Cryptography

Cryptography

Mini Symposium, January 21, 2004

Gerard Tel (Ed.)

Instituut voor Informatica en Informatiekunde
Universiteit Utrecht

Contents

Preface

This syllabus is a result of the graduate course on Cryptography, given in December 2003 and January 2004 at Utrecht University. In addition to studying the text book [Tel02], the participants were supposed to connect theory to practice in an individual literature study. The results of this study were presented both in an oral presentation (given on January 21, 2004) and a paper; this document collects the papers.

Fourteen presentations were given on the Symposium, arranged in three tracks. The track on *Systems* was chaired by Hans Bodlaender and contained the presentations by Jan-Willem van den Broek, David Hörchner, Andrew Koster, Dirk Smit, and Leonard Tersteeg (Chapters 1 to 5). The track on *Implementations* was chaired by Marinus Veldhorst and contained the presentations by Gommaar van Strien, Stefan Holdermans, Robert Krenn, Dennis Leman, and Joost Verhoog (Chapters 6 to 9). The track on *Financial Cryptography* was chaired by Thomas Wolle and contained the presentations by Bart de Boer, Els Maes, Jaap Jan Nagel, and Shay Uzery (Chapters 10 to 13).

At `http://www.cs.uu.nl/~gerard/FotoAlbum/F2004/Uithof` a few photos of the symposium can be seen.

This photo shows twelve of the speakers. Top row: Jaap Jan Nagel, Dirk Smit, Joost Verhoog, Robert Krenn, Els Maes, Bart de Boer, Leonard Tersteeg, Dennis Leman. Bottom row: Shay Uzery, Andrew Koster, Jan-Willem van den Broek, Gommaar van Strien. Stefan Holdermans and David Hörchner had left the scene.

I hope that the reader will get an impression of what we did in the course and in the symposium.

Gerard Tel, March 2004.

Chapter 1

Key Scheduling in RC4

Written by *Jan-Willem van den Broek*

RC4, or Ron's Code #4, is a stream cipher[1] designed by Ron Rivest (of RSA fame) in 1987.

The algorithm has been kept a trade secret by RSA laboratories to this day. However, on September 13, 1994, the code for the algorithm was anonymously posted on the internet newsgroup `sci.crypt`[2]. RSA laboratories has not explicitly confirmed or denied the validity of this code, but has responded to critique on RC4 based on this code (see [Riv]). Because of this and the fact that there are no known instances in which this code has produced different results than official RC4 implementations, it is generally believed that the code is indeed valid.

Due to its simple central concept[3], RC4 can be efficiently implemented in software applications. This has resulted in a great popularity of the algorithm[4]. The cipher is used in applications such as Oracle SQL and Microsoft Windows and is featured in protocols such as SSL (Secure Socket Layer), which is used for secure internet connections, and WEP (Wired Equivalent Privacy), which is a security protocol in the IEEE 802.11 (Wi-Fi) standard for mobile networks.

1.1 Description of the RC4 algorithm

The RC4 algorithm can be seen as consisting of two distinct parts. The first part is the so called "KSA" or "Key Scheduling Algorithm". This part of the algorithm uses a variable size key to derive the initial internal state.

[1] Readers may be familiar with other stream ciphers such as A5, which is used in GSM mobile phones. For those who are not familiar with the concept: A stream cipher uses a key to generate a pseudo random stream of values which are then pairwise combined with the values of the plaintext to form the ciphertext. For instance, each value of the plaintext could be combined with the corresponding value of the stream through a xor operation. If the stream were completely random, then the ciphertext would be perfectly secure, as with the appropriate stream, every plaintext of the right length could be derived for any ciphertext.

[2] This is not surprising for those who are familiar with Kerckhoffs' principle, which states that one should always assume that an attacker knows the algorithm that is used, as it is both difficult to keep secret and difficult to replace when discovered.

[3] So simple in fact, that the entire algorithm can be implemented in only 162 bytes of perl code, as has been demonstrated in [Bac].

[4] Some even claim that RC4 is the most popular stream cipher in software applications [FMS01], or indeed the most popular stream cipher overall [Man01].

The second part of the algorithm, the so called "PRGA" or "Pseudo Random Generation Algorithm", then uses this internal state to derive a stream of pseudo random values. These values are xor-ed with the values of the plaintext (usually simply the bytes interpreted as numbers) to derive the ciphertext.

Since xor-ing with some value can be inversed by another xor operation with the same value, the plaintext can be regained by simply applying the algorithm (with the same key, of course) on the ciphertext.

1.1.1 The KSA

It is not hard to see that RC4 would not be a very useful cipher if the number of different streams that can be produced with it were relatively small or if these streams were relatively predictable (that is to say, that it is possible, given part of a stream, to make a reasonable guess as to what the next part of this stream will be). Since the PRGA is completely deterministic, knowing the internal state of the algorithm suffices to accurately determine the stream it produces.

Because of this, it is very important that this state is not easy to determine. In order to accomplish this, RC4 has been designed to have a very large state. This state consist of an array $S \in S_N$, which is a permutation of all possible interger values in $[0 \dots N)$ and two indices $i, j \in [0 \dots N)$, with $N = 2^n$. Since n is typically chosen as 8, this state is very large indeed![5]

Of course, this begs the question of how we get the initial state. This initial state should ideally remain completely secret. Using the entire state as a secret key would not be very practical, however. It is simply much to big for that. Thus, we need to find a way to transform some key K of some reasonable size that is a multiple of n (in practice this is usually 40-256 bits for RC4) into a valid initial state *in such a way that the initial state will appear completely random.*

In other words: what we need here is a hash function. This is exactly what the KSA is designed to do.

The KSA is a function that takes a key K and produces an array S as described above. It does not produce values for the indices i and j (the PRGA will initially use default values of 0 for both these indices). The way it does this is probably best illustrated with a bit of pseudo-code.

The code labeled algorithm 1.1 is a slightly modified (for clarity only) version of that in [Man01]. The variable ℓ is the length of the key K devided by n (so if $n = 8$, then a key of length 24 has $\ell = 24/8 = 3$). We use this variable because RC4 can use a key of any length that is a multiple of n.[6]

As is (hopefully) readily apparent from the code, the KSA simply fills S with the required values and then creates a permutation of these values, by swapping values in a way that is controlled by the key. The entire process can easily be done in $O(N)$ time.

[5] Of course, n is not randomly chosen as 8. The reason for doing so, is that in this way all elements of S, as well as i and j can be represented with exactly one byte. This is especially important since the values of S will later be used to xor the message with. If the plaintext consists of bytes of data (which is, of course, usually the case), then we want the algorithm to occasionally xor with every different value that can be represented with a byte (0 to 255). Henceforth, unless explicitely stated otherwise we will assume that $n = 8$.

[6] Of course, using a key with $\ell > N$ would be pointless.

KSA(K)
Initialization:
    $S \leftarrow \langle 0, 1, \ldots, N-1 \rangle$
    $j \leftarrow 0$
Scrambling:
    For $i \leftarrow 0 \ldots N-1$
        $j \leftarrow (j + S[i] + K[i \bmod \ell]) \bmod N$
        Swap $S[i]$ and $S[j]$

Algorithm 1.1: KSA(K)

1.1.2 The PRGA

So, now that we have a valid initial state, how do we proceed? This is where the PRGA comes in.

As with the KSA, the PRGA is best explained with a bit of pseudocode. Again, this code (1.1) is a slightly clarified version of that found in [Man01].

Like the KSA, the PRGA is a very simple piece of code. It also simply swaps elements of S in a way that is hard to predict without knowing the entire internal state including the pointers. Besides this, in every iteration it also outputs a value z (of size n) that is to be used to xor an n bit value of the plaintext with.

The generation loop can continue as long as more output values are desired. It can, in theory, continue indefinitely. Of course, at a certain point the algorithm will reach an internal state that it has been in before. When this happens it will start to generate an output sequence that it has produced before. This is of course undesirable, as we would like the stream to be as close to random as possible.

Fortunately, this will most likely not be a problem in practice. According to RSA Laboratories [RSA], it is highly likely that the period of RC4 is larger than 10^{100}, which is of course much greater than the size of any reasonable plaintext[7].

[7] In fact, this value, which is known as one "googol" (yes, the name of the popular search engine "google" is indeed inspired by this value) is larger than the estimated total number of fundamental particles in the observable universe. See [Wik].

PRGA(S)
Initialization:
    $i \leftarrow 0$
    $j \leftarrow 0$
Generation loop:
    $i \leftarrow (i + 1) \bmod N$
    $j \leftarrow (j + S[i]) \bmod N$
    Swap $S[i]$ and $S[j]$
    Output $z \leftarrow S[(S[i] + S[j]) \bmod N]$

Algorithm 1.2: PRGA(S)

1.2 Attacks on RC4

Now we arrive at the most important aspect of RC4, which is the security it offers. Is it possible to construct an effective attack on RC4?

The first attack we could try might be to guess (a part of) the internal state. If we could manage to do so, then we could simply apply the PRGA and generate a stream as usual.

Unfortunately for attackers, as has been mentioned before, RC4 has a huge internal state. If we choose $n = 8$, as is usual, then the internal state (consisting of two indices that can take 2^8 different values and a state containing a permutation of all numbers from 0 to $2^8 - 1$) has an effective size of $\log_2((2^8)^2 \times (2^8)!) \approx 1700$ bits (this is the minimum number of bits that is required to give each state a unique number) and this is simply much too large a number of bits to guess, even with very expensive hardware and a lot of time[8]. Besides, as the size of the key is likely smaller than this, we would be better off if we just tried guessing the key.

Even if we could somehow guess some partial state, this would do us very little good since the internal state is transformed in such a complex (from the point of view of an attacker) way. Thus, a partial state will not allow us to generate a useful stream.

These facts suggest that attacking this part of the cipher won't be very effective. Despite much research in this area, for $n = 8$ and suffiently long keys, the best known attack still requires more than 2^{700} time (see [Man01]), which is of course useless in a practical setting.

A much more interesting area of the algorithm to attack is the KSA. The fact that a relatively short key is transformed in such a simple way into this enormous internal state which we cannot effectively attack sounds quite promissing to would-be attackers.

We will now go over the two most dangerous attacks on RC4. Both of these do indeed attack the KSA. Other, less dangerous attacks on RC4 are described in [Roo95] and [GW00]. We will not discuss these here, but they make for a good read to interested readers.

1.2.1 Invariance weakness

A significant weakness of RC4 was discovered by Fluhrer, Mantin and Shamir and described in [FMS01]. In their article they show that for a large number of keys, there is a non-neglible probability that the output stream will have a certain property that can be exploited by an attacker.

The proof of this so-called invariance weakness is quite lengthy and complex, so we will not show it here. Instead we will do the next best thing and prove the weakness for a slightly modified version of KSA, which Fluhrer, Mantin and Shamir call KSA*.

Algorithm 1.3 describes KSA*. As can been seen by comparing it with algorithm 1.1, the only difference between KSA* and regular KSA is that in the former, index i is updated at the *beginning* of the scrambling loop, instead of at the end.

Definition 1.1 *Let S be a permutation of $\{0, \ldots, N-1\}$, t an index in S and b some integer. If $S[t] \bmod b = t \bmod b$, then the permutation S is said to b-conserve the index t. Otherwise S is said to b-unconserve t. The number of indices that S b-conserves is denoted as $I_b(S)$.*

We denote S, i and j after round t of KSA* as S_t, i_t and j_t. For simplicity's sake we will often write I_t for $I_b(S_t)$.

[8] For reference: the estimated total number of fundamental particles in the observable universe can be expressed in about 270 bits.

KSA*(K)
Initialization:
    $S \leftarrow \langle 0, 1, \ldots, N-1 \rangle$
    $i \leftarrow 0$
    $j \leftarrow 0$
Scrambling:
    Until $i > N$
        $i \leftarrow i + 1$
        $j \leftarrow (j + S[i] + K[i \bmod \ell]) \bmod N$
        Swap $S[i]$ and $S[j]$

Algorithm 1.3: KSA*(K)

Definition 1.2 *A permutation S of $\{0, \ldots, N-1\}$ is b-conserving if $I_b(S) = N$.*

Definition 1.3 *Let b, ℓ be integers and let K be an ℓ value key. Then K is called a b-exact key if for every index t, $K[t \bmod \ell] = (1 - t) \bmod b$.*

Please note that this will only be the case if $b|\ell$. This is not a sufficient condition for such a key, however.

Now, we will first prove a lemma that we will need later on.

Lemma 1.4 *If $i_{t+1} = j_{t+1} \pmod b$, then $I_{t+1} = I_t$.*

Proof. The only difference between S_t and S_{t+1} is the swapping operation on indices i_{t+1} and j_{t+1}. If S_t b-conserves i_{t+1}, then S_{t+1} will b-conserve j_{t+1}, since $i_{t+1} \bmod b = j_{t+1} \bmod b$. Of course the same will hold for i_{t+1} if S_t b-conserves j_{t+1}. Since all other indices are unaffected, it follows that $I_{t+1} = I_t$. △

Theorem 1.5 *Let $q \leq n$ and ℓ be integers and $b = 2^q$. Suppose that $b|\ell$ and let K be a b-exact key of ℓ words. Then the permutation $S = KSA*(K)$ is b-conserving.*

Proof. We will prove by induction on t that for any $1 \leq t \leq N$ it turns out that $I_b(s_t) = N$ and $i_t \bmod b = j_t \bmod b$. This implies that $I_N = N$, which makes the initial state generated by KSA*(K) b-conserving.

For $t = 0$ our proof is trivial, since $i_0 = j_0 = 0$ and S_0 is the identity permutation, which is b-conserving for every b.

Now suppose that $i_t = j_t \pmod b$ holds and that S_t is b-conserving. Now $i_{t+1} = i_t + 1$ and $j_{j+1} = j_t + S_t[i_{t+1}] + K[i_{t+1} \bmod \ell] \overset{\bmod b}{\equiv} i_t + i_{t+1} + (1 - i_{t+1}) = i_t + 1 = i_{t+1}$.

Since $i_{t+1} = j_{t+1} \pmod b$ holds, we can now use lemma 1.4 to conclude that $I_{t+1} = I_t = N$ and therefore S_{t+1} is b-conserving. △

On the regular KSA, this property does not hold, though a very similar, but weaker property does hold. The details of this can be found in [Man01].

