems is actually using the features of the chip, they disclose next to nothing about their algorithms, 'for reasons of security.'

2.4.3 Reverse-engineering secret protocols

In 2007, Karsten Nohl and Henryk Plötz (see figure 2.4) worked on reverse-engineering the Mifare Classic card using several techniques[NP]. They presented their results at the 24th Chaos Communication Congress[10], December 2007.

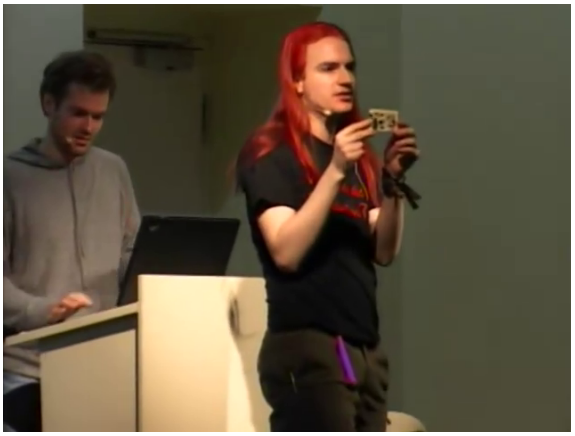


Figure 2.4: Henryk Plötz demonstrates equipment to simulate any RFID chip, with Karsten Nohl in the background, at 24th CCC in Berlin, December 2007

On the one hand, they tried to visually observe the inner workings on the chip. Using a regular optical 500x magnification microscope and polisher paste, they exposed the layers of the RFID chip, with logical gates and transistors on the bottom layer and connection wires on the five layers above. They discovered that the random number generator on the chip effectively has only 16-bit randomness, and that the input to the cipher was based on a 48-bit key, using a simple Linear Feedback Shift Register. Not all of the protocol was discovered yet, though.

On the other hand, they used a computer with RFID antenna to communicate with the card directly, and observed bidirectional communication from the card with a Mifare reader. Using this technique, the researchers not only confirmed that the random number generator was giving only 16-bit of randomness, they also found out that it was only dependent on the time between card power-up and start of communication – something

[7] `www.nxp.com/products/identification/mifare/classic/`
[8] `http://www.translink.nl`
[9] `http://www.nxp.com/products/identification/mifare/classic/`
[10] `http://events.ccc.de/congress/2007`

that is not very random, at least, not at the resolution this is apparently measured by the card.

Combining all this, it is fair to assume that the cryptographic protocol used in the card will be known in the near future, it might even already be known to malicious parties with more time and resources to continue the reverse-engineering. With a key size of 48-bit (8 bits less than DES), and the significantly smaller processor requirements compared to DES, using a computer it should be possible to break the cryptographic security in a matter of seconds using brute-force, if no better attack based on weaknesses in the protocol can be found.

It is not known yet how actual data is stored on this chip. Like with the disposable ticket, certain information and/or a hash over the contents might be stored on the chip, making manipulation and reading of the chip contents harder, possibly computationally infeasible lacking access to private keys stored in readers only. With the disposable ticket itinerary information was stored unencrypted, and on this chip with more built-in security, the same might be the case. But this does not matter: once the card security is broken, it's possible to clone the chip, and the clone is fully functional: all itinerary details can readily be queried on OV-chipterminals, and one can travel with the clone and using the balance on the chip. Because copying can happen without the owner knowing, it is not possible to catch the abusers by looking up the personal details of the cloned card, after cloned use is detected, and alert the police with that information.

2.4.4 Summary and notes

- Security by obscurity is ineffective: Kerckhoff's principle is still valid. The secrecy on the implementation seems to only slow down discovery of the workings of such 'smart' RFID chips, but not prevent it, while with an open architecture, weaknesses can be identified and resolved in a public request-for-comments stage of the design
- Unless a publicly scrutinized cryptographic protocol is employed on the chip, and even then only if the implementation has no flaws, no data should be stored unencrypted and unsigned on RFID chips, but stored as encrypted, signed blobs using keys only known by the reader software, to prevent leaking private information from the RFID chip

2.5 Conclusions

In the world of RFID chips, there are many implementations with inadequate provisions for privacy. Consumers are largely subject to the mercy of big commercial corporations, and often fail to recognise the severe privacy implications themselves. A key aspect of RFID chips is that their owner does not generally know what information is transmitted and what information could be obtained by malicious parties. Without extreme good care, the wireless technique will most of the time be subject to abuse.

It would be good if governments would:

1. Start publicity campaigns to make the public more aware of the risks
2. Introduce legislature forcing implementing parties to implement adequate privacy measures, similar to current data store legislature in effect in many countries, with steep penalties as failures are hard to resolve quickly
3. Introduce legislature forcing a visible mark on objects containing RFID tags, such as the passports already have
4. Introduce legislature to prevent RFID chips from being left enabled after having done their thing: making tags in consumer goods self-destruct no later than selling the good to consumers

In addition, companies wishing to implement protocols based on smart RFID chips should realize that they should not save on cryptographic protocols, in their own interest. Keeping protocols secret is not helpful in the long term, and reputation damage is hard to repair. Especially in large-scale deployments, interest by (often well-intending) researchers is bound to be large, and is better used to your advantage rather than disadvantage.

Chapter 3

Xbox Securtity

Written by *Miloslav Valco.*

In November 15, 2002, Microsoft launched its video game console Xbox. It attempted to use its software developing experience to enter a new market and establish its position among the fierce competition. With its high computational power, great graphics and high speed broadband connectivity options, Xbox proved a worthy opponent to Sony's Playstation 2 and Nintendo's Gamecube. Microsoft's profits from the console's release however were hardly optimal. This is mostly because of the numerous and severe mistakes it made in the implementation of the console's security. This study explores the most important of these and how they can be abused by an attacker. It investigates two methods of encryption: the use of the RC4 algorithm and its successor, the Tiny Encryption Algorithm. It reveals further security deficiencies that allow an attacker to avoid the crypto-system altogether. How severe are the mistakes? Could they be prevented?

3.1 Microsoft's Financial Gambles

Xbox was released at the same time as two other, well established consoles: Sony Playstatition 2 and Nintendo Gamecube. Microsoft did so to make a strong statement:

it is serious about entering the market and wants to be compared to the best. Although Microsoft is a massive multinational corporation, by doing so it made a huge financial gamble that could have disastrous results if Xbox could not live up to the expectations [Gra07].

Expectations were high indeed. For a year before its release, Playstation 2 was flooding the media with game commercials promising never-before seen image quality (not even on newest PCs). Nintendo was very reputable among gamers, especially due to age-proven high game content quality. Gamecube also possessed graphics comparable to Playstation 2 and while Sony set the price tag to `$299` dollars, Nintendo sold Gamecube for a whole `$100` cheaper [Gra07].

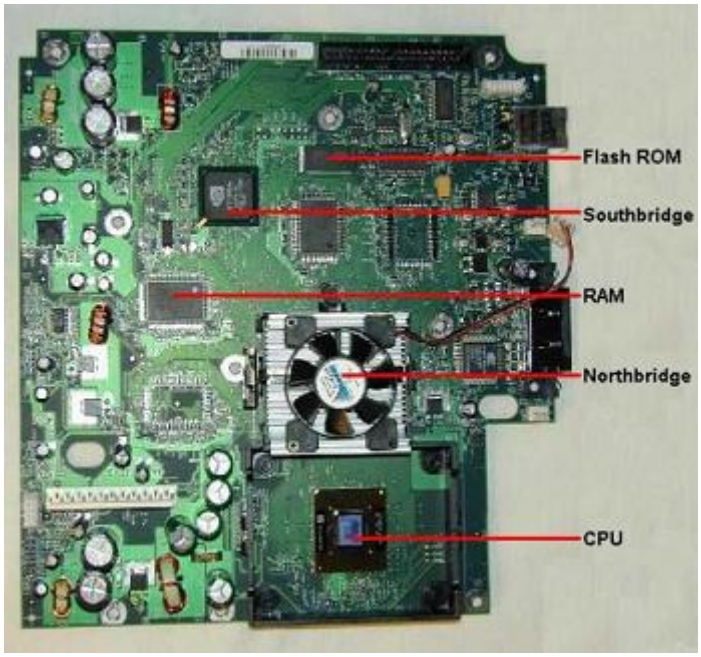
How can a small device for `$200` deliver equal visual performance for video games as a brand new PC? The hardware is simply sold below its real price and owners of the console make up for the difference and secure future profits by purchasing original games [Ste05]. Microsoft decided to match Playstation's price and give away even more for it. Xbox came with equal quality graphics, 10 GB of hard disk space and high speed Ethernet for online play (available a year after console release). By doing so, Microsoft committed billions of dollars to design the product, purchase the parts and set up production lines. Xbox was sold for `$300` dollars (later the prices dropped by an additional `$100`), while the cost of the components was estimated to be `$425`. Furthermore, Microsoft also had to finance a massive advertising campaign to make up for the reputation of its competitors [Gra07].

A year after the release, it seemed that Microsoft had succeeded. The retails placed Xbox in the second place, beating Gamecube and leaving it slightly behind Playstation 2. Microsoft utilized its experience in dealing with competition and succeeded in creating something brand new that attracted a lot of customers. Surviving among the competition however was only a part of the gamble. Microsoft had to keep up the sales for at least three years to make up for its losses [Gra07].

The second gamble Microsoft made is security. Xbox is not scientifically innovative enough to deserve a separate patent (establishing intellectual rights would probably be too costly and take too long anyway). Microsoft therefore could not prohibit the customers to deconstruct their Xboxes. It had to make sure that the components inside Xbox's plastic cover cannot be used for anything else besides running original games. Customers were also expected to later purchase extra features, such as the possibility to use the DVD drive to play movies and to play games online. Xbox had to make sure they would not be able to access these without the purchase [Ste05].

We will see that Microsoft's unconventional approach to security and the resulting mistakes were caused by attempting to maximize the profits. Most of the components were chosen or switched for the cheapest ones possible and important decisions were often taken too hastily.

3.2 Andrew "bunnie" Huang



Andrew "bunnie" Huang was a PHD student at Massachusetts Institute of Technology (MIT) when he decided to deconstruct an Xbox and explore its security. Immediately after exposing the insides, he discovered it looks a lot like a normal PC (see picture). In many ways it is indeed just a normal PC, similarly like Gamecube and Playstation. Sony and Nintendo however took more effort to purchase parts specially customized for their consoles. To save as much money as possible and to meet the competition deadline, Microsoft used unmodified PC parts wherever possible. Andrew quickly figured out an Xbox contains 733 MHz Intel Pentium III processor, nVidia GForce 3 improved chipset, 64 MB RAM, traditional Busses, 100 Base-T Ethernet, USB ports, 10 GB hard drive and DVD-ROM drive [Hua02].

Andrew looked at Xbox startup next. As every PC, Xbox contains a flash ROM - a small memory that contains the boot block. The boot block loads the operating system (Xbox uses windows 2000 kernel) into RAM memory and afterwards surrenders control. When designing the security, Microsoft decided it has to somehow check that windows 2000 kernel is indeed the only executable code at this point. Once it takes over the computer, it can itself check that every other running process is a valid and original. Windows 2000 itself uses a well implemented RSA signature scheme that can be trusted. The flash ROM however cannot. Anyone could change its contents or simply replace it completely and run whatever code he wants. Andrew also started by dumping the flash ROM to see how it is protected (dumping = transporting the contents through a low-speed bus and tapping the bus to see the contents) [Hua02].

At the very beginning of the flash ROM, Andrew found a strange piece of code, which was actually an early version of the startup security algorithm (the encryption algorithm and secret key changed later). He also discovered (through trial and error - removing the flash ROM, rewriting the commands) that this piece of code never gets executed; the flash ROM does not use its first 512 bytes, it skips them (unless the rest

was modified, in which case it panics and crashes). Thanks to the old piece of code however, Andrew gained some understanding of what happens in the start-up process [Ste05].

One way to make a flash ROM "irremovable" is to integrate it into another chip, preferably the processor. This way, the flash Rom would become very temper-resistant: removing, rewriting even reading or finding it would become impossible for everyone except the richest governments in the world (capable of looking at layers of transistors with an electron microscope). Microsoft however could not afford custom made processors and integrating the whole flash ROM into another chip would be too costly anyway (it has to be possible to reprogram the flash ROM during production and repairs). They therefore decided to introduce another element in the chain of trust. They integrated a small (512 bytes) "secret ROM" into "Southbridge" (nVidia MCP: media and communications processor). This small memory contains the new version of the security algorithm. Since 512 bytes of memory is too little to check the entire contents of the flash memory, there is another link in the chain of trust. The secret ROM verifies only a small boot-loader stored the flash memory, which in turn verifies the rest of the flash ROM. Like the windows 2000 kernel, the boot-loader is large enough to use a more secure encryption algorithm (SHA-1) [Ste05].

Andrew realized he is looking for a secret ROM hidden outside the flash ROM and since he could not see it, it was probably integrated in of the other chips. He considered Microsoft's and his own financial options and decided the secret ROM is probably not going to be in the processor or the Northbridge. Customizing these would make them much more expensive and if they were, Andrew did not have the means to tap them. He went for the first option that seemed within his reach: Southbridge. Since the secret ROM is not inside the processor, it needs to send data to it through a bus. Microsoft admitted they considered the HyperTransport bus that is used for this purpose safe; complex data travels through it too quickly [Ste05]. Andrew utilized the facilities of MIT and his own knowledge and talent to construct a custom tap-board, capable of sniffing the data he needed (it cost him some `100$`). He analyzed the data and discovered how the chain of trust works [Hua02].

The chain of trust starts with the secret ROM, which decrypts the boot-loader with a secret key stored inside. Afterwards it hashes the boot-loader and stores in RAM. Since hashing usually uses some form of feedback (making future values dependant on previous ones), it makes sure that if any one byte of the boot-loader was changes, all bytes will change and only the final bytes need to be checked (the "magic number"). If the last few bytes of the hash are correct, the secret ROM surrenders control to the boot-loader. The boot-loader initializes and checks the windows 2000 kernel, which takes over and checks any further processes. If any of these checks fail, Xbox enters panic mode and stops working (flashes a wrong code LED) [Ste05].

```
For i = 0 to 2^(n-1)
S[i] = i
j = 0
For i = 0 to 2^(n-1)
j = j + S[i] + K[i mod l]
Swap(S[i]; S[j])
```

Algorithm 3.1: Initializing S-table

3.3 RC4

Andrew published his discoveries and when more people looked at the data, they quickly saw many weak points in the security set up. Most importantly, the secret ROM uses the RC4 encryption algorithm. RC4 was probably chosen, because it is small enough to easily fit inside the secret ROM. It is used both to decrypt the boot-loader (an attacker must not be able to read its contents) and to hash it (both the secret key and the magic number are stored inside). We will see that due to the properties and weaknesses of RC4, it was the wrong choice for this purpose.

RC4 inside the secret ROM uses a 16 bytes secret key (128 bits; 2128 different possible keys). RC4 is a stream cipher: it uses the key to generate a key-stream, which is combined with the plaintext (using XOR). RC4 operates with words; words are groups of bits of size n (usually 8, 28 = 256). First RC4 takes an l-bytes long key and generates a table S with size 2n-1. S is initialized with words in ascending order. Afterwards two counters and the key are used to exchange the position of the words [Was01]. Algorithm 2.1 shows the procedure written in pseudo-code.

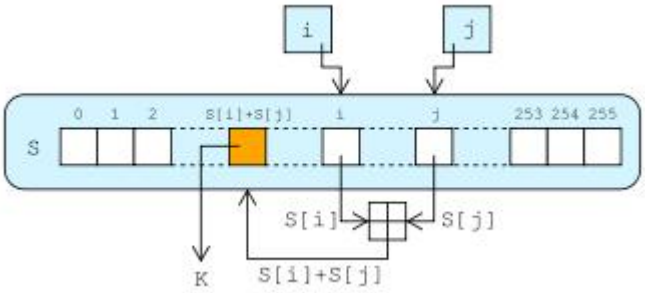
Afterwards RC4 uses the values from the S-table to generate key-stream while continually swapping the values of the S-table (algorithm 2.2 contains this process in pseudo-code).

Finally, the key-stream z is XOR-ed with the plaintext. For decryption the same

```
i = 0
j = 0
while GeneratingOutput
i = i + 1
j = j + S[i]
Swap(S[i]; S[j])
Output z = S[S[i] + S[j]]
```

Algorithm 3.2: Generating the key-stream (one round of RC4)

process is applied (a key will always generate the same S table and the same key stream - XOR twice gives the plaintext back).



RC4 is a permutation: all of the values appear once and no more than once. The strength of RC4 is that it is continually permutated; the output cipher text seems completely randomized and hard to track. RC4 has a number of weaknesses however. Most importantly, an attacker does not need to know the actual key. As long as he can find the initial stage of the S-table, he can decrypt the cipher text. There are a number of attacks possible: given a current stage, the attacker can try to guess what the initial S-table was. He can check the correctness of his guesses by applying the algorithm and seeing if it gets him to the current state. Furthermore, although the output seems random, the probability distribution is not. Some bits have the probability to be followed by a 1 greater than the probability to be followed by a zero (and the other way around). By applying the Bayes theorem and some further non-trivial calculations, an attacker can reduce the space of possible answers considerably [Was01].

In the case of Xbox, the job of an attacker is even easier. Remember that RC4 is not used just to decrypt the boot-loader. It is also used to hash the boot-loader; the last bytes are checked for the magic number. Any stream cipher can be modified a little and applied as a hash. Not every hash is a secure hash however. The professionals from Microsoft somehow overlooked that RC4 does not have any feedback (possibly because they were also considering using RC5, which does use feedback in the S-table generation and would have been a better choice). Without feedback, what is at the beginning of the boot loader does not influence the final few bytes of its hash [Ste05]).

People quickly realized they can add code to the beginning of the boot-loader that will force the windows 2000 kernel to accept non-original DVDs and programs that failed the RSA check. Mod chips appeared: some completely replace the flash ROM, some just altered a few bits traveling through. The first of these could be plugged into the Xbox motherboard with some 31 pins to permanently bypass the security set up. Later people discovered that if the flash ROM is missing completely, the system tries to start up from another location. To ease the production process, it looks for a ROM memory connected to the LPC bus next to the flash ROM. Mod chips that use only 9 cables appeared [Ste05].

This means the security system was a complete failure. People can purchase the hardware for half the market price and use it to run copied games, or any software that the hardware can handle. They therefore also do not need to pay to get the additional features; possibly with the exception online gaming (official servers can check the integrity of every Xbox that tries to connect).

c = c + delta
Y(i+1) = Y(i) + F(Zi, K[0,1], c)
Z(i+1) = Z(i) + F(Y(i+1), K[2,3], c)
where F is defined as:
F(z, K[i,j], c) = (SL4(z) + K[i]) XOR (z + c) XOR (SR5(z) + K[j])

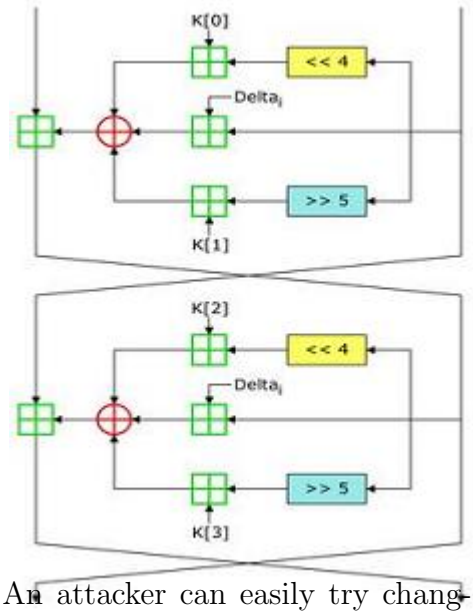
Algorithm 3.3: TEA

3.4 Tiny Encryption Algorithm

Microsoft saw that it is loosing profits because of the flaws in Xbox security. Instead of properly revising the whole system however, they decided to act quickly. RC4 was marked as the only weakness of the system and needed to be replaced. This involved forcing nVidia to discard millions of Southbridges and manufacture new ones [Ste05]. The new Xbox (1.1) was exactly the same as the old one except that RC4 inside the secret ROM was replaced with the Tiny Encryption Algorithm (TEA).

TEA is the smallest and computationally most efficient encryption algorithm up to date. It uses a 128-bit key, which it splits into 4 parts: K[03]. Odd rounds use first 2 parts, even rounds use K[3,4] [KSW97]. Algorithm 2.3 shows what two rounds of TEA look like in pseudo code.

SL4(z) means shifting (not rotating) z to the left by 4 bits, and SR5(z) denotes a shift to the right. Delta is a fixed value: delta = ($\sqrt{5}$ - 1) * 231. TEA is usually applied for 64 rounds.



Decryption means simply inverting the signs. TEA does use feedback and fits in 100 bytes easily. However due to its simplistic key schedule, it has a number of weaknesses. Most importantly, switching one or two bit values of the key in certain positions does not change the output of one round of the algorithms. After 2 rounds then, there is probability ? that the output will not change and after 64 rounds, this probability becomes 2-32. This probability can be further lowered by using differential analysis and a birthday-type attack [KSW97].

These weaknesses make TEA completely inadequate for its task inside the secret ROM. Once again the greatest problem is using it as a hash. People quickly learned switching bits 16 and 31 of a 32-bit word does not influence the outcome of the hash. An attacker can easily try changing bits of the boot-loader, to find one that will cause a jump to an insecure memory location and the final bytes of its hash will still pass the check Steil05.

3.5 The Visor Hack

Replacing RC4 with TEA was not only a bad idea, it was done too early as well. Within weeks after Xbox 1.1 was released, numerous other security deficiencies were discovered. One of these is the panic mode. If a security check fails, Xbox needs to lock down. Shutting the Xbox down at this point presents a number of dangers. It might be the case that an error has occurred and there is no attack. When the processor stops running, the secret ROM could be dumped and read more easily. The secret ROM therefore needs to be turned off, so the secrets stay secret. The problem is that once it is turned off, the processor falls into flash memory and an attacker could take over. The ingenious solution Microsoft came up with is to put the instruction to shut down the secret ROM at the very end of the address space; once it is executed, the processor will try to retrieve data from the following location and since that does not exist it will halt the machine and blink the error light [Ste05].

Visor is an anonymous hacker, who decided to check whether this routine really works. He changed the last few bytes of the boot-loader to cause panic mode. Instead of a lock down, his Xbox simply jumped to the beginning of the address space and continued executing code from there. The reason why this brilliant idea failed in practice is that the processors were changed just before production stage. Intel processors, which ended up inside Xboxes instead of AMD because of better prices still use an old wrap-around memory method. Because of compatibility issues, it was useful to have a cyclic memory space: the end = the beginning. AMD however entered the market much later than that; their processors no longer use cyclic memory, both because there is no longer any need for it and because they believe it might be a security risk. The panic code does not fulfill its purpose; although it switches off the secret ROM so it could not be sniffed, it leaves the rest of the system in the hands of the attacker [Ste05].

3.6 Xbox Linux Project

Xbox Linux Project is a team of people who want to be able to use the Linux operating system on their Xboxes. They also explored Xbox security and among other things they discovered an easier method of dumping the secret ROM once it was changed to use TEA. What they really wanted was to find a method to bypass the security without deconstructing the Xbox: without soldering, mod chips and pins [Ste05].

Xbox Linux Project looked at the windows 2000 kernel and how it runs the games. The games cannot be altered as they come on original DVDs and are checked with RSA encryption. Not everything is checked though. Games are expected to handle their own saved games. The saved games are stored and run from the hard drive; they can even be stored and carried around in USB memory sticks. Not all, but the saves of quite a

few games (007: Agent under Fire, Splinter Cell) can be exploited. It usually involves inducing some minor error or bug by changing the name of the player inside the file for example. Once the game launches the save, it overwrites its stack and eventually executes whatever code was added to the saved game [Ste05].

The problem with this approach was that the game had to be inserted in the DVD drive and run every time. The Linux people found yet another weakness: the dashboard. Dashboard is the desktop of Xbox; it gets executed when there is no DVD in the drive and allows the user to manage his saved games, audio, options The dashboard is protected. It is encrypted and Windows 2000 kernel checks the public keys. The protection has blind spots however. Fonts for example are not checked. The final solution designed by the Xbox Linux Project launches a hacked saved game, which afterwards alters dashboard's fonts. Next time dashboard is launched it crashes because of faulty fonts and executes whatever code is inside the fonts now (such as the Linux installer) [Ste05].

3.7 Discussion and conclusion

Before criticizing, one must realize that by releasing Xbox, Microsoft made a last minute decision to enter a new market of video game consoles. It became a newbie with high aspirations among well established brands such as Sony Playstation and Nintendo Gamecube. Microsoft had to commit billions of dollars to this risky decision. In spite of the relative lack of experience and high time pressure, Xbox was not a complete failure. Perhaps most importantly, Microsoft managed to cover its losses and establish its position among the competition. Furthermore, it utilized the lessons learned in the construction of Xbox 360, witch has a much more robust security (more custom parts, RSA signatures for start up). Finally, building a secure gaming console is no easy task. It is a computer, with all its intricate parts and boot processes. It is indeed very hard to design a wholesome system without mistakes.

Microsoft however did suffer major losses that could be prevented. To make up for the costs of hardware, the precious insides of the Xbox needed to be protected, so they could only be used for the purpose for which they were made. The security system has countless mistakes. Microsoft is prossibly the most experienced software developer; one would expect designing a video game cannot be as challenging as designing an operating system. In spite of the complexity of the task, some of the mistakes are simply too severe.

Xbox security starts with a secret ROM hidden in the nVidia Southbridge, which handles some initializations and most importantly, checks the integrity of the boot process. It decrypts a boot-loader stored in the flash ROM and hashes it to check that it was not altered. The boot-loader then checks and initializes the Windows 2000 kernel, which checks the games and all other processes. This report mentioned only some of the countless mistakes made at every level: secret ROM uses a hash completely useless for its task (and was later replaced by an equally bad algorithm). Even if the hash does not match, instead of crashing, Xbox jumps to evaluating an insecure memory location. Not even the Windows 2000 is secure. Some of its blind spots are the games' save files and

the dashboard program's font files. There are countless ways to permanently bypass Xbox security; a mod chip or a corrupt save file will allow the user to run whatever software the Xbox can handle. Looking at the system from an attacker's point of view would surely have prevented most of these.

Finally, the hackers are not evil people. All Andrew "bunnie" Huang did was revealing how the security works; he did mention possible weaknesses, but never published a way how to abuse them. He only confirmed Kerckhoffs' principle: security through obscurity never works. Sooner or later, someone with the necessary equipment and knowledge will take the effort to explore the system and reveal every hidden detail. Furthermore, once Microsoft realized the mistake it made with the RC4 hash, instead of any discussion with people who could reveal further mistakes and help amend them, it quickly made the same mistake again by using TEA. The Xbox Linux Project (responsible for revealing TEA, the save game and dashboard weaknesses) offered Microsoft not to publicize its findings in exchange for implementing a backdoor for Linux. Microsoft did not respond to their offer and proceeded with some further futile effort to mend the holes without further financial loss. Because of the severity of the mistakes and the number of various attackers with the same goal, Xbox security was broken within month after its release. However hackers gave Microsoft a chance to cooperate and postponed publicizing the weaknesses and ways of exploiting them. The sheer extent of electronic piracy confirms that the Microsoft's policy of "not talking to terrorists" does not yield desired results. Working together with hackers instead would possibly increase product quality with lower financial losses than the ones suffered.

Chapter 4

Multiple Encryption

Written by *Ronald Chu & Mark Jeronimus.*

Cascade ciphers are a class of ciphers where multiple encryption primitives are combined in a sequential configuration in an attempt to increase security. Some research indicates a cascaded cipher improves security by design while others indicate the opposite. Several studies have been done, focussing on different classes of attacks, in an attempt to prove such weaknesses. We will discuss some key points of those security questions and provide the reader with a deeper understanding of the implications a cascade cipher might have.

4.1 Introduction

The general notion of a Cascade Cipher (CC) is a cascade of cryptographic primitives, usually block- or public-key ciphers. These ciphers are executed sequentially, passing the output of one cipher to the input of the next cipher in the cascade (called a *stage* here forth). This is different from the notion of Multiple Encryption, in that the latter can also be arranged in a parallel fashion or a hybrid combination between parallel and sequential [ZHSI03]. Cascaded encryption has also been mentioned by [DK05] as CC for public-key ciphers. Shannon [Sha49] mentioned a Product Cipher as a related configuration. A product cipher describes an iterated cipher like DES, with lots of (almost) equal stages (referred to as *rounds*).

The most apparent reason for using cascaded encryption is to create a stronger encryption, as is the case with Triple-DES. Using different schemes, however, has the extra advantage that the system is secure as long as at least one of the used schemes is secure and therefore protects against future exploits of a scheme as well as mathematical breakthroughs.

4.2 definitions

The following definitions are used throughout the paper:

P, M, C, m The plaintext P or M which consists of m bits (generally referred to as the *length*) and the ciphertext C of the same length. All intermediate ciphertexts between CC stages are also of length m.

K, K_e, K_d, k The key K with length k. For public-key ciphers, the encryption key K_e and decryption key K_d.

n The number of cipher stages in the CC.

l The number of plaintexts in a Chosen-plaintext or Known-plaintext Attack.

t The total number of participants in a Secret Sharing Scheme (SSS). $t \geq n$.

$\mathcal{E}$ A standard encryption scheme $\mathcal{E} = (\mathcal{G}en, \mathcal{E}nc, \mathcal{D}ec)$ consists of a key-generation algorithm $\mathcal{G}en$ generating an encryption and decryption key, an encryption algorithm $\mathcal{E}nc(M, K_e) = C$ and a decryption algorithm $\mathcal{D}ec(C, K_d) = M'$. The encryption and decryption key are usually omitted. $\mathcal{D}ec(\mathcal{E}nc(M)) = M$ for every message M.

Some attacks require the existence of oracles. Other oracles are introduced to aid in the generalization of certain models.

$\mathcal{EO}$ An encryption oracle which receives a plaintext P, returns the ciphertext $C = \mathcal{E}nc(P)$. This emulates the situation where the attacker has knowledge about and access to the encryption algorithm.

$\mathcal{DO}$ A decryption oracle which receives a ciphertext C', returns the plaintext $P = \mathcal{D}ec(C)$ or $\perp$ if $C' = C$. This corresponds to the setting of a Chosen-plaintext attack where the attacker can decrypt any ciphertext except the original.

$\mathcal{KEO}$ A key exposure oracle which receives an index i and returns the key of stage i in the cascade. This oracle is used to emulate the real-world situation where some ciphers are broken. At most $n - 1$ keys of different stages will be returned by this oracle, otherwise it refuses.

4.3 Known attacks

For all attacks, it is assumed that the adversary has at least a description of the CC and all stages used in the CC (in accordance to Kerckhoffs' principle).

4.3.1 Chosen-ciphertext Attacks (CCA)

Although cascading ciphers are the main topic of this paper, they are a subset of multiple encryption and the work done by both Zhang et al. and Dodis and Katz is mainly generic for *all* multiple encryptions and may therefore be remodeled to cascaded ciphers in specific or at least give some insight as to how to approach generic CCA-secure cascaded ciphers. There seems to be no research available on cascaded ciphers specifically.

In parallel multiple encryption, the message M is split into i parts which are encrypted in parallel by (possibly different) encryption ciphers $\mathcal{E}_1 \dots \mathcal{E}_i$. Since the notion of additional security of multiple encryption is lost (an attacker only needs to break one specific cipher to succeed), the plaintext need to be pre-processed with an All-Or-Nothing Transformation (AONT) or a threshold secret sharing scheme. Parallel multiple encryption with AONT preprocessing and cascading ciphers (or sequential multiple encryption) are both methods of multiple encryption where the message is encrypted using multiple schemes which all need to be broken in order to obtain the original plaintext.

Zhang et al. [ZHSI03] presented the first formal model for multiple (both sequential and parallel) encryption CCA security. There has been much work done for specific applications of cascaded ciphers (threshold, broadcast and key-insulated encryption) but since this only gives solutions to specific cases and the models need to be re-engineered for even slightly different scenarios a generic solution would be more useful. Their work was a basis for Dodis and Katz [DK05] to define a stronger generic chosen ciphertext-secure multiple encryption scheme.

IND-MCCA and NM-MCCA security

Zhang et al. focus on *indistinguishability* against chosen ciphertext-attacks security for multiple encryption (IND-MCCA) since this is the strongest notion of security. They also mention the notion of *non-malleability* where an attacker must not be able to create a ciphertext of a message meaningfully related to the original message. We will not discuss this notion since it is proven to be equivalent to indistinguishability under CCA.

The notion of indistinguishability intuitively means it should be infeasible for an adversary to distinguish which plaintext of two different plaintexts corresponds to a given ciphertexts. Formally, given an encryption function $\mathcal{E}nc(M_b) = C$ where C is the encryption of M_b and $b \in 0, 1$ which is determined at random, the encryption is indistinguishable if there is no way to obtain b if M_0, M_1 and C are known. The strongest notion of CCA-security is that of a indistinguishable CCA-secure scheme (IND-CCA).

In the IND-MCCA setting the attacker has access to a slightly different encryption oracle $\mathcal{EO}'$ which takes two plaintexts M_0 and M_1 as input and returns the ciphertext C_b corresponding to M_b. The cascade is IND-CCA secure if the attackers chance to correctly guess b is equal (with a negligible difference) to a random guess.

One may think that any multiple encryption using only IND-CCA-secure schemes is IND-MCCA-secure but unfortunately this is not the case. Since the $\mathcal{DO}$ can be used to decrypt any ciphertext $C' \neq C$ an attacker only needs to forge a ciphertext C'_b where $\mathcal{D}ec(C'_b) = \mathcal{D}ec(C_b) = M_b$. This cannot be done in a single cipher setting since the private key is unknown but in the multiple encryption scheme the attacker can obtain

private keys of at most $n-1$ stages using the $\mathcal{KEO}$. If the attacker obtains the private key of the last stage of the cascade then it is possible to forge the ciphertext C'_b which can be submitted to $\mathcal{DO}$ to obtain the plaintext M_b and therefor b.

This can be solved by introducing a randomness to each stage of the cascade where the randomness in each stage is obtained from the original plaintext M and can thus be verified after decryption. The decryption oracle $\mathcal{DO}$ only returns de decryption of a given ciphertext if the randomness verification succeeds. $\mathcal{DO}$ has become useless for the attacker since it is only possible to create a ciphertext from a known plaintext and no longer from a intermediate result as mentioned in the attack above if it were to generate a succesfull decryption by $\mathcal{DO}$. Therefore b is no longer determinable unless *all* stages are broken meaning the cascade is IND-MCCA-secure.

Weak, normal and strong MCCA security

Dodis and Katz [DK05] divide MCCA security into three categories of security strength, depending on what information is available to the attacker, state that the above construction of cascaded ciphers is only weak MCCA resistant and define a general notion of multiple encryption schemes which are strong MCCA resistant. Their work only covers parallel encryption since this is the strongest variant. Their work does, however, give insight into the current status of MCCA-security and is therefor briefly outlined below.

wMCCA (weak) The attacker only has access to an oracle which performs the entire decryption process without showing intermediate results.

MCCA (normal) The attacker has access to and decryption oracle which also shows the intermediate ciphertexts before combining them into the decrypted plaintext.

sMCCA (strong) The attacker has access to an oracle which can decrypt ciphertexts using any of the used schemes

In order to achieve MCCA-security, a signature of all the encrypted shares is added to the final ciphertext C making it impossible for an adversary or compromised stakeholder to manipulate individual shares without being caught. For strong MCCA security, a signature is added to each share.

4.3.2 Known/Chosen-plaintext Attacks (KPA/CPA)

The Known-plaintext Attack and Chosen-plaintext Attack are very similar. In both attacks, the adversary possesses or knows some information about the plaintext. The KPA is the weakest of the two. If any model is secure against KPA, it is not automatically secure against CPA. The other way around, it is implied that all CPA-secure models are also KPA-secure, as we will discuss shortly. The adversary is assumed to possess some plaintext/ciphertext pairs (KPA) or an oracle (CPA).

In the CPA the adversary will have access to an Encryption Oracle $\mathcal{EO}$. In addition, he/she might also possess a ciphertext C for which the plaintext is unknown. The adversary wins if the key is recovered or if the plaintext of a given ciphertext C is found.

In the KPA the adversary will have access to no oracles, but instead he/she is given a history of l encryptions under the same, unknown key: P_b, C_b with $b \in (1, l)$ and $C_b = \mathcal{E}nc(P_b)$. In addition, he/she might also possess a ciphertext C for which the plaintext is unknown. The adversary wins if the key is recovered or if the plaintext of a given ciphertext C is found.

The KPA is generalizable to CPA with a fixed number of chosen plaintexts. The only difference between KPA and CPA is that with CPA, l can grow polynomial in resources. This makes CPA a much more feasible attack. Some sources (e.g. [ZHSI03]) define the $\mathcal{EO}$ differently.

Attack

For reasons of generality, and to show that the internal structure of a stage need not be weak for a CC to be weak, it is assumed that every cipher $\mathcal{E}nc_i$ is a random block cipher. A random block cipher consists of 2^k permutation of size m, one for each of the 2^k possible keys. Each permutation is randomly and unbiased chosen from the pool of $2^m!$ possible permutations. It is not ruled out that some keys may have the same permutation assigned to it.

Theorem 4.1 *The Random Block Cipher is a mathematically perfect cipher.*

Proof. Because of the lack of internal structure, it is provably impossible to break it in less than 2^{k-1} steps on average, with the additional requirement that m is much longer than the unicity distance. $m \gg \frac{k}{1-\alpha}$ where α is the entropy of the plaintext. △

The chance that two successive stages have a weak combination of permutations is negligible. There are $2^m!$ possible permutations, of which *only* $2 * 2^k$ are used in any two consecutive stages.

Any CPA or KPA attack on a CC with Random Block Ciphers will be a meet-in-the-middle attack [EG85]. An effective implementation of this attack is the cubic root attack by Hellman [Tel07]

Theorem 4.2 *An attack on a CC of length $n = 2$ is just as secure as a single stage.*

Proof. The general advantage of Hellman's attack is that any CC with an even number of stages can be cracked in $O(sqrt[3]2^{\frac{lk}{2}^2})$ time and space, effectively halving the effective key length compared to an attack on a single Random Block Cipher $O(\sqrt[3]{2^k}^2)$. It should be noted however, that most ciphers are not Random Block Ciphers. Especially iterated ciphers, which can be cracked in $O(\sqrt[3]{2^{\frac{fk}{2}}}^2)$ time and space (where f is the number of iterations) do not share this property. △

Hellman's attack on CC's with an odd number of rounds have a similar time and space requirement, although the proof is more involved [EG85].

4.3.3 Ciphertext-only attacks (COA)

The Ciphertext-only Attack is the weakest attack, and tries to attack a (cascade) cipher without plaintexts and without the aid of oracles.

It is assumed the adversary possesses some ciphertext(s) and a description of the language used in the plaintext (note that the language is taken from what is actually feeded into the cipher: the language of a *PKZIP*-compressed message is the language of the .ZIP file format, not the message being compressed). All known implementations to date either use brute force attack, a dictionary attack or exploit the structure of the cipher and the plaintext language.

The adversary wins if the key is recovered or if the plaintext of the given ciphertext is found. In some cases, the adversary also wins if he/she can obtain any information about the plaintext.

Brute-force attack

The brute-force attack is the same for a CC as for a single cipher because it does not rely on internal structure. The brute-force attack requires $O(2^k)$ steps to try all possible keys. On average $O(2^{k-1})$ steps suffice. For each step, the resulting plaintext should be tested for characteristics of the language. The earliest demonstration of a successful attack of this type, came from the Electronic Frontier Foundation (www.eff.org) using the dedicated computer *Deep-Crack* (1998). This computer consists of 768 chips and can try approximately 92 billion DES keys per second. At this speed, it will take 4.5 days on average to find the right key [Wob07].

Dictionary attack

The dictionary attack is the same for a CC as for a single cipher because it does not rely on internal structure. Dictionary attacks are popular with password-feeded applications, such as Unix-login and encrypted archives. These class of attacks make use of the most critical vulnerability in the entire process: the end user (victim). People often use names of subjects, heroes or friends, birthdates, etc as (part of) their password. A dictionary attack is basically a type of brute force attack, but instead of trying all possibilities, it tries to guess passwords using combinations from a dictionary. Often it helps if the dictionary is tuned in accordance to the peculiarities of the victim (who selected the password in the first place). This introduces a manual preprocessing step to the process. When the dictionary is prepared, a program tries every combinations of the entries (including changing lower/uppercase letters, removing dashes in dates, adding punctuation chars between words, add/remove or replace random characters to/from a word, write words backwards, etc.).

There are two classes of dictionary attacks, *offline* and *online* attacks. Offline attacks are feasible only if the adversary obtains ciphertext. With online attacks, the adversary does not have direct access to the ciphertext, but instead has the means to try different keys.

A proven application of the offline Dictionary attack is the popular program *Crack* (www.crypticide.org) that (according to the description) allows unix/linux administra-

tors to find weak user passwords vulnerable to a dictionary attack. It cracks encrypted Unix login entries from the `/etc/shadow` file. In theory, a cracked password does not have to be equal to the password chosen by the victim, as long as it provides access to the victim's unix login (i.e. encrypts to the same value). This program is said to crack about 20% of all used passwords.