The importance of this property is that it causes the PRGA to generate an output stream that has a property that can be exploited in an attack. We will come back to this attack later in this section. First we will proof that this is indeed true for a simplified version of the RPGA, which we call RPGA*. As with KSA* and KSA, a very similar, but weaker property holds for the regular RPGA, which we won't proof here due to the fact that this proof too, is quite lengthy and complex. Interested readers can find it in [Man01].

Definition 1.6 *PRGA* is identical to PRGA, with the exception that it does not perform any swap operations. RC4* is identical to RC4, but uses KSA* instead of KSA and PRGA* instead of PRGA.*

Definition 1.7 *Let q and b be defined as before. Let $\{y_r\}_{r=1}^{\infty}$ be the stream produced by applying the PRGA* to S_0 and let $\{x_r\} = \{y_r\} \bmod b$. We now call $\{x_r\}$ a b-pattern and every stream $\{X_r\}_{r=1}^{h}$ that satisfies $\forall_{r \leq h} X_r = x_r \pmod b$ is called a b-patterned stream.*

Analysis has shown that $\{x_r\}$ is periodic with period $2b$ (see [FMS01]).

Claim 1.8 *Any stream generated by PRGA* from a b-conserving state is b-patterned.*

Proof. Let $\{X_r\}_{r=1}^{\infty}$ be this stream. Denote the state components sequences induced by this stream as $\{S_r\}_{r=1}^{\infty}$, $\{I_r\}_{r=1}^{\infty}$ and $\{J_r\}_{r=1}^{\infty}$. Denote the same sequences induced by the stream modulo b as $\{s_r\}_{r=1}^{\infty}$, $\{i_r\}_{r=1}^{\infty}$ and $\{j_r\}_{r=1}^{\infty}$.

We prove by induction on r that $\forall_r I_r = i_r \pmod b$ and $\forall_r J_r = j_r \pmod b$. Since there are no swap operations, S and s do not change and will remain b-conserving throughout the process.

At $r = 0$, it is the case that $i_r = j_r = I_r = J_r = 0$, so our claim holds.

Now, suppose that (for $r > 0$), $i_{r-1} = I_{r-1} \pmod b$ and $j_{r-1} = J_{r-1} \pmod b$, then

$$I_r = I_{r-1} + 1 = i_{r-1} + 1 = i_r \pmod b$$

$$J_r = J_{r-1} + S_{r-1}[I_r] = j_{r-1} + I_r = j_{r-1} + i_r = j_{r-1} + s_{r-1}[i_r] = j_r \pmod b$$

And we now have

$$x_r = s_r[s_r[i_r] + s_r[j_r]] = s_r[i_r] + s_r[j_r] = i_r + j_r \pmod b$$

$$X_r = S_r[S_r[I_r] + S_r[J_r]] = S_r[I_r] + S_r[J_r] = I_r + J_r = i_r + j_r = x_r \pmod b$$

△

So, we have shown that for certain special keys the output of RC4* has the special property of being periodic under modulo b with period $2b$. This is all well and good, but how could we use this property to attack RC4*?

Most of the Time/Memory/Data tradeoff attacks on stream ciphers are based on the following principle. The attacker keeps a database of $< state, output >$ pairs and looks up every subsequence of the output stream in the database[9]. When a match is found, we can use the associated stream in the database to calculate the rest of the stream[10].

A drawback of this approach is that disk access is quite slow when compared to a computation step. As we need a very large database, looking up pairs can take a significant amount of time and we still might end up without a match (listing all possible combinations is usually infeasible, so in practice these databases only contain a subset of the combinations). As an improvement, we could store only outputs with some rare, but easily recognisable property, such as outputs that start with some predefined prefix. Now we would only have to look up

[9] Of course, to get the output stream we require the plaintext. This is often possible because we only look at relatively small parts of the stream. For instance, we could use parts of the stream that we expect to contain standard headers.

[10] Of course there may be multiple states that produce the subsequence that we investigated, so we are not guaranteed to have found the correct state, even if we have a corresponding pair in the database.

pairs when we encounter a stream sequence with this property and if we use a database of the same size as before, then our chances of finding a match in it are significantly increased.

This solution does come with another problem, which is that it usually takes a lot more effort to find a stream with a specific property, rather than a random stream. For instance, if we assume a random distribution and we need a stream that starts with a specific 16 bit value, then we would expect to need to try $2^{16} = 65536$ states in order to find a single stream that starts with our chosen 16 bit value.

The invariance weakness solves this problem, as we can use it to predict which keys or states will produce streams with a rare, but recognisable property. Due to this fact, RC4* is said to have a low "sampling resistance", while ciphers for which we need to try a large number of keys or states to find a stream with a rare but recognisable property are said to have a high sampling resistance.

Another interesting application of this weakness is to efficiently distinguish a stream generated by RC4* from a truly random stream, or even to distinguish ciphertexts generated by RC4*. We do not have the space here to go into details, so we refer interested readers to [Man01].

Both applications of the invariance weakness that we described here are not only applicable to RC4*, but also in weaker, but still usable versions to true RC4. Due to the increased complexity when compared to RC4*, we will not go into this here. Again, interested readers are referred to [Man01] for the details.

1.2.2 IV weakness

Unlike block ciphers, stream ciphers should aways be used with different keys for each encrypted message to remain secure. Suppose for instance that we have two plaintexts x and y, which we both encode with RC4 using key k. For simplicity's sake, we'll assume that x and y are both of size l. Now let z be the stream of length l that the RC4 cipher produces for key k. It is now the case that $RC4_k(x) = x \oplus z$ and $RC4_k(y) = y \oplus z$. If an attacker should somehow learn y and $RC4_k(y)$, then he will now also be able to determine x from its ciphertext as $RC4_k(x) \oplus RC4_k(y) \oplus y = (x \oplus z) \oplus (y \oplus z) \oplus y = x \oplus (y \oplus y) \oplus (z \oplus z) = x$. This attack is appropriately known as a "known plaintext attack".

In many applications, however, it's not practical to always use a completely new key. These applications include many protocols for wireless devices, such as mobile phones. After all, these devices send many data packages that must be encrypted and they are not capable of storing large amounts of secret keys.

What is often done in such protocols is that the device uses a long term secret key, which is somehow combined with some short term variable that is known to both parties involved in the communication, such as the id number of a data package, to generate the key that is used to encrypt the package in question. This variable is known as the Initialization Vector, or IV.

Unfortunately for many such protocols that use an IV in combination with RC4, Fluhrer, Mantin and Shamir have discovered a weakness that occurs when RC4 is used with an IV. This weakness is described in [FMS01].

Now, what is this IV weakness exactly? As we have demonstrated in section 1.2.1, certain patterns in the key will cause recognisable patterns in the output stream. Since the IV is publicly known (or can at least be guessed with a reasonable chance of success), an attacker may be able to combine this with other knowledge he has to deduce the secret key.

Due to space constraints we will only look at one specific system of combining the secret key and an IV. The attack can however also be applied to other (but not all) systems of doing

this. Also, we will leave many details unproven. Interested readers are encouraged to turn to [Man01] for proofs.

The IV attack requires that the attacker knows the first value of the plaintext of a number of ciphertexts that have been encrypted with the same secret key, but a different IV. Fortunately for the attacker, this is not a very big constraint, as in practice the first value is often (part of) an easily guessed constant such as the date, the sender's id, a predefined header, etc.

Since we are only interested in one value of the output stream, we can simplify our model a bit. The first value of the output stream often depends on only three elements of the internal state. After the KSA has finished, the internal state will look like below. The output value will be Z.

Index	1			X			$X+Y$		
S		X			Y			Z		

If the KSA reaches a stage where $i \geq 1$, $X = S_i[1]$ and $X+Y = S_i[1]+S_i[S_i[1]]$, then we call this situation *resolved*. In such a situation, if we model the remaining swaps in the key setup as random, then with probability greater than $e^{-3} \approx 0.05$ (we will not go into the particulars of this number) none of the elements referenced by these three values will participate in any further swaps. In that case, the value $S[S[1] + S[S[1]]]$ will be output as the first value of the stream (this is not hard to see from the decription of the PRGA). Also, with probability less than $1 - e^{-3} \approx 0.95$ one of these values *will* participate in a swap and this will cause the output value to be effectively random, breaking the resolved condition.

The basic idea behind the attack is that we will look at messages with IV values, such that at some point the KSA is in a resolved condition. In these cases, the value $S[S[1] + S[S[1]]]$ will give us information about the secret key. Of course, we will only get the correct value about 5% of the time. However, all other values have a smaller probability of occuring. Thus, if we repeat the procedure for enough IV values, then the actual value of $S[S[1] + S[S[1]]]$ is expected to occur much more often than any other value.

Now, we'll demonstrate how this can be done in a system where the IV is prepended to the secret key. Assume that we have an IV of length I and a secret key of length ℓ. We will try to derive information on a particular value B of the secret key (so we try to get information on $K[B]$) by searching for IV values such that, after the first I steps, $S_I[1] < I$ and $S_I[1] + S_I[S_I[1]] = I + B$. Then with high likelyhood (probability $\approx e^{-2B/N}$ if we model the intermediate steps as random; again we won't go into the particulars of this number), we will be in a resolved condition after step $I + B$ and then the most probable outcome will be $S_{I+B}[I + B]$.

Also note that at round $I + B$ the following operation will take place (remember that element B of the secret key K is element $I + B$ of the combined key):

$$j_{I+B} = j_{I+B-1} + S_{I+B-1}[I + B] + K[B]$$

And due to the swapping operation:

$$S_{I+B}[I + B] = S_{I+B-1}[j_{I+B}] = S_{I+B-1}[j_{I+B-1} + S_{I+B-1}[I + B] + K[B]]$$

Now, we can make the probabilistic assumption that the output value $Out = S_{I+B}[I + B]$ and then predict the value of $K(B) = S^{-1}_{I+B-1}[Out] - j_{I+B-1} - S_{I+B-1}[I + B]$. Here $S_r^{-1}[V]$ denotes the index of value V within the permutation S_r.

Since our prediction $Out = S_{I+B}[I + B]$ is accurate 5% of the time, that means that we find a correct value of $K[B]$ that often as well. By performing this process on multiple IV values we can increase our chance of correctly predicting $K[B]$ to as high as we like.

With enough messages we can deduce the entire secret key, making this into a very dangerous attack indeed. This is especially true as it is completely passive. If executed properly noone will know that the key has been compromised.

A notable protocol that is vulnerable to this dangerous IV weakness is WEP (Wired Equivalent Privacy), which is used in the IEEE 802.11 (Wi-Fi) standard for mobile networks. In 2001 Stubblefield e.a. proved this by determining a network key using only cheap equipment that they managed to assemble in under a week time. Getting the key took them just a few hours. The details can be found in [SIR01].

In fact, in response to this Ron Rivest and RSA Security have adviced in [Riv] that WEP and its successor WEP2 (which is vulnerable to the same attack) be considered broken.

A notable procol that is *not* vulnerable to this attack, even though it does use IVs with RC4, is SSL (Secure Socket Connection). This is because instead of simply concatenating the secret key with the IV to form the session key, as most protocols do, it uses a hash of the secret key and the IV as the session key. Thus, even though an attacker might know the IV, he will not know any part of the actual key that is used for encryption, since a hash is a one-way function (it is impossible to determine the input of a hash function from the output of the function).

1.3 Conclusion

So, is RC4 still a useful cipher? The answer is yes, but only when used properly.

The two main weaknesses of RC4 are the invariance and IV weaknesses that we described before.

The invariance weakness does significantly reduce the amount of work required to attack RC4, but as long as the cipher is used with sufficiently long keys, the amount of work that is required for an attack will still be too great to be feasible.

More dangerous is the IV weakness. As we have mentioned before, it has been demonstrated in [SIR01] that this weakness can indeed be used to launch a practical attack against WEP. However, this is only so because WEP combines the IV and secret key in a much too simple way. If WEP had hashed them to produce the session key, as is done in the SSL protocol, then the IV weakness would no longer be applicable. Thus, RC4 is not necessarily unsafe because of this weakness.

Besides these weaknesses which are specific to RC4, the cipher also suffers from the weaknesses that all stream ciphers suffer from. Chief among these is that the cipher should never be used with the same key, as this will make it vulnerable to a known plaintext attack. However, we already know that we can work around this problem by using IVs. Provided, of course, that we combine the secret key and the IV in a secure way, such as by hashing them together.

Another problem of stream ciphers is that they are vulnerable to a so called "bit-flipping" attack. Since a stream cipher codes each value of the plaintext independently into a value of the ciphertext, it is possible to change (flip) a few bits of the ciphertext without this being immediately apparent, even after decryption. For instance, if the changed value was some binary number, then the new value will still be a binary number, albeit another one[11]. Of course the attacker still will not know the value after decryption, but if he only wants to subtly cause confusion, then this is all he needs to do.

[11] See [How] for a slightly more elaborate example.

Again, protection to this attack is not hard to achieve. Simply send a hash with each message. If they do not match after transmission, then the receiver will know the message cannot be trusted.

As we have shown, all the potential security issues concerning RC4 can be avoided as long as the cipher is used in a responsible way. Thus RC4 can still be considered secure. And since it can be very efficiently implemented in software applications, it is still a useful cipher for many applications. We can therefore conclude that RC4 still has a place in modern cryptography.

Chapter 2

The Fortuna PRNG

Written by *David Hörchner*

This chapter is about the generation of pseudo random numbers. A specific pseudo random number generator, *Fortuna*, will be explained in detail. Many cryptographic systems use random numbers. Some examples include:

- Generating session and message keys.
- Seeds for algorithms to calculate large prime numbers for *RSA* or *ElGamal* systems.
- Random challenges in authentication protocols.
- Random parameters for digital signatures.

All these applications can be successfully attacked if the random numbers can be predicted or influenced. Random number generators can be considered as fundamental building blocks of many cryptographic systems.

2.1 What is randomness?

The measure for randomness is called entropy. Entropy is expressed in bits and measures the uncertainty an observer has about a value. A n-bit word that is completely unpredictable to the observer has n bits of entropy. If the observer knows, for example, that the word takes on 8 different values, each with an equal probability of occurring, than the word has 3 bits of entropy. A common definition of entropy for a variable X is:

Definition 2.1 $H(X) = -\sum_x P(X = x) \log_2 P(X = x)$

Where $P(X = x)$ is the probability that variable X has value x. There are a lot more mathematical details associated with the concept of entropy, but we do not need them here. The important thing to remember is that the more an observer knows about a value the smaller its entropy value is.