There exist a mathematically perfect (but sometimes infeasible) countermeasure against online Dictionary- and Brute-force Attacks on an *interactive login environment*, called *Key Strengthening* [KSHW97]. The main feature of this protocol is to introduce a necessary, unavoidable delay in the key processing step. An example might be hashing the password 100.000 times. This unavoidable delay (usually taken to be at least 5 seconds) defies the possibility of any attack involving guessing passwords. It can be formulated that only $\log_2(\frac{T}{T_p})$ bits of passwords can be brute-forced in time T when one try takes time T_p.

Generally, any key with less *entropy* than 80 bits will be vulnerable. This follows from the *public knowledge* that at the current state of technology, roughly 2^{80} operations can be performed within reasonable time and cost. For medium- and long-term security, 100 and 128 bits is recommended respectively. *Public knowledge* is emphasized because it is always unknown what secret organizations might have achieved at any time in history, present and future. An entropy of 80 bits roughly translates to passwords with 54 characters of english words, 20 hexadecimal characters, 14 characters of *secure password* (random combination of uppercase/lowercase letters, numbers and punctuation) or 10 full-ASCII bytes.

4.4 Applications

The widely believed advantage of a CC is that it is at least as secure as the strongest cipher used. Others believe or hope the cascade may actually be stronger than the individual stages. Some papers [ZHSI03][EG85] try to prove the contrary, often without formal proof. Another example of improving security is not by the cumulative security of the CC, but by using it as part of an SSS implementation. This way responsibility is distributed amongst different stakeholders [DK05].

The applications are a direct result of the advantage. There are several applications for Cascade Cipher. Many are the same as for Multiple Encryption.

Long time security The CC will be secure even if all but one stages are broken. For this purpose, the ciphers for each stage are usually chosen to rely on different mathematical principles (like DLog). The NESSIE project [Con03] for example recommends double encryption with ACE-KEM and RSA-KEM for very long term security.

Increasing the number of iterations Some ciphers are built with an iterated internal structure, for which the number of iterations is effectively increased by cascading it, while still being compatible with existing specialized hard- and soft-

ware. A historical example is Triple-DES, which is (except from an initial and final permutation) an iterated Feistel network of 16 rounds.

Increasing the effective key length Almost the same reasoning as the previous application. A Triple-DES cascade has a key of 168 bits, which is substantially harder to brute-force than 56-bit Single-DES, even using specialized DES cracking hardware.

Secret Sharing Scheme Although most Secret Sharing Scheme protocols are built on parallel decryption, some can use serial configurations. The keys of each stage within the CC are distributed over several stakeholders. Only if at least n out of t stakeholders participate can the message be recovered.

Key-insulated encryption where the users secret key is updated each time period. The main property of such schemes is that exposing secret information during some time periods leaves all non-exposed periods secure. [DKXY02] defines a generic solution where some l secret keys are given each time period and the sender first splits the message into l shares. This correspondents exactly with a multiple parallel encryption scheme.

Certificate-based encryption where a message is encrypted twice, once by a user's public key K_e, and also using the user's certificate issued by an authority. The aim is that the user cannot decrypt the message without a valid certificate and the authority cannot decrypt it without the user's private key K_d. An example is CSS, used on DVD's. The user, in this case, is the DVD player.

Proxy encryption in which ciphertexts encrypted with one key (or cipher) are transformed to a state in which it is encrypted with another key (or cipher), without revealing the plaintext in the process. Such protocols were first discovered by Mambo and Okamoto [MO97]. A common atomic proxy encrypts the ciphertext with the target cipher, then decrypts it with the source cipher (using commutative cipher schemes). Temporarily, the message exists as CC ciphertext. Both the C of the cascade and the C's of each cipher separately can be leaked to an adversary.

Truecrypt

One system that meets a lot of requirements for a secure cascaded cipher is TrueCrypt [Fou], a popular implementation ofthe encrypted volume. An encrypted volume is a virtual file system which can store any type of file. The ciphertext consists of all data concerning the volume, including the header and file system tables. This program should be relatively secure against the previously mentioned attacks in the following ways:

- **It uses keys with at least 128-bit entropy.** It uses a master key of 128 bit, generated using a 640-bit hash function, with lots of random events as input (system timer, keystrokes, mouse movements). The user key consists of a password along with optional series of keyfiles.

- **it does not leak any information about the plaintext language.** It actually does. The first 288 bits from bit 64 onward are composed of the header, and from bit 512 onward, there is a either a FAT32 or NTFS file system header. However, the key schedule is relatively complex. Upon decryption, the User Key is combined with salt stored in the ciphertext and hashed. The hashed User Key is used to generate a Primary Key and Secondary Header Key. These are used to decrypt the header and retrieve the Master Key, from which the session Key is derived. Both the Master Key and the Secondary Header Key are needed to decrypt data in the file system of the volume.
- **It does not leak any information about the plaintext.** Virtually any type of file can be stored. Even when an adversary can guess that there are some files of popular formats stored on the volume, he/she does not know *where*.
- **It does not involve a functional oracle.** The volume can only encrypt/decrypt when the key has been entered and resides in memory. When the volume is 'closed' the keys in memory are destroyed.
- **The ciphers are mathematically *incompatible* (i.e. based on different mathematical problems).** It uses combinations of the following three ciphers, AES, Serpent and Twofish.
  - AES (the winner of RSA's call for the Advanced Encryption Standard, which was won by Rijndael) uses iterations that consist of different, incompatible operations: XOR, Substitution and Multiplication over a Galois Field.
  - Serpent uses a symmetric combination of bit shifts (permutations) and XOR.
  - Twofish uses a Feistel round with interleaved bit shifts (permutations). Twofish is conjectured insecure in 2007 by a somewhat infeasible *Truncated Differential Attack (CPA)* requiring $2^{57.3}$ time and 2^{51} space [KR00].

4.5 Summary and conclusions

CCA A natural cascade of CCA-secure stages is not automatically secure as shown above. There are, however, constructions available which are proven to be MCCA-secure.

CPA/KPA The complexity of breaking a cascade of n stages is roughly equal to breaking a cipher with a key $\frac{n}{2}$ times as long (as opposed to the intuitive expectation of n times as long).

COA All the described ways of attacking a CC can be generalized to attacking a single cipher. A CC provides no additional security except from the increased key length and when the attack is a function of the internal structure of the system as a whole (either a CC or a single cipher). The best way to defend against COA:

- Having an entropy-rich key
- Avoid ciphers that have been exploited for a COA or might be susceptible to one.
- Use *Key Strengthening*

Goldreigh stated that a cascade cipher is as strong as the strongest cipher (1998). This was later corrected by Zhang, who proved a cascade of CCA-secure stages can be cracked if the last stage (which isn't necessarily the strongest) has been compromised. He did provide a workaround for this problem, however. More generally, he proved that any cascade built with CCA-secure ciphers is not automatically CCA-secure.

There is no general way to determine the strength of any cascade cipher (or any Multiple Encryption scheme). The only systems of which the strength has been assessed are very specific systems (usually single ciphers). Even so, this strength assessment is not final, as cryptographers are continuously searching for better ways to attack known algorithms (the typical notion of an NP-complete problem). The only way we can assess the strength of a cascade cipher is to combine the strengths of the separate stages (assuming all known reinforcements have been implemented) and hope there will be no future breakthroughs against general cascades or your specific implementation during the lifetime of the system.

Our final conclusion is that cascades are still a good idea if you are looking for a stronger system (with the sacrifice of increased resources requirement) as long as the designer also considers the reinforcements to counter know attacks on cascades.

Chapter 5

Phishing

Written by *Marnix Kammer and Barbara Pieters.*

Cryptography deals with all kinds of attempts to learn information that one shouldn't be allowed to learn. This chapter is about such an attempt called *phishing*[1]: the forgery of e-mails or websites of companies or other organizations, so that users are tricked into believing they are communicating with a real company or organization while they aren't. This way, users can be persuaded to give out sensitive information about themselves (such as credit card numbers or passwords), which can be very valuable to the one(s) performing the phishing attack when they exploit it.

5.1 Phishing and cryptography

The field of cryptography deals with computer security. Attacks on computer security can be classified into three types: physical, syntactic, and semantic. Physical attacks try to compromise security by destroying or manipulating hardware of computer systems. Syntactic attacks try to take advantage of all kinds of vulnerability in software, and these are a much more serious threat because they are not location-bound in most cases. Cryptography mainly focuses on preventing these kinds of attacks.

Semantic attacks, however, are entirely different: instead of exploiting vulnerabilities in software, these attacks exploit the way humans interact with computers or interpret messages. Phishing belongs to this third type of attacks. Although some techniques in cryptography used against syntactic attacks can be used in a similar way to combat semantic attacks (such as an anti-phishing e-mail filter), as we will show in the second part of this chapter, cryptography alone is simply not sufficient to ensure protection against most phishing attacks. The danger of phishing lies in the fact that controlling or securing human behavior is much more difficult than securing the behavior of

[1]The term *phishing* first appeared around 1995 and is derived from fishing, because it bears some resemblance to putting a piece of bait on a fishing line (e.g., showing a link) and catching a fish (sensitive information) with it.

computers and software. In this chapter, we will show what additional methods concerning the interaction between humans and computers can do against phishing, and what automated techniques and tools we can use to prevent it.

5.2 Forms of phishing

Virtually all forms of phishing extensively make use of internet, and they exploit the fact that users often lack certain knowledge of computer systems, security and security indicators. In this section, we will describe the various forms of phishing and what kinds of techniques they use to fool users.

5.2.1 Fake e-mails

By fake e-mails, we mean e-mails that appear to be from a company or website the user has a connection with, or just entirely made-up e-mails that appear to be from some professional organization. The latter of course has a harder time convincing users because they involve (made-up) companies or organizations that the user does not know, but still, this type of e-mail is the most common phishing e-mail attack. A phishing attack using fake e-mail can have several purposes with the e-mail. The most common are:

- E-mails trying to pursuade the user into replying with sensitive data or sending money or unwittingly assisting the sender in some scam. A typical example of this would be a company claiming that there was an error in their database and they lost your account info; if you would be so kind as to reply with the relevant information they lost. These phishing attacks are the least successful, because most users don't trust impersonal mail from senders they don't know.

- E-mails containing links to phishing websites. This is an indirect phishing attack because the e-mails themselves do not try to pursuade the user to divulge any information, but they do try to trick the user into thinking that the links contained in the e-mails will lead them to real websites of real companies. Although indirect, this is the most dangerous form of e-mail phishing because the websites the e-mails refer to are a lot more powerful when it comes to interaction with the user, and they can make use of a wider range of techniques to fool users, as we will show in the subsection about website forgery.

- E-mails containing a direct way to let the user enter some sensitive information, for instance a form, and pursuading them that it is urgent to do so. This type of phishing is not very succesful, because many users don't trust e-mails containing forms and, more importantly, many anti-phishing e-mail filters don't either, as we will see in the second part of this chapter.

Casus 5.1: PHISHERS GETTING CATCHED

Phishing does not always pay; this is what Jeffrey Goodin must have thought on January 16, 2006 when he was the first U.S. citizen to be convicted according to the 2003 CAN-SPAM (Controlling the Assault of Non-Solicited Pornography And Marketing) Act. Later in that year, on June 11, he was given a sentence of 70 years in federal prison.
Jeffrey Goodin tricked his victims, all of whom were America Online subscribers, by sending e-mails urging them to "update" their America Online billing information or lose their service, and referred them to forged websites created to collect credit card information. The stolen credit card information was later used by Goodin to make unauthorized purchases.
The financial damage that was done by Goodin was estimated at 1 million U.S. dollars. It is estimated that in the United States alone, citizens lose 3.5 billion U.S. dollars each year as a result of phishing attacks; Businesses lose even an estimated 4.5 billion dollars.

5.2.2 Fake links

The previous section already mentioned them: fake links that trick the user into thinking they are being led somewhere else than where they're really taken to. A link that says "Click here to go to your settings page now" probably will not point to your settings page at all. Also, IP addresses are used instead of domain names. A link that says "To our secure server" and takes you to, for instance, `http://123.132.213.231/securelogin.php`, could appear to users infamiliar with IP addresses as really being a secure server. A common reaction of inexperienced users here would be: "That address looks complicated, so it really must be a secure server". If they even pay attention to the address bar. Inexperienced users might also fall for an URL like `www.rabobank-internetbankieren.nl`, thinking it is part of the real Rabobank website.

More clever tricks exist that fool even experienced users. One such trick is to show a URL on the screen that is almost the same as the real reference in the link, for example: showing `www.bankofthewest.com` and actually leading the user to `www.bankofthevvest.com`, with a double v. One of the biggest scams in the history of phishing was a website with the URL `www.paypa1.com`, with an 1 instead of an l. Many users did not pay attention to this and thought they were dealing with the website of the PayPal company.

Also, possibilities are expanding for domain names to contain *internationalized* (non-ASCII) characters as well, making it almost impossible to see the difference without decoding the URL. For instance, most people will be fooled by a Greek rho pretending to be a p. Fu's paper [AYF05] about *international resource identifiers (IRIs)* gives a good description of what kind of new phishing threat these domain names pose.

Figure 5.2: Similar internationalized characters, but with different Unicodes

5.2.3 Website forgery

Mimicking There exist many techniques a phishing attack can use when it comes to spoofed websites. Most of these are based on visual deception by mimicking legitimate text, layout or images.

- **Deceptive look and feel** A spoofed website may use the same stylesheets and the same images and logos as a legitimate website, at which point the website gains trust from the user because most users think that "a website probably isn't a scam if it looks this professional".
- **Images mimicking windows or dialogs** These look exactly like certain dialogs or windows. Especially 'error dialogs' appear on a lot of sites, quickly tricking the user into clicking on them.
- **Link images** Many sites have links as images, and there exist even some images that users almost automatically recognize (for instance, a company logo) as a link to some specific site or functionality. If these images are copied and if the link around them points to a site that is different but nevertheless looks the same, users are easily fooled.
- **Fake links** The same as in e-mail messages and as described in the previous subsection; the link text differs from the real link reference.
- **Hidden address bar** Javascript enables one to open a new browser window without an address bar. This is to prevent users from seeing the real (illegitimate) address.

Misplaced trust Other phishing strategies involve adding certain items to a website that induce misplaced trust.

- **Padlock icon** Many legitimate websites use *HTTPS* and *Secure Sockets Layer (SSL)*, which a lot of browsers indicate by a closed padlock icon next to the address bar and usually by other means as well. Mozilla Firefox, for instance, also paints the background of the address bar yellow and displays the domain name and a closed-padlock icon at the bottom right of the screen if it is a secure site. Because forged websites mostly do not use SSL, they can try to gain a user's trust by placing a closed-padlock icon somewhere in the body of the page (usually somewhere in a corner or at the top of the page).
- **False SSL certificate** Forged websites can place a trust seal somewhere on a page that links to the website of a *Certificate Authority (CA)*, which provides

verification of the certificate (and an English language description) of the website it was given the URL of. Of course, the CA would not have issued such a certificate for a forged website. But a clever trick is for the phishing website to link to a CA website, giving the CA the URL of the legitimate website instead. Only the more informed (and cautious) users would notice that the URL of the website they are on differs from the URL of the website to which the certificate belongs.

5.2.4 Pharming

Though most forged website can be exposed by observing that the URL does not correspond to the legitimate website, a lot of damage can be done by phishing if the *correct* URL of the legitimate website points to a *forged* website. *Domain Name Service (DNS)* servers map URLs to the actual IP addresses on which the legitimate websites are hosted. If that information is tampered with, a correct URL may point to an IP address on which a forged website is hosted instead.

This is called *pharming*: exploiting some vulnerability in the software of a DNS server. More difficult to detect is modification of a user's *host file* (a desktop PC's own mapping from domain names to IP addresses) or the compromising of a local network router. Nowadays, more and more network routers at people's homes are wireless. Most of the time, people don't bother to change the default login information and passwords for managing the router wirelessly after it has been installed. However, pharming is only possible when the legitimate website does not use HTTPS or SSL, because the server certificate would not be valid for the forged website in that case. Of course, again, this only applies when users pay attention to warnings about certificates.

5.2.5 Other forms of phishing

Other forms of phishing exist, such as *phone phishing*. In a phone phishing attack, a message (most commonly via e-mail) is sent to the user informing him that, for instance, his bank account information may have been compromised and he must call a telephone number in order to verify his account information. When the user calls that number, a prerecorded voice tells him to enter his bank account number and his PIN code first, after which the harm has already been done. Sometimes in this type of attacks, the phisher uses fake caller-ID data so that incoming calls seem to come from a legitimate company or organization. Phone phishing is sometimes called *vishing*, shorthand for voice phishing.

5.3 Changes in human-computer interaction to prevent phishing

Phishing can almost always be recognized as such by humans, by a few simple properties a user can pay attention to. We will go into this more specifically for e-mails and websites.

5.3.1 How to recognize phishing e-mails

- **Language and grammar errors.** As phishing e-mails are often self-written (because the phishers want the user to do something that the legitimate company or organization would normally not ask of the user), they tend to contain a lot of language and/or grammar errors. They are mostly written in English (to reach a wider audience) but English need not be the native language of the phishers at all. In fact, it is unlikely that English is their native language because people living in America have a greater chance of getting caught, because of the infamous Patriot Act and overall increased security awareness after 9-11. So if an e-mail contains a lot of language and grammar errors, that's a good clue it's either spam or phishing e-mail.

- **Urgent action required from user.** Most phishing e-mails urgently press the user in order to get something done from him or her. This is done mostly under the threat that some account will be closed. However, most legitimate companies will never do this because it is actually an advantage for them to keep connections with as many customers or clients as possible. So this is almost an indication that it is probably a phishing attack.

- **Forged headers.** In almost all phishing e-mails, *forged e-mail headers* make the sender seem to be some legitimate company. But in most e-mail clients, one can always check the full header to see where the e-mail really originated from. Or at least, from which server or domain. If this is entirely different from the sender address that has the "From" label, it's not a legitimate e-mail.

- **Unknown sender.** It's obvious that if you do not know the sender and if the sender does not have any valid reason to know you, then the sender would normally not have any reason to e-mail you at all and it's probably a phishing e-mail or just spam.

- **Fake links.** If the e-mail contains fake links that do not match with the website they really refer to and that are definitely not a typing error, do not trust the e-mail.

- **The reason for sending the e-mail.** One must always ask oneself: What reason does this person or company have for sending me this e-mail? If some company claims, for instance, that they have lost your account info and you must re-enter it, then that's not very plausible. Companies always store customer and client info on multiple redundant servers in safe places, and it is very unlikely that they will ever be so stupid as to really lose your account info. And if they do, your e-mail address would probably be gone from their database too, so they wouldn't have a way to e-mail you anyway. So if the reason for e-mailing you is not valid, don't trust it.

5.3.2 How to recognize a phishing website

- **Language and grammar errors.** The same goes here as for phishing e-mails. But there's one difference: phishing websites are mostly, in contrast to phishing e-mails, not self-made but copied from a legitimate website. These perfectly copied website would probably not contain any language and grammar errors. But if they do, they're probably made by phishers and not to be trusted.

- **Incorrect dates.** If websites are copied, they become static unless the phishers add some extra code. Dynamic elements in a legitimate website (such as, showing today's date in a corner of the screen) will be out-of-date on copied, static websites. So if the dates on a website are somehow incorrect, then that's a strong clue it is a phishing website.

- **Strange URL.** The address of the phishing website may have a strange or incorrect URL, as explained in the section about fake links. IP addresses instead of domain names are not to be trusted, and missing address bars (opening new windows without address bars is possible with simple JavaScript code) are definitely a clue that it is a phishing website.

- **Missing HTTPS/SSL.** If a website appears to be the secured section of (for instance) a bank site or the site of some other trusted type of company, then the legitimate version of it will always use HTTPS/SSL security. One can easily see that this is missing by looking at the address bar (the address itself beginning with HTTP, not HTTPS) and a closed-padlock icon that should be somewhere next to the address bar. A website that should be secure but is missing HTTPS/SSL is likely to be a phishing website.

- **Fake closed-padlock icons.** In order to compensate for the previous point of missing closed-padlock icons, or just to induce trust from the user, phishers sometimes place fake icons on their website. These can often be found in one of the corners or at the top of the website, or next to a login button. Closed-padlock icons on the body of a website are almost always an indication that it's a phishing website.

- **Fake links.** Just like phishing e-mails, a phishing website could very well contain false links (probably to other phishing websites).

- **Unsigned or fake certificate.** The certificate of the phishing website may be missing, or unsigned, or self-signed instead of being signed by a legitimate Certificate Authority. Sometimes, as explained earlier, a "certificate valid" image link to a CA's website is placed somewhere on a phishing that really links to the right CA. But if the link given to the CA's website does not match the URL of the referrer website, or if there even exists such a "certificate valid" image link on the website, it is very likely to be a phishing website.

If you have only a vague suspicion of a website being a phishing website, a very good idea is trying to login with false login information (i.e., the right username but a wrong

password) in order to see if the website gives an error message. If it does, is is very likely to be legitimate because you are in their database. If it doesn't, it is certainly a phishing website because it only used the entered login information to store somewhere or send it to the phisher.

In this section, we have covered several very good ways to determine whether or not a phishing attack is taking place. In fact, we believe that using these criteria correctly will always make a user recognize phishing attempts. But most users do not know (or forget) that they have to pay attention to these criteria, and even if they do know, a good phishing website does not even begin to make a user question its validity, or only when it's already too late. This fact calls the need for automated ways, tools and techniques to recognize phishing as such, in order to help users in the battle against phishing. The next section will go further into that.

5.4 Technical solutions to prevent phishing

A nice way to prevent phishing from happening is to have some sort of security such that phishing is no longer possible. As already stated, a big problem with fighting phishing is that there is a lot of human interaction involved. Phishers can produce websites which deceive a user. Or they send some user an urgent e-mail which the user believes. The best solution would be that users do not even get near a phishing website or e-mail

5.4.1 Dynamic Security Skins

Since users are not altogether well suited for the task to recognize when they are phished, some technical support to prevent phishing is welcome. One of the technical solutions against phishing are the *Dynamic Security Skins* from Dhamija and Tygar [DT05]. This scheme allows a remote webserver to prove it's identity to a user, which is easy to identify and hard to bypass. The Dynamic Security Skins (DSS) are a means to create a trusted path between a webserver and a user, without difficult authentication. Furthermore, since the DSS uses imaging for authentication and verification it is also difficult to spoof.

The DSS has the following properties.

- **User authentication** The user only has to recognize an image and remember a password, undeterminant of which server the user wishes to communicate with.
- **Content authentication** The user only needs to match two images to authenticate the content from a server.
- **Hard to spoof** The authentication is hard to bypass.

Further, the DSS also uses an authentication (and verification) protocol to meet the following security properties. This protocol is a *zero knowledge proof* protocol.

- **Mutual authentication** Both the server and the user authenticate to each other.

- **Personal information kept safe** No personal information, such as passwords, are sent over the network.
- **No masquerade** Even after observing any number of successful authentication, an attacker is unable to masquerade the server.

Secure Remote Password Protocol (SRP) The *Secure Remote Password Protocol* (SRP) [Wu98, Wu02] is used to obtain user and server verification. Using this protocol, the server never stores the password of a user, but stores a verifier instead. This verifier is then used to agree upon a session key for the current login session.

Obtaining the verifier Before the actual communication can take place, the user chooses a password P and picks a random *salt* s. Onto the password, the salt and the identity I of the user an hash function is applied: $x = H(s, I, P)$. Then the *verifier* v is computed: $v = g^x$. The user sends the salt s and the verifier v to the server. This is the only moment when the password is actually sent over the internet. Thus, when this connection is not safe, the password might be captured by a malevolent person, such as a phisher. The server saves the verifier and the salt together with the username of the user. These two values together, s and v are the password for the user according to the server. The reason why this verifier is stored, is that users like to have simple passwords, with low *entropy*. These passwords can be easily "guessed" in a brute force manner.

Obtaining a *session key* To agree upon a session key the user sends his identity to the server. Then the server looks up the corresponding salt and verifier and sends the salt back to the user. The user computes value $x = H(s, I, P)$ again and computes public key $A = g^a$ using a randomly chosen private key a. This value A is sent to the server. The server on his turn computes public key $B = v + g^b$ using the verifier and a randomly chosen private key b. The server sends this value B to the user, along with a randomly scrambled value u. Then both parties compute some value S. The user computes S as: $S = (B - g^x)^{a+ux}$. The server, on the other hand, computes S as: $S = (Av^u)^b$. Both values S evaluate to $S = g^{ab}g^{bux}$. The reader is encouraged to verify this. Then they both send each other a message. The user sends the server the message $M_1 = H(A, B, S)$, which the server verifies using values A, B and S. The server sends the message $M_2 = H(A, M_1, S)$ to the user, which the user verifies using values A, M_1 and S. Then both calculate the session key $K = H(S)$, which is never sent over some connection. See protocol 5.3 for an overview.

Since the protocol uses a, b and u as random values every time a session is started, an attacker can not use the previous observed authentication for spoofing, since the used session key does not contain the necessary values to create a new session key. Furthermore, by creating a *trusted path* the user can not be fooled that he or she is not talking to the server, but to a spoofer instead.

Creating a trusted path DSS creates a trusted path to the password window. The password window puts an image over the login fields and throughout the whole window, thus showing the user that the window is to be trusted. Every user is assigned a random image (this can also be changed into a chosen image or photograph). This image is then

User		Server
Login using I	1	Lookup v and s
Receive s	2	Lookup v and s
Compute $x = H(s, I, P)$ and $A = g^a$	3	
Send A	4	Receive A
	5	Compute $B = v + g^b$ and choose random u
Receives B and u	6	Send B and u
Compute $S = (B - g^x)^{a+ux}$	7	Compute $S = (Av^u)^b$
Compute $M_1 = H(A, B, S)$	8	
Send M_1	9	Receive M_1
	10	Verify M_1
	11	Compute $M_2 = H(A, M_1, S)$
Receive M_2	12	Send M_2
Verify M_2	13	
Compute session key $K = H(S)$	14	Compute session key $K = H(S)$

Protocol 5.3: Secure Remote Password

put on top of the window (being transparant) and even is shown on top of the input fields. By doing the latter, it is more difficult for a phisher to fake the login screen.

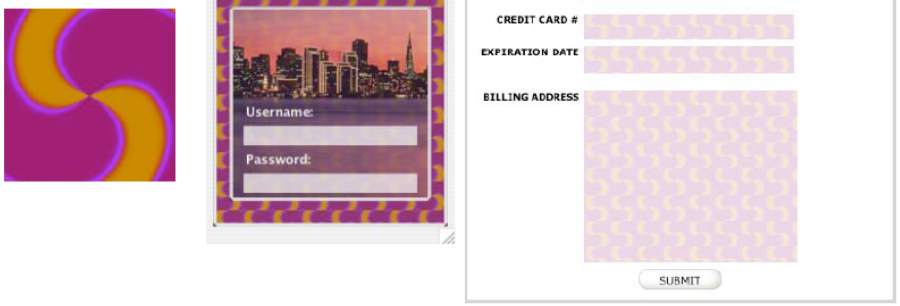
Skins Once the user and the server have verified that they both are who they claim to be, the server can use different images or *skins* to put in the website. These images can be randomly generated by the server, such as a visual hash [PS99].

These skins can be added in several ways in the websites, such as in the border (of the browser) or in input fields. Furthermore, different hash images can be used to identify what kind of security is used by a website. A generated image is created during the last phase of the SRP protocol. The image is made using the hash value of the session key. The server uses this value to generate an abstract image using the visual hash algorithm of Perrig and Song [PS99] – for technical details of the algorithm the reader is redirected to their article. The browser on the other hand can produce the same hash, since the browser of the user also has the session key. Now the user only has to check if the two hash images are the same. Checking an image is, for most human beings, far more easier than reading through certificates and keeping an eye out for certain authentication and security icons. Thus it is believed by Dhamija and Tygar that users will like to use their DSS for preventing themselves from getting spoofed.

Security analysis Let's consider the security issues for the DSS.

- **Leak of the verifier** The verifier is sent only once over the network. Once it is kept safe on the server, an attacker can make no use of the verifier, since it is not the same as a password. But, when the verifier is captured during the transfer from

Casus 5.4: Using hash images in DSS



These pictures give an idea what dynamic security skins do with randomly generated images. The first image is the basic hash image that is calculated using the session key that both parties have agreed upon. The user has to check wether the generated image given by the DSS client is the same as the server shows him. If this is not the case, then the user knows that the server is not the server he believes the server to be. The second and third picture show how the server can use the generated image in the webpages to guarantee the user that he is looking at a safe website and that his personal information can not be stolen. The second image shows the login screen (where the generated image is used in the border of the window). The third image shows the use of the generated image inside textfields. Doing the latter, the user can convince himself that it is altogether safe to enter his credit card information in these fields.

user to server, the attacker can use it to impersonate a server. So, the connection on which the verifier is sent to the server, should be safe.

- **Leak of the images** The personal image of a user is not sent over the network, an attacker only can get this image when he is in the same room as the user or when he has compromised the browser. The skin which is calculated using the session key cannot be used for more than one session, since the session key is never sent over the connection.

- **Man-in-the-middle attack** The SRP itself prevents a classic man-in-the-middle attack. But there also could be a man-in-the-middle attack using a window on top of the login window to trick users into logging into a spoofed server. Using the trusted path to this window, where a user specific image is layed over the window itself, prevents from this kind of man-in-the-middle attack. But when the attacks get so sophisticated that the phisher decides to make the login fields transparant, a man-in-the-middle attack is possible again.

- **Spoofing trusted window** Trusted windows can be spoofed when there are not enough images to choose from. Therefore, DSS supports the ability to use personal images (such as photographs) to be used. In this way, the number of used images grows so large that such a spoof is not that likely anymore.

- **Spoofing visual hashes** The visual hashes can only be spoofed when users do not check their hash with the hash used in the browser. There is no easy way to prevent this since preventing this kind of trickery lays in the hands of the user.

Does it work? The idea is a good one. It is still being tested and finetuned doing usability tests. If the system can be used already or if it is widely supported by servers is not known. And even if it is supported, it is still the responsibility of the users to use it correctly. If not used correctly, this system can not keep users safe from phishers.

5.4.2 Detection of phishing websites using visual similarity

An altogether different way to attack phishing itself is to check several websites to see if they look similar. A human can do this too, but since computers mostly work faster than humans can, it would be nice if a computer could do the similarity check.

Wenyin et al. [WHX+05] propose a means to detect if a website is a forgery. At first, a webmaster or the like can identify URLsthat are suspicious. Then software can be used to determine how much the possible forgery is indeed a forgery. To do so, the true website and the supposed spoofed sites are compared to each other using three aspects: block level similarity, layout similarity, overall style similarity.

Block level similarity The webpage is divided into several blocks. The content of an individual block can be categorized as either text or image. These two kinds of blocks have different features on which the comparison takes place. The features for text blocks are, for instance, color, border style and alignment. Possible features considered when looking at an image consist of overall color of the image, image size and possible text.

The similarity of two blocks is calculated as follows. First feature similarities are calculated. For all features of the two blocks, these similarities are combined into a weighted sum to depict the similarity between two blocks. In this particular implementation the focus is more upon color features.

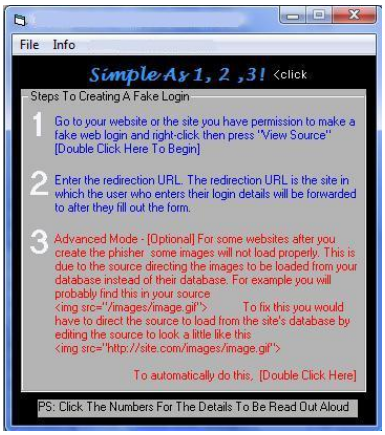
Two blocks are considered similar if their similarity is higher than a given threshold. After all the similarities between the blocks of two webpages are calculated, a matching scheme can be found between two webpages' blocks.

Layout similarity Since it is time-consuming to replicate a website from scratch, it is believed that phishers copy the source code of the original website in order to make the forged website. Then this code can be altered a little to the phisher's wishes.

The site is divided into topic blocks. A topic block is a block where the contents belong to a certain topic, which can be of interest to the reader, such as the sports section. To calculate the layout similarity of two websites, the websites are both divided into matching layout blocks. Two blocks are considered similar if they both have a high similarity and the same relation to corresponding matched blocks in the website. Then, the similarity is calculated as the ratio of the weighted number of matched blocks to the total amount of blocks in the original website. Furthermore, the weights of the different blocks is assigned according to their importance in the whole webpage. These two, the ratio and the weights considering the importance, define the layout similarity.

Casus 5.5: "Do It Yourself"

Nobody will disagree on the fact that phishing is a big problem and it is difficult to do something about phishing. Now that phishing seems popular, also "Do It Yourself" phishing kits have popped up. Using these kits it is even more simple to create a fake website or a fake e-mail. But it becomes funny (and also very unnerving) when phishers also phish other phishers. nu.nl published a newsitem on January 28, 2008, where is stated that many of these DIY kits not only send private information to the initial phishers, but also to the developers of the DIY kit.



Overall style similarity The style of a webpage can be a means of a user to detect if a site is fake. Thus it is important to a phisher to get the style of his webpage to match to the style of the real webpage. This gives a reason why also the overall style similarity is taken into account to see if a website is false.

The overall style is calculated as follows. Style features are determined as font family, background colors, alignments and so on. For each feature value of one feature, the blocks and their weights are used to count how many times they appear. This is used to compute the correlation between the two different webpages.

Performance When these three different similarity values are calculated for two sites, they all give some value. Beforehand, the system is given a certain threshold (a value between 0 and 1). If one of the three values is higher than the predefined threshold, the supposed false site is considered a false site. Obviously this system can give false positives (sites flagged as non-phishing, while they actually are phishing websites) and false negatives (sites flagged as phishing, which they are not). The ratio's of these false negatives and false positives can be manipulated by changing the threshold. The tests which were done with this system used thresholds of 0.7 or 0.9. Both values are not optimal, using a threshold of 0.7 resulted in no false alarms, but some missing. Using a threshold of 0.9 resulted in several false alarms, but no missings. The optimal threshold probably lies somewhere in between.

5.4.3 Detection of phishing e-mails

Spam filters are commonly known nowadays. Spam filters have the ability to detect whether an e-mail is spam or not. Most spam filters focus on the text in the e-mails, which is likely to contain strange words or faulty language. As opposed to spam, phishing e-mails are more likely to contain correct texts inside the body of the e-mail, since it is the objective of the spoofers to lure somebody into trusting the e-mail. In order to gain the trust of a user, the phisher needs to send e-mails that look a lot like an e-mail the spoofed company would have sent. Thus, normal spam filters will not work too well on

phising filtering.

Fette et al. [FST07] try to tackle the phishing e-mail using *machine-learning* . To go short, machine learning is the science to build – mathematical or probabilistic – models to classify data. Their approach is called *PILFER*. PILFER is a classifier which uses a random forest. A random forest creates several *decision trees*, which are pruned afterwards. A decision tree is a tree (graph) where each node is a feature. From each node several branches emerge where each branch is a decision path (such as: 'has an IP-based URL AND has nonmatching URLs'). At the end of such a path a leaf is found. The leaf consists of cases (e-mails) considered phishing and cases (e-mails) considered non-phishing. For further understanding of machine learning and decision trees or random forests, the user is referred to any literature on these subjects.

Although probably many features can be found which can classify a e-mail as being "phising", mostly less features are desireable over more features. The reason is that the model that is learned to classify might be prone to overfitting when too many features are considered. When overfitting is an issue, it might be the case that too many e-mails which are actually phishing are not flagged as such and vice versa. PILFER – which was trained and tested in 2007, using testsets from 2002, 2003 and 2006 – uses only 10 discriminative features. The testsets used are fairly old, but when this kind of filter is used, the filter needs to be trained again on a new dataset. Hence, although this particular instance of PILFER only uses 10 features, it is probable that more features are needed in the feature (or old features are no longer discriminant and therefore can be deleted). Some features are also used to detect if a e-mail is SPAM, but proved to be useful when detecting phishing.

Discriminant features

1. **IP-based URLs** Since attackers wish to lure users into going to their site, but they might not have the required domain name (or any domain name for that matter) phishers can use IP-based URLs. This will probably happen less in the feature, since phishers tend to register a domain name for the use of phishing. But as there are still enough phishers using IP-based URLs, this feature stays important. This is a binary feature.

2. **Age of linked-to domain names** Most of the domain names used by phishers are not old. These domain names might be registered using a stolen credit card, in which case the registrar may cancel the registration. Furthermore, some companies hire other companies to monitor suspicious registrations. Therefore, these domains do not have a long lifespan. PILFER does a WHOIS query on the domain name and gives the e-mail a flag if it is less than 60 days old. This is a binary feature.

3. **Nonmatching URLs** Nonmatching URLs are URLs where the text of the link does not match the URL in the HREF statement.

4. **"Here" links to non-modal domain** Phishing e-mails might contain several links. Links are modal if they are used to give the e-mail a trustfull look and feel, such as the disclaimer and privacy policy. These links probably link to the official site. But there is always a link which the phisher wants a user to click, possibly

surrounded by the text "here", "click" or "link". When there is a link surrounded by these words that does not match the modal domain, the e-mail is flagged. This is a binary feature.

5. **HTML e-mails** An e-mail is flagged if it contains HTML-code in the body. The reason for this is that it is more difficult to fool a user into clicking on a false link when only plain text is used. This is a binary feature.

6. **Number of links** The number of links inside the e-mail is being count. Links are defined as being an `<a>` tag with a HREF property. Also mailto: links are count. This is a continuous feature.

7. **Number of domains** The number of main domains are counted. A main domain is a domain which you can actually register, such as `uu.nl` or `yahoo.co.uk`. This is a continuous feature.

8. **Number of dots** The number of dots in a HREF statement is counted. Attackers tend to produce legitimate looking domains by using subdomains, such as `http://my-bank.update.data.com`. Another way to produce such URLs is to pretend that the site is hosted at a domain, when actually the URL redirects the browser to the malicious website. An example:
`http://www.google.com/url?q=http://www.badsite.com`. While legitimate URLs can contain a lot of dots too, this feature can still be descriptive. This is a continuous feature.

9. **Contains javascript** Attackers can use javascript to perform a sophisticated attack or hide information from the user. Thus the e-mail is flagged if the string "javascript" is found in the e-mail. To the classifier it does not matter if the string appears in a `<script>` tag or an `<a>` tag.

10. **Spam-filter output** When an e-mail client has a spam filter, the output (if a e-mail is spam or not) is taken into account. When a e-mail is considered to be spam, the e-mail is flagged. This is a binary feature.

PILFER can be used to detect phishing e-mails and even gets a higher accuracy when a spam filter is used alongside PILFER. With minor changes to the feature set PILFER can also be used to detect phishing websites. Then the spam filter feature is useless, since spam filters can not filter websites. Some other filters need minor changes. Some new features:

1. **Site in browser history** When a site is already in the browser history, the site is not likely to be a spoof. And a large number of visits indicates that a user has some relation to the site. Hence if the site does not have these features, it might be fake.

2. **Redirected site** Although redirection can be legitimate, it can be that the redirection is illegal. For these situations, this feature is added and records if the user went to a page directly or if the browser was explicitly told to redirect to a page.

3. **tf-idf** tf-idf stands for term frequency-inverse document frequency. This is a measure of importance of a term. This can be used to identify key terms of a page. Furthermore, it can be used to determine if the current page is a copy of a more popular page.

Performance After training and testing PILFER achieves an accuracy of 99.5%, which is very nice. Furthermore, the false positive rate – the proportion of good e-mails classified as bad – is approximately 0.0013. The false negative rate – the poportion of bad e-mails classified as being good – is approximately 0.035. The spamfilter (SpamAssassin) got a false positive rate of 0.0012 and a false negative rate of 0.130, when trained. The performance of PILFER was recorded when it was trained and when it used the spam-filter output as a feature. Both where tested using the same testset. Given these results, PILFER works very well in classifying e-mails into good and bad e-mails and therefore might be a very welcome technical solution to prevent from being attacked by phishers. PILFER was only delevoped in 2007 and no information is found that indicates that one can download this filter at this time.

5.5 Summary and conclusions

Phishing is the forgery of e-mails, website or other information media that companies and organizations use to communicate with their customers and clients, intended to extract all kinds of sensitive information from users, like passwords or credit card numbers. Phishers use this information to aid in some scam, or to gain personal wealth. Phishing attacks are performed mostly in the form of e-mails or websites.

There are many indications that an e-mail or website is part of a phishing attack. Some of these are fake links, mismatching URLs, language and grammar errors, forged e-mail headers so that they seem to come from a trusted sender, missing certificates or fake certificates. Urgent requests to the user to get them to do something for which the sender does not have a logical reason at all are also a very good indication that an e-mail or website is fake.

Practice has shown that many users are easily fooled by phishing attacks because they do not think of (all of) the ways by which phishing can be recognized as such. This fact calls the need for automated ways, tools and techniques to recognize phishing, in order to help users in the battle against phishing.

Some simple anti-phishing software is available in Firefox and Internet Explorer. More sophisticated software to prevent phishing is currently being developed. Dynamic Security Skins are a means to let a user know if he or she is entering a safe website without the tedious job to check certificates and such. But the functionality of DSS depends on how the user interacts with DSS: DSS cannot prevent you from being phished if you use DSS incorrectly.

Also, software is developed to detect if a website or e-mail is false. Software which do such, are the visual similarity system from section 5.4.2 and PILFER. Both systems can detect a phishing website (although PILFER has to be altered to be able to detect

phishing websites), and PILFER can also function as a phishing filter. Using such systems have the major advantage that users are not confronted with phishing e-mails and websites. Hence, the problems encountered when having human interaction are not applicable to these systems. Unfortunately, no information was found whatsoever if these systems are currently deployed.

Chapter 6

Electronic Voting 2.0

Written by *Jeiel Schalkwijk.*

Recently direct recording electronic voting machines have become the cause of controversy. The primary problems are a lack of paper trail and the impossibility of verifying that votes have been correctly registered. Because of these issues all voting machines used in the Netherlands have lost their certification and paper ballots remain the only legal option.

At the same time a lot of research is being done in utilizing cryptographic techniques to create voting systems that are more resistant to fraud and cheaper to use than paper ballots. In this paper we will discuss two of these systems. One of these is ThreeBallot, which does not use modern cryptographic technology. The other utilizes zero knowledge proofs and is based on the hardness of the discrete logarithm.

6.1 The current state of voting machines

With the rise of computers, many countries, including the Netherlands, were eager to use electronic voting machines to conduct elections. Compared to hand counting, electronic tabulation is cheap, fast and free of human error. The machines also removed the possibility of human error when filling in the ballots. Invalid and unreadable ballots, whether accidental or not, became a thing of the past. Voting machines are also immune to some types of voting fraud, such as chain voting[1] and vote spoiling[2].

Voting machines were also thought to increase security. They were seen as secure, tamper-proof black boxes. This was an era before most people had knowledge of viruses, spyware and adware. Computers were likened to bank ATM's: secure trustworthy machines.

[1] In chain voting a vote buyer fills in a ballot and gives it to a voter, who deposits it and brings back an empty ballot, completing the cycle. See also [JON05]

[2] In vote spoiling valid ballots are invalidated, e.g. by vote counters during tabulation.

Lately people have started to realize that this trust in voting machines was misplaced. Direct recording electronic (DRE) voting machines have one serious flaw, namely the lack of vote verification. Specifically, how can we trust the machine to count correctly and not maliciously? How can we know for sure that the correct internal counter was incremented when the voter pressed a button?

With paper ballots, each voter is assured that the ballot she[3] deposited in the ballot box was correctly filled in. This is a very basic form of vote verification. If the election result is very close or if there are suspicions that the votes were not tallied correctly, a recount can be ordered. The ballots form a "paper trail" that can be consulted when in doubt. A paper trail can not prevent all types of fraud, so it has to be augmented with tight voting procedures. For example, to prevent against vote spoiling, multiple people from opposing parties should be present when ballot boxes are opened and when ballots are counted. Receipt free DRE machines offer no such paper trail. The voter has to blindly trust the machine to record her vote correctly.

Ballot printers and scanners provide an alternative to DRE machines. As their name suggest, ballot printers merely print a paper ballot, which the voter can then deposit in a ballot box. Only a deposited printed ballot carries legal weight, so the printer does not need to keep track of what it printed. Any fraud by a ballot printer would immediately be detected by the voter. Ballot scanners scan ballots and tally the result. They can be used on their own or in combination with ballot printers. The ballots can also be counted again by hand to verify that the scanner tallied the result correctly. In the U.S.A. optical scan machines have been used for quite some time. The ballots are strips of paper with perforations. The voter punches a hole corresponding to a candidate and the machines scans the paper strip. These "opscan" machines have often been the subject of controversy. But since there is still a paper trail, a hand recount can always take place to verify whether the ballots were counted correctly.

Ballot printers and scanners can provide most of the advantages that DRE machines have, while still allowing voters to verify that their ballots were cast as intended.

6.1.1 The Nedap ES3B

In the Netherlands paper ballots were slowly being replaced by DRE machines. Of the DRE machines used, the Nedap ES3B had a market share of 90%.

In 2006 DRE's were (finally) introduced in Amsterdam and in that same year a group of concerned citizens formed an organization to protest the use of these machines. The organization[4] was aptly named "We do not trust voting computers" and many of its members are active in computer security. Their objective was to stop the use of DRE machines in the Netherlands, because of the security vulnerabilities of these machines.

To prove their point they performed a security analysis on the Nedap ES3B[GH07]. Their primary result is that the Nedap ES3B could very easily and *undetectably* be reprogrammed to give any voting result. It would be trivial for Nedap to program the

[3]It has become custom to refer to voters in the feminine form, since most voters are women and "he or she" is rather cumbersome.

[4]http://www.wijvertrouwenstemcomputersniet.nl/

Casus 6.1: NEDAP ES3B CHESS

One of the first arguments made against the Nedap voting machines by "we do not trust voting computers" was that they were just computers that could be programmed to run anything, even a chess program.
Nedap maintained that their voting machines were special purpose machines and that it was nonsense that they could run chess.
So when the activists got an ES3B they hacked it to play chess. But with only 16 KB of RAM, it does not have a promising future in the sport.

ES3B with malicious code, but they could also be hacked during transportation[5], while they were storage or in the days before an election.

Another flaw of the ES3B that the authors identified was that it leaked electromagnetic radiation that could be analysed to reveal what a voter was voting. Admittedly, proper shielding could solve this problem. But it is also important to realize that paper ballots can not leak information, with or without shielding.

Because of these issues, the voting machines were decertified in October 2007 and the only remaining legal option for Dutch elections are the tried and trusted paper ballots, filled in with a red pencil. The manufacturer appealed the ruling, but it was telling that the Dutch government did not support their appeal, even though they had supported them during the initial case.

The activist organization had stated that it prefers the use of paper ballots, since that system was well designed and it has worked very well in the Netherlands for many years. The Dutch government stated that they want to use a ballot printer and scanner

[5]The machines were regularly transported to and from Nedap for maintenance, with the only supervision being the truck driver.

system in the future and have planned its introduction in 2010. But according to the activists newsletter the Dutch lower house was not very enthusiastic about it.

It should also be noted that the cost advantage that DRE's provide can only be achieved if the initial cost of purchase is spread out over many elections. Because they have been decertified so soon, the relative cost of these machines is quite high. To avoid the same costly mistake in the future with any new computerized system, perhaps it would be wise to run a pilot program for a long time before widely deploying them. And it's also wise to be very critical of any new system. Since paper ballots are not broken, we should not be eager to fix them.

6.2 Cryptographic systems

A lot of research into voting systems has been published by cryptographers. The objective is to produce a system that is immune to all types of voting fraud. The general idea is that each voter receives a receipt that allows her to verify that her vote has been correctly tallied in the end result. A list of all voters is also published. Removing or modifying a vote would be detected by the voter and could be proven with the receipt. Adding votes would require inflating the voter list with non-existent people or non-voting voters, both of which are detectable.

It is important that these receipts do not allow a voter to prove to anyone how she voted, because then they could be coerced or enticed to vote a certain way. This property is called coercion freeness[6]. Coercion freeness is strictly stronger than the well known voter privacy property, which states that nobody should know how a voter voted.

A voting system is correct if the final outcome is exactly the result of tabulating all valid votes and nothing else. Vote verifiability helps ensure correctness (sometimes along with other anti-fraud combinations). DRE's have zero vote verifiability, while paper ballots have partial vote verifiability (you are guaranteed that your ballot was correctly deposited in the ballot box). The systems discussed in the next sections aim for *universal verifiability*, the strongest form of vote verifiability. In universal verifiability each voter can check to see that there vote was correctly tabulated in the end result, this ensures that the result is correct.

It is trivial to get a correct voting system. Just make every voter-vote pair public, so that everyone can verify that their vote is counted as intended. Coercion freeness is also trivial, DRE machines in private booths do a very good job. The challenge is to have both properties in a single system.

Fianlly, it should be noted that the cryptographic systems compete with paper ballots, so their security properties should be an improvement over paper ballot voting, not just DRE voting. Although paper ballots are neither perfectly coercion free nor perfectly correct, they are pretty good.

[6]Some authors call this property receipt-freeness, but that is a somewhat confusing term, since voters generally do receive receipts.

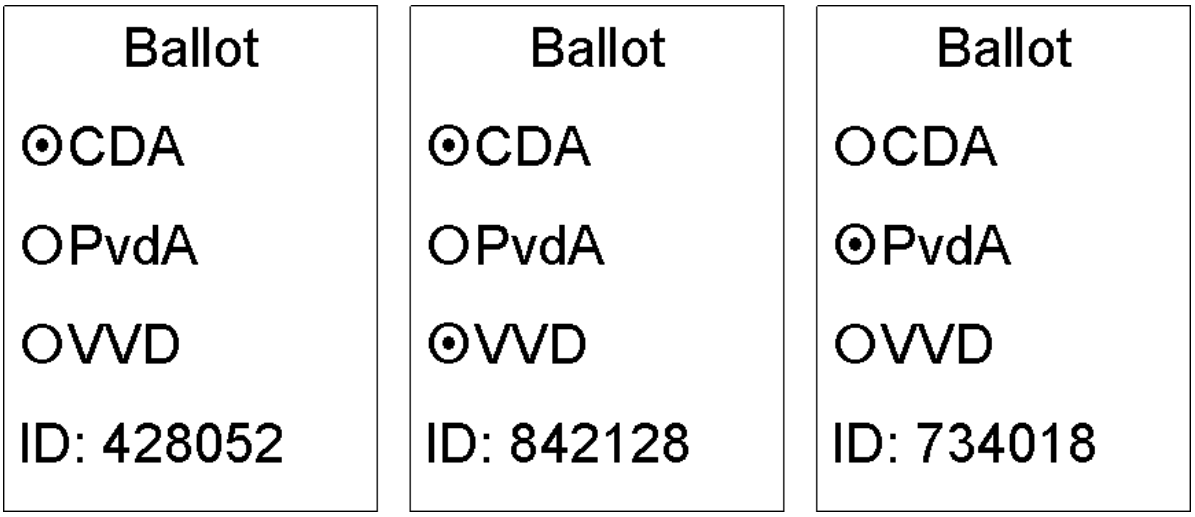


Figure 6.2: A ThreeBallot cast for CDA

6.3 ThreeBallot

Rivest and Smith proposed three voting systems that are voter verifiable, but which do not use cryptography. Their motivation was that, although cryptography has many advantages, average voters and politicians may be reluctant to use a complex system whose inner workings they do not understand. One imagines that the fiasco with DRE's will not make these people less reluctant to try advanced electronic voting machines with obscure number filled receipts.

One of the proposed systems is ThreeBallot. In ThreeBallot each voter receives three identical and unrelated paper ballots, instead of the usual one.

Casting a vote in ThreeBallot To cast a vote for a candidate, that candidate should be marked on exactly two ballots. All other candidates should be marked exactly once. The ballots are placed in a checking machine to ensure that these constraints are met. If the constraints are met, the machine copies one ballot (the voter chooses which) as a receipt and deposits all ballots in a ballot box. The machine must be simple enough so that it can not remember which ballots it copied.

Receipts must be authentic, either by using special paper or by using digital signatures.

Tallying the vote If n people voted, then all candidates will have $n + c_i$ votes, where c_i would be the vote count using traditional ballots. It is thus trivial to convert between a ThreeBallot score and a traditional score. All ballots are published publicly (e.g. through a website), to allow voters to verify that their receipt ballot has been counted in the final tally. The published ballots are not images of the physical ballots, only the contents of the ballots are published. The names of all those who voted should also be published.

Prevention of fraud and coercion Adding ballots is impossible, since extra names would have to be published. As with all voting systems, measures should be taken to prevent ballot stuffing by listing the names of fictional, dead and absent voters. A third of all ballots have been protected by copying and modifying those ballots from the final result can be detected. Because nobody knows *which* ballots have been protected, any fraud can potentially be detected, with high probability. For example, if 6% of ballots have been modified and 50 voters check their ballots online, the fraud will be detected with a probability of 95%, regardless of the number of voters.

Coercion or vote selling is simply prevented by using three ballots. A voter can produce any possible ballot to a coercer, while her other two ballots actually determine her real vote.

6.3.1 Problems with ThreeBallot

ThreeBallot is very dependent on the checker (and copier) to work correctly. A checker could be rigged to allow three votes for a certain candidate or a copier could be rigged to remember which ballots have not been copied. The authors state that the checker and copier should be constructed in a simple (mechanical) manner, so that any tampering will be evident. They should especially not use programmeble electronics.

ThreeBallot is also vulnerable to a specified pattern attack. If there are c candidates, then there are 2^c possible ways to fill in each ballot. If c is sufficiently large, a coercer can demand that a specific and unlikely pattern is published and a 'completing' pattern should be copied by the voter. Even with a relativly small c many voters might not be willing to take the risk that a specified pattern is used by some other voter.

Finally there are also usability issues with an uncoventional system such as ThreeBallot. Rivest and Smith do point out that ThreeBallot and traditional (single) ballots can be seamlessly used together. In the published list of voters it must be recorded wether the voter used a single ballot or ThreeBallot and the ballots should be identical.

6.4 Moran and Naor's protocol

Moran and Naor[MN06] base their protocol on those of Neff[Nef04] and Reynolds[Rey]. The protocol works by exploiting the temporal nature of an interactive zero knowledge proof (ZKP) and combining it with that of a commitment protocol. A commitment protocol works like this:

1. A committer has some secret a
2. A commitment c (derived from a) is published, such that c does not reveal a. At the same time there is no $a' \neq a$ that can be derive the same c.
3. (Optionally) a is revealed, and it can be verified that a derives to c.

Now suppose we have a prover who knows some statement S. The prover wants to prove knowledge of S to a verifier, but without revealing S. The steps taken in a typical ZKP are:

prover Commits to two proofs P_0, P_1, such that both proofs can only be true if the prover knows S. (S can be derived from P_0 and P_1).

verifier Makes a choice $b \in 0, 1$.

prover Reveals proof P_b.

verifier Verifies that proof P_b is valid and that it is actually the committed proofs.

Since the prover has a 50% chance of correctly anticipating the challenge, the ZKP is iterated a few times. The probability of guessing correctly ten times is just 0.1%.

It is trivial to construct a valid P_0 or P_1 without knowledge of S, but it is not possible to construct *both* without knowledge of S. This means that if step two comes before step one, we can construct a fake ZKP that looks exactly like a real ZKP. Note that in a fake ZKP the proof that is never revealed is invalid. But because it is never revealed, we can get away with this 'bluff'.

In Moran and Naor's protocol the verifier is the voter and the prover is the voting machine. The voter wants a real proof for her chosen candidate (by only giving a challenge after the machine commits), but will help the voting machine construct fake proofs for all other candidates by giving challenges before the machine commits.

The protocol is described below. Please note that there are two types of commitments. The first commitment v is a commitment to the voter's vote. This commitment will only be revealed during the final tallying. The other commitment is a commitment to the proofs. Each proof 'proves' that v is a commitment to a vote for a particular candidate. For clarity we only iterate the ZKP part of the protocol once.

1. The voter chooses a candidate.
2. The voting machine makes a commitment v of the vote.
3. The voter provides challenges for the *other* candidates.
4. The voting machine makes a masked v_c for each candidate c. It commits proofs for each candidates. Proof $P_{c,0}$ proves that the v_c is actually a masked v and proof $P_{c,1}$ proves that v_c is actually a vote for c. If both are valid proofs, then v is a vote for c. This is the case for the chosen candidate. But for all other candidates only one proofs is actually valid, the other is invalid! Which is valid depends on the already provided challenges. v and all $v_c's$ are printed.
5. The voter gives a challenge for her chosen candidate
6. All challenges and the corresponding proofs are revealed and printed.
7. The voter verifies that the printed challenges were the ones she provided.

As long as the voter ensures that she gives the challenges in the right order, she can be assured that v is a vote commitment for her candidate. She gets to take the printout home with her, to verify that her vote was counted correctly.

All receipts are published on a publicly accessible website. *Anyone* with enough technological knowledge (e.g. political parties, observers) can verify that all published receipts are correct (i.e. the proofs are valid proofs and correspond to the challenges). Each voter can verify that the published receipt matches the printed receipt. Together this ensures that each vote is correctly registered.

Tallying is also based on a ZKP, this time between the voting machine and the tallying authority. The machine permutes and masks the vote commitments. The tallying authority can choose to (1) reveal the masks, thus establishing that the masked commitments are a permutation of the published commitments, or (2) open the masked commitments, thus revealing the votes, without revealing to which published commitment they correspond. By iterating this a few times, the tallying authority can be sure that the voting machine is not cheating. Of course, this exchange should also be publicly published.

6.4.1 Pederson Commitments

Moran and Naor's protocol utilize Pederson commitments[Ped92]. The commitment protocol looks like this:

1. A value a is selected
2. A Pederson commitment $P(a, r)$ is published, with r randomly chosen.
3. The commitment is opened by publishing (a, r)

It is computationally infeasible to find a different pair (a', r') such that $P(a, r) \equiv P(a', r')$. This implies that once $P(a, r)$ is published, we can be confident that the committer can never change the value of a. That is why it is called a *commitment*, it commits the committer to a particular a.

Furthermore it is impossible to derive what the value of a is from $P(a, r)$ without knowing r.

Technical details: Pederson commitments are based on the hardness of the discrete log. To compute them we start with a cyclic group G of order q and with two generators $h, g \in G$ such that $\log_g h$ is unknown and hard to calculate. G, q, g and h are publicly known. To commit an $a \in \mathbb{Z}_q$, a random $r \in \mathbb{Z}_q$ is chosen and $P(a, r) = h^a g^r$ is sent. To open the commitment (a, r) is revealed.

If the committer knows another (a', r') such that $h^a g^r = h^{a'} g^{r'}$ then he can compute $\log_g h = \frac{r - r'}{a' - a}$, which is assumed to be hard. Because of this, finding two pairs that commit to the same value is as hard as calculating $log_g h$. Pederson commitments are thus computationally binding.

If r is chosen randomly from $\mathbb{Z}_q$ then g^r is a random element of G, because g is a generator of G. In turn, that means $h^a g^r$ is a random element of G. $P(a, r)$ is thus perfectly hiding if r is perfectly random.

We can also *mask* a Pederson commitment with r'. The masked commitment is $P(a, r + r') = P(a, r)g^{r'} = h^a g^r g^{r'}$. Opening the masked commitment does not require revealing r! One only needs to reveal $(a, r + r')$. Similarly one can unmask a masked commitment without opening the commitment by only revealing r'. This proves that the masked commitment is actually the other commitment, $P(a, r)$.

6.4.2 Problems with Moran and Naor's scheme

One security issue is that a corrupted voting machine can break voter privacy by revealing all votes cast on that machine.

A second issue is that the inability of the voter to be objectively random can be exploited. If the voter can base her 'random' challenge on the printed commitments, her receipt becomes a proof that the real challenge was given after the commitment. The receipt is then a proof of how she voted, and a coercer can exploit this. To prevent this, the authors recommend that the printed page is partially hidden behind an opaque shield. The voter can see that something was printed, but can not yet read it.

Bohli et al[BMQR07] also present an attack using either a radio receiver or a pre-recorded audio track. Their recommendation is to use a trusted random number generator instead of allowing the voter to directly input numbers.

It should be noted that these issues only affect the coercion freeness property of the system. The system is computationally correct under all circumstances.

And the final issue is usability. The voting process has little in common with traditional paper ballots and such a big change in the way we vote is sure to meet with resistance. The fact that the system does many complex calculations might also be a cause of concern for those who do not understand the purpose and limitations of these calculations. For example, a vote coercer might convince non-technical voters that he can "decrypt the vote on the receipt". You, the reader, will not believe that. But if other people do, we all lose.

6.5 Conclusion

DRE's have failed in the Netherlands and with good reason, since they can be very easily be hacked. They make voting fraud very easy. Traditional paper ballots should not be dismissed just yet. The protocols surrounding their use are very well thought out and they have used for many years with great success.

Voting systems are actively being researched and can provide better security features than simple paper ballots. Some, like ThreeBallot, are rather simple. Others are more complex, but have even nicer properties. However we should be careful before implementing them. To those who read papers such as this, it might seem the most logical thing to use the best available voting system, but the cost, both social and economic, of switching from traditional ballots to one of the systems discussed should not be underestimated.

Chapter 7

Coupon Systems

Written by *Eric van Dalen.*

Companies that sell to private customers want to have a stable consumer database. People have come up with various methods for this purpose. One of those methods is the principle of coupon systems. Some product is discounted for a certain fixed amount or percentage. This way of marketing is suitable for introducing new products to customers. Not everyone is positive about these coupons, which are often received through mail, since it is a marketing system.

What exactly is a coupon? Searching for the definition of coupon in the Van Dale dictionary teaches us: [Dal]

Definition 7.1 *A coupon is a ticket that grants the right to the purchase of goods or services of a specific business for a certain amount of money.*

So a coupon grants a right to the owner. It is important to note that a coupon, in principle, does not bring the owner any negative effects [1]; the coupon can be discarded if this is the owner's choice. Also is there no third party involved, this is a major difference with ordinary money we see around us, banks and governments are of great influence here. From a technical point of view, we are not interested in the product for which the discount holds, neither the guarantees that an issuing party will deliver the product, but rather the coupon itself.

There are different types of coupons that fundamentally differ from each other in implementation. All of the different types are exclusive member of the analogue or the digital coupons. Although the choice for a analogue or a digital coupon does not have great implications on the security properties that are desired from the coupon, it does have great implications on the manner how these properties are implemented. Although analogue coupons are briefly mentioned in a moment, the digital version will be the centre of this chapter. Some general information about digital coupons is given in section 7.1.2. A special version of the digital coupon, the multi-coupon, is worked out in more detail in section 7.2.

[1] ignoring the chance to get cut by sharp edges

7.1 Coupon Security Requirements

Some security properties that can be questioned for an application of a coupon system are the following:

Redemption limit Should it be possible to redeem the coupon only a finite number of times? If so, should it be possible to discover a coupon's maximum usage?

Unforgeable Should it be that valid coupons can only be issued by a certain agency?

Anonymous Should it be infeasible to know the owner's identity from a coupon?

Non-transferable Should a coupon only be useful for people to whom the coupon is issued?

Limited time span Should a coupon only have a specified time interval during with the coupon can be redeemed?

We shall take a look at the way how current applications of coupon systems are dealing with these properties. Since the type of coupon systems can be quite different, one should deduce that different implementations are required; different circumstances ask for different implementations.

7.1.1 Analogue Coupons

Everybody probably knows some different coupons that are printed on a piece of paper by hart, an example is the Dutch version of the book coupon, which is known are the 'boekenbon'. Although the barcode on this coupon is digital in nature, it shall be classified as a analogue coupon because of the fact that these coupons are only useful for the purchase in an 'in real life' business.

The implementations of the security properties of these coupons are rather basic: The property of non-transferability is often not taken into account. When a system does have this property it often violates the anonymity property, an example of this are coupons from a Dutch lottery called the 'postcodeloterij'.

Although the importance of the anonymity property is negligible for agencies that publish them [2]. For customers it is believed to be a positive property to help trust the coupon. Since customers can always reveal their identity to the agency if they want, the anonymity property can never be negative for customers.

For agencies the property of redemption limit and unforgeability is more relevant. This property is implemented in different ways across different types of physical coupons.

For example, the security of one time use with the boekenbon is granted by the barcode system. When new coupons with barcodes are being published, a new entry in a central database is made that contains the barcode, the value of the coupon, the current state of the coupon (on entry it is active of course) and possibly a time span

[2] and sometimes the whole purpose of the coupon is to obtain personal information

Casus 7.1: Current Internet Coupons

The (lack of) implemented security properties regarding internet coupons have triggered the emerge of special websites. These websites are dedicated to gather as much coupons as possible and redistribute them to visitors. Although a part of the coupons that are found on these coupon sites can simply be requested from the agencies that issue them, it still saves a registration for these agencies. Some of the coupon sites are criticized because they ask for registration too.
In contrast to coupons that are distributed through mail, the internet coupons can save some serious money; discounts that range from 10 to 15 percent on electronic devices are no exception. Some comparisons of websites that gather internet coupons can be found [Coh05][Fox00]. They also tested if the coupons are mostly valid, this seemed to be the case.

during which the coupon is valid. When the coupons are begin spend, a connection to the central database is established, when a query passed the specific requirements, the coupon will be accepted and the state of the coupon in the database will state that it now has been spend. A future attempt to redeem the coupon will fail.

Other coupons have the one time use security implemented in the coupon itself. Certain parts of the coupon have special characteristics that are believed to be difficult to copy [3]. So if such a coupon is redeemed, the characteristics will be examined and when accepted, the coupon will be taken. This is comparable to euro bills.

7.1.2 Digital Coupons

Some of the generally less familiar coupons are the digital coupons. The purpose of these coupons is the same as those of the analogue variant. Digital coupons are mostly issued by web shops for certain discounts. The shapes of these coupons are often strings. However, implemented security properties can range from almost none, to all of them. This can be because certain uses simply do not require many security properties, or currently used systems are lagging behind.

Most of the current internet coupons systems that are being used by web shops are very primitive. Coupons from such systems often have the same security properties as physical coupons: Limited usage and anonymity. Sometimes they have a certain time limit. Some context information can be found in Casus 7.1.

Coupons can be looked at as a special form of money. As stated earlier, a major difference between coupons and money is the lack of a third party (banks and governments) with coupons. There a various existing digital money systems that have most of the security requirements, however non-transferable would be a property that contradicts with money as we know it. Some technical information about digital money can be found [Tel02].

In the following section, we will take a look at a system that has the same properties as digital money systems, but with some form of non-transferability.

[3] or are simply not worth copying at all

7.2 Multi-coupons

Multi-coupons have some additional business advantages over normal coupon, therefore a business can decide to issue multi-coupons instead of normal coupons. First we have to define what a multi-coupon exactly is.

Definition 7.2 *A multi-coupon is a fixed, bundled set of coupons, which can be redeemed separate from each other, but is issued at once.*

Popular examples of multi-coupons are those that can be obtained at cinemas. A consumer buys a multi-coupon that consists of m coupons for the price of $m-n$ coupons, for $n > 0$. The multi-coupon is valid for a certain consumer during a certain time span. Some of the typical extra business advantages for the cinema with this system is the 'locking in' of consumers, this means that it is unlikely that a customer is going to a competitive cinema as long as the customer has some coupons left. Since the customer pays the whole multi-coupon in advance, the cinema also receives money earlier. The advantage for customers is that the total amount of money paid for m shows is lower with a multi-coupon.

Liqun Chen, Matthias Enzmann, Ahmad-Reza Sadeghi, Markus Schneider and Michael Steiner came up with a system for digital multi-coupons [CES+05]. They think that the whole multi-coupon principle is useful for purchases over the internet; viewing prices for similar products from competing web shops required relatively little effort. First we shall state which security properties this system has. Second we shall take a look at how these properties are met.

Unforgeable It is infeasible for others to issue valid multi-coupons. This is one of the security requirements from section 7.1.

No resets It is infeasible to reset redeemed coupons. This is an exclusive property for multi-coupons; it is a form of unforgeability.

One time usage It is possible to discover whether a coupon from a multi-coupon has been used before. This is also one of the security requirements we have seen before.

Anonymous It is not possible to retrieve the owner's identity from a coupon, one of the requirements that has been explained before.

Unlinkable A party that does not own the multi-coupon cannot learn anything such that different coupons from the same multi-coupon can be linked to this same multi-coupon during any of the applied protocols. This property can be seen as an extension to the anonymous property.

Disclosed When a coupon from a multi-coupon is redeemed, the redeemer cannot learn how many active coupons are left in the multi-coupon. This property can also be seen as an extension to the anonymous property.

Weakly Unsplitable Splitting of a multi-coupon is only possible if a group of people trust each other not to redeem each others part. Every member of this group must still hold all the coupons; this is called an 'all-or-nothing' strategy. Weakly Unsplitable is a weaker version of the non-transferable property.

The Limited time span property from section 7.1 is not met. There are various methods to meet the property. A simple example would be to issue multi-coupons until a certain set date has passed, then after a year no coupons from any multi-coupons are accepted. Since this property can be met by means that are not built inside this type of multi-coupon, we will not give it more attention.

7.2.1 How It Works

There are two different kinds of functions the business must implement: coupon issuing and coupon redeeming. The implemented role that does the issuing will be called the issuer. The implemented role that does the redeeming will be called the redeemer. Note that both parties are part of the same business and that information between these parties can be shared freely.

There are mainly two protocols that must be followed by both the customer and the business to meet the properties that are stated above. The result of the first protocol is that a customer ends with a valid multi-coupon; this protocol shall be called the 'issue protocol'. The second protocol will be used to redeem an unused coupon, this protocol shall be called the 'redeem protocol'. We will take a fine-grained look at both protocols.

7.2.2 The Issue Protocol

The purpose of this protocol is that a customer gets a signature over his multi-coupon such that it is valid and can be redeemed. Every coupon shall have the form of a unique serial number x. Given a set coupons $X = (x_1, x_2, \ldots, x_m)$, customer U wants to receive a signature from issuer V such that U is the owner of a valid multi-coupon $(X, Sign_V(X))$.

However, before V can issue any multi-coupon that consists of m coupons, it has to be initialized. The result of this initialisation phase is a public-private key pair (PK, SK). Public key PK shall be of the form (A, b, c, n), where $A = (a_1, a_2, \ldots, a_m)$. The n is a 'special RSA modulus', this means that $n = pq$ for large primes p and q. Although these properties are relevant, they will not be explained in detail since a lot of literature can be found regarding this (e.g. [Tel02]). For completeness, the following must hold: $p = 2p' + 1$, $q = 2q' + 1$ and both p' and q' are primes themselves. Secret key $SK = p$ gives information about the factorization of n (a problem that is infeasible to solve).

$a_1, a_2, \ldots, a_m, b, c$ are all chosen uniformly random from a set QR_n. Although the properties of this set will not be discussed in detail, it will yet again be included for completeness.

Definition 7.3 *for every element a in the set of quadratic residues QR_n there exists an element b in $\mathbb{Z}_n^*$, such that $a = b^2$.*

Obviously, V will run the methods that are required to acquire (PK, SK), V then publishes PK, but keeps SK secret within the business. This initialization of V must be done once [4] before any multi-coupon is issued. For given $X = (x_1, x_2, \ldots, x_m)$ we now want to acquire a valid signature using (PK, SK). The signature system that is applied here is that of Camenisch and Lysyanskaya [JC02]. This system has some unique properties that will be used, the system is also very efficient for this purpose.

We will now take a detailed look at the issue protocol itself. Common input is public key (A, b, c, n).

1. U chooses random $X = (x_1, \ldots, x_m)$. Note that U and not V chooses the serial numbers of the coupon, in section 7.2.4 we will see why. Also note that every x_i should never be chosen before, chances for this to happen can be minimized by choosing numbers uniformly from a large range of number.

2. U chooses random s' for blinding purposes.

3. U sends a commitment value D to V, where $D = \prod_{i=0}^{m} {a_i}^{x_i} b^{s'}$. This D is linked to X such that it is infeasible to retrieve any x_i from D. Also, when X is changed, D is changed.

4. U proofs knowledge of X to V. This means that V knows that U knows the discrete logarithms of D (a problem that is infeasible to solve). This is done through a sub-protocol that uses 'Zero-knowledge proofs', from this V only learns that U knows X, no information is given such that it would be easier for V to retrieve (any part of) X. Techniques for Zero-knowledge proofs are described in [Tel02]. When U fails to proof this knowledge, V knows this and the protocol is stopped.

5. V chooses random large values s'' and e, where e is prime. Purpose of this is that U cannot choose some convenient values x_i and s' such that U can generate a valid signature on his own.

6. V sends (v, e, s'') to U, where $v = (c/(Db^{s''}))^{1/e} \bmod n$. One can ask if U cannot easily calculate this (v, e, s'') on his own since e and s'' are random and U knows every component of which v is dependant (b and c are both part of the public key). This is however *not* feasible, since this would break the Strong RSA assumption.

7. U uses signature $(v, e, s = s' + s'')$ for X. U can verify that (v, e, s) is a signature of X by checking whether $c = v^e D b^{s''}$. It is easy to see that this follows from the way how v is calculated.

This ends the issue protocol, the customer can now, if all went well in the issue protocol, make use of his multi-coupon (X, v, e, s). We will now take a detailed look how the customer can redeem coupons from X, using the redeem protocol.

[4] and only once to keep current coupons valid

7.2.3 The Redeem Protocol

The redeem protocol is used when a customer U wants to redeem a coupon at a redeemer V. The redeemer needs certainty that the coupon is valid. To meet the stated properties, the customer only gives limited information during the protocol. The used signature system of Camenisch and Lysyanskaya (CL) has a nice property that can be used to partially randomize a signature.

Theorem 7.4 *Given valid CL-signature (v, e, s) over X, then another valid CL-signature over X is $(vb^w, e, s - ew)$, for random w.*

This theorem will be used in the protocol we will now see in detail. Common input is public key (A, b, c, n).

1. U chooses a valid coupon x from X. Note that U knows which coupons he has used before, this overhead should not be a problem for U. There exists another technique where U deletes x from X, and still keeps a useable signature (v, e, s). This technique yields overhead too since the signature is only valid under another public key PK which he can compute. Hence, U should communicate this new PK to V first.

2. U sends x and v' to V, where $v = vb^w$ for random w. This prevents V from recognizing v from the issue protocol.

3. U proofs knowledge of X, $Sign_V(X)$ and $x \in X$ to V. Just as with the issue protocol, this is done using zero-knowledge proofs. The v' is used for proving $Sign_V(X)$. Since quite some mathematical details are ignored, one should realize that this protocol is a simplified version, full details can be found in [CES+05]. When U fails to proof knowledge, V knows this and U is rejected.

4. V checks x in database. When an existing entry of x is found, U will be rejected because he tries to redeem the same coupon x for at least the second time.

5. V adds x to database. This prevents U from using coupon x again.

When all goes well, coupon x is accepted by V.

7.2.4 Property Proof Sketches

We will now take a look at how the stated properties are met.

Unforgeable This property follows from the signature system from Camenisch en Lysyanskaya, breaking this would mean breaking the strong RSA assumption.

No resets Correctness of this property can be seen from how a signature is verified: $c = v^e Db^{s''}$, where D is a commitment value to X. So, when someone tries to change a certain coupon x_i from a set of coupons X, this D also changes. An other signature is required.

One time usage This property is met by using a database.

Anonymous A multi-coupon $(X, Sign_V(X))$ only exists of serial numbers and a blindly signed signature, thus does not contain information about the owner.

Unlinkable The only information that a customer sends to the issuer or redeemer which is not blinded is coupon x itself. However, X of which x is an element is signed blindly, thus seen for the first time at the redeem protocol.

Disclosed If any information can be obtained about the amount of remaining active coupons, some information can be linked together during any of the protocols. This would contradict the unlinkability property.

Weakly Unsplitable If someone does not know all separate coupons, he cannot proof knowledge of this in step 3 of the redeem protocol. It is also impossible to replace certain coupons of the multi-coupon with dummy coupons, since this would violate the no resets possible property.

7.3 Conclusion

Coupons have some nice advantages for both businesses and customers. However, current applications, both analogue and digital, are lacking security. For certain internet businesses (mp3 store, etc.) multi-coupons can be extra useful over normal coupons. An digital implementation of multi-coupons from Liqun Chen et. al. has some nice properties, but requires special protocols.

Chapter 8

Broadcast Encryption

Written by *Henri Kas and Jeroen Weijers.*

Broadcast encryption is the problem of sending an encrypted message to a large set of listeners in such a way that the message can only be decrypted by a privileged subset of listeners, while this subset changes dynamically. In this article we first explain the idea, applications and properties a broadcast encryption algorithm should have. Next, we discuss several broadcast encryption schemes, starting with the early schemes of Berkovits and Fiat & Naor. This is followed by the Subset-Cover model, which is used by some schemes that are widely used in practice. We conclude with a discussion of the Free Riders problem.

8.1 Introduction

8.1.1 What is broadcast encryption?

Broadcast encryption adresses the problem of encrypting data that is to be broadcasted such that only people that are allowed to see the data can actually see it. All other persons, those who are not authorized to see the data, can only see the ciphertext but not the actual content. This sounds like a typical encryption problem that can be solved with any form of encryption as long as the right keys are distributed over the people that are allowed to see the data. But there is a catch, broadcast encryption should also be able to handle changes in the group of persons allowed to see the data. When a person's right to see the data is revoked only he must be excluded from the group and none of the other users should be bothered. Another problem is that people may give their key to others, and then other non-authorized people can see the data. This traitor behaviour of people is addressed by some traitor-tracing algorithm's. This introductory section is largely based on the article [JLP02].

Casus 8.1: OCCAM, a different approach to broadcasting a secret

OCCAM is promoted as a means to protect data that is to be broadcast. OCCAM unfortunately in practice is a public key system. The system requires a device that wants to playback some data to be connected to a central server where the user-rights are stored. Based on the identity of the device key the central server might grant the device the right to decrypt the data. Although it is possible to get a ticket for later use the system requires the device to be connected at nearly all time. Also the system doesn't provide anonymity because of the fact that the central system sees what device wants to see what data, and this might easily be resolvable to a single user.

8.1.2 Applications of broadcast encryption

Applications of broadcast encryption range from encryption of 'premium' television channels, to protection of movies on HD-DVD or Blu-Ray discs. Also internet-streams can be encrypted in this form.

Using a broadcast encryption scheme to protect data on a movie-disc has some advantages over the use of a shared-secret scheme, such a scheme was used for encryption of DVD movies. The encryption scheme used for DVD is called Content Scramble System (hereinafter called CSS) and was cracked in 1999 by Jon Lech Johansen. It appeared that the disc-key could be found by using a brute-force technique and exploiting some weaknesses in the algorithm within 18 minutes on a 450MHz Intel Pentium 2 machine (such a machine was recommended for playing a DVD). Cracking this algorithm lead to the use of other cryptographic systems for the successors of DVD. Blu-Ray and HD-DVD both use techniques based on Advanced Access Content System (or AACS for short). AACS is a broadcast encryption based encryption scheme that allows to revoke the rights of some groups of players (for example when one of the decryption keys has been compromised). More information on AACS can be found in casus 8.4.

For protection of data on recordable media (such as: hard-drives, or digital music players) the Content Protection for Recordable Media was invented, this scheme is for example used in Secure Digital cards (although applications of CPRM on SD cards are rare). The CPRM scheme is a non open standard and only after paying for a license one is allowed to look at the implementation of this standard. A very similar standard is used for protection of DVD-Audio, this scheme is called Content Protection for Pre-recorded Media (CPPM) .

8.1.3 Broadcast encryption vs. Public key cryptography

Why would anyone use broadcast encryption when public key cryptography is also available? There are multiple reasons to do so, but sometimes there are also reasons not to use broadcast encryption. An important reason to use broadcast encryption schemes is the fact they are computable faster usually. Most public key systems use relatively expensive exponentiations whereas broadcast encryption schemes usually don't.

A second and also important reason to use broadcast encryption is the fact that

single keys that are compromised can be excluded from the system without directly rendering it completely useless, in contrast to public key systems. When we do provide a public key system with ways to exclude keys from the system the number of keys that are to be stored at each receiving client is rather large, and the communication overhead explodes.

A reason not to use broadcast encryption is the fact that most systems don't provide means to sign something with a signature that cannot be falsified. For most of the applications of broadcast encryption (protecting data on video-discs, or streaming media over the internet) this is not as important as protecting the content itself against inappropriate use. Broadcast encryption is not suited for internet banking, here non-repudiation is important and due to the lack of a way to uniquely sign data by a single user this non-repudiation requirement is not met.

8.2 Algorithms

8.2.1 Broadcasting a secret

The goal for a broadcaster is to broadcast a secret to a subset of his listeners. We denote the set of all listeners with U. This subset is called the privileged set of users, or T. Of course the broadcaster does not like to perform a separate encryption for each of his listeners, because this would require to much bandwith and computational costs. A better idea is to encrypt the message once and take care for all privileged users to receive the proper decryption key, in such a way that other listeners are unable to compute the original message.

So, the broadcaster encrypts the message with a secret key. Next, the broadcaster needs to send the key to each listener, this can be done in either a serial or parallel way. For the serial way, the broadcaster encrypts the key for each user separately, using each user's private key and then communicates the encrypted key to each user. This communicating can be done by using a reserved field within the message itself which contains a list of keys, one for each user.

The parallel way on the other hand, uses a unique integer n for each user which is used by the broadcaster to compute the encrypted key. This key is encrypted in such a way that modulo n, it fits the original key for each privileged user, but does not fit for a user that has not been subscribed. The consequence is that the encrypted key is as large as the product of all ns.

The schemes described above however, are not true broadcasting schemes (schemes in which the message contains the same information for each and every listener), because of the fact that each recipient has to be sent individual information that enables him to recover the secret. Whether this is done as a part of the message or on other ways, the main disadvantage is that, especially for a large set of users, we need to send or compute too much information. What we would like to have is a broadcasting scheme in which the broadcast message contains the same information for each listener and the privileged users are able to compute the message from it, while others are not.

In broadcast encryption, the quality of a scheme is determined by three properties: transmission overhead, storage overhead and computational overhead. The transmission overhead denotes the amount of keys to be transmitted by the center, the storage overhead denotes the amount of secret keys to be stored by the users, and the computational overhead denotes the amount of computations required to obtain the key.