2.2 Real randomness versus Pseudo randomness

By its deterministic nature, a computer can not *generate* real randomness. Real random data can however be extracted from the physical world. We could use for example the thermal noise in electrical circuits or the precise timing of *Geiger* counter clicks. Unfortunately, many cryptographic systems do not have access to these kinds of high quality sources of real random bits. Other possible sources of entropy include the exact timing of keystrokes and the precise movements of a mouse. However, an attacker could measure or influence a source and thereby reduce the amount of entropy the source has.

There are a number of problems associated with the use of real random data sources:

- Data is not always available, for example, when we use the timing of keystrokes a user has to type something in order to generate entropy.
- The amount of data is limited. For example, a *Kerberos* server generating thousands of keys every hour needs a lot of random data.
- It is hard to predict how much entropy a particular source contains.

Because of the disadvantages of the use of real random numbers we often use something called pseudorandom numbers. These are numbers that are deterministically generated using an algorithm. A pseudorandom number generator (PRNG) has internally some (random) data called the seed from which the pseudorandom numbers are generated. If an attacker knows this seed he can predict the output of the PRNG.

A distinction can be made between traditional PRNG's and PRNG's for use within a cryptographic context. Traditionally PRNG's are designed to generate data that is statistically random. For a PRNG used in cryptographic systems we have the additional requirement that the output of the PRNG should be unpredictable to an attacker. In other words, the output should be indistinguishable from real random data, even if the attacker has a lot of computational resources. A PRNG that has this property is called cryptographically strong. *Fortuna* is an example of a cryptographically strong PRNG.

2.3 Attacks on a PRNG

There are a number of possible attacks on a PRNG. See also [KSWH98].

- Direct Cryptanalytic Attack. In this type of attack the attacker can distinguish between real random outputs and the output from the PRNG. This type of attack is of course only possible if the attacker can see the PRNG output. This may not be the case, for example, when the output is used to generate secret keys.
- Input-Based attack. In this type of attack the attacker has knowledge about, or can control, (some of) the input of the PRNG. Input-based attacks can be further subdivided into:
  - Chosen-input attacks, in which the attacker controls some of the input.
  - Replayed-input attacks, in which the attacker inputs previous input again in the PRNG.

  - Known-input attacks, in which the attacker has knowledge about the input. For example because the attacker can perform measurements on the input data.

- State Compromise Extension Attacks. In this type of attack, the attacker knows the internal state of the PRNG and uses this information to predict future output or reconstruct previous output from the PRNG. There are various situations in which the attacker may acquire knowledge of the internal state. The most likely one is during an initialisation of the PRNG (for example after a reboot) when there isn't enough entropy to make guessing of the internal state unfeasible.

It is assumed that an attacker eventually may acquire the internal state. When this happens we say the internal state has been compromised. In a traditional PRNG this means that any future output will be predictable to the attacker. This type of attack is called an permanent compromise attack.

To overcome this problem the PRNG has to recover from a compromised state. To achieve this, real random data is used to mix in with the internal state. This operation is called a reseed. But if only small amounts of entropy are mixed in, the PRNG will still be vulnerable against so called iterative guessing attacks. If the amount of entropy that is mixed in with the internal state is small enough, the attacker can try all possible combinations for the random input data and compare the output for each combination with the actual output of the compromised PRNG. After a match the attacker knows the new internal state. In this way the PRNG will never recover from a compromise.

To defend the PRNG against iterative guessing attacks we must ensure that it is unfeasible to try all possible combinations for the random input with is mixed in with the internal state. This means that an attacker should spend at least 2^{128} steps. Therefore, at least 128 bits of entropy have to be mixed in with the internal state. Because the incoming random data contains only small amounts of entropy, the data is stored in a pool. When the pool contains enough entropy it is mixed in with the internal state.

The concept of an entropy pool leads to another problem. We want the PRNG to recover from a compromise as soon as possible but we do not know exactly when the pool contains enough entropy to protect against iterative guessing attacks. The PRNG *Yarrow* tries to solve this problem by using so called entropy estimators. But it is impossible to estimate the amount of entropy in all situations because it depends highly on the amount of knowledge the attacker has of the entropy sources. The *Fortuna* PRNG solves the problem of entropy estimation by not using it at all.

2.4 Fortuna

Fortuna is a cryptographically strong pseudo random number generator. It is named after the Roman goddess of chance. It was designed by Niels Ferguson and Bruce Schneier[FS03] and improves their previous design *Yarrow*[KSF99]. The main advantage of *Fortuna* compared to *Yarrow* is that *Fortuna* doesn't use entropy estimators.

Fortuna consists of three parts:

- **The generator** which transforms a seed deterministically into a stream of pseudo random output.

- **The accumulator** which functions as an entropy pool storing incoming events and eventually reseeds the generator.

- **The seed file control** which ensures that the generator is in a secure state immediately after a reboot. This mechanism is needed because after a reboot the system hasn't generated enough entropy to securely seed the generator.

Each of the three parts will be discussed in detail in the following sections.

2.5 The generator

The generator is the part of Fortuna that actually generates the pseudo random output data. The pseudo random data is the output of a deterministic algorithm. For this algorithm *AES*, *Serpent* or *Twofish* can be used. The input of the algorithm is a fixed-size internal state consisting of a 256-bit block cipher key K and a 128-bit counter C. The output is generated by using the block cipher in counter mode. This means that the current counter is encrypted with the current cipher key. After that the counter is incremented. One encryption results in a block of 128 bits pseudo random output data. If a user requests more than 128 bits of random data multiple blocks will be generated and appended to the output.

Because of the deterministic nature of the algorithm there is the possibility of a State Compromise Extension Attack as described in a previous section. Iterative guessing attacks will be handled in the accumulator, which will be described in the next section. This ensures that future outputs will be unpredictable. But what about previous outputs? If an attacker knows, at some time, the internal state (consisting of the counter and the block cipher key) he will be able to reconstruct previous outputs of the generator, and perhaps find some values that were used to generated some important secret keys. This type of attack is called a backtracking attack.

The generator is protected against this type of attack by generating after each request 2 extra blocks (256 bits) of data which are used to replace the current key. The old key is destroyed and backtracking attacks are no longer possible.

The counter is not reset after each request. If the counter would reset, and a key value ever repeats and request of a fixed size were used, then the new key value would also be a repeated key value. This would lead to cycles in the output, and to avoid this the counter is never reset.

To defend the generator against Direct Cryptanalytic Attack it must be impossible for an attacker to distinguish between real random data and the pseudo random output data. Therefore the generated data should be statistically random. In a sequence of real random data there will be repeated values, but if we use a block cipher in counter mode there will never be a repeated value in the output. To ensure that this statistical artefact will be hard to detect the output size of a single request will be limited.

According to the *Birthday Theorem*, if we randomly draw from a set with n members, we see a duplication after approximately $\sqrt{n}$ draws. This means that if we generate 2^{64} random 128-bit blocks we expect to see about one repeated block value. So, if an attacker request a number of times an output of 2^{64} blocks he will notice that the output of the generator is not completely random. Therefore the generator is limited to an output size of 2^{16} blocks. This equals to 1 MB of pseudo random data. If a user needs more than 1 MB of data than he can do multiple request.

The probability of a repeated block value in a set of 2^{16} random 128-bit block is about $\frac{(2^{16})^2}{2*2^{128}} = 2^{-97}$. This means that an attacker has to make 2^{97} request of 2^{16} blocks before he

will notice that the output data is not completely random. The total work the attacker has to do will therefore be in the order of 2^{113} operations.

This is not the 2^{128} steps the designers of *Fortuna* were aiming for, and they could have used for example SHA-256, but they argue that this would be a lot slower. Also notice that the attacker has to perform the 2^{113} steps on the computer that is being attacked.

The generator can be used as a separate module for things like *Monte Carlo* simulations and situation in which there is a need for reproducible statistically random numbers.

2.5.1 Reseeding

The generator includes a reseed operation that takes an arbitrary string an uses it to update the block cipher key. This is done by hashing the concatenation of the current key and the input string using the $SHA_d - 256$ hash function. The reseeding procedure will be called by the accumulator or the seed file control after a reboot.

2.6 The accumulator

The accumulator is the part of *Fortuna* that implements the concept of an entropy pool. It collects real random data from the environment and uses it to reseed the generator.

2.6.1 Input of entropy

Fortuna uses multiple source of entropy. Each source is identified with an unique number in the range 0 - 255. A source can be anything, like accurate mouse movements and clicks, precise keystroke timings, events generated by printers or disk drives, etc. A user can include any source as long as he makes sure that there is al least some entropy input in the accumulator.

Each entropy source will generate events. The events are encoded in a fixed form: the first byte encodes the source number, the second byte encodes the number of bytes of event data and the following bytes contain the actual event data.

Data that can be easily predicted by an attacker should not be included in the event. For example, only the least significant bits in a timer event should be included, because it is save to assume that an attacker can, for example, guess the year, month and day of an timer.

It is assumed that the attacker has knowledge of, or controls, some of the entropy sources. Furthermore the attacker can request pseudo random data at any time form the PRNG.

2.6.2 The pool mechanism

To make an iterative guessing attack unfeasible, the accumulator collects entropy which is used to reseed the generator. If enough entropy is gathered between reseeds the PRNG will recover from a possibly compromised state. The problem that arises when using an entropy pool is that it is difficult to determine when to do a reseed. *Yarrow* uses entropy estimators and a number of heuristic rules to accomplish this.

Fortuna uses a much better mechanism of multiple pools. There are 32 pools, numbered P_0, P_1, ..., P_{32}. With each source there is a string associated of unbounded length. Each entropy source distributes its events evenly over the 32 pools in a cyclic fashion. This way the pools will be filled more or less evenly. The incoming events are appended to the string of the appropriate pool. Eventually only a hash of the string will be used. Therefore, in an actual implementation each pool only stores a partial hash of its associated string.

If pool P_0 has reached a certain fixed size there will be a reseed. Reseeds are numbered successively 1, 2, 3, ... etc. The reseed number r determines if a pool is used in a reseed according to the rule:

Pool P_i is included if 2^i is a divisor of r.

This means that pool P_0 is used every reseed, pool P_1 every other reseed, pool P_2 every fourth reseed, etc. After a pool has been used in a reseed its associated string is reset to the empty string.

The rule to determine which pool to include in a reseed ensures an eventual recovery from a compromised state. As assumed, the attacker might have knowledge about, or control, some of the entropy sources.

If only pool P_0 is used in a reseed the attacker could know enough of the data collected in pool P_0 to reconstruct the internal state using an iterative guessing attack. Pool P_1 however contains twice as much data that is unknown to the attacker because twice as much data (containing twice as much entropy) is collected in this pool before it is used in a reseed. Pool P_2 contains four times the amount of entropy of pool P_0, etc. So, even if there exist only a small amount of entropy in the entropy sources, there will be a pool that collects enough (128 bits) entropy to defeat the attacker.

After we reseed with a pool that contains al least 128 bits of entropy the PRNG will recover from a compromised state. The use of this mechanism means that the time between recoveries is dependent on the rate at which entropy is collected in the pools. If an attacker knows a lot about the sources, they contains a small amount of entropy and the rate at which entropy flows into the pools will be low. This leads to longer times between recoveries from compromised states.

We will now derive the exact dependency. We assume the rate at which entropy enters the accumulator is fixed. We call this quantity p . This means that after t seconds the accumulator has collected pt bits of entropy. Because events are evenly distributed over the pools, and there are 32 pool2, each pool collects $pt/32$ bits of entropy in t seconds.

An attacker will be defeated if the generator is reseeded with a pool that contains at least 128 bits of entropy. Let t be the time between reseeds. Because pool P_i is used every 2^i reseeds, it collects $2^i pt/32$ bits of entropy before it is used in a reseed.

There will be a recovery the first time we reseed with a pool P_i where:

$$128 < 2^i pt/32 < 256 \tag{2.1}$$

The upper bound of 256 is derived from the fact that if pool P_i collects more than 256 bits of entropy, pool P_{i-1} has already collected 128 bits of entropy (pool P_i collects twice as much entropy as pool P_{i-1} between reseeds). We can rewrite this inequality:

$$2^i pt/32 < 256$$

which is the same as

$$2^i t < 8192/p \tag{2.2}$$

This result means that the time between recoveries is bounded by the time it takes for the accumulator to collect 8192 ($=2^{13}$) bits of entropy. This is a worst cause bound that can actually happen if we reseed just before a particular pool P_i has collected 128 bits of entropy. In this case a recovery will occur after a reseed that uses the next pool P_{i+1}. At the time of this reseed pool P_{i+1} has collected almost 256 bits. This means that the total amount of

entropy that was collected is about $32 * 256 = 8192$ bits. In an ideal situation 128 bits are needed to recover from a state compromise. So, the solution given here needs only 64 times as much randomness.

A possible attack on the accumulator is to feed it so many known event that even the last pool P_{31} doesn't collect enough (8192 bits) entropy between reseeds. This seems very unlikely but to exclude this possibility the time between reseeds is limited to a minimum of 100 ms. This means that a pool P_{32} would be used in a reseed after more than 13 years.

2.7 Seed file management

When the system that runs the PRNG is rebooted the PRNG can only generate pseudo random data after it has collected enough entropy to reseed the generator. Even after this first reseed it is not sure if the state of the generator is unpredictable to an attacker.

These problems are solved in Fortuna by using a seed file. This is an file that contains enough entropy to get the PRNG into an unknown state. After the seed file has been used it must be rewritten with new data. The concept looks simple but a number of problems have to be solved in an actual implementation.

The first time the PRNG is used, the seed file will be read and a reseed operation will take place using the data from the seed file. Directly after the reseed some pseudo random data is generated to update the seed file.

Here we encounter the first problem. An attacker could request data after the first reseed but before the seed file has been rewritten with new data. Suppose, that before updating the seed file the computer that runs the PRNG crashes. The next time the computer reboots, a user might request data right after the first reseed. Because the seed file is exactly the same as the previous time the computer booted, the state will repeat and the user gets the exact same pseudo random data the attacker got earlier. To solve this problem, the procedure that reads and rewrites the seed file should be atomic. During the use of the PRNG the seed file will be updated occasionally.

To ensure the unpredictability of the output of the PRNG, it is important to avoid repeating the same internal state twice. However, if the file system is restored from a previously made backup, the same internal state will be repeated. The PRNG will be compromised until enough entropy has been gathered. There is no good solution to this problem. The current time could be used hashed together with the current seed file but this doesn't help much if the attacker can control the clock. A solution to this problem will highly depend on the actual platform that runs the PRNG.