8.2.2 Berkovits' scheme

Berkovits [Ber91] has thought of a true broadcasting scheme, which will be explained below. It makes use of Shamir's polynomial interpolation scheme [Sha79], which has been discussed during the Cryptography course. This section works out a $\{k{+}1$ out of $2k{+}1\}$ scheme, in which each privileged user can reconstruct the secret. Say we have k privileged users, and each participant gets a share of the secret. The secret is encoded into a polynomial P of degree k. Every share can be seen as a point on the graph of P, where the secret S is the constant term of P, i.e. the point $(0,\ S)$. When at least $k{+}1$ shares are known, a shareholder is able to interpolate P to recover the secret.

To use this principle in a broadcasting scheme, the broadcaster sends to all users a private share (x_i, y_i), for the k privileged users a real share and a fake for non-privileged users. The broadcaster now searches a polynomial of degree k that passes through $(0,\ S)$ and all points (x_i, y_i) that belong to all k privileged users, but does not cross any point of a non-privileged user. Then he publicly broadcasts k other points on the graph of P, that can be randomly picked. At this moment any privileged user is able to interpolate P to recover the secret S, because together with his own private share he knows $k{+}1$ points on P. Revoked users only know about k points, which is not sufficient for computing P.

In this polynomial broadcasting scheme, the transmission costs are $O(k{+}1)$ and the storage is of $O(1)$. Unfortunately, the polynomial interpolation method is inefficient for a large number of users because of the large computational costs ($O(k^3)$), so for large broadcasting purposes we need a more efficient scheme.

8.2.3 Schemes by Fiat and Naor

Fiat and Naor [FN94] have thought of several schemes and relate them with the term resiliency. A scheme is resilient to a set of non-privileged users if that set cannot recover the secret, despite their efforts to cooperate. So a scheme is k-resilient if it is resilient to any set $S \subset U$ of size k. Also a distinction is made between zero-message schemes and more general schemes. Zero-message schemes have the property that users who are a member of the privileged subset T are able to compute a common key, without any transmission with the center. Of course, in the initialization phase some things have to be transmit from the center to the receiver, but in the future no new information has to be sent to a user anymore. So for a new message, that is probably meant for a different subset of users, the privileged users can themselves compute the key.

A 1-resilient scheme Say, we have a set of users U and a privileged set of users $T \subset U$. The users from T now have to determine a common key which will enable

them to decrypt the messages from the center. First of all, we shall define a 1-resilient scheme that makes clear the idea of zero-message schemes. Later on, we will use this scheme to define some more efficient schemes. The following scheme is based on the Diffie-Hellman key exchange mechanism [DH76]. Every user $u \in U$ is assigned a public number p_u, where p_u, p_v are relatively prime for all $u, v \in U$. (Two numbers p_u and p_v are relatively prime if $\text{GCD}(p_u, p_v) = 1$). The center chooses a composite number N that is composed out of two large prime numbers, and a secret value g. Then for all users $i \in T$ their private key $g_i = g^{p_i}$ is computed. The common key for the subset T is defined as $g_T = g^{p_T} \bmod N$, where $p_T = \prod_{i \in T} p_i$.

Now, every user $i \in T$ can compute the common key g_T as

$$g_i^{\prod_{j \in T - \{i\}} p_j} mod N$$

and is able to decrypt the message with this key.

This system is secure because of the assumption that extracting a root modulo a composite is computationally hard. So we say that this scheme is valid to be a 1-resilient scheme and the user only has to store one key.

Unfortunately, two users can cooperate and can compute g, so this scheme is not 2-resilient.

A k-resilient scheme This 1-resilient scheme seems to be useless, but we can however use the preceding scheme as some sort of building blocks to construct a k-resilient scheme, for $k > 1$. To do this, we consider a set of functions $F : \{f_1, .., f_l\}$, which maps a user to a number from the set $\{1, .., m\}$. These functions have the property that for every subset $S \subset U$ of size k, there is a function $f_i \in F$ such that for all $x, y \in S : f_i(x) \neq f_i(y)$. So, we could also say that the set of functions F contains a hash function for all subsets of U.

We now use an independent 1-resilient scheme $R(i, j)$, for every $1 \leq i \leq l$, $1 \leq j \leq m$ and every user $x \in U$ is sent i keys that belong to the schemes $R(i, f_i(x))$.

The broadcaster generates random strings $M_1, .., M_l$, such that Xored together they form the message M, in other words: $\oplus_{i=1}^{l} M_i = M$. So, this can be seen as an additive secret splitting scheme. The center broadcasts for all $1 \leq i \leq l$ and $1 \leq j \leq m$ the message M_i to the users of the privileged subset $\{x \in T | f_i(x) = j\}$, using scheme $R(i, j)$. Every user $x \in T$ has the keys to obtain the messages $M_1, .., M_l$ and he can Xor them together to get M.

A coalition of users S that do not belong to T cannot get any information about a message M_i, because the keys they together have is at most the keys a single user from S has (since for every user a different scheme $R(i, j)$ is used). M_i is delivered in m independent transmissions and the j^{th} transmission uses the scheme $R(i, j)$. There could be at most one $x \in S$ for which $f_i(x) = j$, so at most one $x \in S$ has the keys of that scheme. Because $R(i, j)$ is a 1-resilient scheme, that single user x cannot recover M_i.

The number of keys a user has to store is m (because a 1-resilient scheme needs 1 key to store) and the length of the transmission is $l \cdot m$ times the length that is used in a 1-resilient scheme. Here $m = 2k^2$ and $l = k \log n$. We won't go into further details about these amounts, but we refer the interested reader to [FN94].

There are several enhancements [FN94] known for this k-resilient scheme, which we shall not describe. There are also schemes that are resilient to any size cooperation of revoked users, (even all r of them). These algorithms are called r-flexible. Examples of them, the Subset-Cover schemes, will be explained below.

8.3 Subset-Cover algorithms

In this section we will discuss the Complete Subtree algorithm and the Subset Difference algorithm from [DNL01], and [Qui07] was used as additional reading.

In many cases, a user does not want to be connected to the broadcast network at all times. In the example of digital TV, the user must still be able to watch the channels he subscribed for after the receiver has been turned off during the night or unplugged from the network for a period of time. This means that the broadcast center cannot send information about new keys when a user is added or removed. All information needed to decrypt the message (given the information in the current broadcast) has to be stored in the receivers from start. We call these receivers stateless. For example at a pay-tv station, a user usually receives a smartcard containing the decryption keys. This is why a user does not have the capacity to store a large amount of keys. At the same time, many of the applications for broadcast encryption require the receivers to store a large number of keys. Fortunately, there exist algorithms that cope with this problem. How this is done will be explained below.

These Subset-Cover algorithms, as the name already suggests, makes use of subsets and a cover. The previous algorithm explained in section 8.2.3 does some sort of grouping of users, it creates keys for subsets of users. Also in subset-cover algorithms we make use of groupings; a user can belong to several subsets. There is also a cover, which is the set of all subsets that contain all privileged users and no revoked ones.

In the Subset-Cover algorithms, each subset is given their own secret key. To do this efficiently, we chose a tree structure, in which every subset is a node in the tree and each user is a leaf. See for an example figure 8.2 which shows the tree structure with users L1 to L8, where L5 and L6 are revoked. In this example the cover consists of the subsets N2 and N7. Because of the tree structure every user has to store $\log n$ keys.

Two Subset-Cover algorithms will be explained in this section: Complete Subtree and Subset Difference. These algorithms are said to be r-flexible, and from this it follows that the storage size at the receiver is not a function of the number of revoked users r. Receivers do not have to be updated about keys after changes in user rights have taken place.

8.3.1 The Complete Subtree method

The Complete Subtree algorithm consists, just like the Subset Difference algorithm, of three phases, namely initialization, encryption and decryption. In the initialization step, the center creates a key for every subset and sends these secret keys to the receivers (one for each subset the user belongs to) through a private channel.

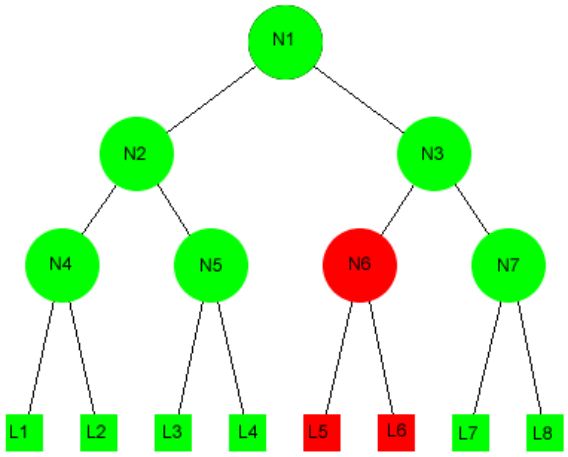


Figure 8.2: Tree-structure. Revoked users are marked red (L5 and L6). In this example, the cover is {subset N2, subset N7}.

For the encryption of the actual message, the center creates a random session key and encrypts the message with it. This could for example be done using a stream cipher encoding, because of the size of the message. Then the session key is encrypted with all secret keys of the subsets that are part of the cover, i.e. the privileged subset. These encrypted session keys are stored in the header part of the message, together with an index. Then the message is broadcast.

When a user receives the encrypted message, he starts searching the header for the encrypted session key corresponding to a subset he belongs to. Once he has found one, the user knows he is privileged and he uses his secret key to decrypt the encrypted session key. With this key he can now decrypt the complete message.

One way to check if the user belongs to a subset is to trace the path from the user to the root, and check if any of the subsets along the path are in the header. Another more efficient way is to represent the index by a binary bitset, where a left node is represented by a 0 and a right node is represented by a 1. For example the subset with index 9 would be represented by 010. To check if a user belongs to a subset, simply check if the subset bitset is a suffix in the user bitset.

8.3.2 Subset Difference method

In section 8.3.1 the Complete Subtree method is explained. This method requires the user to receive key information of $r \log \frac{N}{r}$ per message, where N is the total number of users in the system and r is the number of revoked users. In this section we will discuss the subset difference method that only requires $2r - 1$ of extra key information per message.

The subset difference method uses subsets of the tree just like the complete subtree

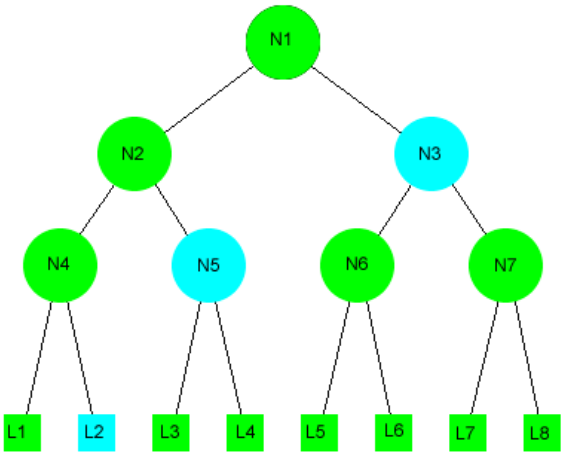


Figure 8.3: The nodes marked in blue are the nodes that hang-off the path from the root to user 1.

method. In this method however a subset S_x corresponds to a group G_1 minus another group G_2 where $G_2 \subset G_1$. These groups correspond to subtrees in a full binary tree such that subset S_x of the tree corresponds to two nodes in the (V_i, V_j) where V_j is in the subtree of V_i and V_j contains the nodes in $\mathcal{R}$. In figure 8.2 an example of such a subset is shown. The presented subset corresponding to this picture is (V_1, V_6) or $S_{1,6}$.

As in the complete subtree method when we want to send a message to all users $u \in N \backslash \mathcal{R}$ we find a group of subsets st. $S_1 \dots S_m = N \backslash \mathcal{R}$. The number of subsets found is at most $2r - 1$ which is an improvement over the $r \log \frac{N}{r}$. A problem however is the number of keys a user needs to store. When we do this in the same way as we did with the complete subtree method we would have to store $O(N)$ keys. For storing the keys within the complete subtree method we used the information-theoretic approach. For optimizing the number of keys stored at each user we use a computational technique involving a pseudo random generator that generates pseudo-random keys from information from the label of its ancestor. Using such a technique requires the user to store only $\frac{1}{2} \log^2 N + \frac{1}{2} \log N + 1$ keys.

To compute keys all nodes in the tree (including the users) are assigned a random label. On this label one can apply a hash function . This hash function should be non-reversible and give a result that is three times as long as it's input. At a player all interim-keys are stored that are on the nodes that 'hang-off' the path from the root to the node itself, for each of the subtrees the player belongs too. Hang-off the path we mean the first node that branches-off to a subtree that doesn't contain the user, see picture 8.3 for an illustration of nodes that 'hang-off'.

An interim key for a node in a subtree is computed by taking the hash of the label from the root of that subtree. Then depending on whether we are on the right or left of that node we take the left third of the hash or the right third of the hash, we hash the

Casus 8.4: Application of subset difference method

An application of the subset difference method is the Advanced Access Content System (AACS) system found in Blu-Ray and HD-DVD players. The content of an AACS protected disc is encrypted using a media-key. This media key is available in encrypted form for a large number of device-groups. All of the devices store a large number of interim keys locally. When a disc is inserted the players finds the encrypted version of the disc-key and computes the right decryption key from its interim keys. If an interim key leaks the providers of Blu-Ray and HD-DVD discs simply exclude all users that use that particular key. This doesn't make the players containing the leaked worthless because they contain multiple keys. Only when all keys in a device are compromised a player becomes worthless and can't decrypt any new films.
The AACS system is rumoured to be hacked, but in fact only the device key of the program Win-DVD was leaked due to a flaw in the software. Those keys are now excluded from use and new Blu-Ray and AACS discs are still very well protected.

result of this and carry on doing so until we arrive at the node we which to compute this intermediate key for. From an interim key we can compute a key for a subset by taking the interim key from the root of the subset and then compute hashes in the same way as before travelling towards the root of the excluded subset. From the hash of the root of the excluded subset we take the middle third part and hash this to retrieve the key for the subgroup. As none of the users has a interim key of its ancestors they can never compute a key for a subgroup they are excluded from.

8.4 Free riders

8.4.1 What are free riders?

Ideally we don't want people that are not meant to see some data, not to be able to see this data. To achieve this encryption is most often used. In broadcast encryption schemes it is also common to prevent unauthorized people from seeing this data. In some cases this actually mostly desirable, for instance when military, financial or other highly classified data is broadcasted we don't want any person that is not authorized to see the data. But sometimes it is not directly a tremendous problem when unauthorized people can see the data, for instance when broadcasting commercial television it is acceptable that 1 on every 10000 viewers is not actually authorized but is still capable of decrypting the data, especially when allowing such users brings performance gains or financial profits. This section will discuss some of topics from [RW06].

8.4.2 Why allow free riders?

As mentioned in the previous paragraph we might consider allowing free riders if doing so gives a performance boost or financial profits. In practice it appears that allowing free riders can give a better performance and possibly as a result of the gain give some financial profit. The performance gain is the result of less communication when determining what broadcast key to use. In algorithms as described in section 8.3 the information used to determine what key will be used is of $O(r \log \frac{N}{r})$ respectively $O(2r-1)$. When we allow free riders we can improve on this. Unfortunately it appeared that an optimal assignment of free riders is NP-complete.

8.4.3 How does it work

The principle of allowing free riders is rather complex and worth a paper in its own right. Instead here we'll discuss what the idea behind allowing free riders is. It appears that allowing free riders can be applied on both the Complete Subtree algorithm as discussed in 8.3.1 and on the Subset Difference algorithm discussed in 8.3.2. The basic idea is that allowing f free riders in a system of n users can lower the amount of communication needed to be done to let everyone know what the session key is. For instance suppose we have a tree of 8 users and only 1 user is excluded (and we'll use the complete subset algorithm). We will now have to devide the tree in three subtrees to mark all the allowd users. So we now have to communicate 3 times as much data as when we would have allowed one free rider. In practice, with larger amounts of users, a larger amount of revoked users and more freeriders results might be more significant as in the example mentioned earlier. A complete overview and discussion on how the free riders algorithms work can be found in [RW06].

Chapter 9

Software Implementation of Cryptographic Algorithms

Written by *Marco Devesas Campos, Jose Fernandes.*

9.1 Introduction

Computers have invaded the homes of countless people throughout the world. Starting with Xerox Alto in 1973 and gaining a huge boost in popularity with the introduction of the IBM PC, personal computers have transformed themselves from luxury to a convenient tool and, more recently, to an indispensable part of everyone's life. Fueling their capabilities were General Purpose Processors (from now on referred simply as "processors"). These processors are capable of performing any type of calculation and thus enable computers to perform any task that users wish them to. A multitude of software applications have been implemented. Word processors offer a more convenient way to compose letters than the typewriters that company-secretaries used in the old days. Spreadsheets help households keeping the balance of their bank accounts. E-mails have replaced, both for business and private users, letters as a communication medium. And internet-access is widely seen as a basic necessity of everybody today.

Apart from their flexibility, the main reason behind the success of PCs was the immense power that they gained throughout the years. Moore's law states that the number of transistors in a processor doubles every year and a half. This exponential growth in complexity and power as meant that PCs are capable of doing calculus in real-time of things that in the past were only possible using super-computers or using dedicated hardware. Today's computers are capable of playing films at high-definition and performing the massive amount of calculations needed to play 3D-games in real-time.

Another type of machine that has gained a part in our everyday life is the Cellphone. From huge bricks, that had to be carried around in suitcases, to small plastic fashion accessories that can be carried in one's pocket or around the neck, cellphones have become more and more ubiquitous. Two advances made this possible. Batteries have become smaller and have bigger capacities. Processors, helped by Moore's Law, have become smaller, more potent while consume less and less power. This has meant that applications that worked only in computers are now available to everyone with a powerful enough cellphone. It is not uncommon for cellphone makers to advertise the ease of use and the practicality of having e-mail applications or web-browsers in their models.

All this has meant that more and more sensitive information passes through such systems. Personal e-mails may contain information that sender and receiver want to remain private. E-Banking allows bank customers to access and use their accounts from the comfort of home or just about anywhere they like, if they access the service by cellphone. This implies that confidential information such as account numbers, credit card numbers and pins has to be transferred through the internet. For businesses, security is even tighter. Workers on the road may carry trade secrets which must be protected against loss or steal of the system where they are stored. Secrecy may even be enforced within companies to guarantee that only the senior responsibles have access to confidential information.

Cryptographic algorithms are a necessity of modern society and must be available to the general public. The security of their information depends on it. Furthermore, it is not desirable that everybody has to carry around a special "encryption/decryption" device, so those algorithms should be made available for usage in their existing computing devices. We know this to be possible because modern computers are run by processors that can do any sort of calculation including, obviously, all the ones needed by cryptographic algorithms.

But, given their general nature, modern processors are not "tuned" to perform as fast as possible the cryptographic functions. The efficiency of the implementation is a fundamental issue. E.g. the IPv6 protocol, which is scheduled to replace the ubiquitous IPv4 [1], explicity mentions the possibility of cyphering *every* packet that leaves a computer. This means encrypting up to 512KB of data per second in a typical 4 Mb connection[2]. If the algorithms involved in encrypting all this data are not efficient then the encryption stage becomes a bottleneck of the communication process. This would mean that the user would never be capable of fully utilizing his network connection and that he was paying for bandwidth that simply did not use use. At the same time, the processor load induced by the calculus of the cryptographic functions would impair the processing of applications that are running concurrently and the user would notice that his system is running slow.

Ergo, there is a cost in adding security to the system. In systems with limited resources to spare, such as cellphones, this cost grows quickly for even small increases in processing load. If the cost becomes too big, users might avoid using any security at all and their data might become compromised. If we want to enforce security in a system

[1] As it has been for a few years now...

[2] We are omitting some details here such as the headers of the other protocols involved in the communication

then we must be sure that this will not collide with the users' other interests. Security must be enforced as transparently as possible and requires that implementations utilize every trick available to run as efficiently as possible.

But this cannot be done blindly. When cryptographic algorithms are run in a modern computer, they may have to share resources with other applications. Applications running at the "same" time share, not only, processor time, but also memory and cache. This leaks information which can result in a security problem. An attacker can run a program at the same time as another user is running an encryption and use these shared resources to gain insight that will help him find the latter's keys.

In this chapter, we will address the issue of implementation of cryptographic algorithms in hardware platforms commonly available to everyone. We start by providing an overview of how modern processors work. We proceed by showing two design techniques aimed at improving the performance of algorithms. We explain these techniques in the context of the implementation of two cryptographic algorithms, AES and DES. We then move on to discuss the potential security issues of software implementations. In concrete, we show how side-channels can be constructed to attack the security of the algorithms. Consequently, we discuss the basis of such attacks and show how they can be used, in practice, to retrieve the keys used in AES and RSA encryptions and decryptions.

9.2 Anatomy of a modern processor

If we want to make good use of the processor's capabilities, we must first comprehend how they are implemented. This section shall shine some light of inner-workings of a processor. We will take special care with the features that help speed up the execution of programs. In later sections, we will see how these features combine to make the execution of cryptographic algorithms really efficient.

9.2.1 Overview of Processor Architecture

Internally, instructions are nothing more than sequences, perhaps of variable length, of bits. The first step for processing an instructions is then to interpret those bits and, according to the instruction they encode, command the processor's units to do what the instruction represents. This is done by a unit called the *Instruction Decoder*.

All of the data that is processed must be stored in special memory areas, contained inside the processor, called *Registers*. This is because the functional units of the processor expect their input and output values to be stored there. The number of bits in a register defines the *word size* of the processor. There are two types of registers: those which are visible to the programmer and those which are not. The not-visible ones can only be modified by the processor as a side-effect of instruction execution. Here, we will only consider register who are visible, and can therefore be explicitly modified by the program. Registers are a serious component of the performance of a processor because

they do loads and stores very fast – much faster than normal memories – but they are a very scarce good.

Processing is nothing more than doing some transformation on the input bits. Processors are equipped with two units which provide such transformations. *Integer Units* (also known as *Arithmetic and Logic Units*) regard input bits as binary numbers and perform the usual operations on them: sums, multiplications, division, etc. They are also capable of doing logic operations such as complementation and conjunction and exclusive-or. *Floating Point Units*, on the other hand, regard the bits stored in registers as being rational numbers in a given format (such as IEEE's 754 Standard). Cryptographic techniques usually deal with either logic or integer operations and so, FPU's are not within the scope of this chapter.

Instructions, commonly, operate over input coming from the outside, e.g. what the user typed on the keyboard or packets coming from the network. This data is stored in Main Memory (usually called RAM) but to be used it must be first, as explained above, be loaded into to registers. When there are not enough registers to hold everything need by a program (a situation called *Register Starvation*), data that is not currently being used has to be stored temporarily in memory. To communicate with memory, the processor has a *Memory Interface* that takes care of all loading and storing of data.

Processors, depending on their target market, come with other operational units not mentioned here, e.g. Streaming SIMD Extensions (SSE) and Altivec. Some might even allow cryptographic algorithms to run faster. But this would imply that programs would be written in a non-portable way. In this exposition, we are interested in general techniques which allow for speedups in many cryptographic algorithms under different yet common architectures.

9.2.2 Performance Improvements

The model we presented in 9.2.1 lends itself to some modifications that can lead up to good performance gains in processing. These modifications are all transparent from the programs point of view. They do not require programs to be modified in order to take advantage of them. But to use them fully, some issues must be taken into account.

From the description above we can see that the units that make up a processor are fairly independent of each other. Granted, the integer unit operation is dependent on the output of the instruction decoder. But, while the integer unit is doing some operation, the instruction decoder could be decoding the next instruction and the memory interface could occupy itself with writing the result of the previous operation to memory. This is *Pipelining*. During normal functioning, all the instructions in the pipeline belong to the same thread. Hyper-Threading removes this limitation, and allows the pipeline to be busy with instructions of several threads.

It is easy to spot a few problems with pipelining. What if an instruction depends on a value that is still being obtained further ahead in the pipeline? The processor is capable of determining such *instruction dependencies* and guarantees that an instruction only runs when all the dependencies are satisfied. In extreme cases, the processor will insert `noop`s into the pipeline, which, in a way, defeats the purpose of the pipeline. Another problem are conditional branches. If the next instruction is depending on some result,

how can the processor know what instruction to load next? The answer is: it guesses. If the guess was correct then all is well and the processor can carry on as if nothing happened; if the guess was wrong, the processor must discard all the computations it did and start all over again from the correct instruction. This is a waste of time and resources. And conditional branches should be kept to a minimum and respect the heuristic behind the processor's guesses.

Even though memories are fundamental for the functioning of a computer, their speeds have not seen an improvement as big as the one of processors. In fact, they lag behind so much, that memory accesses take up several processor cycles. This aggravates even more the problem of dependencies between instructions. To alleviate the problem processors started to include faster memories that store the most used memory positions and have them available, when necessary, more quickly than main memory. These are called *cache memories*, or just caches. Due to their speed, cache memories are also very expensive and therefore, as registers, scarce. Processors have usually multiple caches structured in a hierarchy of access. The bigger the cache size, the slower they are. To the programmer, though, caches are completely invisible (the word cache comes from the French for *hidden*).

Because they are smaller, it is common for caches to become full and have to start removing some of its contents to make way for the new data. If the evicted data is needed again, it must be retrieved again from the upper levels which is a time-consuming operation. This is called a *cache miss* and, to prevent them, there are a few principles that should be taken into account:

Temporal Locality Recently accessed memory positions are bound to be accessed again soon

Spatial Locality Positions near recently accessed positions are likely to be accessed again soon

Sequential Locality Positions are accessed in sequential order (ascending or descending)

Caches are shared among processes that are running simultaneously, even though different processes have different memory spaces. In 9.4.1 we discuss the security implications of this.

9.3 Techniques for Efficient Implementations

9.3.1 Bit-slicing DES

It has been common practice, in the cryptographic circles, to represent algorithms using their hardware description[3]. In the old days, cryptography was implemented using specialized hardware. Case in point: the Enigma machine. The analysis of cryptographic

[3]Electrical (logic gates, relays) and/or mechanical(rotors)

algorithms was then closely connected with the analysis of electric circuits. Nowadays, we are used to see things as S-Boxes and Matrix operations. But every such operation is usually accompanied of the logic circuit that implements it. This is true for the (dying?) king of block cyphers: The Data Encryption Standard (DES).

DES is a Feistel network algorithm([Sti05]). It is composed of several Feistel boxes, connected to each other, and an initial and a final permutation of bits. Each Feistel box (which implements a Feistel Function) is composed by several stages. The first, expansion, expands the number of bits of the input from 32 to 48. The resulting bit string is then `xor`'ed with the rounds sub-key. Finally, the bits pass through 8 S-boxes and a permutation network. In total, there are about 100 logic gates in a DES' Feistel Box.

This raises a problem: how to compute efficiently the logic circuit of the Feistel boxes. The answer is *Bit-Slicing*. The key insight in bit-slicing is that when performing a logic operation on a current processor, with n-bit registers, there are n one-bit, parallel, operations being done. If we dedicate a register to every bit of the Feistel function and execute its logic expression, we can perform the encryption of n blocks *at the same time*. As simple example, to better show the workings of bit-slicing, consider a function $F(x_1, x_2)$ on two bits. If we wish to apply it, simultaneously, to $a = a_1a_2$, $b = b_1b_2$ and $c = c_1c_2$, we must load all the first bits(a_1, b_1, c_1) into register r_1, all the second bits (a_2, b_2, c_2) into register r_2 and run the instructions of F with $x_1 = r_1$ and $x_2 = r_2$. The result will be stored in register $r_F = (F(a), F(b), F(c))$. Adapting DES to this technique results in a similar process. Let us take the example where the register size $n = 64$. We start by loading 64 blocks. We then put all the first bits in one register, all the second bits in another register, etc. We then apply the logic function of the Feistel Box. The result will be stored in 64 registers. In one are all the first bits of the resulting encrypted blocks, in another all the second bits, etc. To obtain the final result, simply put, the bits in their correct order.

There are a few issues with this solution. First is the non-standard representation of blocks. This forces a conversion between block formats that represents a overhead in processing. The implementation explained in [Bih97] required 2500 extra instruction to perform conversion. Also, the register size must be big enough to allow the effects of the obtained parallelization to be big enough to make the process run within acceptable performance figures. This method is very register-intensive, so the number of registers is an important issue. The more the merrier. This allows more data to be stored inside the processor and avoids having to use main memory as temporary storage which slows down, considerably, the whole processing. Finally, it is also possible to parallelize the processing of logic gates. For instance if one gate only needs bits 1 and 3, and another bits 2 and 4 then both computations can be executed in parallel. To do this, it is necessary to have in the processor multiple units capable of performing logic operations.

9.3.2 Table Lookups – The AES example

One, very common, performance optimization in software is to use tables to compute the result of a certain function. First, it is necessary to pre-compute the values for all possible inputs of the function. This table is then loaded into memory whenever the

program is run. To compute the result of the function, simply access the table with the inputs as indices to the desired value. One notable usage for this technique is in the implementation, in a processor, of some complicated mathematical functions, such as divisions and trigonometric functions. The table stores some pre-computed values that are then combined to calculate the result for the given input. An error in one such table was the responsible for the famed bug in Pentium processors ([Hal95]).

We will illustrate the use of this technique in cryptographic algorithms by describing an implementation of the Advanced Encryption Standard (AES) algorithm. AES is the result of an international competition held by the american National Institute of Standards and Technology to replace the (insecure) Data Encryption Standard (DES). The process lasted for a few years, and spread trough a few rounds. All of the final competitors still standing in the final round were deemed secure. The final choice, though, fell over the Rijndael algorithm ([DR]) proposed by the belgian cryptographers Joan Daemen and Vincent Rijmen. One of the reasons behind the choice was the easiness with which Rijndael can be implemented in both hardware and software.

Like DES, it is composed by several rounds. Unlike DES, these rounds are not based on Feistel-boxes. The key size may vary (128, 192 or 256 bits) and, depending on it, so do the number of rounds. Here we will treat the 128 bit case which has 10 rounds. The key is passed to a key scheduler which generates, from it, the keys used in each round. For every round i, the key (K_i) and the output of the previous round are combined, using the operations `SubBytes`, `ShiftRows`, `MixColumns` and `AddRoundKey` (`MixColumns` is skipped in the final round), to yield the state after that round (x_i). The initial state is the `xor` of the plaintext and the key. The cyphertext is the output of the final round.

We might consider, as a first approach, to use a lookup table for every possible combination of key and previous state to compute the round function efficiently. Unfortunately, both inputs, key and state, are 128 bits, as is the output state. This means that the table would need to hold $2^{128} \times 2^{128} \times 128 = 2^{263}$ bits $= 2^{230}$ GBytes. [DR] gives us a better way to do this. The idea is to use four (T_0,T_1,T_2 and T_3) pre-computed tables to calculate the output of each of the inner rounds. Each table is composed of 256 32-bit entries and, thus, they occupy 4KBytes in total. Both input and output states are split into 16 8-bit groups ($x[1]x[2]..x[15]$) and the round key is divided into 4 32-bit groups ($K[0]K[1]K[2]K[3]$). The relationship between inputs and output is given by the following equations

$$\begin{aligned}
(x_{i+1}[0]x_{i+1}[1]x_{i+1}[2]x_{i+1}[3]) &\leftarrow T_0[x_i[0]] \oplus T_1[x_i[5]] \oplus \\
&\quad T_2[x_i[10]] \oplus T_3[x_i[15]] \oplus K_{i+1}[0] \\
(x_{i+1}[4]x_{i+1}[5]x_{i+1}[6]x_{i+1}[7]) &\leftarrow T_0[x_i[4]] \oplus T_1[x_i[9]] \oplus \\
&\quad T_2[x_i[14]] \oplus T_3[x_i[3]] \oplus K_{i+1}[1] \\
(x_{i+1}[8]x_{i+1}[9]x_{i+1}[10]x_{i+1}[11]) &\leftarrow T_0[x_i[8]] \oplus T_1[x_i[13]] \oplus \\
&\quad T_2[x_i[2]] \oplus T_3[x_i[7]] \oplus K_{i+1}[2] \\
(x_{i+1}[12]x_{i+1}[13]x_{i+1}[14]x_{i+1}[15]) &\leftarrow T_0[x_i[12]] \oplus T_1[x_i[1]] \oplus \\
&\quad T_2[x_i[6]] \oplus T_3[x_i[11]] \oplus K_{i+1}[3]
\end{aligned} \tag{9.1}$$

For the final round there are, similarly, four tables that permit the obtention of the cyphertex efficiently.

9.4 Side-channels

When analyzing cryptographic algorithms and protocols from a mathematical perspective, there is usually the implicit assumption that the parties involved only reveal information to others when they want to. In real life, however, that is seldom the case. Imagine that you want to say something to a friend, but nearby is one enemy that you want to prevent to ear what you are saying. Your instinct will probably make you whisper the secret instead of saying it out loud. But your enemy might know how to read lips, so if he can see you talking to your friend, he will find out what you want to say. Even if have your mouth covered, by simply looking at your expressions while you talk, your enemy can see if your are sad, happy or angry and from there discover, at least, the topic of the conversation. In this situation, your face is said to form a side-channel of your main communication channel (your mouth).

The same happens with every cryptographic machine that you can think of. All machines use electric power to fuel its functioning. Internally, all data is represented by binary numbers. The binary digits are represented using electric signals, e.g. $0V$ for digit 0 and $5V$ for digit 1. So, when preforming a calculation, the power-consumption of the machine will depend on the number of 0s and 1s in the system. If someone has access to the power-consuption, he may be able to rule out some of the possible inputs to the machine because they would require the machine to use significantly more or significantly less power than used.

If an attacker wants to further reduce the number of possible inputs to the machine, he can turn to several other "hints" that the machine provides. Let's imagine that the machine we are talking about performs RSA decryptions. RSA decryption of a message x^e, with a private key (d, n), consists of calculating

$$D_{(d,n)}(x^e) = (x^e)^d \ mod \ n = x \tag{9.2}$$

The machine is known to perform exponentiation according to the following recursive formula

$$x^e \ mod \ n = \begin{cases} 1 & \text{if } e = 0 \\ x^{e/2}.x^{e/2} \ mod \ n & \text{if } e \text{ is even} \\ x^{\lfloor e/2 \rfloor}.x^{\lfloor e/2 \rfloor}.x \ mod \ n & \text{if } e \text{ is odd} \end{cases} \tag{9.3}$$

If e is represented by a binary number then $\lfloor e/2 \rfloor$ corresponds to the shift right operation. Checking if a binary number is even or odd is just verifying if the least significant bit is 0 or 1, respectively. This means that the running time of an exponentiation depends only in the bits of the exponent and the time it takes to perform a multiplication. The total time is given by

$$\text{Time} = \text{MultTime} \times (3 \times \text{nr of 1's in exponent} + 2 \times \text{nr of 0's in exponent}) \tag{9.4}$$

If the attacker knows the time it takes by the machine to perform a multiplication and the total time it takes to perform the decryption, he can infer a minimum and a maximum number of 1s in the private key d. This may allow him to perform brute-force attacks by reducing the number of possible keys that he has to test.

Due to security risk that they impose, side-channels have been studied extensively. Other types of side-channels include van Eck radiation(which uses the electromagnetic signals generated by the machine when doing computations) and acoustic cryptanalysis (which relies on the sounds generated by the functioning of the machine).

9.4.1 Cache Attacks

In the rest of this section, we will explore another type side-channel that leaks information through the cache. One of the most significant parts of any modern processor, when it comes to performance, is the cache(s). Cache memory permits the processor to access more quickly the (hopefully) more relevant parts of the memory for execution. This is done by maintaining in a copy of the said relevant parts in a faster memory array than the main memory.

Because they are so fast, the cost of manufacturing cache memories is also very high. Modern processors have caches that are only a fraction of the size of main memories. This means that whenever a word in memory needs to be accessed the probability of not being in the cache (a *cache miss*) is very high. To reduce the probability of it happening several techniques are employed by processors. Here we will consider the case of *m-way set associative* caches, because it is the most commonly found type.

A common memory access pattern involves accessing, sequentially, adjacent memory words. For example, when scanning an array, a program typically accesses the first position of the array, then the second, then the third, etc... A similar pattern occurs while reading the instructions that compose a program. Typically, nearby memory accesses in time tend to be to positions whose addresses are also nearby to each other. This is called the *Principle of Locality*. Caches explore this principle to reduce cache misses by grouping adjacent memory positions into blocks with the same size as as *cache lines*. When one memory position from a block must be read from memory, the entire block is transfered to a cache line. Depending on its address, blocks are mapped to different cache lines. Blocks that are mapped to the same cache lines are said to belong to same *cache set*. For each cache set there are several line in the cache where a block from a set are mapped to. No two cache sets share a cache line. The number of lines in a cache line is called the associativity of the cache and is the m in "m-way".

9.4.2 Synchronous Attacks

Recall the general operation of the AES cypher. In each round, the new state is computed by accessing the records of four tables T_0 to T_3. Whenever the algorithm reads one position from a table, the relevant memory block is transfered to the cache and remains there until some other block, belonging to the same set is needed. The positions of the tables that are accessed depend only on the plaintext and the encryption key. This is specially easy to see in the first round of AES. The initial state is simply the xor of the key and plaintext.

The strategy for the attack is the following: run an encryption with known plain-text; check which memory blocks were used in the first round; calculate the keys that make the algorithm access those memory positions. Repeat this several times with different

plaintexts until the right key is found.

The big question in this strategy is how to detect which blocks were used in encryption. This is done by timing memory accesses. One simple way to know if a table entry was used or not during encryption is to read it after the encryption happened. If the operation goes really fast then it was already in the cache; if not, the table entries were not used in the computation and they had to be read from memory. This is not possible in practice: the operating system provides each application with its own addressing space. This way, one running application cannot interfere with another one by writing on the latter's address space. This technique is part of *Virtual Memory*.

One solution is to have the spy – the attacker's program – do reads and writes in its own address space but in addresses that belong to the same cache set as the memory addresses of the AES tables. Before the encryption happens, the spy program fills the cache with memory positions of its own address space. When the encryption runs, some of those positions will have to be evicted from the cache because the algorithm needed to read values from the tables. After the encryption algorithm runs, the spy program reads from memory, the same positions as before. If for some position the operation takes too long then it means that some table entry belonging to the same cache set was used during encryption. Using some address algebra it is not difficult to find out which table entries might have been responsible for that. In [OST06] this approach is called *Prime+Probe*.

Another approach begins by running the encryption first. This ensures (modulo cache collisions due to the disposition of the tables in memory) that all. We then repeat, for the same plaintext, the encryption and time it. Then to test if an entry of the tables was accessed we simply read m (were m is the associativity of cache) different blocks of memory that belong to the same cache set as the target table entry. We therefore know for sure that that entry is not present in the cache. If we repeat the encryption, one of two things can happen with the execution time: if it remains the same then target table entry was not used be the encryption; otherwise, the time increases and that can only be because the entry, or more exactly, one the entries that are on the same block as the target entry, was needed by the algorithm and had to be fetched from memory into the cache. This alternative is also explored in [OST06] and, unsurprisingly, it was shown to have worse performance than the previous strategy.

The descriptions above assume some highly idealized circumstances. For one, we assume that there are only two processes running; in real life, every system runs several programs at any given time, some of each are not even explicitly ran by any user, e.g. cron jobs and system daemons. While running, these programs access memory and cache and may cause memory blocks of the tables to be evicted. This introduces noise in the measurements which makes detecting which blocks were used in an encryption even harder. Nonetheless, it is also possible that the these processes do not touch the cache in a way that compromises the measurements, and in cryptography it is standard practice to take special care with the worst case scenarios.

A second assumption is that all encryptions are triggered explicitly by the spy program. This may seem unrealistic but there is at least one real-life scenario to which it applies. The Linux `dm-crypt` module allows all the contents of a filesystem to be encrypted before being written to disk. The filesystem can be written and read by

anyone as long as they have the proper permissions to do it. The keys are hidden by the module. If the user that runs the spy program has write-access to one file in the encrypted filesystem, then he can use `write`'s to that file to trigger the encryptions with know plaintext. As we have shown, the spy program can therefore retrieve the keys, which the user can use to have access to the remaining contents of the filesystem.

The third, and final, observation is related with the memory addresses of the tables. Even though the tables are constant, their size and contents do not change, their placement in memory is defined by several factors: the compiler is responsible to define where in the code the tables come; the Operating System uses this information, stored in the program's binary, to decide what is the table's address in the virtual memory of the system and this in turn depends on many other external factors. Without this information, there is simply no way to discover which entries of each table were accessed and no way to correlate the information we get from the measurements with the actual key used. The authors of [OST06] claim to have solved this problem, but have not made any mention to the fundaments of their solution.

9.4.3 Asynchronous attacks

The attacks explained in the previous section are part of the class of synchronous attacks. They rely on the explicit triggering of encryptions (or decryptions) by the spy process. It is also possible to have the spy process running continuously and do the measurements when it concludes that an encryption is taking place. On Unix systems we can detect that an encryption is taking place by looking at the output of a command such as `ps aux | grep "openssl"` (or any other program that runs encryption). As soon as it is detected, the spy program starts to monitor the memory accesses of the victim program.

Unlike the previous attacks, in asynchronous attacks, the spy program does not have any access to the plaintext nor to the cyphertext of the encryption/decryption. To steal the key, we have to detect memory accesses that are only dependent on the key. This means that, under the right conditions, we are in the presence of a very powerful attack, stronger than KPA, CPA, CCA and KCA, because we have access to none of the information of the algorithm. RSA implementations have been shown to be vulnerable to this attack ([Per05]).