A third problem arises the first time a computer is booted. In this case there no seed file that can be used to reseed the generator. New computers are often pre-installed with exactly the same data. To make things worse, right after the first boot a number of important keys have to be generated. A solution to this problem is to use a second machine that generates a unique seed file for each separate machine during installation. Another option would be to request a human tester to generate some random data by moving the mouse or pressing some random keys.

Chapter 3

Captcha'ring a robot

Written by *Andrew Koster*

This chapter describes how problems from Artificial Intelligence can be used to prevent robots on the Internet from performing actions we want to reserve for Humans. We begin with a global explanation of what Captchas are (Section 3.1) and what their various uses are and how they can be used in fighting spam and other intruding programs. Then we explain the formal definition of a Captcha and show how they are useful in aiding AI research (Section 3.2). We also show a formalization of an example of an aural Captcha (Section 3.3). Finally we show what methods there are to solve Captchas (Section 3.4) and show they're not as secure as initially thought.

3.1 What is a Captcha?

Casus 3.1 discribes a relatively harmless example of a couple of students having a bit of fun with so called bots, but there are far more serious problems with these bots roaming the internet. The main problem is that they open free mail accounts and send out junk mail to unsuspecting users, but there are a myriad of other applications for bots and there are more being thought of every day.

To counter the use of these bots on the internet the Captcha was invented, as is detailed in [ABHL03]. Captcha stands for Completely Automated Turing test to tell Computers and

Casus 3.1: How to win votes for your University

When the wellknown website www.slashdot.com posted a poll asking what the best University in the USA was, they were expecting a lot of students to vote. They took the usual precautions in that IP addresses were logged and couldn't vote twice.
They were unprepared for what really happened, though. The students at Carnegie Mellon University wrote a clever voting program that managed to bypass all security and vote for CMU. Soon afterwards people at the MIT caught on and proceeded to do the exact same thing. When the poll closed MIT had won with 21,156 votes with the CMU chasing closely at 21,032 votes. All other universities had less than a 1000 votes each.

Casus 3.2: An example of the Gimpy Captcha

In this Captcha the user is asked to type three words appearing in the picture.

Humans Apart and is also often referred to as a Reverse Turing Test (or RTT).

A Captcha is a computer program that can automatically generate tests, that:

1. Humans can pass, but
2. Current computers programs cannot.

The reason it is commonly referred to as a *Reverse* Turing Test is clear: where the test devised by Alan Turing in [Tur50] entailed a test being posed by a human being and having the goal to test the intelligence of a computer, a Captcha is a test posed by a computer and has the goal to keep other 'dumb' computer programs out.

There are already Captcha programs in use. Internet Companies like Yahoo! use a Captcha called EZ-Gimpy to keep bots from opening email accounts and programs such as MailFrontier require the sender of an email message to answer a Captcha before the message is accepted.

3.2 How does a Captcha work?

The idea of a Captcha is to create a set of problems that cannot be solved by current computer programs, but are relatively easy for humans to solve. Luckily there's a whole field of research devoted to exactly these problems, the field of Artificial Intelligence and we have a rich source from which to select our problems. Current Captchas are mainly based on problems in vision and object recognition, but progress is being made in the creation of aural Captchas as well as Captchas based on pattern recognition, face recognition and other such wellknown *Hard* AI problems. We can't just use any AI problem, though. We need to be able to generate our instances of the problem with a computer program and furthermore require the problem to be public. We can't use the secrecy of the working of our program as a valid argument

Casus 3.3: What is wrong with a Captcha

If we don't publicize our algorithm, but keep it secret, we can see from a simple example that it is neither secure, nor does our program contribute to AI research. If for example we built a Captcha that had a secret text. It either generated a paragraph using a text-generation algorithm, or chose a random paragraph from this text and then asked the user whether this paragraph made sense (the most sophisticated text-generator known at the moment cannot create a complete sensible paragraph), it would be able to discern between computers and humans reliably (there is no known algorithm to discern between sensible text and gibberish). This Captcha, though, would only work as long as our text was secret. If the code was reverse engineered and the text retrieved, it would be very easy to break the Captcha by matching the paragraph to the text. Now we would have solved the problem without any algorithmic advances.

for the functioning of our Captcha, nor can we use a secret database. The security of the program may only be based on a small amount of randomness in generating the instances. This because of techniques like reverse engineering to discover the code used, but also because our Captchas serve a second purpose, we wish to advance research in Artificial Intelligence with them. We want solutions to our problems to be useful in a general field and **we *want* our Captchas to be solved!**

3.2.1 The main goal of Captcha*s*

What we actually want is to solve these Hard AI problems and advance the research into these problems. The use of Captchas for this purpose generates a win-win situation. Until our problem is solved, we have a reliable way of discerning between computer programs and humans. When we find a solution, though, there are many applications for the underlying AI problem. In [ABHL03] there is an application of the solution for a promising Captcha given for use in robust steganography (see chapter 7 for more details on the problems of steganography), but solutions obviously also have a use in the application of AI from where they originally came. Object and pattern recognition are important problems in Computer Vision for instance.

The use of Captchas in the solution of Hard AI problems is obvious. There are many malicious programmers who would like to (ab)use the sites protected by a Captcha and they will set their minds against these programs and therefore, indirectly, be contributing to the AI community.

What is needed to further this goal is a precise way of defining our Captcha. We need a formal definition of a Captcha thus that if the Captcha can be beaten by a computer, it provides a solution to the underlying AI problem.

3.2.2 Formal Definitions

Probabilistic programs and success

If a program $P(\cdot)$ is a probabilistic program, we will use $P_r(\cdot)$ to denote the deterministic program resulting from P when random coins r are used. $\langle P_{u_1}, S_{u_2} \rangle$ is the output of the interaction of two probabilistic programs P and S using random coins u_1 and u_2. We call a program V a Test or Verifier, if for all P, u_1 and u_2 the interaction between P_{u_1} and V_{u_2} terminates and $\langle P_{u_1}, V_{u_2} \rangle \in \{accept, reject\}$. In this case P is called the Prover.

We define the *success* of an entity A over a test V by:

Definition 3.1 $\boldsymbol{Succ}_A^V = \Pr_{r,r'}[\langle A_r, V_{r'}\rangle = accept]$

Hard AI Problems and Human Executability

In the AI community there is no proper definition of an AI problem and the following definition probably doesn't capture all the problems that fall in their domain. It is, however accurate and adequate for defining the class of problems we will be using in this chapter.

Definition 3.2 *An AI problem is a triple $\mathcal{P} = (S, D, f)$, where S is a set of problem instances, D is a probability distribution over S and the function $f : S \to \{0,1\}^*$ answers the problem instances.*

We require for a problem $\mathcal{P}$ that a fraction $\alpha > 0$ of the humans H for any problem $x \in S$ holds that for a certain δ:

$$\Pr_{x \leftarrow D}[H(x) = f(x)] \geq \delta$$

We call such a problem (δ, τ)-*solved* if there exists a program A, running in time at most τ on any input from S, such that:

$$\Pr_{x \leftarrow D, r}[A_r(x) = f(x)] \geq \delta$$

Such a program A is called a (δ, τ)-solution to problem $\mathcal{P}$

Definition 3.3 *A problem $\mathcal{P}$ is called a* (δ, τ)-hard AI Problem *if there currently is no (δ, τ)-solution known AND the AI community agrees it is hard to find such a solution.*

Apart from needing a formal definition of a Hard AI problem we also need a formal definition of when a test V as seen in 3.1 can be passed by humans.

Definition 3.4 *We call a test V $(\alpha, \beta) -$ human executable if at least an α fraction of the human population it holds that:*

$$\mathbf{Succ}_H^V > \beta$$

We can give no formal proof that a test V is actually (α, β)-human executable, but we can give an empirical proof. The success over a test will often depend on the population's origin, culture, sensory disabilities, etc. If we have a test based on the English language it will have a very low success rate in China, for example.

CAPTCHA

With the above definitions we can give a formal definition of a CAPTCHA as follows:

Definition 3.5 *A* (α, β, η)-CAPTCHA *is a test V which is (α, β)-human executable and has the additional property:*

There exists a (δ, τ)-hard AI problem $\mathcal{P}$ and a program A, such that if for a program B holds that $\mathbf{Succ}_B^V > \eta$, then A^B is a (δ, τ)-solution to $\mathcal{P}$.

Figure 3.4: The EZ-Gimpy Captcha

A^B is the program A integrating program B, which is the case-specific solution to our Captcha, to generate a general solution to $\mathcal{P}$.

There are two additional requirements for a Captcha that we don't formalize as above. In our definition of human executable problems we have made no requirements as to the time it takes humans to solve the problem. For a Captcha it is crucial that this time is short, otherwise they have no practical value. Noone wants to puzzle for an hour before opening an email address.

There is another requirement on the problems we can use, namely that we have to generate instances of these problems automatically and the security must be in the small amount of randomness used.

This definition of a Captcha makes it so that not nearly all Hard AI problems qualify, but there are still enough left to keep us busy. Especially problems with object and text recognition are popular, but we will also give an example of an aural Captcha.

3.3 Actual Captchas

As seen in [ABHL03] it is now easy to give a formal definition of a Captcha and prove it has the required properties. This also means that it, if ever solved by a computer, gives a solution to the underlying AI problem, which is exactly what we want. Furthermore, they show that the solution to their Captcha can be used in steganography. We would like to do something different here. In [KLS02] Kochanski, Lopresti and Shih introduce a design for creating an aural Captcha, using speech. The idea is very similar to the Captcha mentioned before, EZ-Gimpy, but instead of transforming the image with the written word in it, we use a soundbyte to which noise is added. Empirical tests presented in their paper show good results using current speech recognition programs, but give no formal evidence it is an effective Captcha.

3.3.1 The AI problem family

In this section we will generalize the the abovementioned Captcha to an actual problem in the AI community, namely speech recognition. We will use a simple definition of speech and proceed from there.

Definition 3.6 *A soundbyte $b \in \mathcal{B}$ is a table of length n describing the waveform of the corresponding sound. The index of the vector may be seen as time, where the value is the amplitude.*

Definition 3.7 *A soundbyte b will be recognized as speech if the corresponding sound s of the waveform that is represented in b is such that: there is a human language $\mathcal{L}$ such that a portion $\alpha \sim 1$ of the human native speakers of $\mathcal{L}$ recognize s as words in $\mathcal{L}$.*

Definition 3.8 *A transformation of a soundbyte is a function $f : \mathcal{B} \to \mathcal{B}$. It takes a soundbyte a and outputs a soundbyte b, not necessarily the same length. Examples are adding noise, changing the pitch, mixing with other sound, etc.*

State of the art speech recognition programs can recognize plain text to a reasonably reliable extent, but if we distort the sound they have problems, as demonstrated in [KLS02]. We formalize our problem family as:

Problem Family $\mathcal{P}$: Choose a soundbyte b and a transformation on this soundbyte t with output $t(b)$. $\mathcal{P}$ consists of writing a program that takes $t(b)$ and outputs b.

Or formally: Let $S_{\mathcal{B},\mathcal{T}} = \{t(b) : t \in [\mathcal{T}] \text{ and b} \in [\mathcal{B}]\}$, $D_{\mathcal{B},\mathcal{T}}$ be the distribution on $S_{\mathcal{B},\mathcal{T}}$ and $f_{\mathcal{B},\mathcal{T}} : S_{\mathcal{B},\mathcal{T}} \to [\mathcal{B}]$ be such that $f_{\mathcal{B},\mathcal{T}}(t(b)) = b$. Then $\mathcal{P}_{\mathcal{B},\mathcal{T}} = (S_{\mathcal{B},\mathcal{T}}, D_{\mathcal{B},\mathcal{T}}, f_{\mathcal{B},\mathcal{T}})$.

It is extremely simple to make a (δ, τ)-solution by guessing the value of b as the byte with the highest probability. Then $\delta_{\mathcal{B}} = max\{\Pr_{i \leftarrow \mathcal{B}}[i = b]\}$ and $\tau_{\mathcal{B}}$ is the time it takes to describe b. We therefore require a solution to have $\delta > \delta_{\mathcal{B}}$ and $\tau > \tau_{\mathcal{B}}$.

We must also remark that Ahn, Blum, Hopper, and Langford had foreseen it was possible with sounds too, but chose images to demonstrate their theory. In this chapter we choose sounds, but the definitions are analogous to theirs.

3.3.2 A CAPTCHA based on this problem

We will now show that the CAPTCHA based on speech proposed in [KLS02] is an actual CAPTCHA according to the definition in Section 3.2.2. Kochanski, Lopresti, and Shih propose we synthesize a random 5 digit sentence as our soundbyte b and have a table of transformations, using white noise, mixing in songs, chants, cutting out bits of the soundbyte and other such transformations. The prover P has to state what the 5 digits were. We enhance this CAPTCHA by requiring a larger database of words for the language. For example the complete Webster English dictionary.

We formalize this: an instance of our CAPTCHA is a tuple $\mathcal{S} = (\mathcal{B}, L, \mathcal{T}, \tau)$ with $\mathcal{B}$ the soundbytes produced by a generic male voice speaking the words in database L and $\mathcal{T}$ the transformations in the table mentioned above. τ is a deadline. The program starts by choosing 5 random elements of $\mathcal{B}$ and sticking them together to produce soundbyte b. Then choosing a random transformation $t \in \mathcal{T}$ and calculating $t(b)$. The CAPTCHA verifier sends the prover $t(b)$ and asks him to type the 5 digits in the correct order. If the digits match the labels in the correct order the verifier accepts, if the time runs out it rejects and if the digits are incorrect it rejects.

Theorem 3.9 *Any program B that has success greater than η over $\mathcal{M}$ can be used to (δ, τ)-solve $\mathcal{P}$, with $\delta = \eta$.*

Proof. Let B run in time at most τ and have success $\sigma_B \geq \eta$ over $\mathcal{M}$. Using B we construct a program A^B that is a (δ, τ)-solution to $\mathcal{P}$. We note that B recognizes all words in the database L. Therefore all A^B has to do is to pass on input j to program B. It will recognize the words in j with a percentage $\sigma_B \geq \eta = \delta$. Because we made no requirements on L, in our example it is English, but it can be any language, or the table of transformations $\mathcal{T}$ a solution to this CAPTCHA is a valid solution to $\mathcal{P}$. △

Figure 3.5: Effectivity of the aural Captcha

Furthermore we need to show that humans can solve the Captcha. For this purpose we refer to [KLS02] where they show in more detail when it gets hard for humans to solve the problem and what the gap is between current programs and humans.

The use of the solution to this CAPTCHAis obvious. There is alot of research being done into this AI problem already and speech recognition is a 'hot topic'.

3.4 Breaking a Captcha

In this section we look at a few ways in which attackers have tried to circumvent the difficulties posed by Captchas.

3.4.1 The legitimate research

The first way of breaking a Captcha is the way it was meant. AI research advances and finds clever algorithms and methods to render the Captcha useless. This is what happened to EZ-Gimpy. Although Yahoo! still uses it, a decent OCR-program will recognize the text without too many problems. The actual Gimpy Captcha as demonstrated at [AHL00] has been solved as well, although it can still be used in many applications. Mori and Malik give a detailed method in [MM03] for getting a 33% chance success over Gimpy and text recognition in cluttered backgrounds is not seen as a big problem in AI anymore.

3.4.2 Stealing cycles from humans

The usual workaround methods for beating Captchas are something completely different. Malicious programmers wishing to beat Captchas aren't interested in advancing AI research. All they want is for their spam-bots to generate mass emails. Therefore in stead of solving the Captcha computationally they involve a human 'in the loop'. Two methods:

Captcha farms: By exploiting children in 3rd world countries CAPTCHAs can be solved. The idea is simple: labour is cheap, so fill a classroom with children and every time a bot encounters a CAPTCHA it will send it to a child there. He will solve it and send the bot the answer. There is no evidence that this method is actually being used, but it is plausible.

Pornographic websites: They work in the same way as above. Someone wanting to solve CAPTCHAs opens up a pornographic website and if a bot encounters a CAPTCHA it sends it there. A random user then gets the message: "if you wish to view this picture, solve this CAPTCHA!" and unwittingly releases another spam bot onto the net. This method is being used in practice and has proven successful.

An innocent bystander can look at this method as "cheating", but apart from the question whether it is or not, do we care? Is it really so bad to use humans in the loop when solving CAPTCHAs? From the perspective of the AI community we could see this as the ideal cooperation between humans and computers. All problems that can easily be solved by computers are done so faster and better, while problems that prove insurmountable hurdles are passed on to humans. But, better still, there is masses of human input generated, we could use this to train a computer program to solve the problem itself. There are many learning algorithms like neural networks, which can be trained in this manner. Say for example you need a new jersey and ask your computer program what colour looks better on you. At the start the program won't have a clue, but now say we take various photos of you wearing different colour jerseys and post them on internet and ask people to vote which one looks best. If enough people vote we can train our program with this information and in the future it can decide itself if you need a new t-shirt. This can also be done with CAPTCHAs. Feed a learning program enough solutions to a CAPTCHA and it might learn to solve them. Nothing is guaranteed, but learning algorithms are improving all the time and who knows what will become possible with the new technologies being developed.

3.5 Conclusion

By giving a formal definition of CAPTCHAs there is a clearly defined research area to which they can be applied. Within this area they essentially create a win-win situation, where, as long as they remain unsolved we have a way of stopping unwanted computer programs, but when they are solved a valuable research problem has been solved too. The CAPTCHA as a way of fighting spam is insecure, because of the problem outlined in the last section, but although it doesn't stop them, it does make it harder for malicious programmers to successfully create spamming robots. As a way of having 'free' research done, the CAPTCHA is very successful. It interests programmers, who would otherwise not care, for AI research areas.

Chapter 4

The NESSIE project

Written by *Dirk Smit*

An initiative by the European Commission resulted in a portfolio of strong cryptographic primitives, which ought to be implemented by industry all over the world. In this paper we will see the reasoning behind this initiative (Section 4.1), the way the project board dealt with the submissions (Section 4.3) and what the results effectively are (Section 4.4).

4.1 Introduction

The European Commission set out various projects to make Europe the world's most competitive and dynamic economy. Part of this vision is the aim to have easily accessible Information Technologies which can be used throughout the European business and society. Therefore, the Information Society Technologies (IST) [COR00] was funded to structure these activities in four-year programmes. In the latest program, the European Commission is concerned with the quality of life for European Citizens in a global information society. One of the projects in this latest program is called NESSIE: New European Schemes for Signatures, Integrity and Encryption.

In this paper, I will guide you through a couple of topics within the NESSIE project. First I will introduce the NESSIE project. Questions like what is the purpose of this project, who is participating and what will be the result of the project will be answered. After this introduction, I will highlight the most interesting part (I think) of this project: the performance and security evaluation. The last section is about the results of the project at this time, January 2004. This paper will not review the amount of cryptographic primitives, which was sent in during the project. This is not interesting, because it will look like an enumeration of primitives, which can be found on the site of the project (www.nessieproject.org) [Nes00] and if I describe one recommended primitive, I should describe them all. Furthermore, this paper will not review the precise reasoning behind the choice to recommend some primitives for further research, which had to be done by the participants of the project.

Casus 4.1: NESSIE Project Timeline

January 2000: Start of the project. Creation of an industry board.

March 2000: Call for cryptographic primitives.

September 2000: Submission deadline.

June 2001: Preliminary assessment of submissions.

July 2001: Begin of second phase.

February 2002: Preliminary selection of submissions.

May 2002: Standardisation plan.

December 2002: End of the project.

March 2003: Final selection of submissions. Final report of NESSIE project.

4.2 The first phase of the NESSIE project

4.2.1 Main goals

The main goal of the project is described as "to put forward a portfolio of strong cryptographic primitives that has been obtained after an open call and been evaluated using a transparent and open process". This description has several things of interests in it. First the project wants to find a portfolio of strong cryptographic primitives, which means that not only algorithms including block ciphers were asked, but also algorithms including stream ciphers, hash functions, MAC Algorithms, digital signature schemes and public key-encryption together with identification schemes. This is a much broader scheme than the registration of the AES-algorithm. In a sense, you can say that the NESSIE project is a continuation of the AES-project, but the algorithm, which was put forward as the AES-algorithm *Rijndael*, is also included in the NESSIE project. Another difference with AES is the purpose of the algorithm. AES was intended to produce algorithms for government standards. The results of NESSIE will not be adopted by any government or by the European Commission. The intention is that certain industry boards will adopt (one of) the results, so results will be wide spread. To facilitate this, the project board is set up around 20 leading European companies in the area of encryption. A critical evaluation of this results will follow in the last section of this paper.

Second, the project wants to obtain primitives after an *open call*. The project call was in February 2000, where the cryptographic society was asked to sent in strong cryptographic standards on the areas described above. The deadline was in September 2000, and after that, the project board organised a couple of workshops for the submitters. Every submitter was invited to present his/her submission in these workshops. The submissions were put on the website of the project and the evaluation of each algorithm is also available for everyone. The timeline of the project is depicted in Casus 4.1. From this casus, it is clear that the project ended up in December 2002, but the latest report of the project board is dated March 2003.

The third thing of interest in the main goal, stated by the project board, is an *open and transparent process* to evaluate the submissions. Part of this statement is the call for

evaluation methodologies for these methodologies together with the desired algorithms. The submissions for this call were also announced on the website, just like the evaluation of the evaluation methodologies. As stated above, the evaluation of the algorithms was published to reach an open process. The algorithms are compared to existing primitives. The proces is transparent meaning that from the original timeline it is obvious when the project board wants to receive certain results, and when they offer results. It is also transparent which person did the evaluation of a certain primitive.

4.2.2 Selection Criteria

Not every submission was accepted by the project. The project board stated that every proposal should respect the criteria of:

- long-term security
- market requirements
- efficiency
- flexibility

Security is the most important criterion, because security of a cryptographic primitive is essential to achieve confidence and to build consensus around that primitive. A second criterion relates to market requirements. Market requirements are related to the need for a primitive, its usability, and the possibility for world-wide use. The performance of the primitive in a specified environment is meant by efficiency. For software, the range of environments considered include 8-bit processors (as found in inexpensive smart cards), 32-bit processors (e.g., the Pentium family) to the modern 64-bit processors. A fourth criterion is the flexibility of the primitive. It is clearly desirable for a primitive to be suitable for use in a wide range of environments. The evaluation section of this paper contains more criteria, such as security requirements and implementation criteria.

4.2.3 Respons to the Call

With respect to the various requirements and selection criteria, the project board received thirty nine algorithms and one testing methodology. This is quite a good respons, when bearing in mind that the submitters had only six months to develop and test their algorithm.

Within the 39 algorithms, there were 26 symmetric algorithms. Block ciphers gained most attention, because 17 (out of 26) symmetric algorithms make use of this technique. This is not surprising given the earlier calls for DES and AES, which caused an increasing interest in block ciphering. Six algorithms used stream cipher methodology, two MAC algorithms were received and one hash-function has been sent in. The other 13 asymmetric algorithms consist of 5 asymmetric encryption schemes, 7 digital signature algorithms and one identification scheme.

Exploring the origin of the submissions, 6 are originated Europe, 9 North America, 9 Asia, 3 Australia and 3 South America. It is striking that the origins are so well spread, so this could refer to a great interest in the results of the NESSIE-project. A great part of the submissions were sent in by industry (27), only 7 by academia and 6 by co-operation between industry and academia. In this case, it is interesting to see that industry showed so much effort. This could mean that industry is searching for stronger cryptograhic standards with better performance.

After the security and performance evaluation, which will be described in the next section, the project board selected candidates for the second phase. Not only the results of this evaluation were involved in the decision to send a algorithm to the second phase, but also the main goal of the project "to put up a portfolio") and the claims made by the designer were taken into account. During this selection process, designers of the algorithms were allowed to make changes to their algorithm. The purpose of this was to upgrade the overall performance of the algorithm without losing the general requirements. This is also part of the open process, which was mentioned in the main goal. In total 24 algorithms were selected to the second phase, and all topics were covered by these algorithms. The only hash function that was sent in will be compared to an existing one. Selecting 24 out of 39 submissions is a large fraction and this shows that the quality of the submissions was rather high (or the industry board did not want to choose..).

4.3 Evaluation Methodologies

All submissions should be evaluated in the same way in order to compare the algorithms in a fair way. Because of the various types of primitives that the board wants to receive, it seems to be very difficult to approach every algorithm in the same way. In this section, I will focus on the security and performance evaluation the project board did on every algorithm and I will try to make clear to which extend the methodologies are standardized.

To do this evaluation, the project board made use of tools. There are two classes of tools. The first class includes tools that are for general use, not for a particular algorithm. Logically, the second class of tools deals with specialized tools for a particular submission. The generals tools were available from earlier projects and include for example the frequency test and the dependence test. I will not go further into detail about this tests. At the other hand, the project board used special tests for the evaluation of random bitstrings. Furthermore, the project developed a tool to do differential and linear cryptanalysis on block ciphers. Contrary to the main goal of the project, these tools are not available outside the project, but the results of the evaluation remains public.

4.3.1 Security Evaluation

A submission will be checked twice in the project board, and both parties will summarize their findings in one document. Based on this procedure, a submissions was rejected or selected for the second phase.

In general, every submissions should respect the following security criteria:

- An attack should be at least as difficult as the generic attacks against that type of primitive, for example against a birthday attack.
- Every algorithm will be evaluated against the security claims of the submitter. If the claim is wrong, the submissions will be rejected.
- Primitives will be tested against attacks within the stated environment.

Some issues within the security evaluation are generally applicable. To compare all the submissions, these issues are enumerated below. This enumeration will be followed by a summary of issues for particular primitives.

Casus 4.2: Side-channel attack

A side-channel attack is an attack which make use of information gathered from for example time-analyses or power consumption. It is generally known that for a side-channel attack, you do not have to know a plain- and ciphertext.

1. *Resistance to cryptanalysis*: It is clear that a primitive should be resistant to cryptanalytic attacks. However, other factors such as the volume and type of the data required to facilitate the attack are considered.

2. *Design philosophy and transparency*: Again, transparency of this is of big importance in this project, not only because it is a demand of this project, but also because of the comparisons between primitives. Without transparency of an algorithm, algorithms cannot be compared thorough. The design philosophy is asked to have confidence in the submissions. The argumentation why the creator has designed the algorithm like this contributes to that, just like the mathematical standards he used do.

3. *Strength of modified primitives*: The strength of a primitive is particular assessed by changing components or by reducing the number of rounds. If the inference between the original and the modified primitive is straightforward, conclusions can be made based on the tests on the modified one.

4. *Relative security*: What is the security in comparance with established primitives? This clearly has to do with the design transparancy, meaning that a transparent algorithm can be compared on the same level as a established one. Based on this, we could conclude that relative security is very difficult to do.

5. *Cryptographic environment*: A cryptographic algorithm could be assigned for a particular environment. For example, it is not useful to do a side-channel attack (Casus 4.2) on a smart card. The level of security depends on the type of environment, it is designed for.

Above this, it is useful to do statistical testing in order to find pitfalls in the design of the algorithms.

For every area of cryptographic primitives, the project board assessed some particular security levels. For example, a block cipher receives the security level *high* if the key is of at least 256 bits and the block length is at least 128 bits. A Hash functions has security level *high* if the output length is of at least 256 bits. For every area, special attacks were used to specify the order of security. It is not interesting to describe all the attacks, because I will not use the results in the remainder of this paper.

Assymmetric algorithms are usually based on mathematical problems that are believed to be "unsolvable". However, the project board tested these primitives against current techniques to test the unsolvability. Algorithms that used the same type of assumptions were compared to each other and the report was based on this comparison.

4.3.2 Performance evaluation

A primitive is practically usable if it has a good performance on several platforms. For that reason, the submissions must be tested on various platforms, but for the aime of comparison, only a few platforms were used for evaluation. This surely has to do with the competences of the project board but also with the limited amount of time.

The project board developed a framework to compare the primitives on a fair and equal basis. The first part of this framework includes the dessecting of the algorithm: setup (independent of key and data), precomputations (for example key schedule) and the algorithm itself. Every part of each algorithm was fully examined in order to determine the degree of optimisation. Besides, each candidate is tested on four platforms. Here, it is interesting how the performance is measurable on each platform. Parameters such as RAM, speed, code size etc. can be measured on every implementation, but for some implementations, you need extra parameters. After this, the influence of attacks, of encryption and decryption and of verification is analyzed. Hardware implementations were not taken into account due to limited resources.

We should make some remarks on the way of comparison. First, some algorithms are compared not only to other submissions but to existing ones, for example a block cipher can be compared to Rijndael. Secondly, the goal of the measurements is not to measure a precise value of the speed, but place the speed in a certain range due to imprecisions in the measurements. Above this, the project only implemented the critical part of the algorithm. For the primitives that were selected to the second phase of the project, the whole implementation was done in this phase.