The RSA crypto-system is a public key crypto-system based on the idea of *inverse exponents*[4]. To encrypt a message x, we simply raise x to the public exponent e (modulo a public n) of the receiver, and get cyphertext $y = x^e$. To decrypt, the receiver of the message computes y^d, modulo n. Because e and d are inverse exponents, $x^{e.d} = x^1 = x$ and he has, therefore, access to the message x. To do the exponentiations efficiently, the algorithm pre-computes a table. The accesses made to this table during exponentiation are dependent only on the private exponent d. By tracking those accesses, it is possible to infer d, or at least reduce the number of possible keys to a value low enough to permit using other attacks.

The attack we present here was first introduced in [Per05]. In this paper, the author presents the applicability of the attack on systems with processors with the Hyper-

[4] Our name

Threading Technology (HTT). Pipelining permits performance gains by allowing several instructions to be run at the same. The only limitations are that no two instructions can use the same part of the processor at the same time and special care must be taken with the dependencies between instructions (no instruction can use the value of a register if a previous instruction, that changes the value of that register, hasn't finished computing). HTT extends pipelining, allowing two threads to use the processor simultaneously. To accomplish this, only certain parts of the processor's logic need to be duplicated. But most resources, such as the arithmetic units of the processor, the registers and caches, are simply shared during execution. This allows further performance improvements, at the cost of only a slightly bigger *die area*. To the operating system, the processor appears as a normal dual-cpu. The attack explores the fact that the spy and the victim program (can[5]) run, for all intents and purposes, at the same time and a real-time profiling of the memory usage patterns can be done.

To maximize efficiency, RSA implementations[6] use two techniques when performing exponentiation. First, to reduce the overall complexity, instead of one exponentiation modulo n, two exponentiations modulo p and q (with p and q prime and $n = pq$) are performed. This is a result of the Chinese Remainder Theorem. Second, instead of equation 9.3, a *sliding window* approach is used. Assume we are calculating $y = a^e \ mod \ p$. The main idea, as presented in [Koc95], is to re-write e as a sequence of adjacent groups of bits with at most m bits. Also, every such group must represent an odd number or a zero. For example, for $m = 3$ and $e = 3665_{10} = 111001010001_2$ we could have the groups $e = (111)(00)(101)(0)(001)$. We then process these groups from left-to-right(MSB to LSB) to find the value of the exponentiation. We first pre-compute the table $A = \{(0, 1), (1, a^1), (3, a^3), \ldots, (2^m - 1, a^{2^m-1})\}$, modulo p. The reason why we only compute this set for odd exponents and zero is that our groups are only odd numbers or zero. Initially, we set $y = 1$. We then process each group g iteratively, from left to right, setting $y = y^{(2^{sb(g)})} \times a^g$ (all operations with the appropriate modulo), where $sb(g)$ is the size in bits of g. The value of a^g is pre-computed and this can be implemented using table lookup $A[g]$. One further optimization is to compute $y^{(2^{sb(g)})}$ as a sequence of squarings, because squaring can be done more efficiently than ordinary multiplication.

The accesses to the table A depend only on the bits of the exponent, which in RSA decryption is the private key. We can therefore analyze the cache accesses of the decryption to discover, at least, some of the bits of d. According to [Per05], OpenSSL implementation of squaring also makes use of extensive the cache which is suitable to cryptanalysis. Using the combined effects of these two sources of information, about 310 of a 512-bit private key can be obtained.

---

[5]We assume, again, the best case scenario for the attacker

[6]We are thinking OpenSSL

9.5 Summary and Conclusions

Information security is a latent necessity of our modern society. This requires that everyone has access to cryptographic techniques capable of protecting their information from unwanted eyes. Furthermore, cryptographic algorithms should be incorporated as transparently as possible in people's workflows. This implies that software implementations should be as efficient as possible, so as to interfere as little as possible with the functioning of other programs. To achieve this, the right implementation techniques must be chosen. Bit-slicing and table lookups are two such techniques.

We must also be careful with these implementations. If they are not well thought off, side-channels may be created that permit attackers to gain information that they are not supposed to. In particular, caches leak a lot of information about the functioning of programs. Attacks using such side-channels have been showed to be feasible in practice against otherwise secure algorithms such AES and RSA.

Chapter 10

Reputation-Based Trust

Written by *Wesley van Beelen.*

Reputation is the general estimation in which someone or something is held by others. It can be used in markets as feedback for potential buyers on how trustworthy the seller is.
A reputation system is a system where a buyer can give a satisfaction rating for the seller. The reputation rating gives future buyers additional information to help them decide from which party to buy something. Examples of auction websites that use reputation systems are ebay.com and tweakers.net. On such websites users can give feedback on the salesperson after the user bought something of him which will be used to compute a reputation for the salesperson over all the previous sales he engaged in. Thus the reputation says something about the history of the salesperson.
Having a higher reputation is profitable because people will trust the salesperson more and are willing to pay a higher price for his salesware. This means people will try to attack the reputation system to give them selfs a higher reputation or give others lower reputations. This chapter will first describe which types of attacks exist, followed by a example in which we will give a reputation system which can counter these attacks.

10.1 Reputation system attacks

There are a number of ways to attack a reputation system:

Badmouthing: Badmouthing is done by giving another party a unearned bad feedback, with the purpose of giving this party a worse reputation then he should have.

Ballot stuffing: Ballot stuffing is the other way around. Another party is given unearned positive feedback, with the purpose of giving this party a better reputation. Note that ballot stuffing is not always done to give yourself a higher reputation but could also be done by a group of befriended parties that give each other higher reputations making it harder to detect.

Cybil attack: In a cybil attack a party makes one or more fake identities which could then be used to do ballot stuffing on the original identity or to badmouth your competitors.

Mobile virus attack: A party actively searches for weaknesses in the system to corrupt other parties or take control over them.

We want to design a reputation system that is secure against all these attacks. In human-driven reputation systems, where buyers and sellers are humans like the ones on ebay.com or tweakers.net, these attacks are corrected on a case by case basis by a team of (hired) personnel.
Can we also use a reputation system for an automated distributed environment, where servers offer services for payment and in a distributed way together decide which is the best server to complete the service given certain constraints on price and time demanded by the buyer? This sounds like a difficult task because any of these servers could be malicious and could try to disrupt the system in his favor. In the next section we will show that such a system does exist by offering an example of an automated distributed computation environment which incorporates such a secure reputation system.

10.2 Automated Distributed Computation Environment

First, what is an automated distributed computation environment exactly? It is a virtual market in which competing computing servers offer computation power for payment and they also must be able to decide together which server can do a computation given constraints on cost and computation time by a service requester (a buyer). Because the server are competing, some might be programmed to be malicious to receive more profit. There are a couple of ways a malicious server could act dishonest:

- If a malicious server is selected as the one to do the computation, he could not do the complete computation, but be lazy and return random numbers as solution. In this way he could do his "service" very fast and cheap, but it's not really what the service requester wants..
  This means we need to check if the returned solution is correct. We do not want the service client to do any calculations himself, so this check also has to be done by the servers.

- A server could advertise a very short computation time for computations to get more request, but in reality take more time then advertised. This could be punished financially (the service requester pays a lower price) but the case remains the requester receives the solution later then planned and miss one of his own deadlines.

The solution to these problems is to introduce a reputation system into the environment, which gives each server a reputation based on whether his previous services where satisfactory. Because these reputations have to be stored on the servers them selfs we have to use cryptography techniques to make sure malicious servers don't temper with them.

10.2.1 Assumptions

We will make a few assumptions about the environment:

- We have n average number of computing servers. This is average because more servers might join or server might stop offering their service or they might have crashed.

- We assume c to be the maximum number of faulty servers, which could be servers that are malicious or have crashed. In practice this number is hard to determine, but it's best to make this sufficiently large.

- Let a be the number of input data elements for the computation. A computation is done by iterating one by one over the list of input data elements. The solutions are collected and stored in a list of solutions corresponding to the input data element list.

- We assume each input data element takes even time to compute.

- We have a trusted server that can give secret key shares to each server.

10.2.2 Initialization

Each server gets a secret key share from the trusted server and it's public key. An important property of this secret key share is that only $\mathbf{c+1}$ or more servers can sign anything with the combined secret key shares. However any server can check the correctness of the sign alone using the public key. Each server also stores for all other servers:

- The current reputation of the servers signed with the secret key.
  This means that only c+1 servers can temper with the reputation values, but because of the assumption that c is the maximum number of faulty servers this is impossible.
- Time needed to compute a instruction (like X minutes per instruction).
- Price needed to pay for computing during a certain time length (like X dollars per hour).

The last two are needed to calculate the cost to do a certain computation on a server.

When a new server is added *2c+1* servers will send the above data to the new server. The new server will wait until he received c+1 equal data packages from the servers, in which case we knows that the data is correct. Why? Again because we assumed that only c server could send faulty data, we always still have c+1 servers that are honest. The new server then sends his time per computation and price constants to all the other servers. He could lie about the time per computation that he needs but this would be bad for him on the long run. The reason is that this would be detected after he receives a computation job and he would be given a negative reputation. New servers start at a certain default reputation value and they will receive a secret key share.

10.2.3 The algorithm

The algorithm for the reputation system for this automated distributed computing environment consists of 6 phases. It start with the service client (which from now no we will call Bob) sending in his computation, input data elements and time and price constraints and ends when he receives the correct solutions back.

Phase 1: Witness selection

In the witness selection phase service client Bob randomly selects 2c+1 servers as witnesses. This means at least a majority c+1 of the witnesses are honest (non-faulty) according the faulty server assumption. There is no way to find out which (if any) of these witnesses are malicious, because they might act totally honest until they can take advantage of being malicious.

Phase 2: Computation request

Bob now sends a computation request to all the witnesses he chose, containing:

- The computation itself.
- The input data elements.
- The maximum cost Bob is willing to pay for the computation.
- The time limit in which the computation must be finished.

Phase 3: Distributed server selection

In the distributed server selection phase the witnesses select the most suitable server to compute Bob's computation in a distributed way.
On every server the following algorithm is executed: First we make a list with the indexes of all the servers in the environment. From this list we delete the indexes of the servers that take too much time or when their cost is too great for the given computation. From the remaining server indexes we chose the server with the best reputation. We will call this server Alice. The index of Alice is send to the rest of the witnesses.
Because a majority is of the witnesses is honest and they all do the deterministic algorithm on the same data, the correct server Alice is always chosen by a majority of the witnesses.

Phase 4: Threshold witnessing

In the threshold witnessing phase we let Alice do the computation and test if the solution is correct.

First all (honest) witnesses send the computation and input data elements to Alice together with one or more computed *ringers*. Ringers are used to (cheaply) get proof of if a computation is done correctly.
A ringer a precomputed solution to a randomly chosen input data element. Witnesses will take one of the input data elements and calculate the solution for this data element. Now Alice must return the index of the input element which the ringer corresponds to after she has computed the result of all the input data.
Alice could be malicious and just compute the ringers backwards to get the input data and return random numbers as the rest of the solutions. To counter this a one-way hash function is used on the ringer, which make sure Alice can't calculate backwards. Alice must compute the solution all the input data elements, do the same one-way hash function over them and compare the results with the hashed ringer.
But what is malicious Alice is lazy and stops computing real solutions when she found the solution to all the ringers she received? The countermeasure to this problem is that witnesses could send a couple of fake ringers, unknown to Alice, which look like real ringers but are just random and don't have a corresponding input data element. Because Alice never knows if she has all the real ringers she must compute all the computation solution.
Ringers give a pretty good chance that Alice will be caught is she does not do all the computations, and because the penalty on being caught acting malicious can be severe (Burn server Alice and spread her ashes across the sea), she will probably do her work.

Phase 5: Ringer checking

After her computations Alice returns her solution for all the input data elements, plus the answers to all the ringers she received to all the witnesses. The witnesses then individually check if Alice returned the right answer to their ringer(s). If a ringer is not correctly answered, then the witness will notify the other witnesses of this. Because the report of one witness can never be trusted, the other witnesses will check if the ringer is

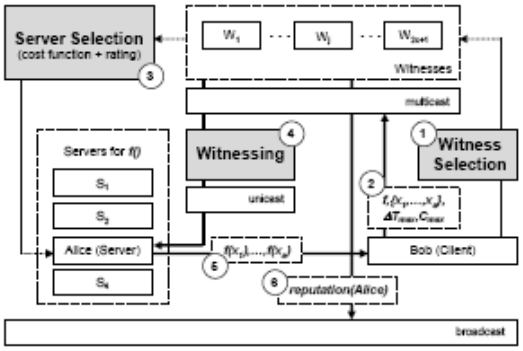


Figure 10.1: Overview of the system.

correct and Alice did indeed give the wrong answer. If Alice is indeed found guilty, she is again burned and has her ashes spread across the sea. Else the witness who incorrectly reported that Alice answered a ringer incorrectly is malicious and faces her fate instead.

Phase 6: Yielding the solution

If Alice answered all the ringers correctly, all her computation answers are considered correct and the solutions are sent to client Bob.

Bob gives feedback of Alice's work accordingly to whether she did her job within the time limit. The witnesses will update Alice's reputation using the feedback of Bob. Finally this new reputation is signed with the secret key shares and send to all servers so they can update their reputation value for Alice.

10.2.4 key renewal

To counter against mobile virus attacks the secret key shares are recalculated and renewed by the trusted server after a predefined time in which no vulnerabilities can be used such that more then c servers are attacked. This method makes the system secure against mobile virus attacks.

10.3 Summary and Conclusions

In this chapter we first have explained what reputation systems are and how they could be exploited. In the second part we have given a detailed example of a reputation system in an automated distributed computing environment. We can prove that the given system is secure against:

- **Bad mouthing and Ballot stuffing:**
  A reputation can never be changed by malicious servers because they can't break signed reputation values which they store. This means reputations can only be

changed by a client after he paid for a computation using his feedback which is how it should work.

- **Cybil attacks** Cybil attacks do not work since any fake participant can own at most a non-unique secret key share (which is of the malicious entity that created it), and thus it will be rendered unsuitable to act as an independent witness.
- **Mobile virus attacks** This is countered by renewing the key shares regularly, which does not give the attacker the time to corrupt enough servers to do any damage.

We also considered the system is secure in all the cases in which a server in the distributed computing environment could be malicious by not doing all his work or accusing others from being malicious. We can conclude the system is secure. More information can be found in [CS06].

Chapter 11

Advanced Encryption Standard

Written by *William van Hemert.*

11.1 History about AES

During the 20^{th} century there was a need for securing data flows, more and more data was going through public channels. That's why the National Bureau of Standards (NBS) in 1972 made a call for submission of encryption protocols. From this call the DES algorithm was chosen as the best option and became the official encryption standard. When DES was chosen as the official standard it was secure, because an exhaustive search was impossible to do, but when the years went by the DES standard became weaker and weaker.

In the 1990's an exhaustive search became more and more reasonable to do, because computing power was growing rapidly. In 1997 the National Institute of Standards and Technology (NIST) asked for handing in encryption algorithms that could be used as new standard. Fifteen algorithms where submitted, including the Rijndael algorithm which was made by Joan Daemen en Vincent Rijmen, two Belgian cryptographers. All fifteen algorithms where taken for a review and cryptographers looked at security, performance, possible settings, different environments and possibilities to work in limited hardware or software. Rijndael was chosen as the best solution to the above criterions. Some small changes in block size where made and in 2000 Rijndael became the new official standard for encryption, named as "Advanced Encryption Standard" (AES). In the rest of the paper we will speak about AES instead of Rijndael, because AES is of course the Rijndael algorithm.

After introduction AES became National Security Agency (NSA) approved, this means that the encryption standard would be used by all US Government Departments. As an advice the NSA gave the following secret levels:

- 128 bit key length or above where approved for the level "Secret"

- 192 bit and 256 bit key lengths where approved for the level "Top Secret", later they changed the level of "Top Secret" to only allow 256 bit key lengths.

11.2 General properties of AES

11.2.1 Key length

The AES algorithm uses a minimal length of 128-bits, making it much safer than the forgoing DES standard, which makes use of a 56-bits key. The allowed key lengths are 128-bits, 192-bits or 256-bits, the different versions will further be pronounced as AES-128, AES-192 and AES-256. The key length is used as a property for making the algorithm strong. Trying all possibilities means checking lots and lots of different keys, even for the minimal key length this is a huge amount of work. Let's say that we have a personal computer that can check one billon keys a second (which he can't at the moment) then we will need the following times for an exhaustive search:

Type	Possible Keys	Time needed in years
AES-128	$2^{128} = 3{,}4028 \cdot 10^{38}$	$1{,}0790 \cdot 10^{22}$
AES-192	$2^{192} = 6{,}2772 \cdot 10^{57}$	$1{,}9904 \cdot 10^{41}$
AES-256	$2^{256} = 1{,}1579 \cdot 10^{77}$	$3{,}6717 \cdot 10^{60}$

11.2.2 Rounds

AES uses multiple rounds to encrypt the data blocks, in each round the following steps are done: SubBytes, ShiftRows, MixColumns, and AddRoundKey. All four steps are done in each round, except the last round: the MixColumns is then skipped, what each step means will be discussed in the next section. The number of rounds depends on the version that is used for the algorithm, here is a short overview of the number of rounds per version:

- AES-128: 9 encryption rounds and one final round
- AES-192: 11 encryption rounds and one final round
- AES-256: 13 encryption rounds and one final round

11.2.3 Block cipher

AES uses block cipher to encrypt and decrypt the information. AES uses a block size of 128-bits, thereby the original text is spitted into blocks of 128-bits that are represented as a hexadecimal 4 x 4 matrix (4 bytes x 4 bytes = 16 bytes x 8 bits = 128 bits). Each block is encrypted en decrypted separately and the output is the combination of the separate blocks.

11.2.4 Fast in hardware and software

Processors work at byte level and therefore need extra time to work with underlying bits, AES therefore uses bytes in the matrix to work with, and DES uses single bits which makes it slower. At hardware level the AES standard works at the same speed as DES, because wiring decides how the bits flow, and this is independent of bytes.

11.2.5 Confusion and Diffusion

Shannon described two important properties for block ciphers, namely:

- Confusion: "The relationship between the plaintext, cipher text, and key should be complex", this means that it must be hard to see overlapping patterns between the plain text and the cipher text. It also means that information about the key that is used cannot be derived from the cipher text.
- Diffusion: "All of the cipher text should depend on all of the plaintext and all of the key" This basically means that if one bit is changed in the plain text or key multiple bits must be changed in the cipher text.

Further, he assumed that when an encryption system is created at least one of the two properties must be implemented in the algorithm to make it a good system. That's good news, the two properties stated above are used in the AES algorithm, namely in the SubBytes, ShiftRows and MixColumns steps.

11.3 Different components of AES

The AES algorithm uses the Key Schedule, SubBytes, ShiftRows, MixColumns and AddRoundKey steps to encrypt and decrypt. At each step simple operations are done, these steps are then combined to a round and the number of rounds depends on the key, how the steps are combined will be explained after the individual steps are discussed.

11.3.1 Key Schedule

The key schedule is a step that only takes place once in the encryption or decryption process. The idea is to create different keys from a small key, so that these different keys can be used in the rounds during the process. The single round keys are created from the expanded key and this expanded key is generated follows:

Version	a	b
AES-128	16	176
AES-192	24	208
AES-256	32	240

1. The first a bytes are copied from the original key
2. The value used for rcon is set to one (rcon is an internal function)

3. A 4-byte variable is created, we call this variable c.
4. c gets the value of the forgoing 4-bytes in the expanded key
5. The Key Schedule Core (KSC) is used to randomize c, with the help of the value of rcon (KSC is an internal function)
6. We increment the value used for rcon by one, making the next result from the rcon totally different.
7. We XOR the value of c with the expanded key
8. The value of c is stored as 4 new bytes in the expanded key
9. Steps 4 and 8 are done an extra three times
10. The following steps are taken, depending on the key length:
    - If we are using AES-128:
        - No extra steps are done
    - If we are using AES-192:
        - Steps 4 and 8 are done an extra two times
    - If we are using AES-256:
        - Step 4 is done
        - Variable C is going through the Rijndael S-Box
        - Step 8 is done
        - Steps 4 and 8 are done an extra three time
11. We repeat step 3 till 10 until B bytes are calculated

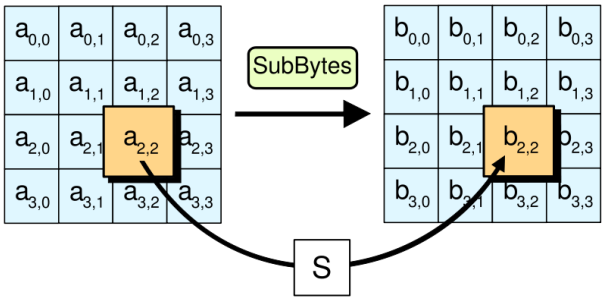
As you can see above the steps 4 and 8 are done some extra times when the key is longer. The reason is that there are more round key are needed, since the number of rounds depend on the secret key length. The above steps may look a bit confusing, maybe this figure can help you to understand it:

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
62 63 63 63 62 63 63 63 62 63 63 63 62 63 63 63
9b 98 98 c9 f9 fb fb aa 9b 98 98 c9 f9 fb fb aa
90 97 34 50 69 6c cf fa f2 f4 57 33 0b 0f ac 99
ee 06 da 7b 87 6a 15 81 75 9e 42 b2 7e 91 ee 2b
7f 2e 2b 88 f8 44 3e 09 8d da 7c bb f3 4b 92 90
ec 61 4b 85 14 25 75 8c 99 ff 09 37 6a b4 9b a7
21 75 17 87 35 50 62 0b ac af 6b 3c c6 1b f0 9b
0e f9 03 33 3b a9 61 38 97 06 0a 04 51 1d fa 9f
b1 d4 d8 e2 8a 7d b9 da 1d 7b b3 de 4c 66 49 41
b4 ef 5b cb 3e 92 e2 11 23 e9 51 cf 6f 8f 18 8e
```

Example of an extended key that is generated for AES-128

11.3.2 SubBytes step

In this step the sixteen bytes that are stored in the matrix will be substituted to a new value, that new value is then stored. The substitution is defined by the Rijndael S-Box, this S-Box is member of the multiplicative inverse of the finite field and has good non-linearity properties. The values of the S-box can be looked op in the figure about Rijndaels S-Box.



Graphical view of the SubBytes step

On the values of the field matrix multiplication with the S-Box is applied, and every result is then XORed with the multiplication pairs and a predefined value of the S-Box. This XORing with the predefined value is needed to invert the value during decryption, that's why the inverse S-Box is different from the original S-Box. Since confusion is used here, the output will look random making it difficult to gain information about secret information or the input.

$$\begin{bmatrix} b7 \\ b6 \\ b5 \\ b4 \\ b3 \\ b2 \\ b1 \\ b0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} a7 \\ a6 \\ a5 \\ a4 \\ a3 \\ a2 \\ a1 \\ a0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Rijndaels S-Box

$$\begin{bmatrix} a7 \\ a6 \\ a5 \\ a4 \\ a3 \\ a2 \\ a1 \\ a0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} b7 \\ b6 \\ b5 \\ b4 \\ b3 \\ b2 \\ b1 \\ b0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

Rijndaels inverse S-Box

Decryption: This step works almost the same, only this time other values of the S-Box are used, namely the inverse of the original S-Box.

Proof: To proof that the inverse holds we will label the S-Box with X and the inverse S-Box with Y. The constants used in the S-Box will be represented as C and the constants used in the inverse S-Box will be represented as D. The following calculation can made: We can view the S-Box output as: B' = XB $\oplus$ C (*) Assume that the inverse looks like: Y = X^{-1}: YX = E (Unity matrix) Multiplying both parts of (*) by Y, we have: YB' = YXB $\oplus$ YC = B + YC and B = YB' $\oplus$ YC = YB' $\oplus$ D

Example: Let's take the hexadecimal value CA as example. This value is looked up in a Galois Field multiplicative inverse table. In the field (C,A) the value "53" is showed, representing the binary value "01010011". The calculations are shown in the tables below:

Bit number	Input	Calculation	Output
7	1	1·1 ⊕ 1·1 ⊕ 1·0 ⊕ 1·0 ⊕ 1·1 ⊕ 0·0 ⊕ 0·1 ⊕ 0·0 ⊕ 0	1
6	1	0·1 ⊕ 1·1 ⊕ 1·0 ⊕ 1·0 ⊕ 1·1 ⊕ 1·0 ⊕ 0·1 ⊕ 0·0 ⊕ 1	1
5	0	0·1 ⊕ 0·1 ⊕ 1·0 ⊕ 1·0 ⊕ 1·1 ⊕ 1·0 ⊕ 1·1 ⊕ 0·0 ⊕ 1	1
4	0	0·1 ⊕ 0·1 ⊕ 0·0 ⊕ 1·0 ⊕ 1·1 ⊕ 1·0 ⊕ 1·1 ⊕ 1·0 ⊕ 0	0
3	1	1·1 ⊕ 0·1 ⊕ 0·0 ⊕ 0·0 ⊕ 1·1 ⊕ 1·0 ⊕ 1·1 ⊕ 1·0 ⊕ 0	1
2	0	1·1 ⊕ 1·1 ⊕ 0·0 ⊕ 0·0 ⊕ 0·1 ⊕ 1·0 ⊕ 1·1 ⊕ 1·0 ⊕ 0	1
1	1	1·1 ⊕ 1·1 ⊕ 1·0 ⊕ 0·0 ⊕ 0·1 ⊕ 0·0 ⊕ 1·1 ⊕ 1·0 ⊕ 1	0
0	0	1·1 ⊕ 1·1 ⊕ 1·0 ⊕ 1·0 ⊕ 0·1 ⊕ 0·0 ⊕ 0·1 ⊕ 1·0 ⊕ 1	1

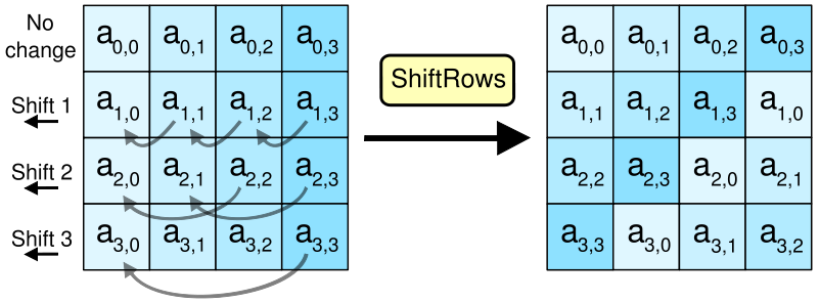
After encryption the result is then used to decrypt the value with the inverse S-Box:

Bit number	Input	Calculation	Output
7	1	0·1 ⊕ 1·1 ⊕ 0·1 ⊕ 1·0 ⊕ 0·1 ⊕ 0·1 ⊕ 1·0 ⊕ 0·1 ⊕ 0	1
6	1	0·1 ⊕ 0·1 ⊕ 1·1 ⊕ 0·0 ⊕ 1·1 ⊕ 0·1 ⊕ 0·0 ⊕ 1·1 ⊕ 0	1
5	1	1·1 ⊕ 0·1 ⊕ 0·1 ⊕ 1·0 ⊕ 0·1 ⊕ 1·1 ⊕ 0·0 ⊕ 0·1 ⊕ 0	0
4	0	0·1 ⊕ 1·1 ⊕ 0·1 ⊕ 0·0 ⊕ 1·1 ⊕ 0·1 ⊕ 1·0 ⊕ 0·1 ⊕ 0	0
3	1	0·1 ⊕ 0·1 ⊕ 1·1 ⊕ 0·0 ⊕ 0·1 ⊕ 1·1 ⊕ 0·0 ⊕ 1·1 ⊕ 0	1
2	1	1·1 ⊕ 0·1 ⊕ 0·1 ⊕ 1·0 ⊕ 0·1 ⊕ 0·1 ⊕ 1·0 ⊕ 0·1 ⊕ 1	0
1	0	0·1 ⊕ 1·1 ⊕ 0·1 ⊕ 0·0 ⊕ 1·1 ⊕ 0·1 ⊕ 0·0 ⊕ 1·1 ⊕ 0	1
0	1	1·1 ⊕ 0·1 ⊕ 1·1 ⊕ 0·0 ⊕ 0·1 ⊕ 1·1 ⊕ 0·0 ⊕ 0·1 ⊕ 1	0

11.3.3 ShiftRows step

This is a simple step, each row is shifted a number of places to the left depending on the row number. The numbers of shifts that are made are summarized below:

- The first row shifts no places, since the row number is 0
- The second row shifts one place to the left, since the row number is 1
- The third row shifts two places to the left, since the row number is 2
- The fourth row shifts three places to the left, since the row number is 3



Graphical view of the ShiftRows step

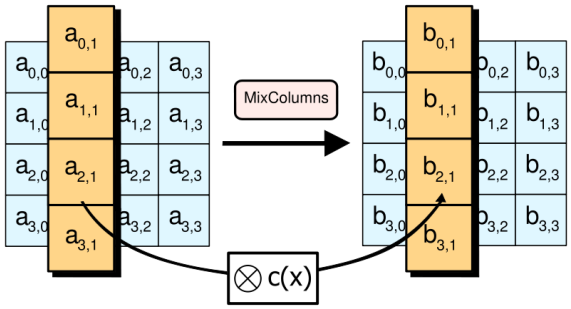
Decryption: This step is the almost the same as in the encryption step, but the direction of the shifts is reverted. This time the rows are shifted to the right, creating the inverse step from encryption.

Proof: This can easily been proven by a simple example:

Input				Encryption	Internal				Decryption	Output			
54	62	13	21	0 to the left	54	62	13	21	0 to the right	54	62	13	21
98	96	09	83	1 to the left	96	09	83	98	1 to the right	98	96	09	83
48	72	35	76	2 to the left	35	76	48	72	2 to the right	48	72	35	76
23	45	76	23	3 to the left	23	23	45	76	3 to the right	23	45	76	23

11.3.4 MixColumns step

In this step each value of a field is multiplied by a number, this is done in the GF(2^8) this Galois field is a member of the finite fields.



Graphical view of the MixColumns step

The number used during the multiplication depends on the field, the values are:

Multiplication				Hexadecimal				Mathematical
2	3	1	1	02	03	01	01	$b_0 = 2a_0 + a_3 + a_2 + 3a_1$
1	2	3	1	01	02	03	01	$b_1 = 2a_1 + a_0 + a_3 + 3a_2$
1	1	2	3	01	01	02	03	$b_2 = 2a_2 + a_1 + a_0 + 3a_3$
3	1	1	2	03	01	01	02	$b_3 = 2a_3 + a_2 + a_1 + 3a_0$

Decryption: Decryption follows the same steps as encryption, but the multiplication values are different:

Multiplication				Hexadecimal				Mathematical
14	11	13	9	0E	0B	0D	09	$b_0 = 14a_0 + 9a_3 + 13a_2 + 11a_1$
9	14	11	13	09	0E	0B	0D	$b_1 = 14a_1 + 9a_0 + 13a_3 + 11a_2$
13	9	14	11	0D	09	0E	0B	$b_2 = 14a_2 + 9a_1 + 13a_0 + 11a_3$
11	13	9	14	0B	0D	09	0E	$b_3 = 14a_3 + 9a_2 + 13a_1 + 11a_0$

Proof: Every column will be treated as a polynomial over Galois field GF(2^8). It's multiplied modulo $x^4 + 1$ by using a fixed polynomial. During the encryption the following formula is used: c(x) = '03'x^3 + '01'x^2 + '01'x + '02', the inverse of this formula is c^{-1}(x) = '0B'x^3 + '0D'x^2 + '09'x + '0E', which is indeed the decryption formula used. For example, field (0,0) can be verified as follows: 0E·02 + 0B·01 + 0D·01 + 09·03 = 0E·02 + 0B + 0D + 09·03 = 1C + 0B + 0D + 1B = 01

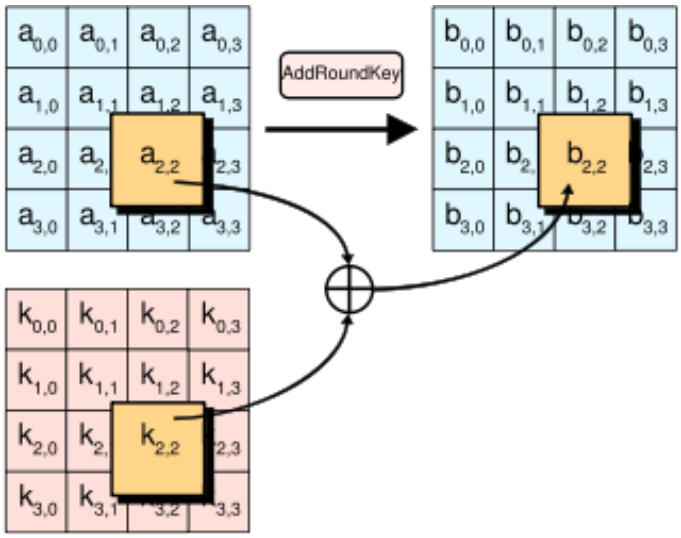
Example: Let's say we have a column with the following hexadecimal values: {F2,4C,E7,8C} The calculation is done as follows: b_0 = 2·F2 + 8C + E7 + 3·4C =

40 b_1 = 2·4C + F2 + 8C + 3·E7 = D4 b_2 = 2·E7 + 4C + F2 + 3·8C = E4 b_3 = 2·8C + E7 + 4C + 3·F2 = A5

The inverse is calculated as follows: b_0 = 14·40 + 9·A5 + 13·E4 + 11·D4 = F2 b_1 = 14·D4 + 9·40 + 13·A5 + 11·E4 = 4C b_2 = 14·E4 + 9·D4 + 13·40 + 11·A5 = E7 b_3 = 14·A5 + 9·E4 + 13·D4 + 11·40 = 8C

11.3.5 AddRoundKey step

In this step every bit of the byte in the matrix is XORed with the corresponding bit in with the secret round key. For those who are not familiar with XOR: The XOR port gives the value 1 if only one input is 1 and the other is 0.



Graphical view of the AddRoundKey step

Decryption: Decryption follows the same steps as encryption.

Proof: Suppose we have two bits: a bit x and a bit k. From the XOR rules follow: ((x ⊕ k) ⊕ k) = x. The input is equal to the output, since the input is inverted twice or stays equal twice.

Example: Let's take the random binary value "11100101" and the random binary key "01011100". The results of the steps are shown below:

Bit number	Input	Key	Encryption	Internal	Decryption	Output
0	1	0	1 ⊕ 0	1	1 ⊕ 0	1
1	1	1	1 ⊕ 1	0	0 ⊕ 1	1
2	1	0	1 ⊕ 0	1	1 ⊕ 0	1
3	0	1	0 ⊕ 1	1	1 ⊕ 1	0
4	0	1	0 ⊕ 1	1	1 ⊕ 1	0
5	1	1	1 ⊕ 1	0	0 ⊕ 1	1
6	0	0	0 ⊕ 0	0	0 ⊕ 0	0
7	1	0	1 ⊕ 0	1	1 ⊕ 0	1

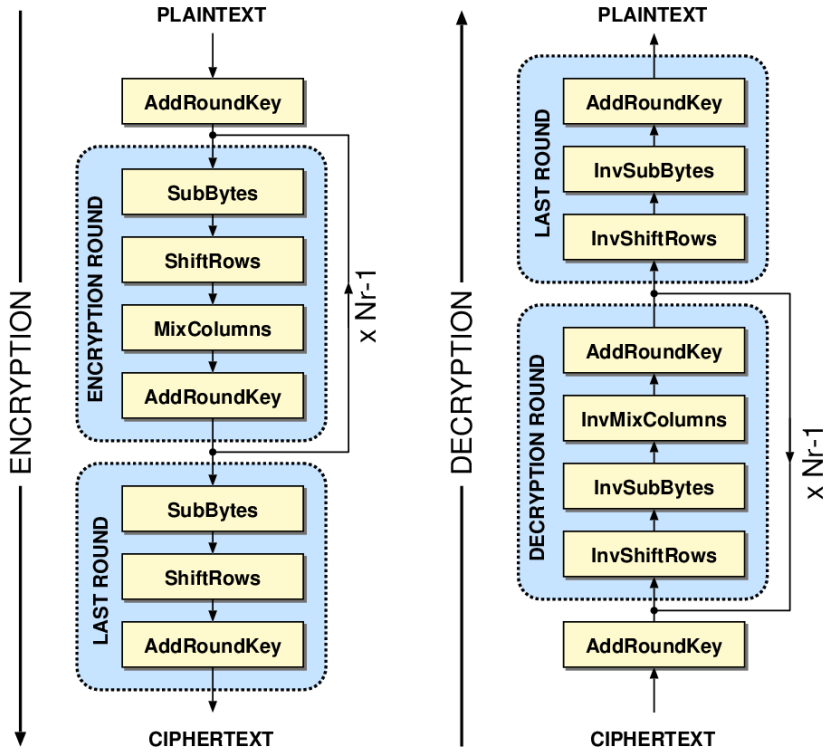
11.4 The encryption and decryption process

An encryption round consist of the following steps: SubBytes step, after that the ShiftRows step, followed by MixColumns step and finally the AddRoundKey step. A final round consist of the following steps: SubBytes step, after that the ShiftRows step and finally the AddRoundKey step.

The procedure runs as follows:

1. Input data is going in.
2. KeyExpansion: using the key schedule, the key is generated and will be use in the next encryption rounds and the single final round.
3. AddRoundKey: The input data is XORed with the first key of the expanded key
4. Multiple encryption rounds, the number of rounds depend on the key. Each time an encryption round is processed the corresponding key of the expanded key is used.
5. Final Round: This round is done one time, independent of the key-size used.
6. The output data is going out.

In each round, diffusion and confusion is applied by the SubBytes, ShiftRows and MixColumns step, making it hard for an attacker to gain important information about the key or plain text. The encryption process can be optimized when used on a system with 32-bit words or larger. The optimized version can combine the SubBytes and ShiftRows with MixColumns. This can be achieved by changing the procedure to a number of table lookups, these tables will consume 16kb of memory.



A schematically overview of the encryption and decryption procedures

11.5 Research about AES

There has been and will be a lot of research about the AES algorithm. Even on different fronts of research, the main groups are summed below:

- Attacks: Will be discussed in the next section.
- Hardware Architectures: Much research is done about the implementing of AES in different hardware. These investigations vary from implementations in small and limited hardware till optimization of AES in hardware and from FPGA implementations till the Rijndael Processor.
- High Speed Implementations: These publications can be used to streamline your designing and making a fast implementation for AES. Of course the optimization step of the AES algorithm is discussed in these publications.
- Low Cost Implementations: This subject contains implementation of AES with minimal costs, thereby optimizing some structures of AES, like the S-Boxes or MixColumns step.
- Software Implementations: Studies about different implementations of AES in software, the limits of some platforms and the optimization in software.
- Side-Channel Analysis: This term will be discussed in the next section.

As you can see lots and lots of publications are done about AES. Most of them are about the implementation and optimization in hardware or software. We will not discuss any further details of the individual publications, except the once that have to do with the security of the algorithm.

11.6 Known vulnerabilities

As you could read in the previous section a lot of research is going on about attacks on AES. Time has proven no real successful attack on AES has been found, but there are some attacks that where effective when fewer rounds are applied during the encryption.

In 2002 Nicolas Courtois and Josef Pieprzyk published an article that the AES algorithm contained a potential weakness which they call eXtended Sparse Linearization attack (XSL attack). The attack depends on solving multivariate quadratic equations (MQ), this problem is NP-hard. The solution to this NP-hard problem is linearization, it replaces each quadratic term by an independent variable, after that the Gaussian elimination algorithm is used to solve the problem. This type of approach was improved in 2000 by Courtois et al, he named the algorithm XL, which stands for eXtended Linearization. This improvement increased the number of equations in the attack, but estimations showed that this approach would not work on AES.

After that improvement the XSL attack from Nicolas Courtois and Josef Pieprzyk came to life and took advantage of the equations produced in the XL algorithm. Several cryptographers have read the publication and found out that there was a problem with the mathematics used to prove the attack. They even found out that the algorithm does not work in the time limit that is shown in the paper and therefore it isn't an improvement at all. Therefore the XSL attack is not universally accepted. It's unlikely that this type of attack can be used in practice and therefore the AES algorithm is safe against such an attack.

Since the submission in 1999 there have been seven possible attacks published, the types of attacks used are: Related Key attack, Square Attack, Impossible Differential and Truncated Differential. The two most successful attacks are the Related Key attack done by Biham et al in 2005 and the square attack done by Ferguson et al in 2000. Another attack that is often used is the side channel attack. We will explain these three possible attacks below:

11.6.1 Side channel attack

There are some possible attacks on the AES algorithm, the one that is discussed often is the so-called "side channel attack". When side channel attacks are applied the device leaks information that later can be used to gain information about the secret key or plain text that is used. The type of information that is leaking depends on the hardware implementation, here are some examples: timing, power usage, electromagnetic radiation and sound. Fortunately, whether or not a device is leaking information depends on the

implementation of het device itself, so if you can create hardware that does not leak (or can hide the leaking) then this attack is not possible.

Two known possibilities for side channel attack have been published by Bernstein and Shamir, both in 2005. Bernstein used the cache timing attack on a special server that used Open SSL with AES encryption. The server was forced to leak information about the timing, this information was used together with 200.000.000 plain texts to reproduce the secret information. The special server used makes it hard to translate this attack to a practical solution in real life, since the timing information is not available then. Later that year Shamir published an article that also uses the cache timing as a weak point, he was able to do an attack in just 65ms with 800 writes. The only problem with this attack was that the attacker must break into the system that encrypts the data.

11.6.2 Square attack

A chosen plain text can be used to gain information about the key that is used. In such an attack the structure of the outputted cipher text is investigated for structures that can revile the key or parts of the key that is used. This problem was already known during the design of the algorithm, it was known that this attack was a major problem. In the time of the development there was a possibly to gain the key when 4 rounds where used, the developers even noticed that it was possible to extend these attacks to 5 or 6 rounds. (Since the introduction more and more research is done about this attack making it possible to break the AES-128 and AES-192 in 7 rounds and breaking the AES-256 in 9 rounds) That's why the developers of AES have chosen for more rounds, making it impossible to use this attack on the full number of rounds, since the information that can be used after each round is less and less. Ferguson et al proofed in 2000 a possibility to do such an attack and thereby reducing the time by a factor of 10, this looks convincing but the time needed for all different AES-128 keys to be checked under this assumption will still be $1{,}0790 \cdot 10^{21}$ years.

11.6.3 Related-key attack

This attack can be used when the attacker has the possibility to make changes in the key. (If dedicated hardware is used with a fixed key this attack is not possible) There are two types of this attack, the attacker must supply a specific plain text and a specific constant:

- The attacker asks for the encryption of the text where the secret key XORed with the constant.
- The attacker asks for the encryption of the text where the constant is added to the secret key.

AES is not vulnerable for both of the attack separately, but when the two types of attack are used together it becomes possible to gain secret information. Biham et al published an article where he made it possible to do an attack on the AES-192, when

using 9 rounds, but this is still safe for AES, since the number of rounds is 12 for AES-192.

There are researchers that are critical about the global security of the AES algorithm, since it's possible to create successful attacks with fewer rounds than done by the AES algorithm. Examples are the Related-key attack and the Chosen-plaintext attack, researchers think that it will be possible to improve these attacks and thereby making a successful attack on the full number of rounds done by the AES algorithm. Furthermore, AES has a neat algebraic structure, unlike other block chipper algorithms, that's why other researchers target their research on this algebraic structure. No attacks of this type are known yet, but there might be a possibility that such an attack will be found in the future.

11.7 Summary and conclusions

The information is divided in different blocks, every block is encrypted by using multiple rounds and each round consists of different steps to encrypt or decrypt the data. During the rounds the output of one round is the input of the next round, making it hard to gain information after different rounds and therefore AES is a strong algorithm against square attacks. This type of attack was already known during the invention of the algorithm and therefore the number of rounds where made larger than the number of rounds that could be vulnerable against this sort of attacks.

There has been done a lot of research done about AES. There were different practical attacks and some theoretical attacks made, but none of them where real successful. The theoretical attacks where defended by cryptographic experts and where proven wrong. Some practical attacks depend on hardware therefore making it impossible to use when some hardware implementation is used. If different hardware is used or the leak of information is removed this attack has no meaning anymore. The one of the few good attacks is the square attack, reducing the size by a factor of ten, but even then it is infeasible to check all the keys in an exhaustive search!

AES is one of the most popular algorithms used, it's already widely used throughout the world in commerce and government agencies. His main competitor is 3DES which is stronger than the AES-128, namely 2168 possible keys against 2128 keys for the AES-128. 3DES does not have the possibility to used longer keys than 168 bit, thereby making it weaker than the AES-192 and AES-256. One of the reasons for the popularity of AES is that it is an official standard in many government agencies, these agencies have investigated the quality of the algorithm. Like 3DES AES can also be implemented in minimal or dedicated hardware and software, one big advantage if AES is the calculating with bytes instead of bits like his predecessors did, they consumed extra time in software needed for converting bits to bytes and vice versa.

AES is a secure algorithm, it uses different techniques as the Galois field and XOR to confuse and diffuse the data and these are important properties that make it difficult to gain information about the key or original text.

11.8 Used resources

Main AES information used
http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf
http://www.quadibloc.com/crypto/co040401.htm
http://si2.epfl.ch/ gurkayna/acacia/acacia.html
http://cmpe.emu.edu.tr/chefranov/cmpe553_04/LectureNotes/AES CIPHER.Part2.doc

Main information about attacks and research
http://www.ece.mtu.edu/ee/faculty/mishra/Publications/DFT2002/DFT2002_Paper.pdf
http://dspace.lib.fcu.edu.tw:8080/dspace/bitstream/2377/1516/1/ce07ics002002000179.PDF
http://www.iaik.tu-graz.ac.at/research/krypto/aes/
http://www.lois.cn/LOIS-AES/data/AES-256.pdf
http://eprint.iacr.org/2002/044.pdf

Other AES information used
http://en.wikipedia.org/wiki/Advanced_Encryption_Standard
http://en.wikipedia.org/wiki/Advanced_Encryption_Standard_process
http://nl.wikipedia.org/wiki/Rijndael
http://en.wikipedia.org/wiki/Rijndael_S-box
http://en.wikipedia.org/wiki/Rijndael_key_schedule
http://www.cacr.math.uwaterloo.ca/conferences/2004/ecc2004/robshaw.pdf

Other information about attacks
http://en.wikipedia.org/wiki/XSL_attack
http://www.secure-data.us/templates/Advanced_Encryption_Standard.html#Optimization_of_the_ciph
http://www.derkeiler.com/pdf/Newsgroups/sci.crypt/2003-04/0092.pdf
http://www.crypto.ruhr-uni-bochum.de/en_sclounge.html
http://www.cryptosystem.net/aes/#OfficialStatus
http://www.conxx.net/AES.php

General information used
http://en.wikipedia.org/wiki/Side_channel_attack
http://en.wikipedia.org/wiki/Related-key_attack
http://en.wikipedia.org/wiki/Chosen-plaintext_attack
http://en.wikipedia.org/wiki/Confusion_and_diffusion
http://www.nku.edu/ christensen/diffusionandconfusion.pdf
http://en.wikipedia.org/wiki/Block_cipher

Chapter 12

Certificates and Root Keys

Written by *Bas van Schaik and Steven Woudenberg.*

12.1 Introduction

In the early days of the internet it was mostly used for publishing and locating information but nowadays internet is much more: it is possible to do all your bank business online and it is possible to buy or sell items, just like an electronic flea market. Because lots of money can be involved with these actions it became very necessary to protect these actions against dishonest trading people and hackers.

Netscape solved the security problem by introducing the Secure Socket Layer protocol (SSL). When this protocol is used, authentication and integrity are guaranteed because of the use of both symmetric and asymmetric cryptography. In SSL symmetric cryptography is used because of the speed disadvantage of asymmetric cryptography, but first the client and the webserver must agree on this symmetric key. The easiest way to achieve this is to start the communication with asymmetric cryptography, agree on a key for symmetric cryptography and then use the agreed key for symmetric cryptography during the rest of the communication.

The first part of the communication between a client and a webserver is described in protocol 12.1 and until the verification of the certificate of the webserver (step 6) it is all unencrypted. The 'Finished' messages (steps 7 and 8) are encrypted and include a digest of all the communication from step 1 through 6 to make sure that none of the communication between the client and the webserver was altered in flight. Because these steps are encrypted it is not possible anymore for hackers to intervene (at least not when they don't have the private key) and when both the client and the webserver are sure that all communication went right, they decide on a key for symmetrical encryption which will be used from that point on.

Client		Webserver
Connect to server's HTTPS TCP port (443)	1	Accept connection
Send client_hello:	2	
– Highest SSL version		
– Ciphers supported		
– Random data		
	3	Send server_hello:
		– Selected SSL version
		– Selected cipher
		– Random data
	4	**Send certificate:**
		– Public key
		– Authentication signature
	5	Send server_done
Verify certificate and accept website	6	
Send 'Finished'	7	
	8	Send 'Finished'

Protocol 12.1: SSL HANDSHAKE PROCEDURE

When taking a close look at this handshaking scheme of the SSL protocol it becomes clear that a certificate is something which is fairly important: when the client receives the certificate and accepts the certificate both parties can set up secure communications.

12.2 Certificates and Certificate Authorities

12.2.1 Introduction to certificates

The purpose of certificates is to prove that a key *really* belongs to a certain individual or organization. In that sense, a cryptographic certificate is comparable to a certificate of e.g. a Bachelor's degree which gives a certain individual the right to call himself 'Bachelor'. Such a degree certificate is always signed by the chairman of the exam committee of an institution who thereby guarantees the validity of the certificate. An employer trusts the exam committee and accepts a request for a salary raise.

This *real world* example clearly illustrates the importance of *trust*: the chairman of the exam committee signs the certificate and an employer trusts him. In cryptography the concept of *trust* is a bit more complicated, especially since all cryptographers tend to act a bit paranoid: do not trust anyone, until that person has proven to be trustworthy.

The website `www.bankofchina.ru` of the largest Chinese bank will identify itself with a certificate stating that `www.bankofchina.ru` really belongs to the Bank of China, but

Bob		trusted third party
Create RSA key pair (n_b, d_b) and (n_b, e_b)	1	
Send Certificate signing request:	2	
$((n_b, e_b)$, identifying information, $\dots)^{d_b}$		Receive request
	3	Identify and validate user
	4	Create certificate:
		(version, validity, public key, $\dots)^{d_t}$
Receive certificate	5	Send certificate

Protocol 12.2: Obtaining a certificate

can this certificate be trusted? Can the entity who signed this certificate be trusted? Or is this a typical example of a phishing website, set up by a group of Russian hackers?

12.2.2 How certificates work

Suppose Bob wants to run a webserver which offers the same services as for example `bankofchina.cn`. Because a lot of money is involved, he wants to obtain a certificate for his public key from a trusted third party. In order to obtain a certificate Bob must create a RSA key pair with private key (n_b, d_b) and public key (n_b, e_b) and send a certificate signing request for his public key.

The trusted third party also has a RSA key pair with a public key (n_t, e_t) and a private key (n_t, d_t) where the public key is public to anyone and the private key is used for signing certificates. The protocol for requesting a certificate is shown in protocol 12.2.

Now Bob has a certificate from a trusted third party and every time a client wants to connect to his webserver the certificate is presented to the client. One day Alice wants to connect to Bob's webserver to transfer some money from her bank account to another bank account and of course she wants to make sure that Bob's webserver is trusted before she logs in to make the transfer. The communication between Bob and Alice is shown in protocol 12.3.

The 'body' of the certificate (which is used in step 4) is the first part of the certificate: this is where all the information of the certificate is. Next to that there is a signature in the certificate: this is an encrypted hash of all the information in the body of the certificate. When the signature is decrypted (by using the public key of the trusted third party) the hash of the body of the certificate must appear, otherwise the trusted third party did not sign that specific certificate.

When Alice checks the validity of the certificate she also checks whether or not the certificate is revoked. Revocation is discussed in section 12.4.

Alice		Bob
Connects to Bob	1	Accepts the connection
Receives Bob's certificate c	2	Sends his certificate c
$s =$(signature of $c)^{e_t}$	3	
$f = H$(body of c)	4	
Checks the validity of c:	5	
– $s == f$?		
– Is c still valid?		
– Is c revoked?		
Accept the certificate	5	Accept Alice

Protocol 12.3: Checking a certificate

12.2.3 Certificate Authorities

In the world of certificates, the role of trusted party 'Theo' is realized by so-called *Certificate Authorities*. The CA performs checks on the requestor of a certificate to establish the requestor's identity and to determine if the requestor is trustworthy. Furthermore, the CA checks if the applicant have the right to use the domain name he is trying to secure and if the applicant's company is a legally and lawfully-formed organization.

These checks are quite strong, but not very thorough, which is why Extended Verification (EV) Certificates were introduced in 2007. To obtain such an EV Certificate the CA will perform some extra checks:

- a more rigorous validation of the legal and physical existance of the entity
- a verification of the entity and its physical address

Further details on the EV Certificate specification can be found in [For07]

For now, lets assume that the Certificate Authority states that the applicant is trustworthy and signs a certificate for the applicant's key. That is great! But can the CA be trusted? Maybe the certificate is signed by some mendacious Russian company! As can be realized, this problem has no simple solution and is similar to the famous "chicken or egg" dilemma.

The solution is as simple as unsafe: nowadays every web browser is shipped with the public keys of all major Certificate Authorities. Once a certificate is signed with the private key (sometimes called *root key*) of such a CA, the browser will recognize the signature and trust the certificate. If the browser does not recognize the signature, it will warn the user and ask if it should continue loading the page (figure 12.4).

12.2.4 Certificate format: X.509

Cryptographic certificates are almost always set up using the X.509 format, which was standardized by the International Telecommunications Union in 1988. Since then two new versions of the standard were released, version 3 being the latest as of 2008.

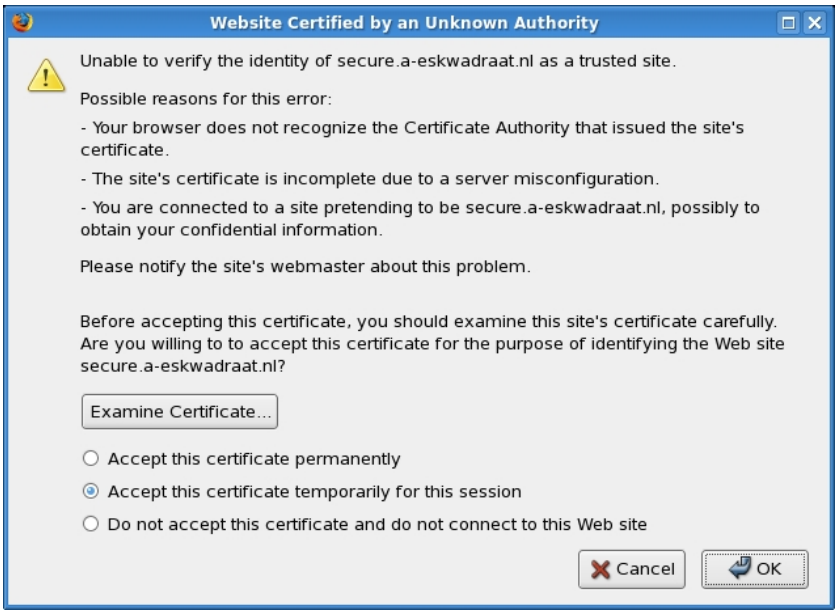


Figure 12.4: Firefox warns that the CA is unknown

A X.509 certificate consists of the following (mandatory) fields:

Version The certificate version (v1, v2 or v3)

Serial number An unique serial number

Algorithm ID The hashing algorithm used (for example: `md5WithRSAEncryption`)

Issuer The name and identity of the Certificate Authority

Validity The dates which mark the validity of the certificate

Subject The name and identity of the owner of the certificate

Public key Information about the key (modulus and public exponent)

Signature The hashed values of the other fields encrypted using the Certificate Authority's root key

12.2.5 Certificate costs and free alternative

Commercial CA's

Since Certificate Authorities are 'normal' (commercial) companies, their services are not free. The prices of signed certificates vary from $ 99,- (single domain) to $ 899,- (wildcard) per year, which is too expensive for some organizations or individuals.

CACert

For this reason, the CACert association was incorporated on the 24^{th} of July, 2004. This community driven association issues certificates for free, with the only requirement that there are other community members who have validated your identity. The more acknowledgements of your identity are given, the more *points* you gain, the stronger the generated certificates will become.

Casus 12.5: SAFENET-LUNA

A totally other approach to protect a root key is provided by companies like SafeNet, who sell hardware security devices to store secret keys. After the secret key is stored it can only be used after entering a secret PIN code.
The private key itself is stored in a tamper-proof chip which can be accessed by multiple persons, each with their own PIN code. Since the device is also able to generate a certificate for a certain key, there is no need to be able to extract the key from the device. the SafeNet Luna device is a typical example of a *closed system*.

At first, the generated certificates are *weak*: only one field (domain name) is filled in the *Subject* field, since other information about the requestor are not validated. When more points are gained, more fields (such as a name) can be filled in and the certificate gains strength.

The largest disadvantage of using CACert certificates is that the CACert root keys are not shipped with modern browsers like Internet Explorer and Mozilla Firefox. This means that a browser will warn the user (see figure 12.4) when a site using CACert certificates is encountered, because the browser does not recognize the CACert signature.

12.2.6 Protecting the root key

Once the (public) key of a CA is shipped with major browsers it is of extreme importance that the private part of this key is kept secret. Therefore, Certificate Authorities have very strict security policies to avoid revealing this important secret.

First of all, most Certificate Authorities do not work with their *real* root key but with a set of keys signed by that root key, the so-called *sub-root keys*. The obvious advantage is that the *real* root key is not needed for every new certificate and can safely be stored in a bank vault on a piece of paper. Once one of the *sub-root keys* leaks out, it can be revoked using the *real* root key. Certificate revokal will be subject to further discussion in section 12.4.

Furthermore it is good practice not to connect the machine containing the root key to a network or the internet, since that would yield a great security risk. Although most Certificate Authorities have decided to treat information about root key storage as company confidential (security through obscurity?), CACert has published some details[CAC08] about their setup.

12.3 Possible attacks

12.3.1 Introduction

There are many ways to set up an attack against certificate cryptography, but only a few of these attacks really focus on attacking the *concept* of certificates. Probably the most easy and practical way attack is installing a fake CA key on a computer.

12.3.2 Installing fake root keys

Attacks against the way Windows manages its CA keys are a typical example of attacks against a *specific implementation* of certificate cryptography, which are quite uninteresting but easy to deploy. If a hacker gains access to a victim's PC, he can install a manually generated root key to the Windows key store. From that moment on, the browser will no longer warn the user if a website is encountered which presents a certificate signed by the hacker, simply because Windows thinks the hacker is trustworthy.

Alsaid and Mitchell[AM05] describe a way to install a fake root key into Windows, without the user being warned. Although this can cause very dangerous situations (like a phishing site identifying itself with a trusted certificate), this type of attacks can hardly be seen as an attack against the concept of certificate cryptography. At the end, the "to trust or not to trust" problem is widely known and cannot be solved without trusting at least one party.

12.3.3 Colliding X.509 certificates

In their paper[LWW05], Lenstra et al. describe a way to generate two *different* certificates for two *different* keys, both signed with the same signature:

- Generate a (special) RSA keypair k_1
- Generate a slightly different RSA keypair k_2
- Get a certificate c_1 for k_1 from a commonly trusted Certificate Authority
- *Manually* craft a certificate c_2 for keypair k_2 by slightly modifying certificate c_1

The last step (crafting certificate c_2) depends on a weaknesses in the MD5 hash function, which allow a collission to be created. By manipulating the genuine certificate c_1 in a special way it is possible to control this collission in such a way that another public key is allowed to be inserted while the hash remains the same.

Although this attack looks quite dangerous, it is quite hard to use in real life. To be able to create the second certificate c_2, the attacker must have generated the two keypairs k_1 and k_2 himself and requested a certificate for pair k_1. After the certificate c_1 is generated and signed, an attacker can modify the certificate body in such a way that the signature of c_1 also signs the public part of k_2. However, the modifications in the certificate body needed to match the MD5 hash can not be chosen freely, which makes it unfeasible to generate a certificate c_2 which secures the website for `www.bankofchina.ru`.

Despite of the practical problems, still this attack might be the first step of something bigger. On the other hand it is quite easy to replace MD5 in the certificate for a different hash function without known weaknesses, like SHA1.

12.3.4 Stealing a keypair

When a attacker succeeds in stealing the keypair and certificate of our favorite Chinese bank, he can of course set up a similar looking website and register all login attempts of users. This way, bank clients will supply their account information without knowing that they are actually communicating with the attacker.

12.4 Certificate revocation

12.4.1 Introduction

When a private key leaks out or an attacker has managed to find a colliding X.509 certificate, the only possible thing to do is to revoke the certificate of the compromised key. But how can the rest of the world be informed about this revokal? And how can a joker be prevented from revoking the key of `www.bankofchina.cn`?

The last problem is quite easy to solve: when a webserver wants his certificate to be revoked, he will send a message to its CA, signed with the webservers private key. Because only the webserver knows it's secret key, only the webserver can send such a revocation request.

Certificate Revocation Lists (CRL)

One way to revoke a certificate is to use Certificate Revocation Lists (CRLs). These lists are signed and published by Certificate Authorities and contain all the serial numbers of the certificates which are revoked. When a client validates a certificate of a webserver, the client must also check whether or not the serial number of the certificate is an entry of the CRL of the Certificate Authority of the webserver. When the serial number is an entry in the CRL, the webserver is not authorized to conduct the transaction.

The CRLs have quite a lot disadvantages, for example they must be issued very often so there is as little time as possible for hackers to abuse a secret key. When a CRL is re-issued every day a hacker can hack a system just after the new CRL was issued and he can abuse the secret key for an entire day. Another disadvantage is that the CRLs are very vulnerable to DOS (Denial of Service) attacks when the CRL cannot be reached by clients they cannot check whether or not the webserver they want to work with has a valid certificate. The last disadvantage which will be discussed here is the length of the CRLs: they can become very, very long: in an investigation of the (American) Federal PKI's cost lists it appeared that the CRLs were by far the largest entry...

Online Certificate Status Protocol (OCSP)

In the Online Certificate Status Protocol (OCSP) the Certificate Authority will receive a question from a client whether or not a certain certificate is valid. When the certificate is valid the Certificate Authority will give its own signature on the validity status of the certificate at the current time. The disadvantages of CRLs are not present in OCSP, except for the DOS attack: it is still possible to generate a lot of requests such that the OCSP server cannot respond to all of them in time. However, in that case the clients will simply not get an answer from the CA and do not know whether or not to trust the webserver.

Unfortunately, the OCSP has some other problems which the CRL's don't have. Signing a piece of data with the RSA cryptosystem is quite expensive and when a lot of requests are fired at the CA-server simultaneously the server must be able to take care of all these requests. Another problem is the bandwith (because of this a DOS attack becomes easier): when a RSA system is used the message will have a minimum size of 2048 bits and again when a lot of requests are fired at the CA-server at the same time there might occur a problem. A solution to this problem is to use a distributed system but this will probably result in more problems than it can offer solutions: every server must have its own secret key to sign the answers to the requests and when one server is compromised, the entire system is compromised. There are solutions for this problem, but the costs are quite high.

Novomodo

[Mic02] describes NOVOMODO which is a more up to date system for certificate validation than CRLs and OCSP. Instead of a signature, as is used in the OCSP, Novomodo uses a *one-way-hash* function, which is of course very hard to invert (hence the name *one-way*-hash function). The hash function is much faster than signing with a RSA key and the output of the hash function is 20 bytes, which is a lot smaller than the minimum size of the answer of the OCSP.

With Novomodo two extra values are added to a certificate: Y_1 and X_{365}, where 365 stand for the number of days the certificate must be valid. First the CA randomly selects two different 20-byte numbers, X_0 and Y_0 and next the values which must be added are calculated. Y_1 is calculated by hashing Y_0: $Y_1 = H(Y_0)$ and X_{365} is calculated by repeatedly hashing X_0. First X_0 will be hashed and the outcome is X_1, then X_1 is hashed and the outcome is X_2, etcetera, until X_{365} is reached: $X_1 = H(X_0), X_2 = H(X_1), ..., X_{365} = H(X_{364})$. Now there is an X-hash for each time period (in this case: day).

When a client wants to verify whether or not a certificate is still valid he will send a request to the CA. The CA will return Y_1 if the certificate is still valid and Y_0 if it is revoked. When the certificate is still valid the CA will also send X_{112}, assuming that the certificate was given $365 - 112 = 253$ days ago and it is valid for 365 days. The user now can check whether or not it is true what the CA says: when Y_0 is returned he will calculate $Y_1' = H(Y_0)$ and if $Y_1' == Y_1$ the certificate is revoked. When $Y_1'! = Y_1$ the client will not trust the CA, but now the client does not know whether or not to trust the certificate...

When the certificate is still valid the CA will return Y_1, which is also in the certificate and can be checked by the client, together with X_{112} which must be hashed 253 times by the client. When $X'_{365} == X_{365}$ the CA has proven that the certificate is valid and the client will continue. When $X'_{365}! = X_{365}$ something is wrong: maybe some hacker has guided the client to a dishonest CA and used the publicly known Y_1 together with some other value of X to fool the client. Because it is a one way hash function no one, except for the CA who has calculated and stored all the X- and Y-values can know the inverse of a hash-value.

Because only the CA knows Y_0 and X_0 it is not necessary to sign the messages which are transmitted the clients: nobody else can calculate the correct values. In the example a daily timestamp is proposed, so when the certificate must be valid for one year X_0 will be hashed 365 times. This is not very safe because the Russian hackers can exploit their knowledge for an entire day. It is also possible to use a timestamp of one minute in which case X_0 must be hashed at most 527040 times (depending on whether it is a leap-year or not). In [Mic02] they claim that these calculations will take about one second which is very reasonable for both the server and the client, so even when a very small timestamp is used, Novomodo can be feasible.

Novomodo is not used by large CA's at the moment, although it seems to work better than both CRL's and OCSP. The certificate standard must be changed a bit because the Y_1 and X_{365} must be added in order to use Novomodo, but this will not have a large influence on the certificates.

12.5 Applications

Apart from the application of certificates in the world of web browsing, there are some other applications and interesting implementations of certificate cryptography.

12.5.1 The web of trust: PGP/GnuPG

In the early days of Pretty Good Privacy (1991), the goal of its creator Philip Zimmermann was to provide a way in which all people around the world could communicate freely without being proscecuted by governments. This required a decentralized system since a central server could be shut down, that is where the *web of trust* came into play.

The web of trust is the concept of people declaring trust in eachother in a way everyone else can validate. This is implemented by promoting every person in the web of trust to a certificate authority: everyone can sign everyone's public key, and everyone can check the signatures of everyone's public key. The only weakness of this system is again the question of trust: do you trust the key p_a belongs to Alice just because it has a certificate of Bob?

After being proscecuted for exporting munitions himself in 1996, Philip Zimmermann incorporated a new company which would later be merged with Viacrypt. A few years later PGP would become an official standard (RFC 2440) under the name OpenPGP. Not much later, PGP became closed source software and the GnuPG (GNU Privacy

Guard) project was started as an open source alternative. Up till today, GnuPG is still widely used by the open source community, not only online, but also in real life during *key signing parties* (see figure 12.6)

Figure 12.6: A typical keysigning party at Utrecht University

12.5.2 Client side certificates: VPN & SMTP

Up till now, certificates were only used to verify the identity of a server, but in some cases a server wants to verify the identity of a client which basically means: using certificates the other way around. Imagine the situation of a VPN server recieving a request for connection from a VPN client. How can the server determine if the client is authorized to connect to the VPN? By providing a certificate to the client, signed by the server itself! Once a client with a certificate signed by the key of the VPN server connects, the server can safely assume that this client is allowed to connect to the VPN.

Another application of client side certificates can be found in the SMTP protocol using TLS (basically a slightly different and improved version of SSL). Well configured SMTP servers are supposed to check the identity of a SMTP client before relaying an e-mail. In this scene, client side certificates can again be used to identify a SMTP client.

12.5.3 E-Government

Governments also possess keys which the use to sign official digital 'documents', for example the chip inside the new RFID-capable passports. By providing an encrypted signature over information inside the chip, customs officers all over the world can verify the integrity of the contents of the chip.

12.6 Other solutions

In this paper the basic way of authenticating a webserver is via a certificate which is signed by a Certificate Authority. Because the client trust the CA he will also trust that the webserver is who he claims to be and the public key from the certificate is the public key of the webserver. This is one way of solving the security issues involved with, for example, online banking and signing e-mails. There are other solutions for these security issues which are discussed in [LB04]. In this paper only PGP and IBE are mentioned.

12.6.1 PGP