To present the information gathered from this tests, the project build up a performance template. The template should provide a clear view on the collected information and on the basic operations of each algorithm as shifting, permutations, inversion and many more. In this template, it is possible to emphasize the most important test results, because not every platform is widely used in industry. At the end, the results of this evaluation were presented in a report, which is available on the website of the Nessie-project. The project board tried to select the algorithms which were:

- at least acceptable on each platform
- performing above the average with significant difference
- flexible

4.4 Results

The project ended in march 2003, at least the last document was published by that date. It is thus interesting to see what results the project delivered and if they have achieved their goals. Let us first explore the achievement of the main goals.

One of the goals was to create a portfolio of strong cryptographic primitives. The number of submissions shows that the creation of a portfolio has succeeded. On all areas, there were sufficient submissions and after the evaluation of the second phase, the project chose an algorithm to recommend in the portfolio in one of the various areas with respect to various security levels. The project board stated that the primitives clearly benefit to the cryptographic research community. The call was a open call, which is proven by of the origin of the submitters and by the various types of industries. The evaluation process was even open. The security and performance evaluation reports are available for everyone, and the project board organised several workshops for the submitters. It should be a disappointment that only one testing methodology was sent in, because such methodologies could support better understanding in security and performance measurements. But overall, the way the project was organised was a great success.

Next, the intention was to spread out the algorithms widely by inviting industry into the project board. This is a much more complicated goal of the project, because it is not easily measurable if the industry has adopted the primitives of the project. Industry is always very reserved about information about their internal methodologies mainly because of competition reasons. After some research on the internet and in magazines, I could not find one firm or organisation that adopted the ideas of NESSIE. In that sense, I could not conclude that the project itself is a great success. Maybe it is too early to draw such a conclusion, because in industry improvement processes take a lot of time, but it is astonishing to see that there is so little published about the NESSIE project besides the reports on the website of NESSIE.

Chapter 5

Best known attack on AES

Written by *Leonard Tersteeg*

AES is the current standard for symmetric encryption. In this chapter the structure of the algorithm is explained. There are no current known short-cut attacks. It is an algorithm with a surprisingly elegant algabraic structure and some cryptanalysts think that this can't be a sign of a good algorithm. There are some theories about how to make use of the structure of AES. However nobody has seen an opportunity to make a practical use of this structure. There exist some possibilities to implement a physical attack, but the assumptions that have to be made, to make such an attack practically feasible are too strong. That's too bad, because the only reason these physical aspects are interesting is because of their practical value. Some specialist do think that although no weaknesses have been found till now, the current standard won't be the standard for the estimated 20 years of the NIST. Especially the authors of the AES standard strongly disagree on that (they should disagree of course).

5.1 Introduction

In our digitised society cryptography is nowadays used on a much wider scale then 30 years ago. We can divide it in two mainstreams: symmetric and asymmetric encryption. Of these streams the symmetric system is by far the oldest. In 1974 DES was developed, became the (American) standard for encryption and has had this title for more than 20 years.

With this standard there was supposed to be clearity about the question which algorithm to use for data protection. After time passed by various algorithms were in use for symmetric encryption, so there had to be a renewal of the standard.

In 1997, NIST (National Institute of Standards and Technology) in the USA set up a program to find a new algorithm for data encryption. The need for the new standard was only more urgent when somebody developed a system that could actually break DES by bruteforce. Eventually out of 15 submissions and after 2 year of analysis by cryptanalyists all over the globe, they choose the algorithm Rijndael [DR99], developed by 2 Belgians: Joan Daemen and Vincent Rijmen, to be the new encryption standard. Is it possible that this successor of DES would hold the title for as long as DES did?

In order to answer the question I will first discuss the working of the AES algorithm Rijndael (Section 5.2). Then there will be an analysis on the various possible cryptographical attacks

that are possible on Rijndael (Section 5.3). After this analysis there comes a consideration of all attacks (Sections 5.4 and 5.5) and it is tried to determine if there's a best attack (Section 5.6).

5.2 Working of Rijndael

In order to compete in the contest of the NIST, the submission had to fullfil certain requirements. Rijndael did this and had some other extra relevant properties :

1. Variable keysizes are allowed; in case there would be proven that the standard was breakable, they could easily raise the keysize in order to increase security.
2. Variable blocksizes can be chosen starting at 128 bits.
3. Variable number of keyiterationrounds are allowed. However in normal circumstances the algorithm would use the following amount of rounds:
   - AES with 128 bits key - 10 rounds
   - AES with 192 bits key - 12 rounds
   - AES with 256 bits key - 14 rounds
4. Not a real property, more an assumption, but the authors of the algorithms claim that it is resistant to any known attack on the date of publicity.

The AES algorithms only uses blocksize 128 bits as input and gives 128 bits as output. The plaintext is cutted into slices of 128 bits and they are mapped to a 4x4 bytes matrix. This is called the state . This state is changing through all the sequential subfunctions of a round. From now one when I'm talking about "the algorithm" I mean the AES-variant and not Rijndael, so I will only consider the variant with a fixed blocksize.

As you can see in fig. 5.1 there are multiple rounds to be run. And in each run all bytes (and thus all bits) have the opportunity to change, this is a big difference with a Feistel network as DES is, where each round only half the bits change.

I'll focus with my explanation on the repeated rounds, cause they are the algorithmic centre. If you're interested in a precise working of the whole algorithm I'd like to refer to the definition of the algorithm [DR99].

5.2.1 Byte substitution

When considering a round the first function applied is ByteSub (in an update the authors have called this SubBytes in order to increase readibility, however because the old specification is still available, the names are often mixed), this is a fixed byte substitution for every byte (see fig. 5.2, every inputbyte has an own S-box value). The S-box looks a bit like the S-box of DES, because it is possible to give a table that for every possible bit ($2^8 = 256$ possibilities) has a table entry with the corresponding substitution value. However It is possible to decompose the working of the S-box into 2 transformation functions, just like the authors of the algorithm did. This substitution is constructed by 2 different transformations:

1. Compute inverse:
   Each byte can be described as a polynomial representation:

Figure 5.1: Working of Rijndael

Byte $(b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0$ With $b_i \epsilon \{0, 1\}$ Example: the binary 01010101 would yield the following polynomial $b(x) = x^6 + x^4 + x^2 + 1$ Given the binary as a polynomial function (to compute the actual binaryvalue use $x = 2$) compute the inverse $b^{-1}(x)$ of this function. Given modulus $m(x)$ we must compute polynomials $a(x)$ and $c(x)$ with the extended Euclides algorithm such that
$b(x)a(x) + m(x)c(x) = 1$

Figure 5.2: BYTESUBSTITUTION

$(b(x)a(x)) \bmod m(x) + (m(x)c(x)) \bmod m(x) = 1 \bmod m(x)$
$b(x)a(x) \bmod m(x) = 1$
$b^{-1}(x) = a(x) \bmod m(x)$
In this way we can compute the inverse of a binary.

2. Apply an affine transformation:
   Given $b^{-1}(x)$ as $x_0, x_1, ..., x_7$ The following transformation is applied:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

   So the result of the original $b(x)$ will be $(y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7) \bmod 2$, the modulus must be applied to every individual y_i so the notation might not be totally correct, but this step is necessary for making it the transformed binary value.

After applying these transformation functions to all bytes in all states, phase 1 of the iteration round is complete. Note that the phase 1 function is non-linear, where the other functions are linear (having the following layout $f(x) = ax$ and non-linear (affine) functions look like this $f(x) = ax + b$). This is important becaus if they were all linear rounds, then after applying all those rounds again then for some r $r \times a \bmod m(x) = 0$ and that would mean that the original

5.2.2 Row rotation

Then the ShiftRow function is applied, fig. 5.3 gives a clear picture of what's happening. Every row of a 4×4 State is shifted to the left over some steps. The first row is not shifted at all. The second row is shifted 1 step to the left making the originally first byte of the row now the last one. The third row applies 2 shifts of 1 step making the original first byte the third byte after shifting and the original second byte the last byte after shifting. The fourth row shifts 3 places making the last byte before shifting the first byte after shifting.

Figure 5.3: Row rotation

5.2.3 Column multiplication

MixColumn takes a column of a state and multiplies that with a function $c(x)$ mod x^4+1 (see fig. 5.4). This function is defined as: $c(x) = 03x^3 + 01x^2 + 01x + 02$. The difference between this formula and the polynomial definition of a binary in section 5.2.1 is that the function coefficients are now not binary, but hexadecimal. Let's take a column of a state $a(x)$ with bytes a_0, a_1, a_2, a_3, the multiplication would be this:

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

Now every original a_0 is replaced by b_0 which is a result of the multiplication: $01a_0 + 03a_1 + 01a_2 + 01a_3$ and so on for all other multiplications as described in the matrix above.

Figure 5.4: Mix Columns

5.2.4 Adding key

At last the key is added, although key generation has possibly taken place before. I won't explain the exact key expansion algorithm, because it is not very relevant considering the possible attacks that are performed. When interested in details I suggest reading the AES proposal [DR99]. The related key attack could be seen as a possible attack, but because of the diffussion and non-lineairity of the algorithm, such an attack is not likely to succeed. Even small changes in key will deliver a complete other cipher. Here is the concept:

1. The number of bits of a round key is equal to the block length, 128 bits, times the number of rounds plus 1, $10 + 1 = 11$, makes a total number of key bits of 1408 bits.
2. The cipher key is expanded into an expanded key.
3. Round keys are taken from the expanded key by taking the next 4 4-byte words ($4 \times 4 \times 8 = 128$ bits). So round 1 will get bits 1–128, round 2 129–256 etc.

Note that adding the key removes similarity between different rounds, because the other transformation functions treat all states the same. Also by having a round constant dependent on the round in the key generation, the similarity between key adding to the outcome of the transformation functions is removed.

5.3 Known attacks

5.3.1 Differential and linear attack

The differential attack looks at differences between rounds of encryption. Oscar offers more than 1 slightly different plaintexts to the encrypter Bob, and looks at the diffence of the encrypted message by XOR'ing the results. A linear attack tries to find correlations between input and output over most rounds (not necessarily all rounds). It is proven in the proposal [DR99] that a differential attack has a probability of at most 2^{-150} for any 4-round differential trail and a correlation of at most 2^{-75} for any 4-round linear trail. Which are both too low to be considered as a serious attacks.

These attacks themselves might not be so effective, but their ideas are a source for new variants of the theme that might be succesfull in breaking AES[ODR02]. One of these variants is considered as the best attack nowadays on AES: the so-called square attack.

5.3.2 Square attack

The idea of the square attack is the following: Use 2 plaintexts (a, b, c, d) and (a', b, c, d) that differ only at one byte a and a'. After 1 round they differ 4 bytes, because of linearity of MixColumn()[BK00]:

$$(02_x \bullet (a - a'), 01_x \bullet (a - a'), 01_x \bullet (a - a'), 03_x \bullet (a - a'))$$

After round 2 they difer at all bytes, because of shiftrow, the columnbytes that were once different have been transposed to all 4 columns. When MixColumn() is applied afterwards to the output of shiftRow() all bytes will differ from each other. So the plaintext differ on all bytes on all possible values. However the differences are balanced, because both can have all values. When a XOR is applied to these collection bytes a 0 will result. For four bytes that

will deliver $0 \oplus 0 \oplus 0 \oplus 0 = 0$ with certainty $\frac{255}{256}$. By using an additional plaintext for every time the XOR doesn't deliver a 0, finally the 4^{th} roundkey can be derived and with that the cipherkey. Normally a ByteSub would destroy this feature, but assuming that this is the last round that will not happen.

This idea can be extended to a 5^{th} and 6^{th} round, with more balancing and more checking. You can find more info in [BK00]. The attack is able to break AES, but only till 6 rounds and AES starts with 10 rounds. This relaxation makes it not feasible to do a practical attack yet.

5.4 Theoretical problems

The square attack can only break a 6 round version of AES. The following theories have not yet led to an actual attack, but it might happen in the recent future. They try to take advantage of the algabraic structure of AES. Courtois and Pieprzyk [CP02] show that certain cipher-block encryption algorithms can be described as a system of multivariate boolean quadratic functions. They also claim that because it is overdefined, which says that there are more equations than variables, it is solvable within a time that's faster than the exhaustive keysearch and with it should be able to break Rijndael with a 256 bit key. A multivariate system of quadratic equations is a NP-hard problem, so coming up with a subexponential algorithm would prove $P = NP$ and that's not very likely. The authors have seen this and claim that because it is overdefined it will decrease its hardness till below NP-hard. Critic reviewers say that Courtois and Pieprzyk claim too many independent equations and thus is the problem a lot harder than Courtois et al. think.

Where Courtois et al. have defined the AES encryption as a set of boolean multivariate equations. Ferguson, Schroeppel and Whiting [FWS01], try a different approach and find one formula with a fraction that contains a fraction for every round. They came to this equation by first taken the formula for a S-Box, see [DR99]:

$$S(x) = w_8 + \sum_{d=0}^{7} w_d x^{255-2^d}$$

Starting from this formula all functions are applied in a mathematical functional way, this results is in the following description after 5 rounds:

$$x = K + \sum \frac{C_1}{K^* + \sum \frac{C_2}{K^* + \sum \frac{C_3}{K^* + \sum \frac{C_4}{K^* + \sum \frac{C_5}{K^* + p^*}}}}}$$

Every K^* is a part of the expanded key for a known *, furthermore p^* contain the plain-textbytes. If you would write this out completely you would have an equation with 2^{25} ($32 (= 2^5)$ copies and 5 summations means $(2^5)^5 = 2^{25}$) terms. By combining two of these equations: one covering round 1 till 5, taking plaintext as input and delivering ciphertext as output and the second one would cover round 6 till 10 with the same variables, with this difference that it takes ciphertext as input. These two combined would yield 2^{26} unknowns. It would need 2^{22} pairs of plaintext/ciphertext to solve this problem. There is no algorithm that solves these kind of problems, but if it were to be made in the future it might beat the exhaustive search. AES relies on the fact that this can't be done, but that hasn't been proven, neither is the opposite that it is possible.

5.5 Fault-based Attacks

As is shown by Blömer and Seifert [BS03], AES is vulnerable to fault-based attacks. It means that if it is possible to change bits in the key generation process at certain times, which they prove they can, that the key can be found after 256 encryptions with a fault. Their idea is not a mathematical approach, but a physical approach. They try with light, electricity and magnetism to alter bits at specific times.

Fortunately for the designers of Rijndael such attacks are usually blocked by hardware implementations or by software that checks on consistency. The ways of altering data in hardware become more sophisticated and that's why there is a need for an extra software check on correctness. These software checks are not included in Rijndael and are therefore now a responsibility for the implementers. They don't always pay attention to this, because they trust on AES.

It looks like it is very easy to break AES in this way, but there are some assumptions to the break after 256 encryptions, such as choosing to alter round keys to a specific value. Most of these assumptions are not very likely to happen in practice. So it is in no way proven that AES can always be broken in a physical way, it just depends on the implementation.

5.6 Conclusion

AES is a nicely composed algorithm, which is, in my opninion, a very understandable but strong algorithm. Nowadays no known shortcut attacks exist for AES. There are some possibilities to write down AES in an elegant algabraic structure, but if this guarantees feasibility is very hard to say. Most researchers say that more research is necessary, I agree on that, but I still doubt that there will be a solution to the mathematical problem of breaking AES. There might exist some shortcut attack, but I don't think, if it exists, that it would be much faster than an exhaustive search. The current best attack is the square attack, that's able to break relaxed forms of AES faster than exhaustive search. It is still no guarantee that a shortcut attack will be found for AES in the upcoming 15-20 years.

When doing research for this paper I was most likely suprised by the physical approach. I liked the idea that they don't look at the mathematical background of AES, which is mainly covered by the authors, but to look on implementation possibilities. The more surprised was I with the announcement that is possible to break AES physically, but after reading on I saw that breaking in practice was not feasible and that was a relief. Because a lot of things in our society are guarded with AES (e.g. smartcards) and the idea that other people with mean intentions are able to make use of the weaknesses of AES I didn't like at all.

The only thing society can do, is stimulating cryptographic research in order to answer the question: Is there no better way than exhaustive search to break AES? Will this question ever been answered? It could be that computers are fast enough in the future to perform such an exhaustive search. I can't say what will happen, the only thing we can do is pray (and do research) and hope that our bank accounts and other privacy-sensitive information is save and the future will tell us.

Chapter 6

Design of Block Ciphers

Written by *Gommaar van Strien*

Block ciphers are an essential part of many cryptographic systems. In this chapter I'll consider some important design issues that should be taken into account in order to create optimal security using (symmetric-key) block ciphers. After that I shortly discuss some practical implementations of block ciphers in cryptography. Furthermore I'll review some attacks on which one should be alert.

When we consider modern cryptographic systems their essential parts consist of block ciphers. One single block cipher can provide some security on a block of data. But by combining many of them it's possible to create hash functions, stream ciphers or even pseudo random generators and finally of course encryption/decryption systems.

Not all block ciphers are equally useful for all applications, even if they provide a high level of security. Computational costs, memory usage and speed issues can limit the practical use of different ciphers. So in general some tradeoff between the provided security level and the efficiency has to be found.

In this chapter I will focus on block ciphers for use with cryptographic systems which use a symmetric key, but some references may help you in finding information on other types of use.

6.1 What is a Block Cipher?

I just said that block ciphers form an important part of cryptographic systems, but now some formal definition would be appropiate. A block cipher is actually a function. An input message is considered as one or more sequences of bits and this message is mapped to some other message using a key. This mapping forms the encryption, and likewise the decryption is formed.

First I'll introduce some notation used. Say we have a vectorspace V which consist of all possible bitstrings. Now we can have a string in plaintext or ciphertext, called a of length n when we have $a \in V_n$. Furthermore we have the keyspace $K \subseteq V_i$ for keys of length i. The keys are used as a parameter of the encryption function E and decryption function D. So, for encrypting string a using key k we have $E_k(a)$ and for decryption of a we of course have $D_k(a)$.

Finally for a message in plaintext we often use P and C to denote a message in ciphertext, with of course $\forall P, C \in V$.

Definition 6.1 *A block cipher consisting of n bits is a function $E : V_n \times K \rightarrow V_n$ such that for all $k \in K$, E_k is an invertable mapping from plaintext to ciphertext, and D_k is its inverse. Thus, for all $k \in K$ and $P \in V$, $D_k(E_k(P)) = P$.*

As said before all kinds of block ciphers can be used to encrypt a plaintext. For normal block ciphers the blocksize is at least 64 bits. For block ciphers used as streamciphers on the other hand the size is probably lower. I will discuss this further on in some more detail. When we want to encrypt texts of longer size we just split the whole plaintext up into parts of exactly the blocksize. To combinate all blocks different methods can be used which shall also be introduced later on.

Now the most general and also the best block cipher is a rather theoretical one. Every bijection between plaintext and ciphertext is implemented. This results in $2^n!$ possible keys for the 2^n possible elements. With this mapping every plaintext can be mapped to every possible ciphertext with equal probability. This perfect cipher is invented by Shannon.

For any block of normal length this method is of course useless because of the storage/computation costs of keys...

Now the main goal is to make a cipher look like completely random without knowledge of a key using some fixed encryption strategy with this key. In order to make the encryption look like random most strategies are based on a few principles introduced by Shannon.

6.2 Shannon's Principles

Claude Elwood Shannon (1916–2001) is considered as one of the founders of information theory. He combined mathematical theories with electrical engineering principles and so helped in the development of the digital computer already in the 1940s. He also introduced the Boolean algebra into computer science. In 1945 he also formulated some principles which form the foundation for the creation of block ciphers. I shall introduce the most important ones one by one.

6.2.1 The Avalanche Effect and Completeness effect

An important feature of a cipher is that it's hard to retrieve either the key or the plaintext from a generated ciphertext. So if we have quite similar plaintexts from which we create ciphertexts with the same key, it should be very difficult to extract information about this key from the ciphertexts. If this is not the case differential attacks on ciphertext become relatively easy. Vice versa, two similar keys used for encryption on the same plaintext should produce two very different ciphertexts. Lack of these difference reduces the keyspace and thereby makes life easier for a cryptanalyst. This kind of properties can be summarized by the so called avalanche effect.

Definition 6.2 *The avalanche effect says: a minor change in the plaintext or in the used key should result in a significant and random-looking change in the ciphertext.*

To create a measure for this significance we can define the strict avalanche criterion.

Definition 6.3 *A block cipher satisfies the strict avalanche criterion if and only if every 1 bit change in the plaintext leads a probability of 0.5 for every output bit to change.*

When we look at DES for example, a 1 bit change in either the key or the plaintext produces on average a change of 35 bits in the output. With a block size of 64 we see that the DES algorithm satisfies the strict avalanche criterion.

Somewhat related to the avalanche effect is the completeness effect. It states that every bit in the ciphertext is a complex function of all input bits in the block. As a result of course every change in the plaintext block result in changes in the ciphertext. So a change in ciphertext is uniformly distributed for each changed bit of the plaintext given as input.

6.2.2 Confusion

As we have seen from the avalanche and completeness effect, we want a mixture between key and plaintext such that the ciphertext looks like completely random. So we want a high entropy in the ciphertext. As a result we want to hide the statistical structure from the plaintext in order to prevent an cryptanalyst to gain advantage from it.

Confusion is a first step in doing this. It creates some state of chaos by substitution of bits and it's meant to make the statistical relationship between key and ciphertext as complex as possible.

A famous example of a simple cipher using confusion is the well known Caesar cipher. This cipher works directly on the letters in the plaintext and not on bits. The encryption is done by shifting every letter of the plaintext a few positions in the alphabet. Besides the fact that a brute force analysis is quite easy against this mechanism, as there only 26 keys possible a frequency analysis can still be done on the ciphertext. This is because every letter is shifted the same number of positions.

To make better use of substitution special substitution tables are used in modern block ciphers for doing this. These tables specify for each position with what other position a switch has to be made. With DES the substitution tables are the 'S-boxes'. Of course the substitutions are done on a combination of (parts of) the key and (parts of) the plaintext. A way to do this is by XOR-ing keybits with plaintext bits before or as part of the substitution. By using the key in this way, frequency analysis becomes already harder as a same letter in plaintext is possibly substituted by different letters on each occurence in ciphertext.

6.2.3 Diffusion

By only using confusion we still have some quite fixed relationships between pairs of bits, of course dependent on the number of substitutions made. To spread influence of each inputbit (either former plaintext or keybit) throughout the whole ciphertext we use transpositions/permutations.

By doing this complete blocks are put into new positions. By iterating many rounds this spreading effect can be enlarged heavily. On this I'll return shortly after. Just as with simple substitutions modern systems use P-boxes which are lookup tables defining all permutations that are to be made. In the long run this has the effect that every plaintext digit influences different parts of the ciphertext. So with diffusion statistical properties of the plaintext become hidden in the ciphertext. Some more detailed view on S-boxes and P-boxes can be found at [XH02].

6.2.4 The Product Principle

As we just have seen with diffusion, it is important that each input bit influences many output bits. Before we saw that confusion was required to create some 'chaos' in our ciphertext. The product principles now says to use both to get a better encryption!

Definition 6.4 *The product principle says that different methods of encryption have to be cascaded in a systematical way to improve the provided security by the system as a whole.*

Shannon was the first to use cascaded ciphers to improve the cipher strength. Consider ciphers A_1 and A_2 both of different classes. Now we can cascade both by multiplying them. This process can actually be seen as some sort of matrix multiplication. For matrices, multiplication is associative, and in most cases not commutative. This taken into account, it's easy to understand that cascading the same encryption method is of no use, as $A \times A = A^2$. Thus, the result is an idempotent class of functions. An example of this is e.g. the permutation cipher. A sequence of transpositions can be replaced by a single one doing the different moves at once. For substitutions holds the same. Two substitutions after another form a new single substitution. But by layering them after another together with good mixing with parts of the key or even data from other blocks of data there is an increase in security. In DES there are 16 iterations to create confusion each round by using substitutions of databits created by XORing keybits and databits together with diffusion in which the influence of a single inputbit is spread out through the whole block. Note that the S-boxes and P-boxes are known, but due to the use of parts of the key in each round this won't help much in breaking the encryption.

6.3 Encryption using Block Ciphers

Having seen the standard principles used for creating a block cipher it's possible to build a good encryption algorithm. As said in the last section iterating over many different cycles and by repeating different ciphers after another result in a good ciphertext. The remaining problem lies in the fact that the plaintext usually is much larger than the normal 64 or even 128 bits length of a block cipher. To encrypt such texts it's necessary to split them up into parts and encrypt each apart from the others using our favourite block cipher. This method in which each block is encrypted independent of other blocks, is called the electronic CodeBook method, also known as ECB.

This directly gives rise to a new problem. When we take into account Kerckhoffs' principle the attacker knows the method of encryption, and he thus knows our blocksize. Now frequency analysis can be done on ciphertext level, when we use the same key for each block. The cryptanalyst can also guess pairs of ciphertext-plaintext and substitute blocks of ciphertext in an intercepted message with other of which he has guessed the plaintext equivalent. In this way he can sabotage communication.

Apart from al these disadvantages there's one small advantage. In case of dataloss during transfer only the corrupted cipherblocks can not be recovered correctly.

6.3.1 Cipher-block chaining

Having seen the large disadvantage of the Electronic CodeBook method we can be glad that are better ways of using block ciphers on bigger amounts of data. One of such methods is called Cipher-block chaining(CBC), which I will describe now.

INPUT: For a message of length n we use an Initialization string I, the ciphertext blocks $c_1, \ldots, c_n$ and the plaintext blocks $x_1, \ldots, x_n$.
Encryption:

- $c_0 = I$
- $\forall i, 1 \leq i \leq n$: $c_i \leftarrow E_k(c_{i-1} \otimes x_i)$

Decryption:

- $c_0 = I$
- $\forall i, 1 \leq i \leq n$: $x_i \leftarrow c_{i-1} \otimes D_k(c_i)$

As we can see the ciphertext of each block depends on that of all blocks before. Disadavantage of this is of course that in case of dataloss every block after the first corrupted block can no longer be decrypted correctly. On the other hand, we can exactly see where the datacorruption has occured, which is of course useful in re-transmitting the message. Further, note that the initialization string not needs to be secret, but the correct version is of course required to make decryption possible. A hacker can sabotage decryption by modifying this string, allthough this is signalled immediately by the receiver.

Apart from this little problem we have solved other problems that the ECB method had. The only case where we get the same ciphertext is when we encrypt the same *total* plaintext using both the same key and initialization string. Then the whole ciphertext is the same. Possible doubly occuring blocks of ciphertext don't have the same equivalent in plaintext. So statistical analysis becomes almost impossible using Cipher-Block Chaining.

There are also some variants on CBC which work almost the same, but in which not the whole plaintext is encrypted at once but groups of blocks at a time. In such cases possibly different initialization strings are used, or only parts of earlier sent blocks are used in the next ones. In all cases there's a step in which different blocks influence each other during encryption. This can be seen as some sort of extra diffusion at cipher-block-level.

6.4 Attacks on Block Ciphers

When attacking block ciphers there are some methods of which a few are well-known

- Brute force
- known-plaintext attack
- ciphertext-only attack

The easiest way to break any encryption is by trying every possible key. So the chance of succes depends only on the keysize of the used cipher. In a normal case this task should be undoable. So some more intelligent attacks have been invented. All are based on analysis of pieces of ciphertext, possibly paired with a correct piece of plaintext. We can roughly split cryptanalysis into two types: differential cryptanalysis and linear cryptanalysis.

6.4.1 Linear Cryptanalysis

With cryptanalysis one tries to find a method to create a way to decrypt a ciphertext without knowing the key. Decryption then looks like some unknown function for the analyst: $D : C_n \rightarrow P_n$.

The main idea of linear cryptanalysis now is to find a simpler function approximation of the parameterized block cipher as a whole. This can be done by trying to find a function mapping out of many pairs of plaintext/ciphertext using a simpler cipher then the one which was used to create it. This idea can be summarized by the following slogan:

> Many one-dimensional slices lead to a two-dimensional total view.

For single block ciphers this method can be used, but for highly and well constructed layered block ciphers like DES or AES, linear cryptanalysis is often too difficult and the search space almost becomes as big the keyspace. Furthermore *many* pairs of ciphertext/plaintext are required. But when these demands are fulfilled linear cryptanalysis is more powerful than differential cryptanalysis on which I come now. A more detailed description of how doing a linear cryptanalysis see Heys [Hey].