PGP is explained in section 12.5.1 and is an alternative for the Certificate Authorities as described in this paper: with PGP there is not one large CA but there is a 'web of trust'. Because of this fundamental difference PGP is considered an alternative for Certificate Authorities, although both alternatives use 'certificates'.

12.6.2 Identity Based Encryption (IBE)

A certificate is needed to make sure that a given public key really belongs to the party you are communicating with and to make sure that that party is indeed who he claims to be. With IBE (Identity Based Encryption) the public keys are not just any random number of lots of bits but they are the identities of the party you want to communicate with. So Alice can encrypt an e-mail message to Bob with Bob's identity.

Now Bob can only read the message if he can indentify himself to the keyserver: he must prove that he is really Bob and when the keyserver believes him it will give Bob the secret key so Bob can decrypt the message he received from Alice. Big advantages are that the administrative costs are far less compared to Certificate authorities, no certificate lookup is needed before sending an e-mail and Alice can send encrypted messages to Bob while Bob does not yet have a private and a public key (and because of this Bob does not has a certificate either). Bob can request a private key at the keyserver: the keyserver can compute all private keys so it must be a very, very safe server which is a big disadvantage.

Another problem which can occur is when identity theft takes place. When someone else but Bob, say Oscar, can identify himself as Bob he can ask for the private key at the keyserver and read all secret messages which Alice sends to Bob. Of course there are a lot of security measures to minimize the chance of an identity theft, for example the principle of zero knowledge proofs.

So IBE is an alternative for Certificate Authorities but let's take a closer look at the part where the keyserver sends the private key to Bob: this part of the communication must be encrypted. Fortunately SSL was created for this purpose, but in SSL the idea of Certificate Autorities is quite essential... So actually IBE is not a complete alternative for CA's. Nevertheless it is a nice idea.

More information about IBE can be found on the website of Voltage Security Inc: [Inc07].

12.7 Summary

Certificates are used all over the (digital) world and are a part of lots of applications and protocols such as the Secure Socket Layer protocol. When a webserver has a certificate which is signed by a party the client trusts (a trusted third party) the client will believe that the webserver is not lying about his identity and the webserver is trusted. Most of the time a trusted third party is a Certificate Authority.

A webserver can request a certificate from a CA like VeriSign by sending a certificate signing request to the CA. When the CA has checked the identity of the webserver, a X.509 certificate is issued for a certain amount of time. The body of such a certificate contains fields like the expiration date, a serial number, who the CA is and what the public key of the webserver is. Next to the body, the certificate has a signature of the CA over the body so clients can check that the certificate is actually issued by the CA and can be trusted.

Quite a number of public keys of CA's are incorporated in webbrowsers so the user will automatically accept a certificate when it is signed by one of these CA's. This is very user friendly, but problems can occur when an attacker installs a fake CA key in the browser. Because a lot of public keys of CA's are distributed along with web browsers it is very important that CA's take good care of their secret key (also named 'root key'). Good security of the root key can be achieved by using sub-root keys and by using the root key as little as possible.

They only known attacks are not on the cryptographical principles of certificates but on the use of the certificates, like installing fake root keys in browsers or using MD5 weaknesses. Of course it is also possible to steal a private key, but then the certificate will be revoked.

Certificate revocation is reached by using Certificate Revocation Lists and the Online Certificate Status Protocol. They both have their advantages and disadvantages and new solutions are in development, like Novomodo.

There are some alternatives for using certificates of CA's like PGP, where each member acts like a CA and can sign the keys of other members, and Identity Based Encryption where no certificates are needed because of the use of identities as public keys. Unfortunately alternatives like IBE need secure communication and use SSL to achieve that. As can be seen in the beginning of this chapter, SSL uses certificates, so IBE uses something for which it wants to be an alternative.

Chapter 13

Quantum Factorisation

Written by *Bas den Heijer and Christiaan Ypma.*

RSA is an asymmetric encryption algorithm, based on the computational impossibility of factoring the modulus used in the algorithm. Over a decade ago, Shor proposed a quantum algorithm for factoring RSA moduli. By lack of quantum computers, the algorithm was not yet put into practice, but recently, researchers claimed the completion of a successful machine. What are the current quantum computers, and how safe is RSA in the decades to come?

13.1 The Quantum World

13.1.1 Quantum Mechanics

Quantum Mechanics are best described as "the physics of the really small", where "really small" is atomic and sub-atomic level ($< 10^{-10}m$). The 'normal' Newton laws do not apply at that scale. All things that happen behave in a more probabilistic way, the laws applied here are non-deterministic (in contrary to the for instance $force \times mass = acceleration$). For example, electrons would rapidly travel towards and collide with the nucleus when normal Newtonian laws would apply. However, the electrons normally remain in an unknown orbital path around the nucleus. As it is almost impossible to completely grasp this, because our senses are based on the macroscopic rules, it is also known as a nonsensical physics. This is acknowledged by a leading figure in quantum mechanics, Richard Feynman, who stated "I think I can safely say that nobody understands Quantum Mechanics". Why then are we, computer scientist, interested in quantum mechanics? In the last 30 years, the number of transistors (bits) per chip roughly doubled every 18 months, amounting to an exponentially growing power of classical computers (Moores law). Eventually this law will be violated, since the bit size will reach the limiting size of one atom in about 10 years. Around that point, we will have to handle quantum mechanics.

13.1.2 Quantum Computers History

The first thoughts about quantum computing were in the seventies, as in 1971 Holevo published an article in which he proved that n qubits cannot carry more information than n classical bits (Holevo's Thorem) [Hol73]. Later, in 1975, the Russian Poplavskii published an aricle which already showed the infeasibility of simulating a quantum computer on a classical system, due to the notion of superposition [Pop75]. Then, in 1981 Richard Feynman proposed a basic model for a quantum computer at the First Conference on the Physics of Computation at MIT. He gave general arguments supporting the idea that a quantum device would be exponentially faster than a classical computer.

In the years to follow a lot of theoretical research was done in this field, and different algorithms were discovered, like in 1994 Shor's algorithm, which allow one to render most current cryptographic methods useless, when a quantum computer of reasonable size is available, and a database search algorithm by Grover in 1996. In 1998 the first 2 qubit and later 3 qubit computer was working (and excecuted Grover) at Stanford.

13.1.3 Qubits and Gates

A qubit is the same as an ordinary bit, as in representing 0 or 1. The difference is, that it has at the same time a superposition of anything between 0 en 1, with a chance a. This superposition is one of the true powers we can use for specific algorithms. Another property of a qubit is entanglement, which is only explainable looking at multiple qubits in one register (for instance 2): $|\psi\rangle = \alpha|10\rangle + \beta|11\rangle + \gamma|00\rangle + \delta|01\rangle$, with the constraint $|\alpha|^2 + |\beta|^2 + |\gamma|^2 + |\delta|^2 = 1$. The notation used here is one common used by physicists for notating the state, it is called *Ket* notation. Thus, ψ is a vector in a two-dimensional complex vector space, where $\alpha|10\rangle + \beta|11\rangle + \gamma|00\rangle + \delta|01\rangle$ forms an orthonormal basis, called the computational basis. The variable α state the chance the state will be $|00\rangle$ when read.

David DiVincenzo stated five 'basic' criteria for realizing a quantum computer [DiV00] :

1. A scalable physical system with well characterized qubits
2. qubits can be initialized to arbitrary values
3. Long relevant decoherence times, much longer than the gate operation time
4. A universal set of quantum gates
5. A qubit-specific measurement capability

It is obvious that a system has to be scalable. But the next statement, well characterized qubits, is perhaps more difficult to explain. The physical parameters have to be known, interactions with other qubits and coupling to external fields used to manipulate the qubit. Looking at the second requirement, it is impossible to calculate anything without being able to have control over the starting values. The third criteria

is technically a challenge. Because of the quantum mechanics, a qubit will change state (decoherce) after a certain amount of time, becuese of influences of the enviroment. This influance can be minimilized, and has to be, to increase the time to be much longer than the gate operations, to allow computations. This brings us to the fourth criteria. To be able to calculate classic algorithms, a universal set of gates is needed, functioning with qubits. The last criteria implies that it has to be possible to (easily) read individual qubits, to know an outcome of an algorithm.

13.2 Integer Factorization

Integer factorization is the process of finding non-trivial divisors of a composite integer. The amount of words in this definition may mislead an unsuspecting reader into thinking it's easy to do this process, it is not. For large integers, the best published algorithm is the number field sieve [Pom96], which has a runtime of $O(e^{c(n)^{\frac{1}{3}}(logn)^{\frac{2}{3}}})$ for an n-bit number (and a positive c): the time required to factor an integer grows exponentially with the number of bits the number has. For large numbers, integer factorization is considered computationally infeasible.

13.2.1 RSA cracking

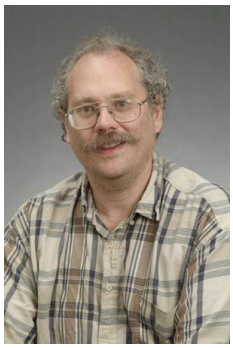
This property has made the RSA assymetric encryption protocol possible. As a service to the forgetful reader, a summary of the RSA key generation, encryption and decryption procedures [Tel07]:

- **Key generation:**
  1. Choose primes p and q.
  2. Calculate $n = p \cdot q$ and $\phi(n) = (p-1)(q-1)$
  3. Pick $e \in \mathbb{Z}^*_{\phi(n)}$
  4. Calculate $d = e^{-1}$ in $\mathbb{Z}^*_{\phi(n)}$

  The public key is (n, e). The private key is (n, d).
- **Encryption:** Encyption of x: $y = x^e$.
- **Decryption:** Decryption of y: $x' = y^d$. This is the original message: $y^d = x^{d^e} = x^{d \cdot e} = x^1 = x$.

Knowing e from the public key does not imply knowledge of the private value d. To calculate d, Oscar (and everybody else) would have to perform an algorithm in $\mathbb{Z}^*_{\phi(n)}$ which requires $\phi(n)$ which in turn requires p and q, the factors of n. Fortunately for the communicating parties, they're protected by the problematic nature of integer factorization. So should computational power ever grow large enough to perform this operation within a practical timespan, they'd just double the number of bits of n and

Casus 13.1: Peter Shor



Peter Shor (born August 14, 1959) won the Rolf Nevanlinna Prize for his factorization algorithm. This prize is awarded once every four years for outstanding contributions in Mathematical Aspects of Information Sciences by the same organisation that awards the Fields Medal for mathematics. Shor was working for AT&T Bell Laboratories when he published his paper on factorization on a quantum computer. Bell Laboratories was the place of origin for several revolutionary technologies including the transistor, the laser, the UNIX operating system and the C programming language. Currently he is a professor of applied Mathematics at MIT. Apart from being an award-winning computer scientist, Shor is also an aspiring poet:

If computers that you build are quantum,
Then spies of all factions will want 'em.
Our codes will all fail,
And they'll read our email,
Till we've crypto that's quantum, and daunt 'em.

rest peacefully again as the number of steps required to factor their new key just grew exponentially.

It's obvious that the safety of RSA depends on the validity of the difficulty assumptions of integer factorization. If a practical technique would be found (or was found) to factor integers quickly, this would make any message encrypted with RSA readable. In light of RSA's huge popularity world-wide this would have very serious implications: Crackers could do anything from reading your e-mail to stealing money from your bank account.

13.2.2 Shor's algorithm

While the world was resting peacefully, Peter Shor embraced the unintuitive nature of quantum computers to design an algorithm that can (theoretically at least) factor integers in a stretch of time polynomial in their number of bits [Sho94]. Algorithm 13.2 is a pseudo-code version of Shor's algorithm.

A couple of aspects deserve some remarks: The rather vague instruction "Find the order of a" will require a more concrete specification and will actually take exponential time with any known classical algorithm. Also, it is possible for the algorithm to terminate without returning an answer (fail).

This last property makes Shor's algorithm probabilistic. It is hard to prove what the probability is that r will be even and $a^{r/2} \not\equiv -1$ modulo N, but unless N is a power of a prime[1] or N is even (in which case 2 is a non-trivial divisor) the probability turns

[1]Efficient classical algorithms exist for the case that N is a power of a prime.

NonTrivialDivisors(N) :
  $a :=$ *random*(N) (* Random integer between 0 and N *)
  if gcd$(a, N) \neq 1$ **then return** gcd(a, N)
  else
    Find r, the order of a: $a^r = 1$
    if odd(r) **or** $a^{\frac{r}{2}} \equiv -1$ modulo N **then fail**
    else return gcd$(a^{\frac{r}{2}} - 1, N)$, gcd$(a^{\frac{r}{2}} + 1, N)$

Algorithm 13.2: Shor's algorithm for integer factorization.

out to be $1 - 1/2^{k-1}$, with k the number of prime factors of N (See [EJ96]). In the case of RSA this is means the probability of failure is less than or equal to $1/2$. This will not hinder it's feasibility much since repeating the algorithm multiple times for different values of a, the probability of failure shrinks exponentially (for instance smaller than the chance humanity will be eradicated by nuclear holocaust before you can finish reading the e-mail you're trying to decrypt).

Let's consider the mathematical basis for algorithm 13.2.

Theorem 13.1 *If NonTrivialDivisors(N) returns without failure, the returned value or values are non-trivial divisors of N.*

Proof. (Proof provided by [Lav03]) In the corner case that gcd$(a, N) \neq 1$, the result is a non-trivial divisor by definition of gcd(). If however the algorithm terminates as planned a and r are such that

$$a^r \equiv 1 \,\%\, N,$$

and since r is even

$$a^{r/2} \equiv y \,\%\, N$$

for some $y \neq \pm 1$. Note that y satisfies

$$y^2 - 1 = (y - 1)(y + 1) \equiv 0 \,\%\, N,$$

which means that N devides $(y - 1)(y + 1)$. Since $1 < y - 1 < y + 1 < N$, N cannot divide $y - 1$ and $y + 1$ seperately. This means that $y - 1$ and $y + 1$ must have factors of N (that yield N when multiplied), and therefore gcd$(y + 1, N)$ and gcd$(y - 1, N)$ must be non-trivial factors of N. △

This has actually been known for several centuries and did not constitute to much of a threat to RSA until Shor found a way to calculate r quickly using a quantum computer. He uses a trick that begins as follows: the quantum register is intitialised to a superposition of all states with equal probability, this is easy. Then he computes $f(|x\rangle)| = a^x$ modulo N, by repeated squaring, much in the same way as classical computers would, except that since the input is a superposition of all possible x, the result is that the quantum register is now in a superposition of every power of a modulo N. This effect is sometimes called *quantum parallelism*.

If the register was read at this stage, the result would be a random power of a, which is worthless. The powers of a are not interesting by themselves, the interesting thing is the period by which they occur. Shor devised a way to perform a Quantum Fourier transform in polynomial time using a polynomial amount of gates[2], this operation basically yields the desired period r or information about r that can be used to make a second attempt more likely to succeed.

Since it's publication several improvements have been made to Shor's algorithm and Shor himself suggests several avenues for investigation. It's running time depends on a couple of implementation choices but has a worst case of $O(n^3)$ and $O(n)$ quantum gates are required to factor an integer of n bits. The number of required qubits is about $2n$.

13.2.3 Complexity classification

So integer factorizaton does turn out to require only polynomial time! Not quite, since Shor's algorithm is probabilistic it is not a proof that integer factorization is P[3]. Actually integer factorization is in the complexity class *BQP*.

Definition 13.2 *The computational complexity class BQP is the class of decision problems solvable by a quantum computer in polunomial time with an error probability of less than* $1/2$ *for all inputs. BQP stands for "Bounded error, Quantum, Polynomial time".*

The number $1/2$ in the definition is arbitrary: as long as the majority of the results are correct or if the incorrect results are easily identifiable (as is the case in Shor's algorithm), it's possible to construct an algorithm with an arbitrary low chance of succes by repeating the original algorithm a constant number of times and combining the results.

Shor also explains in his paper how to perform the discrete logarithm in polynomial time on a quantum computer using a very similar approach. The discrete logarithm is closely related to integer factorisation: the security of the assymetric encryption protocol ElGamal depends on it's infeasibility and it too is in BQP.

13.3 Quantum Computer Implementations

13.3.1 Qubit Implementations

There are currently a lot of diffenent ways to implement qubits, trying to obey all criteria of DiVincenzo.

1. *Trapped ion* Here an ion is used to store quantum information. The ion is "trapped" in a electro-magnetic field, hence the name.

[2]The classical equivalent is the Discrete Fourier transform, which is possible to execute and variation are actually used a lot in for example the JPEG compression alogorithm, but takes exponential time.

[3]It's also suspected that integer factorization is not NP-complete, so no millenium prize for Shor, fortunatly for him he can just steal their money as soon as he has a quantum computer.

2. *Quantum Dot* It is a semiconductor nanostructure that can confine electrons in all three directions using a electrostatic potentials. This creates a really precise readable qubit, and the use of silicon simplefies creation (standard lithografy cna be used).
3. *Nuclear magnetic resonance (NMR)* This is the way IBM build a quantum computer. It is comparable to the way MRI (Magnetic Resonance Imaging) works. It uses the spin-states of molecules as qubits.
4. *Combination (Solid State NMR)* Used by the so-called "Cane Computer", by a large Australian quantum computing research group [4].
5. *Adiabatic quantum computation* This is way beyond the scope of computer science, and according to many quantum experts currently impossible. It is said to be used in th quantum computer of D-Wave, see for more information casus 13.3.

These five are not all possiblilities, there is lot more options being researched. The diversity of different options shows that the field of quantum computing is still in its infancy stage.

13.3.2 Implementations today

In 2007, IBM demonstrated a 7 qubit quantum computer, based on nuclear magnetic resonance. [Van01]. On this computer was the Shor Algorithm run, using a N of 15. After a stunning 720 ms, the answer (a surprising 5 and 3) was given. This quantum computer didn't have the *entanglement* property, so it wasn't really a quantum computer, without this property it couldn't scale.

In 2007 D-Wave systems presented their Orion computer, a 16 qubit quantum computer. It played a sudoku game during the presentation. The problem with this computer is that D-Wave didn't show the "real thing". More on that in casus 13.3.

There are a lot more science groups around the globe which have working NOT-gates, or other *parts* which are needed for a quantum computer, but not a working one (i.e. able to run an algorithm).

So with todays technology, there are no worries for the users of the RSA algorithm (at this time that includes almost everybody). This is also because for Shor's implementation of the factoring algorithm, you need to build a new gate set (read: Quantum computer) for every different N you want to factorize. Perhaps not very feasible.

13.3.3 Implementations tomorrow

There is at the moment a lot of academic research in the field of quantum mechanics, so there will be more breakthroughs in the future. Even here in the tiny Netherlands the TU Delft came recently in the news demonstrating a controlled-NOT quantum gate [Pla07]. It is with no doubt that one day a working quantum computer will emerge, but it will take time. Big companies (like IBM) fund a lot of research, and taking into account

[4] http://www.qcaustralia.org/home.htm

Casus 13.3: About D-Wave

In januari 19, 2007, the world was surprised by the presentation of a commercialy viable 16 qubit quantum computer. The company responsable for this, was D-Wave Systems, Inc. This company, a spin-off from the departement of Physics at the University of British Columbia (Canada), presented later that year, november 12 at the SCO7 Conference, even a 28 qubit computer, which featured an image recognition algorithm. They have plans for a 512 and a 1024 qubit system in 2008.
There is a lot of criticism and disbelief, mainly because D-Wave hasn't made there system public (yet), and there hasn't been any peer review. The technlogy they claim to use (superconducting adiabatic quantum computer) requires, according to Scott Aaronson, a major breakthrough in physics. Read more on his blog[a]. But multiple (large) investors have shown their interest by investing millions of dollars, so they probably are convinced enough by the presentations (or maybe some inside information).

[a] http://scottaaronson.com/blog

the increasing interest in quantum computing, this funding will increase and so will the speed of research.

In China there is at the moment also a large research group trying to create a working quantum computer. A lot of Chinese researchers have re-immigrated to China, because of the higher wages made possible by the Cinese goverment. Also rumors go that there is secret goverment project alongside this research group.

There is offcourse the D-Wave computer, but taking in account all the critics, and the ignoring the commercial chitchat, it is probably an "extended" convential computer, so RSA-users have nothing to fear from D-Wave systems.

A perhaps far-fetched thread is the National Security Agency (NSA), which has, being part of the U.S. Departement of Defense, virtually unlimited funding and over the years attracted the best researchers in the world. So they could already have a quantum computer, or really fast develop one when somewhere in the world a research group publishes a good scalable working example.

Chapter 14

Steganography and Steganalysis

Written by *J.S. de Wit.*

In this digital age we have plenty of cryptographic techniques to effectively conceal information from all but a few selected recipients who have a key or code to reveal the hidden content. Assuming outsiders dont have the means to obtain the decryption key from one of the people who were trusted with it and large enough keys are used, we now consider such secrets computationally safe from attackers. It can, however, be preferable to hide the fact that any exchange of secrets is going on, e.g. in (corporate) espionage. This is what the science (or as some prefer; the art) of steganography deals with; hiding the fact that secrets are transferred from one party to another. Naturally when dealing with human perception one can think of many different methods of fooling other people and hiding the actual information being sent. Documented primitive methods go back as far as the time of the Roman Empire, where information was tattooed on the scalps of soldiers after which they let their hair grow back over it, to more modern techniques in WWII and the cold war. In discussing steganography we consider all these methods primitive and we will focus on the kind of steganography that can only be accomplished with the use of computers.

14.1 Hiding Information

Like the more ancient forms of steganography, in computers information can be hidden in plain text too, but this is technologically less interesting. We will look at cases where the so-called *cover medium* is a binary file. The cover medium is the file in which the information will be hidden. This can be an audio, video, still image or other file. The type of medium isnt essential to most techniques, since most methods are based on bit manipulations. The software using the techniques will of course have to know how to handle specific media in order to avoid destroying the perception of the cover medium. For our examples we will use images, since they are easier to embed in papers.

14.1.1 Hiding in Plain Sight

One method slightly less primitive than methods is to just add the information in the cover medium that can just be seen by people if they know where to look. You could write some text in the smallest font you can select in the bottom few pixels of a random image that can only be read when zooming in on the affected area. A nice example is the rumors of hidden messages that could be heard when playing certain Beatles records backwards, hinting at the death of the real Paul McCartney[1]. True or not, this is an existing technique to hide information in audio streams.

14.1.2 Redundant Bits

While embedding data in plain sight can be an easy (and fun) way to hide data in different media, you always run the risk of people accidentally discovering it. This is why we prefer to hide information in a way undetectable by the human eye/ear. One way to do this is by replacing *redundant bits* in the cover medium. What bits are considered redundant depends on the medium used. In wave audio files there is data that cant even be heard by normal humans. Changing bits in these areas of the file cannot be detected, but because our hearing is not as developed as our sight, also small local changes in audible parts of audio media will not result in perceptual detection. Another advantage of using audio media is that there are a lot of redundant bits to be found. Just a few seconds of uncompressed audio contains several million bits. On bitmap images a good choice would be the least significant bit (LSB) of each pixel. This does require the number of bits per pixel not to be to low. If only 4 bits are used a change in even the least significant bit can cause visual distortion detectable by the human eye. On 8 bit (grayscale) bitmaps and upwards these distortions become more and more negligible. Unfortunately it has been shown by Johnson and Jajodia[JJ98] that all palette-based images leave distortions that can relatively easily be detected. Filtering the image to just show the LSB can reveal distortion not seen in the original picture. When trying to embed information in the bits of an image, the JPEG format is a more suitable candidate. I will not discuss the precise working of the JPEG algorithm, for the purposes of this paper it suffices to know that not the color values of bits are stored, but rather a characterization of blocks of 8x8 adjacent pixels is made, calculated with a *discrete cosine transform* (DCT). Changing the least significant bit of such a quantized DCT value causes a small distortion over all 64 bits in the block. This prevents simple detection techniques like LSB extraction. Even though every LSB can be changed without noticeably altering the appearance of the medium, changing some bits, for example in low-density areas of the medium, is discouraged. Such bits can be marked as locked so that they wont be selected for replacement.

[1]Conspiracy theorists suggest that Paul McCartney died in 1964 and was replaced by a Paul McCartney look-a-like. This is most likely just an urban legend, but if it were true it would be the greatest hoax ever (assuming Neil Armstrong did really land on the moon in 1969).

14.1.3 Redundant Bit Selection

Because in most normal situations the secret will be a lot smaller than the cover medium (it actually *needs* to be for good steganography) we dont have to use all the redundant bits of the cover medium. If we dont want to bother with bit selection we can just append the secret with random bits to match the number of redundant bits. But as we will see later on, using all redundant bits actually makes the process very susceptible to a certain method of detection. Another simple method is to just select the first n redundant bits, where n is the length of the secret. This again could be more easily detected by LSB extraction and visual detection than other methods of selection. A better is to select random LSBs throughout the medium, but this will require extra meta data to be embedded in the file for reconstruction of the secret. Another possibility is to have the selection of what redundant bits to change depend on a user-supplied password upon encryption. Once the bits have been selected there is little more to do than to replace the redundant bits with the bits from the secret. The selection of the bits candidate for replacement is therefore the most important step of steganographic methods using redundant bits. How the redundant bits selected for replacement are chosen can be done in a variety of ways. As mentioned some methods are better than others, but there is no standard all steganographic software adheres to, this is up to the programmer. We will discuss one more technique to avoid a specific method of steganography detection further on.

14.2 Steganography Detection

Where we have people trying to hide information, there exist people who want to recover it. So along with the creation of steganographic techniques came methods of detecting such content. Important is that the goal of steganography detection is purely to discover *that* some hidden information has been passed along. Naturally, the goal of an individual will likely be to read the hidden information. But once some secret interaction has been detected, this results in a cryptographic problem, which is beyond the scope of this paper.

14.2.1 Perceptual Detection

A simple, but not very interesting, method of detection is perceptual detection by humans. If a medium seems garbled for no apparent reason (e.g. clear image distortion in a certain section of an image), one can suspect a medium may contain hidden information and nominate that medium for further, more sophisticated, investigation. Another trivial reason to suspect hidden data is to find the original cover media. If someone has two seemingly identical images, but one differs slightly from the other in a few bits or in byte size, the original was modified. Therefore it is good practice to destroy cover media when done with the encryption process.

14.2.2 Signature Scanning

On the World Wide Web many pieces of software for steganographic encyption and decryption can be found. Many of these leave a detectable signature in the encoded medium. Some do this on purpose, for example a lot of home made software where the author leaves his calling card in the source of the medium. Or commercial products where a freeware or trial version of the software will leave such signatures, while the full paid version does not (or at least not on purpose). To see whether or not unintentional chunks of signature code exist the encoding process of a piece of software, but for several popular programs such artifacts have been found.

14.2.3 Steganalysis

Since anyone can write a piece of software and publish it on the internet it is not possible to know all software and inspect them. What we want is to find a method that can generally be applied, independent from the software that was used for the possible encryption. This can be done with statistical analysis of the medium. One such analysis method we will examine is checking the relative frequencies of colors in digital images, assuming the steganographic technique used changes (some of) the least significant bits in an image. Say that n_i and n_i^* are the number of occurrences of color indexed i before and after the encryption. If $n_{2i} > n_{2i+1}$ then on average more pixels will be changed from color n_{2i} to n_{2i+1} than the other way around. This means that

$$|n_{2i} - n_{2i+1}| \geq |n_{2i}^* - n_{2i+1}^*|$$

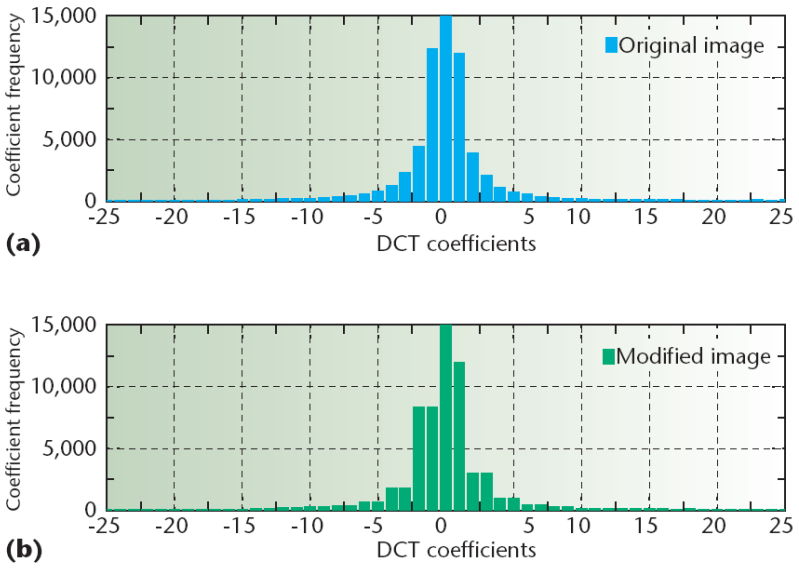
So changing least significant bits will lower the difference between the frequencies of adjacent colors. When we examine the case where we embed a secret that uses all redundant bits we find something interesting. If we replace a bit with a random bit there is a 50% chance that it will change, so when replacing all redundant bits we effectively change about half of them. Now for all adjacent colors we have

$$n_{2i}^* = 0.5 * n_{2i} + 0.5 * n_{2i+1}$$

$$n_{2i+1}^* = 0.5 * n_{2i+1} + 0.5 * n_{2i}$$

$$n_{2i}^* = n_{2i+1}^*$$

This works not only for images where color bits are directly changed, but also for the JPEG format. Here we dont examine the frequencies of colors, but the frequencies of DCT coefficients. Below we see graphs of the DCT coefficient frequencies of the unmodified image (a) and the same image after steganographic embedding (b).



We see that the relative frequency of adjacent DCTs has significally decreased. Of course the smaller the secret hidden, the smaller the change in adjacent frequencies, but there will still be changes in the frequencies. Westfeld and Pfitzmann[WP99] devised a method where they used a χ^2 test to measure if a frequency distribution differs from its expected distribution, shown below.

$$y_i^* = \frac{n_{2i} + n_{2i+1}}{2}$$

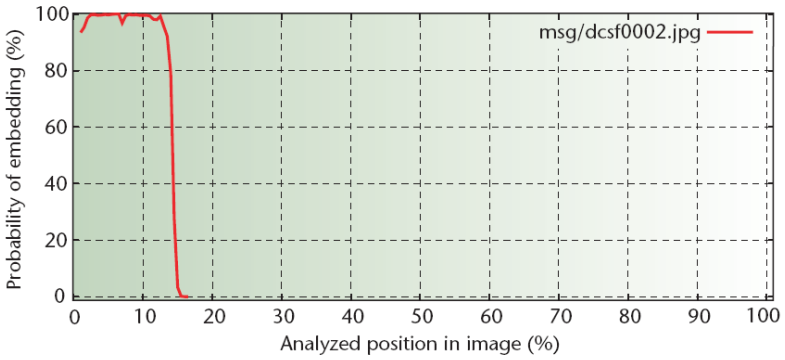
$$y_i = n_{2i}$$

$$\chi^2 = \sum_{i=1}^{v+1} \frac{(y_i - y_i^*)^2}{y_i^*}$$

where y_i^* is the expected distribution, y_i is the observed distribution and v is the number of categories in the DCT histogram. The probability p that the two distributions are equal is

$$p = 1 - \int_0^{\chi^2} \frac{t^{(v-2)/2} e^{-t/2}}{2^{v/2}\Gamma(v/2)} dt$$

where Γ is the Euler gamma function. The probability that an image contains steganographic content is determined by calculating p for a sample starting at the beginning of the image and increasing the sample size for each subsequent measurement. The results of such an analysis can be seen below.



Steganalysis has been proven to be a very powerful tool in discovering steganographic content based on LSB replacement.

14.2.4 Avoiding Steganalysis

Knowing how steganalysis works has prompted software that was vulnerable to it to change the way the redundant bits are selected. Before the encryption takes place the cover medium can be analyzed in the same way the steganalysis takes place. Redundant bits are now no longer selected randomly or according to a predefined function, but are selected in such a way so that the observed frequencies in the cover medium are maintained. This way steganalysis will be far less likely to discover the presence of hidden data in your media.

14.3 Conclusion

The fields of steganography and steganalysis are a cat-and-mouse game with steganalists trying to find new methods of detecting new and more refined methods of steganography. With the vasteness of the World Wide Web, an individual should feel safe that a message properly encoded inside some medium on his or her website will not be stumbled upon by anyone any time soon. Big companies, governments or other parties with higher stakes, however, should not get a false sense of security. Even though something can be hidden from human perception with almost absolute guarantee, techniques and software for detection of steganographic content do exist and new methods are also continuously created and refined.

Chapter 15

Multivariate Cryptography

Written by *Merel Rietbergen.*

In many public key cryptosystems, such as RSA, the plaintext is represented by a single number x. In such systems the encryption is done through a univariate computation, that is, a function which takes a single number as input. Therefore an attacker faces the challenge of solving an equation with a single unknown.

In this chapter we will discuss multivariate cryptosystems, in which the plaintext is represented by several numbers $x_1, \ldots, x_n$ and encryption is done through multivariate quadratic formulas. An attacker of such a cryptosystem must solve a system of quadratic equations with n unknowns.

The idea of using multivariate quadratic polynomials in cryptography is based on the fact that the general problem of solving a system of multivariate quadratic equations is NP-complete. However this fact does not guarantee that a multivariate cryptosystem is secure. To illustrate this we will discuss the multivariate cryptosystem HFE and an attack on this system.

We begin with some preliminaries in section 15.1. In section 15.2 we will discuss multivariate quadratic cryptography and the HFE cryptosystem. We finish this chapter with a desciption of an attack on HFE using Gröbner bases in section 15.3.

15.1 Preliminaries

Before we can discuss multivariate quadratic cryptography we need to introduce some mathematical concepts, namely finite fields, affine transformations and multivariate quadratic polynomials.

15.1.1 Finite Fields

We begin with the definition of fields, and related terminology and properties, since multivariate cryptography is based on this concept.

Definition 15.1 *Let $\mathbb{F}$ be a set of q elements. $\mathbb{F}$ is a* field *if $\mathbb{F}$ has an additive operation $+ : \mathbb{F} \times \mathbb{F} \to \mathbb{F}$ and a multiplicative operation $\cdot : \mathbb{F} \times \mathbb{F} \to \mathbb{F}$ such that the following axioms hold:*

- *Additive commutativity: $\forall a, b \in \mathbb{F} : a + b = b + a$;*
- *Additive associativity: $\forall a, b, c \in \mathbb{F} : (a + b) + c = a + (b + c)$;*
- *Additive identity: $\exists e \in \mathbb{F} : \forall a \in \mathbb{F} : a + e = a$, we denote this e by 0;*
- *Additive inverse: $\forall a \in \mathbb{F} : \exists a' \in \mathbb{F} : a + a' = 0$, we denote this a' by $-a$;*
- *Multiplicative commutativity : $\forall a, b \in \mathbb{F} : a \cdot b = b \cdot a$;*
- *Multiplicative associativity: $\forall a, b, c \in \mathbb{F} : (a \cdot b) \cdot c = a \cdot (b \cdot c)$;*
- *Multiplicative identity: $\exists e \in \mathbb{F} : \forall a \in \mathbb{F} : a \cdot e = a$, we denote this e by 1;*
- *Multiplicative inverse: $\forall a \in \mathbb{F}^* : \exists a' \in \mathbb{F}^* : a \cdot a' = 1$, for $\mathbb{F}^* := \mathbb{F} \backslash \{0\}$, we denote this a' by a^{-1};*
- *Distributivity: $\forall a, b, c \in \mathbb{F} : (a + b) \cdot c = a \cdot c + b \cdot c$ and $a \cdot (b + c) = a \cdot b + a \cdot c$.*

The number of elements of a field $\mathbb{F}$ is called the *cardinality* of $\mathbb{F}$ and is denoted by $|\mathbb{F}|$.

Definition 15.2 *If a field $\mathbb{F}$ has finite cardinality q then $\mathbb{F}$ is a* finite field*, denoted by $\mathbb{F}_q$.*

The cardinality of a finite field is always a prime or a power of a prime. For every prime power q^n there exists exactly one finite field, denoted by $\mathbb{F}_{q^n}$. In every finite field $\mathbb{F}_q$ it holds that $\forall x \in \mathbb{F}_q : x^q = x$ and $\forall x \in \mathbb{F}_q \backslash \{0\} : x^{q-1} = 1$.

The smallest positive number p such that $\underbrace{1 + 1 + \ldots + 1}_{p \text{ times}} = 0$ in $\mathbb{F}_q$ is called the *characteristic* of $\mathbb{F}_q$. If no such p exists then $\mathbb{F}_q$ has characteristic 0. The characteristic of a finite field is always a prime number.

Definition 15.3 *If a subset $\mathbb{F}'$ of the elements of field $\mathbb{F}$ is itself a field then $\mathbb{F}'$ is a* subfield *of $\mathbb{F}$.*

For example, the field of real numbers $\mathbb{R}$ is a subfield of the field of complex numbers $\mathbb{C}$. For field $\mathbb{F}_{q^n}$, where q is a prime, there exists a subfield $\mathbb{F}_{q^m}$ for every m dividing n.

Definition 15.4 *If $\mathbb{F}'$ is a subfield of field $\mathbb{F}$ then $\mathbb{F}$ is a* field extension *of $\mathbb{F}'$.*

Every field extension of field $\mathbb{F}$ has the same characteristic as $\mathbb{F}$. Every element of the field extention $\mathbb{F}_{q^n}$ is of the form

$$a_{n-1}t^{n-1} + \ldots + a_1 t + a_0$$

with $a_i \in \mathbb{F}_q$. E.g. every complex number $c \in \mathbb{C}$ is of the form $a_1 i + a_0$ with $a_1, a_0 \in \mathbb{R}$. Every element $b \in \mathbb{F}_q^n$ can be represented as $(b_1, \ldots, b_n)$ where $b_i \in \mathbb{F}_q$. Therefore there exists an isomorphism onto the n-dimensional vector space $\mathbb{F}_q^n$ over $\mathbb{F}_q$ for every field extension $\mathbb{F}_{q^n}$ of field $\mathbb{F}_q$, namely the canonical bijection:

Definition 15.5 *Let $\mathbb{F}_{q^n}$ be a field extension over $\mathbb{F}_q$ and let $\mathbb{F}_q^n$ be the corresponding vector space. Then isomorphism $\phi : \mathbb{F}_{q^n} \to \mathbb{F}_q^n$ with*

$$\phi(a) := b, \text{ and } b_i := a_{i-1} \text{ for } 1 \leq i \leq n,$$

is called the canonical bijection *between $\mathbb{F}_{q^n}$ and $\mathbb{F}_q^n$.*

The inverse of the canonical bijection is denoted by $\phi^{-1} : \mathbb{F}_q^n \to \mathbb{F}_{q^n}$.

The dimension (n) of the vector space, over field $\mathbb{F}_q$, corresponding to the extension field $\mathbb{F}_{q^n}$ is called the *degree* of the extension.

15.1.2 Affine Transformations

Another mathematical concept that plays a role in multivariate cryptography is that of affine transformations.

Definition 15.6 *An* affine transformation *of a vector space V^n to a vector space V^m is a map $T : V^n \to V^m : \vec{x} \mapsto A\vec{x} + \vec{b}$, where A is an $m \times n$ matrix with matrix elements $a_{ij} \in V$, and $\vec{b}$ is an m-dimensional vector with vector elements $b_i \in V$.*

In multivariate cryptography we want to use affine transformations which have an extra property, namely that of invertibility.

Definition 15.7 *If the matrix A in affine transformation T is invertible, which requires that $n = m$, then T is an* invertible affine transformation.

Note that an invertible affine transformation is a bijection. We denote the set of invertible affine transformations by Aff^{-1}. The inverse of invertible affine transformation $T(\vec{x}) := A\vec{x} + \vec{b}$ is itself an affine transformation $T^{-1}(\vec{y}) := A^{-1}(\vec{y} - \vec{b}) = A^{-1}\vec{y} - A^{-1}\vec{b}$.

15.1.3 Multivariate Quadratic Polynomials

A *multivariate quadratic polynomial* in n variables $x_1, \ldots, x_n$ over a field $\mathbb{F}_q$ is defined as follows:

$$p(x_1, \ldots, x_n) := \sum_{1 \leq j \leq k \leq n} \gamma_{j,k} x_j x_k + \sum_{j=1}^{n} \beta_j x_j + \alpha,$$

with *coefficients* $\gamma_{j,k}, \beta_j, \alpha \in \mathbb{F}_q$. A product of variables $x_i^r x_j^s$, with $1 \leq i \leq j \leq n$ and $r, s \in \{0, 1\}$, is called a *monomial*, and a monomial together with its coefficient is called a *term*.

A polynomial can be written in different ways, e.g. $2x^2 + x + 5$ and $x + 5 + 2x^2$. To ensure a polynomial is always written in the same way we need a convention for the order of the terms. We define our lexicographic ordering $\succ$ of monomials for multivariate quadratic equations as:

$$\begin{aligned} x_i x_j \succ x_{i'} x_{j'} \quad &\text{iff} \quad i < i', \text{ or } i = i' \text{ and } j < j' \\ x_i x_j \succ x_k \succ 1 \quad &\text{for all} \quad 1 \leq i, j, k \leq n \\ x_i \succ x_{i'} \quad &\text{iff} \quad i < i' \end{aligned}$$

where $i < j$. Therefore the terms are ordered by their monomials as follows:

$$x_1^2 \succ x_1x_2 \succ \ldots \succ x_1x_n \succ x_2^2 \succ x_2x_3 \succ \ldots \succ x_{n-1}x_n \succ x_n^2 \succ x_1 \succ x_2 \succ \ldots \succ x_n \succ 1.$$

The $\succ$-largest monomial of a polynomial f is called the *leading monomial* and is denoted by $\mathrm{lm}(f)$, e.g. $\mathrm{lm}(2x^2 + x + 5) = x^2$. The coefficient of the leading monomial is called the *leading coefficient* and is denoted by $\mathrm{lc}(f)$, e.g. $\mathrm{lc}(2x^2 + x + 5) = 2$. The term consisting of the leading monomial and its leading coefficient is called the *leading term* and is denoted by $\mathrm{lt}(f)$, e.g. $\mathrm{lt}(2x^2 + x + 5) = 2x^2$.

A system of multivariate quadratic polynomials is formed by m multivariate quadratic polynomials in n variables $x_1, \ldots, x_n$ over a field $\mathbb{F}_q$:

$$\begin{aligned}
p_1(x_1, \ldots, x_n) &:= \sum_{1 \le j \le k \le n} \gamma_{1,j,k} x_j x_k + \sum_{j=1}^{n} \beta_{1,j} x_j + \alpha_1 \\
&\vdots \\
p_i(x_1, \ldots, x_n) &:= \sum_{1 \le j \le k \le n} \gamma_{i,j,k} x_j x_k + \sum_{j=1}^{n} \beta_{i,j} x_j + \alpha_i \\
&\vdots \\
p_m(x_1, \ldots, x_n) &:= \sum_{1 \le j \le k \le n} \gamma_{m,j,k} x_j x_k + \sum_{j=1}^{n} \beta_{m,j} x_j + \alpha_m
\end{aligned}$$

with $\gamma_{i,j,k}, \beta_{i,j}, \alpha_i \in \mathbb{F}_q$.

The vector $\mathcal{P} := (p_1, \ldots, p_m)$ is a *polynomial vector* and $\mathcal{P} \in \mathcal{MQ}(\mathbb{F}_q^n, \mathbb{F}_q^m)$, where $\mathcal{MQ}(\mathbb{F}_q^n, \mathbb{F}_q^m)$ is the class of polynomial vectors of m polynomials in n variables over $\mathbb{F}_q$. If $m = n$ then $\mathcal{MQ}(\mathbb{F}_q^n, \mathbb{F}_q^n)$ is denoted by $\mathcal{MQ}(\mathbb{F}_q^n)$ for simplicity.

15.2 Multivariate Quadratic Cryptography

Using the mathematical concepts introduced in 15.1 we can define the general multivariate quadratic cryptosystem and how encryption and decryption work.

15.2.1 $\mathcal{MQ}$-Trapdoor

We create a public key multivariate cryptosystem by introducing a trapdoor $(S, \mathcal{P}', T)$, i.e. a one-way function, using multivariate quadratic polynomials over a field $\mathbb{F}_q$. The trapdoor $(S, \mathcal{P}', T)$ is an $\mathcal{MQ}$-trapdoor if S is an invertible affine transformation $S : \mathbb{F}_q^n \to \mathbb{F}_q^n$, T is an invertible affine transformation $T : \mathbb{F}_q^m \to \mathbb{F}_q^m$ and $\mathcal{P}'$ is a polynomial vector of the class $\mathcal{MQ}(\mathbb{F}_q^n, \mathbb{F}_q^m)$.

In a public key multivariate cryptosystem with $\mathcal{MQ}$-trapdoor $(S, \mathcal{P}', T)$, illustrated in Figure 15.1, the private key is the triple $(S, \mathcal{P}', T)$. The public key is a system of

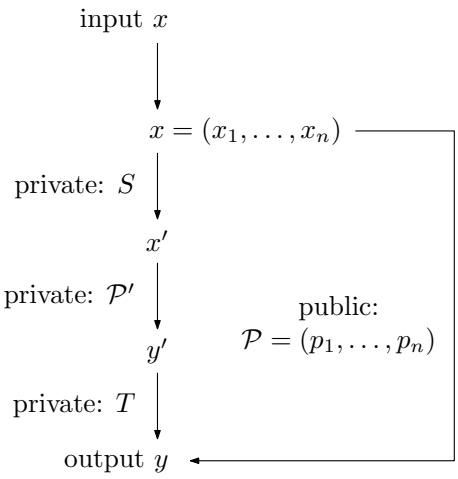


Figure 15.1: Graphical representation of the $\mathcal{MQ}$-trapdoor

polynomials $\mathcal{P} \in \mathcal{MQ}(\mathbb{F}_q^n, \mathbb{F}_q^m)$, which can be obtained by using a vector $\vec{x} \in \mathbb{F}_q^n$ of variables $(x_1, \ldots, x_n)$ as input for the $\mathcal{MQ}$-trapdoor, i.e. by evaluating $T \circ \mathcal{P}' \circ \mathcal{S}$ at formal variable $\vec{x} \in \mathbb{F}_q^n$.

On inspection of the private key $(S, \mathcal{P}', T)$ and the public key $\mathcal{P} = (p_1, \ldots, p_n)$ we see that the component of the private key $\mathcal{P}'$, sometimes called the *central polynomials*, and the $\mathcal{P}$ are both a system of polynomials of the class $\mathcal{MQ}(\mathbb{F}_q^n, \mathbb{F}_q^m)$. As mentioned before, solving such systems is an NP-complete problem. For the public key we are very happy with this property, since it will make it hard for an attacker to find a solution. However, we need to be able to invert the central polynomials $\mathcal{P}'$ during decryption. How this problem is solved is explained in 15.2.3.

15.2.2 Encryption

To encrypt a message $\vec{x} = (x_1, \ldots, x_n) \in \mathbb{F}^n$ we use the public key $\mathcal{P}$ and simply compute $\vec{y} = \mathcal{P}(\vec{x})$. Note that for any plaintext $\vec{x}$ there exists only one ciphertext $\vec{y} \in \mathbb{F}^m$. However, since $\mathcal{P}$ is not necessarily injective, a ciphertext $\vec{y}$ may have several preimages. Below we shall explain how a redundancy $\tilde{x}$, computed during encryption, allows the receiver to choose the correct preimage during decryption. This redundancy $\tilde{x}$ is computed using some hash function $H : \mathbb{F}^n \to \{0,1\}^h$, where $h \in \mathbb{N}$ is the length of the hash string. Therefore the encrypted message is a pair $(y, \tilde{x}) \in \mathbb{F}^m \times \{0,1\}^h$.

The frame below shows an overview of encryption in multivariate cryptosystems.

Encryption of $\vec{x} \in \mathbb{F}_q^n$
- Compute $\vec{y} = \mathcal{P}(\vec{x})$ - Compute hash string $\tilde{x} = H(\vec{x})$ Encrypted message: $(\vec{y}, \tilde{x})$

15.2.3 Decryption

To decrypt a message we need to invert the components of the private key. Since S and T are invertible affine transformations they can be easily inverted. To be able to easily invert the polynomial vector $\mathcal{P}'$ we need to know how it is constructed. However the construction method for $\mathcal{P}'$ varies between different multivariate cryptosystems. We will show the construction method for one cryptosystem, namely HFE, in 15.2.4. We will assume that the person who has the private key knows how its central polynomials were constructed and can therefore invert it.

As mentioned above $\mathcal{P}$ is not necessarily injective, and if it is not then neither is $\mathcal{P}'$. Thus when we are decrypting a message $(\vec{y}, \tilde{x})$ and have computed $Y = T^{-1}(\vec{y})$, we may find that there are several $X \in \mathbb{F}_q^n$ for which $\mathcal{P}'(X) = Y$. We can collect these Xs in a set $Q = \{X : \mathcal{P}'(X) = Y\}$ and compute $\vec{x}' = S^{-1}(X)$ for every $X \in Q$. This will give us a set $Q' = \{\vec{x}' = S^{-1}(X) : X \in Q\}$ of preimages of $\vec{y}$. Now we can use the redundancy $\tilde{x}$ that was computed during encryption to choose the correct preimage from the set Q' by computing $H(\vec{x}')$ for every $\vec{x}' \in Q$, where H is the same hash function as used during encryption.

The frame below shows an overview of decryption in multivariate cryptosystems.

Decryption of $(\vec{y}, \tilde{x})$
- Compute $Y = T^{-1}(\vec{y})$ - Compute $Q = \{X : \mathcal{P}'(X) = Y\}$ - Compute $Q' = \{\vec{x}' = S^{-1}(X) : X \in Q\}$ - Select $\vec{x}' \in Q'$ such that $H(\vec{x}') = \tilde{x}$

15.2.4 HFE: Hidden Field Equations

The multivariate quadratic cryptosystem HFE, i.e. Hidden Field Equations, was proposed by Jacques Patarin at Eurocrypt 96 [Pat96]. For simplicity we shall discuss basic HFE over a field with cardinality and characteristic 2, namely $\mathbb{F}_2$, and we shall take the multivariate quadratic polynomials of class $\mathcal{MQ}(\mathbb{F}_2^n)$. Regrettably, more complex fields and classes of polynomial vectors are beyond the scope of this chapter, however this is discussed to a certain extent in [Wol05].

In HFE the central polynomials are constructed using a univariate polynomial over the field extension $\mathbb{F}_{2^n}$.

Definition 15.8 *Let $\mathbb{F}_2$ be a field and let $\mathbb{F}_{2^n}$ be its n-th degree extension. Let $f(X) : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ be a univariate polynomial over $\mathbb{F}_{2^n}$ of the form*

$$f(X) = \sum_{2^i+2^j \leq d} \gamma_{i,j} X^{2^i+2^j} + \sum_{2^k \leq d} \beta_k X^{2^k} + \alpha$$

with $\gamma_{i,j}, \beta_k, \alpha \in \mathbb{F}_{2^n}$, for $i, j \in \mathbb{N}$ and degree $d \in \mathbb{N}$. This polynomial f is called the central function. *The central polynomials are of* HFE-shape *if $\mathcal{P}' := \phi \circ f \circ \phi^{-1}$, where ϕ is the canonical bijection between field extension $\mathbb{F}_{2^n}$ and vector space $\mathbb{F}_2^n$.*

Note that the central function is not necessarily injective.

The basic HFE cryptosystem is described in the frame below.

Hidden Field Equations	
Computations in: Encryption - field $\mathbb{F}_2$ and vector space $\mathbb{F}_2^n$ Decryption - field $\mathbb{F}_2$, vector space $\mathbb{F}_2^n$ and field extension $\mathbb{F}_{2^n}$	Procedure for key generation: 1. Choose $n \in \mathbb{N}$ 2. Choose $S, T \in \text{Aff}^{-1} : \mathbb{F}_2^n \to \mathbb{F}_2^n$ 3. Choose HFE-shaped polynomial $f : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ 4. Compute $\mathcal{P} = (T \circ \phi \circ f \circ \phi^{-1} \circ S)(\vec{x})$ Public key: $\mathcal{P}$ Private key: S, T, f
Encryption of $\vec{x} \in \mathbb{F}_2^n$: 1. Compute $\vec{y} = \mathcal{P}(\vec{x})$ 2. Compute hash string $\vec{x} = H(\vec{x})$ Encrypted message: $(\vec{y}, \tilde{x})$	Decryption of $(\vec{y}, \tilde{x})$: 1. Compute $\vec{y}' = T^{-1}(\vec{y})$ 2. Compute $Y = \phi(\vec{y}')$ 3. Compute $Q = \{X : f(X) = Y\}$ 4. Compute $Q' = \{\vec{x}' = \phi^{-1}(X) : X \in Q\}$ 5. Compute $Q'_s = \{\vec{x}_s = S^{-1}(\vec{x}') : \vec{x}' \in Q\}$ 6. Select $\vec{x}_s \in Q_s$ such that $H(\vec{x}_s) = \tilde{x}$

15.3 Attack on HFE: Gröbner Bases

One of the attacks that can be used on multivariate quadratic cryptosystems, and which seems to work particularly well on HFE, is one that uses Gröbner bases. The idea behind this attack, which is the only one we will discuss, is to discover the inverse of the public key without recovering the private key.

15.3.1 What is a Gröbner Basis?

A Gröbner basis is a mathematical concept which can be used to find the roots of a system of polynomials. Before we can define Gröbner bases we need two other mathematical concepts.

We begin with the concept of the S-polynomial.

Definition 15.9 *The* S-polynomial *of two polynomials* f *and* g *is defined by*

$$\text{spoly}(f, g) = \left(\frac{J}{\text{lt}(f)} \cdot f - \frac{J}{\text{lt}(g)} \cdot g \right)$$

where $J = \text{lcm}(\text{lt}(f), \text{lt}(g))$*, which is the least common multiple of* $\text{lt}(f)$ *and* $\text{lt}(g)$*.*

Note that the leading terms of f and g are eliminated in the S-polynomial.

Example 15.1 *We can compute the S-polynomial of* x_1+x_2+1 *and* $x_1x_2+x_3+1$ *using the formula in Definition 15.9 as follows.*

$$\begin{aligned}\text{spoly}(x_1+x_2+1, x_1x_2+x_3+1) &= \frac{x_1x_2}{x_1}(x_1+x_2+1) - \frac{x_1x_2}{x_1x_2}(x_1x_2+x_3+1)\\ &= x_1x_2 + x_2^2 + x_2 - x_1x_2 - x_3 - 1\\ &= -x_3 + x_2^2 + x_2 - 1\end{aligned}$$

Example 15.2 *We can also compute the S-polynomial of* $x_1 + x_2 + 1$ *and* x_2x_3.

$$\begin{aligned}\text{spoly}(x_1+x_2+1, x_2x_3) &= \frac{x_1x_2x_3}{x_1}(x_1+x_2+1) - \frac{x_1x_2x_3}{x_2x_3}(x_2x_3)\\ &= x_1x_2x_3 + x_2^2x_3 + x_2x_3 - x_1x_2x_3\\ &= x_2^2x_3 + x_2x_3\end{aligned}$$

The other concept we need is the normal form.

Definition 15.10 *The* normal form *of a polynomial* f *with respect to a polynomial* g *is defined by*

$$\text{normalf}(f, g) = f - \frac{t}{\text{lt}(g)} \cdot g$$

where t *is a term in* f *which is divisible by* $\text{lt}(g)$.

Note that if no term t in f is divisible by $\text{lt}(g)$ then f is itself the normal form of f with respect to g.

The normal form of a polynomial f with respect to a set of polynomials $G = \{g_1, \ldots, g_n\}$, denoted by $\text{normalf}(f, G)$, is a polynomial obtained by recursively computing the normal form with respect to one of the polynomials g_i using a finite number of steps. None of the terms in this polynomial are divisible by a leading term of a g_i.

We can now define Gröbner bases.

Definition 15.11 *A finite set of polynomials* G *is a* Gröbner basis *iff*

$$\text{normalf}(\text{spoly}(f, g), G) = 0$$

for all $f, g \in G$.

By itself this means very little, however Gröbner bases have a very nice property. We can construct a Gröbner basis for every system of polynomials, and the polynomials in the Gröbner basis have the same roots as the polynomials in the original system. Since the Gröbner basis generally contains some polynomials which are simpler than the original polynomials, it makes it much easier to find the roots. We will demonstrate this in 15.3.2.

A Gröbner basis for a set of polynomials can be computed using Buchberger's Algorithm, Algorithm 15.2. Note that in Buchberger's Algorithm $\text{normalf}(\text{spoly}(f, g), GB)$ is added to GB if it is not equal to 0. Therefore it holds that $\text{normalf}(\text{spoly}(f, g), GB) = 0$ for the new GB. Thus the output of Buchberger's Algorithm is indeed a Gröbner basis.

Gröbner basis(G):
    $GB \longleftarrow G$
    $B \longleftarrow \{(f, g) : f, g \in G, f \neq g\}$
    while $B \neq \emptyset$ **do**
        $(f, g) \longleftarrow$ select a pair in B
        $B \longleftarrow B \backslash \{(f, g)\}$
        $h \longleftarrow \text{normalf}(\text{spoly}(f, g), GB)$
        if $h \neq 0$ **then**
            $B \longleftarrow B \cup \{(f, h) : f \in GB\}$
            $GB \longleftarrow GB \cup \{h\}$
    return GB

Algorithm 15.2: Buchberger's Algorithm

15.3.2 The Attack

We shall demonstrate the attack on HFE with Gröbner bases on a public key $\mathcal{P} \in \mathcal{MQ}(\mathbb{F}_2^3)$ and a ciphertext $\vec{y} \in \mathbb{F}_2^3$. Recall that $\mathbb{F}_2$ is a finite field with characteristic and

Casus 15.3: The HFE challenges

Jacques Patarin, the designer of HFE, proposed challenges for HFE in 1998, in the extended version of [Pat95].

First challenge:

- 80 equations with 80 unknowns over $\mathbb{F}_2$
- degree of univariate polynomial is 96
- challenge: solve the equations

This challenge was broken in 2002 by Jean-Charles Faugère in 96 hours using a 833 MHz Alpha workstation with 4 GB of memory. The algorithm that was used is the F5/2 algorithm, which is a fast algorithm to compute Gröbner basis based on Buchberger's Algorithm (15.2).

Second challenge:

- 32 equations with 36 unknowns over $\mathbb{F}_{16}$
- degree of univariate polynomial is 4352
- challenge: solve the equations

This challenge has not yet been broken, although some have claimed they managed it. The reward for breaking the challenges is $500 each.

cardinality equal to two. Therefore we have that $\forall x \in \mathbb{F}_2 : x^2 = x$ and $1 + 1 = 0$. As the public key $\mathcal{P}$ and the ciphertext $\vec{y}$ we take

$$\mathcal{P} = \begin{pmatrix} x_1 + x_2 + 1 \\ x_1 x_2 + x_3 \\ x_2 x_3 + 1 \end{pmatrix} \quad \text{and} \quad \vec{y} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

respectively. To find the plaintext we must solve the system of equations

$$\begin{aligned} x_1 + x_2 + 1 &= 0 \\ x_1 x_2 + x_3 &= 1 \\ x_2 x_3 + 1 &= 1, \end{aligned}$$

which means we need to find the roots of the polynomials

$$\begin{aligned} f_1 &= x_1 + x_2 + 1 \\ f_2 &= x_1 x_2 + x_3 + 1 \\ f_3 &= x_2 x_3. \end{aligned}$$

Of course it is quite easy to find the roots of this system of polynomials, however it is of convenient size to demonstrate the use of the Gröbner basis. Therefore we will apply Buchberger's Algorithm (15.2) to the system $\{f_1, f_2, f_3\}$. We shall not give the entire computation for each S-polynomial and normal form, but trust that the reader can do this for him/herself.

We begin with $GB = \{f_1, f_2, f_3\}$. We have already computed $\text{spoly}(f_1, f_2)$ in Example 15.1 (remember that $x^2 = x$ and $2x = 0$).

$$\begin{aligned} \text{spoly}(f_1, f_2) &= -x_3 - 1 \\ \text{normalf}(-x_3 - 1, GB) &= -x_3 - 1. \end{aligned}$$

So the polynomial $f_4 = -x_3 - 1$ is added to GB and we have $GB = \{f_1, f_2, f_3, f_4\}$. We have also already computed $\text{spoly}(f_1, f_3)$ in Example 15.2 (again remember that $x^2 = x$ and $2x = 0$).

$$\begin{aligned} \text{spoly}(f_1, f_3) &= 0 \\ \text{normalf}(0, GB) &= 0. \end{aligned}$$

Thus GB remains the same.

$$\begin{aligned} \text{spoly}(f_1, f_4) &= -x_2 x_3 + x_1 - x_3 \\ \text{normalf}(-x_2 x_3 + x_1 - x_3, GB) &= -x_2. \end{aligned}$$

The normal form is computed recursively using f_3, f_1 and f_4 respectively. The polynomial $f_5 = -x_2$ is added to GB, which becomes $GB = \{f_1, f_2, f_3, f_4, f_5\}$.

$$\begin{aligned} \text{spoly}(f_1, f_5) &= 0 \\ \text{normalf}(0, GB) &= 0, \end{aligned}$$

and

$$\begin{aligned} \text{spoly}(f_2, f_3) &= 0 \\ \text{normalf}(0, GB) &= 0. \end{aligned}$$

So nothing is added to GB.

$$\begin{aligned} \text{spoly}(f_2, f_4) &= x_1x_2 \\ \text{normalf}(x_1x_2, GB) &= 0. \end{aligned}$$

The normal form is computed recursively using f_2 and f_4. The rest of the normal forms of the S-polynomails are also 0:

$$\begin{aligned} \text{spoly}(f_2, f_5) &= -x_3 - 1 \\ \text{normalf}(-x_3 - 1, GB) &= 0, \text{ using } f_4; \\ \text{spoly}(f_3, f_4) &= x_2 \\ \text{normalf}(x_2, GB) &= 0, \text{ using } f_5; \\ \text{spoly}(f_3, f_5) &= 0 \\ \text{normalf}(0, GB) &= 0; \\ \text{spoly}(f_2, f_4) &= -x_2 \\ \text{normalf}(-x_2, GB) &= 0, \text{ again using } f_5. \end{aligned}$$

So the Gröbner basis of the system of polynomials $\{f_1, f_2, f_3\}$ is $\{f_1, f_2, f_3, f_4, f_5\}$. Finding the roots is now very easy. The equations $f_4 = -x_3 - 1 = 0$ and $f_5 = -x_2 = 0$ directly give us that $x_2 = 0$ and $x_3 = 1$. Substituting $x_2 = 0$ in f_1 leaves the equation $x_1 + 1 = 0$, which gives us that $x_1 = 1$. Thus the plaintext is the vector

$$\vec{x} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

Although this result could have easily been obtained directly, this shows that the Gröbner basis can be very useful in a Ciphertext Only Attack on basic HFE, see also Casus 15.3. A result of this, which is outlined in [GJS06], is that an attack on basic HFE can be subexponential.

15.4 Conclusion

We have seen that basic HFE over the field $\mathbb{F}_2$ can be attacked using the Gröbner basis, however, as Casus 15.3 illustrates, it still takes a considerable amount of time to do so. To protect HFE from its weakness to attacks using Gröbner bases several adaptations have been made, some of which are discussed in [Wol05] along with some other ways to attack HFE. The security of HFE is discussed in [Cou01], which also suggests some boundaries for the number of equations and variables as well as the degree of the central function for HFE. All in all there appear to be enough methods to make HFE sufficiently secure.

Chapter 16

Montgomery Multiplications

Written by *Tessa van der Hoorn & Hugo Duivesteijn.*

In modern cryptography a lot of crypto-systems compute modulo a large number N. Exampels of systems that use modular calculations are RSA and Elgamal, the way these systems work was explained in the course. For some calculations these systems use modular multiplications.

Multiplying modulo a large number N is in general an expensive operation, it takes a lot of time to compute. Montgomery discovered a way of multiplying two numbers modulo any number, which is less expensive.

In this chapter our goal is to find a faster way for modular multiplications. We will discuss the way Montgomery's algorithm works, give an example of the algorithm used in practice and look at the way to imply the algorithm into hardware.

16.1 Montgomery

Peter Lawrence Montgomery is a American Mathematician, mostly active in the field of cryptography. He's particulary known for his contributions to the elliptic curve method of factorisation. Except of this he is also the inventer of the Block Lanczos algorithm, which is an algorithm to find nullspaces of finite matrices. At this moment, Block Lanczos is the best algorithm to find such a nullspace. Montgomery is still active on the scientific front. We found out he currently also works at the CWI in Amsterdam. Peter L. Montgomery has Erdösnumber 1.

Casus 16.1: ERDÖSNUMBER

Erdösnumber: This is an number with which Mathematician indicate their collaborative distance in regard to mathematical papers, between an author and the Hongarian mathematician Paul Erdös. The number is defined as follows: Erdös himself has number 0. If the lowest Erdösnumber of a coauthor is k, then the author's Erdösnumber is k + 1. If not, then the author's Erdösnumber is infinite. This basically comes down to 511 people who were direct collaborators, of which Montgomery is one, with number 1.

16.2 Description of the algorithm

To understand why we even need some other algorithm for modular multiplication let us first take a look at the normal algoritm for multiplying two integers modulo some number N.
Given two integers x and y, let algorithm 16.2 give us $x * y \bmod N$.

We see that in this algorithm we have to divide by N. In general division is a very slow operation, exept for some numbers. We'll try to find an algorithm which avoids division by N and does not affect the algorithms for modular addition and substraction.

First we want to fix $N > 1$ and we select a number R such that the greatest common divisor of N and R is 1, $R > N$ and computations modulo R are inexpensive to process. Now we let R^{-1} and N' be integers satisfying $0 < R^{-1} < N$, $0 < N' < R$ and $RR^{-1} - NN' = 1$. For $0 \leq i < N$, let i represent the residue class containing $iR^{-1} \bmod N$. Now let us define the algorithm REDC as in algorithm 16.3.

16.2.1 function REDC

Looking at algorithm 16.3, t can be any number, but we want t to be an integer. For t to be an integer $T + mN$ must be divisable by R. We can easily check that $mN \equiv TN'N \equiv -T \bmod R$. So $T + mN$ becomes $T + -T = 0$ which is divisable by R. Also we see $tR \equiv T \bmod N$, so with simply multiplying by R^{-1} at both sides, we get $t \equiv TR^{-1} \bmod N$. So with REDC we have found an easy way of computing $t \equiv TR^{-1} \bmod N$ which is in the same residue class as T.

$m = x * y$
$d = m \div N$
return $m - d * N$

Algorithm 16.2: X*Y MOD N

$m = (T \bmod R)N' \bmod R$
$t = \frac{(T+mN)}{R}$
if $t \geq N$ return $t - N$ else return t

Algorithm 16.3: REDC(T)

16.2.2 Multiplying

Now we have found the algorithm REDC, we are going to use it to multipy two numbers modulo N.

Let x and y be numbers between 0 and $N-1$, including 0 and $N-1$. Now z=REDC(xy) is the poduct of x and y modulo N. Let's prove this.

Theorem 16.1 *Given x, y and z=REDC(xy), $xy \equiv z$ mod$N$.*

Proof. z=REDC(xy), this means $z \equiv (xy)R^{-1} \bmod N$. Now we know $N|z-(xy)R^{-1}$ and also $N|R^{-1}(z-(xy)R^{-1}) = zR^{-1} - (xR^{-1})(yR^{-1})$, which gives us $(xR^{-1})(yR^{-1}) \equiv zR^{-1} \bmod N$. Because x, y and z represent the residue classes containing $(xR^{-1}), (yR^{-1})$ and zR^{-1}, we now have $xy \equiv z \bmod N$. △

16.2.3 The addition and substraction algorithms

Comming back to the claim we made in the beginning, we still have to prove this way of representing residue classes does not effect the modular addition and substraction algorithms.

We want to prove that $xR^{-1} + yR^{-1} \equiv zR^{-1} \bmod N$ if and only if $x + y \equiv z \bmod N$. Writing this as true mathematicians we want:

Theorem 16.2

$$xR^{-1} + yR^{-1} \equiv zR^{-1} \bmod N \Leftrightarrow x + y \equiv z \bmod N$$

Proof. $' \Rightarrow '$

Given $xR^{-1}+yR^{-1} \equiv zR^{-1} \bmod N$, we know $N|xR^{-1}+yR^{-1}-zR^{-1} = R^{-1}(x+y-z)$. Because N does not divide R^{-1}, N must divide $x+y-z$ so $x+y \equiv z \bmod N$ △

In this proof I made the claim that N does not divide R^{-1}. This claim still has to be proven.

Lemma 16.3 *N does not divide R^{-1}*

Proof. We have choosen R^{-1} as $0 < R^{-1} < N$ and $RR^{-1} - NN' = 1$. This gives us $R^{-1} = \frac{1+NN'}{R} = \frac{1}{R} + \frac{NN'}{R}$. So now the question is: does N divide $\frac{1}{R} + \frac{NN'}{R}$? Because $N|\frac{NN'}{R}$ we also need $N|\frac{1}{R}$. But we choose R such that the greatest common divisor of N and R was 1. This means N does not divide R so also N does not divide $\frac{1}{R}$. Which means N does not divide R^{-1}.

$' \Leftarrow'$

Given $x + y = z \bmod N$, we know $N|x + y - z$, this also means $N|R^{-1}(x + y - z) = xR^{-1} + yR^{-1} - zR^{-1}$ so $xR^{-1} + yR^{-1} \equiv zR^{-1} \bmod N$. △

16.2.4 substraction

The proof of the substraction algorithm is exactly the same, so we are not going to write it down. The only thing we have to do different from the proof of the addition algorithm is to replace all of the + with −. Notice that if $x - y \equiv z \bmod N$, we have $N|x - y - z = x - z - y$ which also gives us $x - z \equiv y \bmod N$.

16.3 Example

Now we have seen the algorithm, let us try to understand what it actualy does. To do this we will first take a look at an example of normal multiplication.

The idea of Montgomery actually is very simple but seems a bit strange because it computes the answer to a completly different multiplication to the one we asked for.

For example: if we want to know 789098×123456, we will compute $789098 \times 123456 \div 1000000$ instead. To do this, we'll use the shift-and add technique, but instead of starting from the left-hand end of 789098 and multiplying by 10 each time, we'll start from the right-hand and divide by 10:

$$\begin{aligned}
8 \times 123456 &= 987648 \\
\div 10 &= 98764.8 \\
9 \times 123456 &= +1111104 \\
9.8 \times 123456 &= 1209868.8 \\
\div 10 &= 120986.88 \\
0 \times 123456 &= +0 \\
0.98 \times 123456 &= 120986.88 \\
\div 10 &= 12098.688 \\
9 \times 123456 &= +1111104 \\
9.098 \times 123456 &= 1123202.688 \\
\div 10 &= 112320.2688 \\
8 \times 123456 &= +987648 \\
8.9098 \times 123456 &= 1099968.2688 \\
\div 10 &= 109996.82688 \\
7 \times 123456 &= +864192 \\
7.89098 \times 123456 &= 974188.82688 \\
0.789098 \times 123456 &= 97418.882688
\end{aligned}$$

It seems like this is just doing an old multiplication sum done backwards, but there is one important difference. When we use addition, the carries always propagate from right to left, and in the original method this meant that we couldn't be certain of the value of any digit of the result until we had calculated the entire sum. But now we generate the result from the right-hand end while the carries still flow to the left. In other words this means that no digit in the result changes once it has gone past the decimal point.

16.3.1 Montgomery multiplication in modular form

But we don't want to make normal multiplications, we want to compute the multiplication modulo some number. Let us take a look at another example: we'll compute $789098 \times 123456 \div 1000000$ modulo 876543. In this example $R = 1000000$. The defenition of modular division is quit simple: $X \div Y (\mathrm{mod} R)$ is simply the number Z for which $Z \times Y = X (\mathrm{mod} R)$.

Let us work out this example as we did with the first one.

$$8 \times 123456 \quad = \quad 987648$$

Since we want to divide by 10, but we can't do so because 987648 is not divisible by 10. To fix this problem we add a multiple of 876543, which we are allowed to do because we're working modulo 876543. From here we call this multiple of 876543: $n \times 876543$ in which n isn't necessarely the same in al cases, although both of us believe there is an example to be made up in which this is the case.

$$\begin{aligned}
& & & +3506172 \\
8 \times 123456 & = & & 4493820 \\
\div 10 = 0.8 \times 123456 & = & & 449382 \\
9 \times 123456 & = & & +1111104 \\
9.8 \times 123456 & = & & 1560486 \\
9.8 \times 123456 + n \times 876543 & = & & 8572830 \\
\div 10 = 0.98 \times 123456 & = & & 8572830 \\
0 \times 123456 & = & & 0 \\
0.98 \times 123456 & = & & 857283 \\
0.98 \times 123456 + n \times 876543 & = & & 8746170 \\
\div 10 = 0.098 \times 123456 & = & & 8746170 \\
9 \times 123456 & = & & +1111104 \\
9.098 \times 123456 & = & & 1985721 \\
9.098 \times 123456 + n \times 876543 & = & & 4615350 \\
\div 10 = 0.9098 \times 123456 & = & & 461535 \\
8 \times 123456 & = & & +987648 \\
8.9098 \times 123456 & = & & 1449183
\end{aligned}$$

$$\begin{aligned} 8.9098 \times 123456 + n \times 876543 &= 9338070 \\ \div 10 = 0.89098 \times 123456 &= 933807 \\ 7 \times 123456 &= +864192 \\ 7.89098 \times 123456 &= 1797999 \\ 7.89098 \times 123456 + n \times 876543 &= 7933800 \\ \div 10 = 0.789098 \times 123456 &= 793380 \end{aligned}$$

To check this result, multiply it with 100000 and find out it's equal to 770211 mod $876543 = 789098 \times 123456 \bmod 876543$.

16.3.2 I didn't ask for that!

Most of you will still think: "I didn't ask for $A \times B \div 1000000$, I've asked for $A \times B$". Well, you are absolutely right. Let's look what happends if we multiply both A and B by 1000000:

$$1000000 \times A \times 1000000 \times B \div 1000000 = 1000000 \times A \times B$$

In normal words, if you multiply both A and B by 1000000 you won't get the answer you asked for. Instead you get a million times their multiple. How do we correct this?

1. Multiply each of them by 1000000 (modulo R)
2. Do the Montgomery multiplication
3. Divide the result by 1000000 (modulo R)

If you look at Montgomery multiplication just as a form of modular multiplication, then there isn't much use for it, but, when you need to do a great many consecutive modular multiplications, for example doing modular exponentiation, it will come into it's own. Let us , for example, raise x^{25}. We'll use $\otimes$ to indicate a Montgomery multiplication modulo R

$$\begin{aligned} X &= 1000000 \times x (\mathrm{mod} R) \\ X^2 &= X \otimes X \\ X^4 &= X^2 \otimes X^2 \\ X^6 &= X^4 \otimes X^2 \\ X^{12} &= X^6 \otimes X^6 \\ X^{24} &= X^{12} \otimes X^{12} \\ X^{25} &= X^{24} \otimes X^1 \\ x^{25} &= X^{25} \div 1000000 (\mathrm{mod} R) \end{aligned}$$

For the first stage we could use a Montgomery multiplication: $X = x \otimes 1000000000000$. This because it would automatically be divided by 1000000 and so it would give us the wanted $1000000 \times x$. There is only one little problem, namely, Montgomery multiplication doesn't work with numbers larger then the modulus. To solve this we define

$a = 1000000000000(\mathrm{mod}R)$. Now it's just a matter of rewriting the first stage: $X = x \otimes a$
Exept for the first and the last stage, all the stages are the same as we would have to compute if we would be using 'normal' modular multiplication. So we give a little extra attention to the first and the last stage.
We can compute the last stage easily by doing a single Montgomery multiplication: $x^{25} = X^{25} \otimes 1$.
This all together let us make the conclusion that Montgomery multiplication takes two extra multiplications and the precomputing of a, which only has te be done once for any R.

16.4 Montgomery Multiplications with binary numbers

In our examples we used a decimal way of representing the numbers. This is because the idea is easier to explain in a decimal way. But most of the time it is necessary to implement the algorithm on hardware, and we will have to use a binary number representation. Luckily we can make an easy translation of Montgomery's algorithm to an algorithm that can be used on binary numbers.

Suppose $R = 2^n$ and N is odd (it is not hard to see that $gcd(R, N) = 1$). Let both x and y be bigger, or equal than zero, but smaller than N. Now algorithm 16.4 computes $xyR^{-1} \bmod N$.

In this algorithm n is the number of bits. In every i^{th} step we add y to our previous value if $x_i = 1$, if this gives us an even number we divide by 2, els we add N and divide by 2. By induction we can prove $2^i S_i \equiv (x_{i-1}...x_0)y \bmod N$ and $0 \leq S_i < N + y < 2N$. Therefore $xyR^{-1} \bmod N$ is either S_n or $S_n - N$.

For Montgomery's algorithm translated to multiprecision in general, we refer to [Mon].

16.5 Computation time

We started this chapter with wanting to find a way of multiplying two numbers modulo some large number N which is less expensive than the normal method. We have discussed Montgomery's algorithm, but is it in fact faster, and maybe even more important are there methods which are faster?

As we already mentioned, if you want to make just one multiplication, using Montgomery's algorithm is not going to help. The algorithm itself is rather fast, but before we can use it, we first have to find R, R^{-1} and N', this also takes some time to compute. But if we want to make more multiplications, like when making exponentiations, Montgomery's algorithm is faster, because instead of calculation modulo a random number N, we have choosen to calculate modulo R which is choosen in such a way that computation is fast.

$S_0 = 0$
for $i =: 0$ step 1 to $n-1$ do
if $\frac{S_i + x_i y}{2} \in \mathbb{Z}$
$S_{i+1} \leftarrow \frac{S_i + x_i y}{2}$
else $S_{i+1} \leftarrow \frac{S_i + x_i y + N}{2}$
end if
end for
if $S_n > N$
return $S_n - N$
else
return S_n
end if

Algorithm 16.4: HARDWARE IMPLEMENTATION

A way to get around the time it takes to compute R^{-1} and N', is to precompute a lot of them. When calculating this can save you time, but we also have to take the amount of storage in consideration. Because of this a number of new algorithms, inspired by Montgomery's algoritm, have been developed, trying to avoid this problem. Because of that these algorithms have turned out to be faster than Montgomery's. Also several implementations of Montgomery's algoritm on hardware and software have been develloped. For readers who are interested in these kind of algorithms we recomend to take a look at [SD], [BLH] and [FSV].

16.6 Concluding remarks

Montgomery's algorithm gives us a fast way of multiplying, and is easy to implement on hardware and software. But the algorithm was developped in 1985 and since then the algorithm has been improved many times to make it faster, easier to implement, etc. Because Montgomery's algorithm is a base of the other algorithms, knowing and understanding Montgomery's algorithm makes it a lot easier to understand the new algorithms.

Bibliography

[AM05] Alsaid, A. and Mitchell, C. J. Installing fake root keys in a pc.

[AYF05] Anthony Y. Fu, Xiaotie Deng, W. L. A potential iri based phishing strategy. *Lecture Notes in Computer Science* **3806/2005** (2005), 618–619.

[Ber91] Berkovits, S. How to broadcast a secret. *Advances in Cryptology EUROCRYPT 91* (1991), 535 – 541.

[Bih97] Biham, E. A fast new des implementation in software. *Fast Software Encryption: 4th International Workshop, FSE'97, Haifa, Israel, January 1997. Proceedings* (1997), 260–.
http://www.springerlink.com/content/6grhkuk2d7u1jj0b.

[BLH] Blümel, R., Laue, R., and Huss, S. A. A highly efficient modular multiplication algorithm for finite field arithmetic in gf(p).

[BMQR07] Bohli, J.-M., Müller-Quade, J., and Röhrich, S. Bingo voting: Secure and coercion-free voting using a trusted random number generator. In proc. *VOTE-ID* (2007), vol. 4896 of *Lecture Notes in Computer Science*, Springer, pp. 111–124.

[CAC08] CACert. Cacert root key storage (http://www.cacert.org/help.php?id=7), 2008.

[CES+05] Chen, L., Enzmann, M., Sadeghi, A.-R., Schneider, M., and Steiner, M. *A Privacy-Protecting Coupon System*. Springer, 2005, pp. 93–108.

[CJ98] Chabaud, F. and Joux, A. Differential collisions in SHA-0. In proc. *Advances in Cryptology CRYPTO '98* (1998), vol. 1462, Springer Berlin / Heidelberg, pp. 253–261.

[Coh05] Cohen, A. Coupon sites. *PC Magazine* (2005), 90–91.

[Con03] Consortium, N. Portfolio of Recommended Cryptographic Primitives.

[Cou01] Courtois, N. The Security of Hidden Field Equations (HFE). In proc. *Topics in Cryptology CT-RSA 2001* (2001), vol. 2020 of *Lecture Notes In Computer Science*, Springer Berlin/Heidelberg, pp. 266–281.

[CS06] Carbunar, B. and Sion, R. Uncheatable reputation for distributed computation markets. Tech. rep., 2006.

[Dal] Dale, V. W. Coupon (Van Dale). `http://www.vandale.nl/vandale/opzoeken/woordenboek/?zoekwoord=tegoedbon`.

[DH76] Diffie, W. and Hellman, M. E. New directions in cryptography. *IEEE Transactions on Information Theory* **IT-22**, 6 (1976), 644–654.

[DiV00] DiVincenzo, D. P. The physical implementation of quantum computation. *arXiv* **http://arxiv.org/abs/quant-ph/0002077** (2000), 1.

[DK05] Dodis, Y. and Katz, J. Chosen-Ciphertext Security of Multiple Encryption.

[DKXY02] Dodis, Y., Katz, J., Xu, S., and Yung, M. Key-Insulated Public-Key Cryptosystems.

[DL05] Daum, M. and Lucks, S. Attacking hash functions by poisoned messages. Eurocrypt 2005 presentation, 2005. `http://www.cits.rub.de/MD5Collisions/`.

[DNL01] Dalit Naor, M. N. and Lotspiech, J. Revocation and Tracing Schemes for Stateless Receivers. *Lecture Notes in Computer Science* (2001), 41–62.

[DR] Daemen, J. and Rijmen, V. Aes proposal: Rijndael. `citeseer.ist.psu.edu/daemen98aes.html`.

[DT05] Dhamija, R. and Tygar, J. D. The battle against phishing: Dynamic security skins. In proc. *SOUPS '05: Proceedings of the 2005 symposium on Usable privacy and security* (New York, NY, USA, 2005), ACM, pp. 77–88.

[EG85] Even, S. and Goldreich, O. On the power of cascade ciphers. *Transactions on Computer Systems* **3** (1985), 108–116.

[EJ96] Ekert, A. and Jozsa, R. Quantum computation and Shors factoring algorithm. *Reviews of Modern Physics* **68** (1996), 733–753.

[FN94] Fiat, A. and Naor, M. Broadcast encryption. *Lecture Notes in Computer Science* **773** (1994), 480–491.

[For07] Forum, C. Guidelines for the issuance and management of extended validation.

[Fou] Foundation, T. Truecrypt: Free open-source disk encryption software. `http://www.truecrypt.org/docs/`.

[Fox00] Fox, L. Cents and sensibility. *Yahoo! Internet Life* (2000), 32.

[FST07] Fette, I., Sadeh, N., and Tomasic, A. Learning to detect phishing emails. In proc. *WWW '07: Proceedings of the 16th international conference on World Wide Web* (New York, NY, USA, 2007), ACM, pp. 649–656.

[FSV] Fan, J., Sakiyama, K., and Verbauwhede, I. Montgomery modular multiplication algorithm on multi-core systems.

[GH07] Gonggrijp, R. and Hengeveld, W.-J. Studying the Nedap/Groenendaal ES3B voting computer: a computer security perspective. In proc. *EVT'07: Proceedings of the USENIX/Accurate Electronic Voting Technology on USENIX/Accurate Electronic Voting Technology Workshop* (Berkeley, CA, USA, 2007), USENIX Association, pp. 1–1.

[GJS06] Granboulan, L., Joux, A., and Stern, J. Inverting HFE is Quasipolynomial. In proc. *Advances in Cryptology - CRYPTO 2006* (2006), vol. 4117 of *Lecture Notes In Computer Science*, Springer Berlin/Heidelberg, pp. 345–356.

[Gra07] Grant, R. M. *Contemporary Strategy Analysis; Case 11: Rivalry in Video Games*. Blackwellpublishing, 2007.

[Hal95] Halfhill, T. R. The truth behind the pentium bug, 1995. `http://www.byte.com/art/9503/sec13/art1.htm`.

[Hol73] Holevo, A. S. Bounds for the quantity of information transmitted by a quantum communication channel. *Probl. Pered. Inform* **9** (1973), 3.

[Hua02] Huang, A. Keeping secrets in hardware: the Microsoft Xbox case study. 1–12. `ftp://publications.ai.mit.edu/ai-publications/2002/AIM-2002-008.pdf`.

[Inc07] Inc., V. S. What is voltage identity-based encryption (ibe)? (http://www.voltage.com/technology/ibe.htm), 2007.

[JC02] J. Camenisch, A. L. *A signature scheme with efficient protocols*. Springer, 2002.

[JJ98] Johnson, N. and Jajodia, S. Steganalysis of images created using current steganographic systems. In proc. *2nd Int'l Workshop in Information Hiding* (1998), Springer-Verlag, pp. 273–289.

[JLP02] Jeffrey Lotspiech, S. N. and Pestoni, F. Broadcast Encryption's bright future. *IEEE Computer* (2002).

[JON05] JONES, D. W. Chain voting. http://vote.nist.gov/threats/papers/ChainVoting.pdf, 2005.

[Knu97] Knuth, D. E. *Art of Computer Programming, Volume 2: Seminumerical Algorithms (3rd Edition).* Addison-Wesley Professional, 1997.

[Koc95] Koc. Analysis of sliding window techniques for exponentiation. *CANDM: An International Journal: Computers and Mathematics, with Applications* **30** (1995).
`citeseer.ist.psu.edu/koc95analysis.html`.

[KR00] Knudsen, L. and Rijmen, V. Ciphertext-only attack on akelarre. 135–147.

[KSHW97] Kelsey, J., Schneier, B., Hall, C., and Wagner, D. Secure application of low-entropy keys (isw'97). 121–134.

[KSW97] Kelsey, J., Schneier, B., and Wagner, D. Related-key cryptanalysis of 3-way, biham-des, cast, des-x, newdes, rc2, and tea. vol. 1334 of *Lecture Notes in Computer Science*, U.C. Berkeley, pp. 8–13.

[Lav03] Lavor, J. C. Shors algorithm for factoring large integers. *arXiv* **http://arxiv.org/abs/quant-ph/0303175v1** (2003), 1.

[LB04] Linn, J. and Branchaud, M. An examination of asserted pki issues and proposed alternatives.

[LWW05] Lenstra, A., Wang, X., and Weger, B. de. Colliding x.509 certificates.

[Mat06] Matsui, M. How far can we go on the x64 processors? *Fast Software Encryption* (2006), 341–358.
`http://dx.doi.org/10.1007/11799313_22`.

[Mic02] Micali, S. Scalable certificate validation and simplified pki management. *First Annual PKI Research Workshop Proceedings* (2002), 15–25.

[Mir05] Mironov, I. Hash functions: Theory, attacks, and applications. Tech. rep., Microsoft Research, Silicon Valley Campus, 2005.
`http://research.microsoft.com/users/mironov/papers/hash_survey.pdf`.

[MN06] Moran, T. and Naor, M. Receipt-free universally-verifiable voting with everlasting privacy. In proc. *CRYPTO 2006* (2006), C. Dwork (ed.), vol. 4117 of *Lecture Notes in Computer Science*, Springer, pp. 373–392.

[MO97] Mambo, M. and Okamoto, E. Proxy cryptosystems: Delegation of the power to decrypt ciphertexts. In proc. *IEICE Trans. Fund. Electronics Communications and Computer Science, E80-A/1* (1997), pp. 54–63.

[Mon] Montgomery, P. L. Modular multiplication without trial division. *Mathematics of Computation* **44**, 519–521.

[Mor] Morrissey, S. Are microchip tags safe? http://www.verichipcorp.com/news/1192716360.

[MPP+05] Matusiewicz, K., Pieprzyk, J., Pramstaller, N., Rechberger, C., and Rijmen, V. Analysis of simplified variants of sha-256. In proc. *WEWoRC 2005 - Western European Workshop on Research in Cryptology* (2005), pp. 123–134.

[MPRR06] Mendel, F., Pramstaller, N., Rechberger, C., and Rijmen, V. Analysis of step-reduced sha-256. In proc. *Fast Software Encryption* (2006), pp. 126–143.

[MR] Mark Roberti, R. j. A 5-cent breakthrough. http://www.rfidjournal.com/article/articleview/2295/.

[Nef04] Neff, C. A. Practical high certainty intent verification for encrypted votes. http://www.votehere.net/vhti/documentation/vsv-2.0.3638.pdf, 2004.

[NIS95] NIST. Fips publication 180-1: Secure hash standard. Tech. rep., National Institute of Standards and Technology (NIST), 1995. http://www.itl.nist.gov/fipspubs/fip180-1.htm.

[NIS02] NIST. Fips publication 180-2: Secure hash standard. Tech. rep., National Institute of Standards and Technology (NIST), 2002. http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf.

[NP] Nohl, K. and Plötz, H. Mifare, little security, despite obscurity. http://events.ccc.de/congress/2007/Fahrplan/events/2378.en.html.

[NSS+06] Naito, Y., Sasaki, Y., Shimoyama, T., Yajima, J., Kunihiro, N., and Ohta, K. Improved collision search for sha-0. In proc. *Advances in Cryptology ASIACRYPT 2006* (2006), pp. 21–36.

[NXP] NXP B.V. MF0 IC U1 functional specification contactless single-trip ticket IC. http://www.nxp.com/acrobat/other/identification/M028634_MF0ICU1_Functional_

[Org] Organization, I. C. A. Machine readable travel documents.

[OST06] Osvik, D., Shamir, A., and Tromer, E. Cache attacks and countermeasures: The case of aes. *Topics in Cryptology* (2006), 1–20. http://dx.doi.org/10.1007/11605805_1.

[Pat95] Patarin, J. Cryptanalysis of the Matsumoto and Imai Public Key Scheme of Eurocrypt88. In proc. *Advances in Cryptology CRYPT0 95* (1995), vol. 963 of *Lecture Notes In Computer Science*, Springer Berlin/Heidelberg, pp. 248–261.

[Pat96] Patarin, J. Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): Two new Families of Asymmetric Algorithms. In proc. *Advances in Cryptology EUROCRYPT 96* (1996), vol. 1070 of *Lecture Notes In Computer Science*, Springer Berlin/Heidelberg, pp. 33–48.

[Ped92] Pedersen, T. P. Non-interactive and information-theoretic secure verifiable secret sharing. In proc. *CRYPTO '91: Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology* (London, UK, 1992), Springer-Verlag, pp. 129–140.

[Per05] Percival, C. Cache missing for fun and profit. In proc. *BSDCan* (2005).

[Pla07] Plantenberg, J. H. Demonstration of controlled-not quantum gates on a pair of superconducting quantum bits. *Nature* **447** (2007), 836–839.

[Pom96] Pomerance, C. A tale of two sieves. *Notices of the AMS* **43** (1996), 1473–1485.

[Pop75] Poplavskii, R. P. Thermodynamical models of information processing. *Uspekhi Fizicheskikh Nauk* **115:3** (1975), 465.

[PS99] Perrig, A. and Song, D. Hash visualization: a new technique to improve real-world security. In proc. *International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC '99)* (1999), pp. 131–138.

[Qui07] Quirke, J. AACSs subset-cover scheme explained. `http://www.jquirke.com.au/index.php/2007/02/12/aacss-subset-cover-scheme-explained/` [2008-01-30], 2007.

[Rey] Reynolds, D. J. A method for electronic voting with coercion-free receipt. Presentation: http://www.win.tue.nl/ berry/fee2005/presentations/reynolds.ppt.

[Rij07] Rijmen, V. Current status of SHA-1. Tech. rep., Rundfunk und Telekom Regulierungs-GmbH, Austria, 2007.

[RO05] Rijmen, V. and Oswald, E. Update on SHA-1, 2005. `http://eprint.iacr.org/2005/010.pdf`.

[RRTV] Rahimi, A., Recht, B., Taylor, J., and Vawter, N. On the effectiveness of aluminium foil helmets: An empirical study. `http://people.csail.mit.edu/rahimi/helmet/`.

[RSA78] Rivest, R. L., Shamir, A., and Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* **21**, 2 (1978), 120–126.

[RW06] Ramzan, Z. and Woodruff, D. P. Fast Algorithms for the Free Riders Problem in Broadcast Encryption. *Lecture Notes in Computer Science* (2006).

[SD] Simka, M. and Drutarovský, M. Montgomery multiplication coprocessor on reconfigurable logic.

[Sha49] Shannon, C. Communucation theory of secrecy systems. *Bell System Technical Journal* **28** (1949), 656–715.

[Sha79] Shamir, A. How to share a secret. *Commun. ACM* **22**, 11 (1979), 612–613.

[Sho94] Shor, P. Algorithms for quantum computation: Discrete logarithm and factoring. *Proc. 35th Annual Symposium on Foundations of Computer Science* **1** (1994), 124–134.

[SKPI07] Sugita, M., Kawazoe, M., Perret, L., and Imai, H. Algebraic cryptanalysis of 58-round sha-1. In proc. *Fast Software Encryption* (2007), pp. 349–365.

[SLW06] Stevens, M., Lenstra, A., and Weger, B. de. Target collisions for md5 and colliding x.509 certificates for different identities. Cryptology ePrint Archive, Report 2006/360, 2006. `http://eprint.iacr.org/`.

[SS] Siekerman, P. and Schee, M. van der. Security evaluation of the disposable ov-chipkaart. `http://staff.science.uva.nl/~delaat/sne-2006-2007/p41/report.pdf`.

[Ste05] Steil, M. 17 mistakes Microsoft made in the Xbox security system. 1–15. `http://events.ccc.de/congress/2005/fahrplan/attachments/591-paper_xbox.pdf`.

[Sti05] Stinson, D. R. *Cryptography: Theory and Practice, Third Edition (Discrete Mathematics and Its Applications)*. Chapman & Hall/CRC, 2005.

[Tel02] Tel, G. *Cryptografie: Bescherming van de Digitale Maatschappij*. Addison-Wesley, 2002 (363 pp.).

[Tel07] Tel, G. *Cryptografie: Beveiliging van de digitale maatschappij*. Departement of Computer Science, Utrecht University, 2007.

[Ten] Tentacle, P. Hitachi develops rfid powder. `http://www.pinktentacle.com/2007/02/hitachi-develops-rfid-powder/`.

[Van01] Vandersypen, L. M. K. Experimental realisation of Shors quantum factoring algorithm using nuclear magnetic resonance. *Nature* **414** (2001), 883887.

[Was01] Wash, R. Lecture notes on stream ciphers and RC4. Reserve University, pp. 1–19.

[WHX+05] Wenyin, L., Huang, G., Xiaoyue, L., Min, Z., and Deng, X. Detection of phishing webpages based on visual similarity. In proc. *WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web* (New York, NY, USA, 2005), ACM, pp. 1060–1061.

[Wob07] Wobst, R. *Cryptology Unlocked*. 2007.

[Wol05] Wolf, C. Multivariate Quadratic Polynomials in Public Key Cryptography. Master's thesis, Katholieke Universiteit Leuven, 2005.

[WP99] Westfeld, A. and Pfitzmann, A. Attacks on steganographic systems. In proc. *Proc. Information Hiding - 3rd Int'l Workshop* (1999), Springer-Verlag, pp. 61–76.

[Wu98] Wu, T. The secure remote password protocol. 97–111.

[Wu02] Wu, T. Srp-6: Improvements and refinements to the secure remote password protocol. Submission to the IEEE P1363 Working Group.

[WYY05a] Wang, X., Yin, Y. L., and Yu, H. Finding collisions in the full sha-1. In proc. *Advances in Cryptology - CRYPTO 2005* (2005), pp. 17–36.

[WYY05b] Wang, X., Yu, H., and Yin, Y. L. Efficient collision search attacks on sha-0. In proc. *Advances in Cryptology - CRYPTO 2005* (2005), pp. 1–16.

[ZHSI03] Zhang, R., Hanaoka, G., Shikata, J., and Imai, H. On the security of multiple encryption or CCA-security+CCA-security=CCA-security?

Index