6.4.2 Differential Cryptanalysis

A slightly different method that is often used against many block ciphers is the differential cryptanalysis. Goal is to find changes or repetitions between different ciphertexts. Cipher-block chaining makes this process difficult, as then only the first block of data can be used easily for checking on this. Another idea behind this is to look for some isolated points in which the block cipher behaves like a simple function.

This can occur when diffusion is not well enough; as a result some outputbits are influenced only by a few input bits. For this method it may be clear that some pairs of plaintext together with ciphertext are required.

A large differential cryptanalysis was first described by Biham and Shamir, see e.g. [BS91]. They tried the method on DES and their goal was to retain keybits used in the last encryption round. The number of iterations needed, depended on the number of known or chosen plaintexts. With chosen plaintexts the process is a lot simpler! As the number of rounds the algorithm uses, the searchspace increases at least exponential in most cases when using this kind of attack.

6.5 Conclusion

Block ciphers are widely used for symmetric encryption. Some well known block ciphers include DES, 3DES, RC5, SAFER and IDEA. They are used on fixed amounts of data, but can be used for streaming data. Then the design is slightly modified and the actual work is in that case not done on all data but on some long random-looking string which is mixed up with the actual data.

Implementation on hardware or software is relatively easy as only elementary bit transformations are done by a block cipher. Strength is gathered by cascading different methods, like XOR-ing the key with parts of the data, substitute bits to create the confusion effect and finally by using permutation on the data for diffusion. To increase the effect of all this, these methods are repeated in many rounds to get an optimal mixture between key and plaintext.

The success and safety depends of course on keysize and block size, but in most cases the number of encryption rounds used also plays a big part in finding suitable attacks on the cipher. As a result of this, implementation of block ciphers is relatively easy but the desgin is very difficult as the function to be built is highly complex. This is because the output is very dependendent on all inputbits of both the key and plaintext. Each cycle of the algorithm should have enough possible effect on the ciphertext to make the computational cost affordable. A more detailed explanation on the design of block ciphers was presented by Robshaw [Rob94].

Chapter 7

Steganography and steganalysis

Written by *Robert Krenn*

7.1 What is steganography?

Steganography, coming from the Greek words *stegos*, meaning *roof* or *covered* and *graphia* which means *writing*, is the art and science of hiding the fact that communication is taking place. Using steganography, you can embed a secret message inside a piece of unsuspicious information and send it without anyone knowing of the existence of the secret message.

Steganography and cryptography are closely related. Cryptography scrambles messages so they cannot be understood. Steganography on the other hand, will hide the message so there is no knowledge of the existence of the message in the first place. In some situations, sending an encrypted message will arouse suspicion while an "invisible" message wil not do so. Both sciences can be combined to produce better protection of the message. In this case, when the steganography fails and the message can be detected, it is still of no use as it is encrypted using cryptography techniques.

Therefore, the principle defined once by Kerckhoffs for cryptography, also stands for steganography: the quality of a cryptographic system should only depend on a small part of information, namely the secret key. The same is valid for good steganographic systems: knowledge of the system that is used, should not give any information about the existence of hidden messages. Finding a message should only be possible with knowledge of the key that is required to uncover it.

7.1.1 New technology?

Steganographic techniques have been used for centuries. The first known application dates back to the ancient Greek times, when messengers tattoed messages on their shaved heads and then let their hair grow so the message remained unseen. A different method from that time used wax tables as a cover source. Text was written on the underlying wood and the message was covered with a new wax layer. The tablets appeared to be blank so they passed inspection without question.

In the 20th century, invisible inks were a widely used technique. In the second world war, people used milk, vinegar, fruit juices and urine to write secret messages. When heated, these fluids become darker and the message could be read.

Even later, the Germans developed a technique called the *microdot*. Microdots are photographs with the size of a printed period but have the clarity of a standard typewritten page. The microdots where then printed in a letter or on an enveloppe and being so small, they could be sent unnoticed.

Recently, the United States government claimed that Osama Bin Laden and the al-Qaeda organization use steganography to send messages through websites and newsgroups. However, until now, no substantial evidence supporting this claim has been found, so either al-Qaeda has used or created real good steganographic algorithms, or the claim is probably false.

Steganographic techniques have been used with success for centuries already. However, since secret information usually has a value to the ones who are not allowed to know it, there will be people or organisations who will try to decode encrypted information or find information that is hidden from them. Governments want to know what civilians or other governments are doing, companies want to be sure that trade secrets will not be sold to competitors and most persons are naturally curious. Many different motives exist to detect the use of steganography, so techniques to do so continue to be developed while the hiding algorithms become more advanced.

7.1.2 Uses of steganography

With steganography you can send messages without anyone having knowledge of the existence of the communication. There are many countries where it is not possible to speak as freely as it is in some more democratic countries. Steganography can be a solution which makes it possible to send news and information without being censored and without the fear of the messages being intercepted and traced back to you.

While sending messages can be useful, it is also possible to simply use steganography to store information on a location. For example, several information sources like your private banking information, some military secrets and your mothers special pancake recipe, can be stored in a cover source. When you are required to unhide the secret information in your cover source, you can easily reveal your banking data and the recipe and it will be impossible to prove the existence of the military secrets inside. Steganography can offer *denieable* storage of information. The *Rubberhose* project (http://www.rubberhose.org) offers an implementation of this principle.

Because you can hide information without the cover source changing noticebly, steganography can also be used to implement watermarking. Although the concept of watermarking is not neccessarily steganography, there are several steganographic techniques that are being used to store watermarks in data. The main difference is on intent; while the purpose of steganography is hiding information, watermarking is merely extending the cover source with extra information. Since people will not accept noticeable changes in images, audio or video files because of a watermark, steganographic methods can be used to hide this.

7.2 Implementing steganography

Secrets can be hidden inside all sorts of cover information: text, images, audio, video and more. Most steganographic utilities nowadays, hide information inside images, as this is relatively easy to implement. However, there are tools available to store secrets inside almost any type of cover source. It is also possible to hide information inside texts, sounds and video films for example. The most important property of a cover source is the amount of data that can

be stored inside it, without changing the noticeable properties of the cover. When an image is distorted or a piece of music sounds different than the original, the cover source will be suspicious and may be checked more thoroughly.

7.2.1 Hiding a message inside a text

Since everyone can read, encoding text in neutral sentences is doubtfully effective. But taking the first letter of each word of the previous sentence, you will see that it is possible and not very difficult. Hiding information in plain text can be done in many different ways. The first-letter algorithm used here is not very secure, as knowledge of the system that is used, automatically gives you the secret. This is a disadvantage that many techniques of hiding secrets inside plain text have in common.

Many techniques involve the modification of the layout of a text, rules like using every n-th character or the altering of the amount of whitespace after lines or between words. The last technique was successfully used in practice and even after a text has been printed and copied on paper for ten times, the secret message could still be retrieved.

Another possible way of storing a secret inside a text is using a publicly available cover source, a book or a newspaper, and using a code which consists for example of a combination of a page number, a line number and a character number. This way, no information stored inside the cover source will lead to the hidden message. Discovering it, relies solely on gaining knowledge of the secret key.

7.2.2 Images

Hiding information inside images is a popular technique nowadays. An image with a secret message inside can easily be spread over the world wide web or in newsgroups. The use of steganography in newsgroups has been researched by German steganographic expert Niels Provos, who created a scanning cluster which detects the presence of hidden messages inside images that were posted on the net. However, after checking one million images, no hidden messages were found, so the practical use of steganography still seems to be limited.

To hide a message inside an image without changing its visible properties, the cover source can be altered in "noisy" areas with many color variations, so less attention will be drawn to the modifications. The most common methods to make these alterations involve the usage of the least-significant bit or *LSB*, masking, filtering and transformations on the cover image. These techniques can be used with varying degrees of success on different types of image files.

Least-significant bit modifications

The most widely used technique to hide data, is the usage of the LSB. Although there are several disadvantages to this approach, the relative easiness to implement it, makes it a popular method. To hide a secret message inside a image, a proper cover image is needed. Because this method uses bits of each pixel in the image, it is neccessary to use a lossless compression format, otherwise the hidden information will get lost in the transformations of a lossy compression algorithm.

When using a 24 bit color image, a bit of each of the red, green and blue color components can be used, so a total of 3 bits can be stored in each pixel. Thus, a 800×600 pixel image can contain a total amount of 1.440.000 bits (180.000 bytes) of secret data. For example, the following grid can be considered as 3 pixels of a 24 bit color image, using 9 bytes of memory:

```
(00100111  11101001  11001000)
(00100111  11001000  11101001)
(11001000  00100111  11101001)
```

When the character A, which binary value equals `10000001`, is inserted, the following grid results:

```
(00100111  11101000  11001000)
(00100110  11001000  11101000)
(11001000  00100111  11101001)
```

In this case, only three bits needed to be changed to insert the character succesfully. On average, only half of the bits in an image will need to be modified to hide a secret message using the maximal cover size. The resulting changes that are made to the least-significant bits are too small to be recognised by the human eye, so the message is effectively hidden.

While using a 24 bit image gives a relatively large amount of space to hide messages, it is also possible to use a 8 bit image as a cover source. Because of the smaller space and different properties, 8 bit images require a more careful approach. Where 24 bit images use three bytes to represent a pixel, an 8 bit image uses only one. Changing the LSB of that byte will result in a visible change of color, as another color in the available palette will be displayed. Therefore, the cover image needs to be selected more carefully and preferably be in grayscale, as the human eye will not detect the difference between different gray values as easy as with different colors.

Disadvantages of using LSB alteration, are mainly in the fact that it requires a fairly large cover image to create a usable amount of hiding space. Even nowaways, uncompressed images of 800 x 600 pixels are not often used on the Internet, so using these might rise suspicion. Another disadvantage will arise when compressing an image concealing a secret using a lossy compression algorithm. The hidden message will not survive this operation and is lost after the transformation.

Masking and filtering

Masking and filtering techniques, usually restricted to 24 bits or grayscale images, take a different approach to hiding a message. These methods are effectively similar to paper watermarks, creating markings in an image. This can be achieved for example by modifying the luminance of parts of the image. While masking does change the visible properties of an image, it can be done in such a way that the human eye will not notice the anomalies.

Since masking uses visible aspects of the image, it is more robust than LSB modification with respect to compression, cropping and different kinds of image processing. The information is not hidden at the "noise" level but is inside the visible part of the image, which makes it more suitable than LSB modifications in case a lossy compression algorithm like JPEG is being used.

Transformations

A more complex way of hiding a secret inside an image comes with the use and modifications of discrete cosine transformations. Discrete cosine transformations (*DST)*), are used by the JPEG compression algorithm to transform successive 8 x 8 pixel blocks of the image, into 64 DCT coefficients each. Each DCT coefficient $F(u, v)$ of an 8 x 8 block of image pixels $f(x, y)$ is given by:

$$F(u,v) = \frac{1}{4}C(u)C(v)\left[\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y) * cos\frac{(2x+1)u\pi}{16} cos\frac{(2y+1)v\pi}{16}\right]$$

where $C(x) = 1/\sqrt{2}$ when x equals 0 and $C(x) = 1$ otherwise. After calculating the coefficients, the following quantizing operation is performed:

$$F^{Q}(u,v) = \left\lfloor \frac{F(u,v)}{Q(u,v)} \right\rfloor$$

where Q(u, v) is a 64-element quantization table. A simple pseudo-code algorithm to hide a message inside a JPEG image could look like this:

Input: message, cover image
Output: steganographic image containing message
while data left to embed **do**
    get next DCT coefficient from cover image
    if DCT $\neq$ 0 and DCT $\neq$ 1 **then**
        get next LSB from message
        replace DCT LSB with message bit
    end if
    insert DCT into steganographic image
end while

Although a modification of a single DCT will affect all 64 image pixels, the LSB of the quantized DCT coefficient can be used to hide information. Lossless compressed images will be suspectible to visual alterations when the LSB are modified. This is not the case with the above described method, as it takes place in the frequency domain inside the image, instead of the spatial domain and therefore there will be no visible changes to the cover image.

7.2.3 Audio and video

Hiding information inside audio files can be done in several different ways. Using the least-significant bit is possible, as modifications will usually not create audible changes to the sounds. Another method involves taking advantage of human limitations. It is possible to encode messages using frequencies that are inaudible to the human ear. Using any frequencies above 20,000 Hz, messages can be hidden inside soundfiles and will not be detected by human checks.

Also, a message can be encoded using musical tones with a substitution scheme. For example, an *F-sharp* tone will represent a 0 and a *C* tone represents a 1. A normal musical piece can now be composed around the secret message or an existing piece can be selected together with an encoding scheme that will represent a message.

Video files are generally a collection of images and sounds, so most of the presented techniques on images and audio can be applied to video files too. The great advantages of video are the large amount of data that can be hidden inside and the fact that it is a moving stream of images and sounds. Therefore, any small but otherwise noticeable distortions, might go by unobserved by humans because of the continous flow of information.

7.3 Detecting steganography

As more and more techniques of hiding information are developed and improved, the methods of detecting the use of steganography also advance. Most steganographic techniques involve changing properties of the cover source and there are several ways of detecting these changes.

7.3.1 Text

While information can be hidden inside texts in such a way that the presence of the message can only be detected with knowledge of the secret key, for example when using the earlier mentioned method using a publicly available book and a combination of character positions to hide the message, most of the techniques involve alterations to the cover source. These modifications can be detected by looking for patterns in texsts or disturbings thereof, odd use of language and unusual amounts of whitespace.

7.3.2 Images

Although images can be scanned for suspicious properties in a very basic way, detecting hidden messages usually requires a more technical approach. Changes in size, file format, last modified timestamp and in the color palette might point out the existence of a hidden message, but this will not always be the case.

A widely used technique for image scanning involves statistical analysis. Most steganographic algorithms that work on images, assume that the least-significant bit is more or less random. This is however, an incorrect assumption. While the LSB might not seem to be of much importance, applying a filter which only shows the least-significant bits, will still produce a recognizable image. Since this is the case, it can be concluded that the LSB are not random at all, but actually contain information about the whole image. When inserting a hidden message into an image, this property changes. Especially with encrypted data, which has a very high entropy, the LSB of the cover image will no longer contain information about the original, but because of the modifications they will now be more or less random.

With a statistical analysis on the LSB, the difference between random values and real image values can easily be detected. Using this technique, it is also possible to detect messages hidden inside JPEG files with the DCT method, since this also involves LSB modifications, even though these take place in the frequency domain.

7.3.3 Audio and video

The statistical analysis method can be used against audio files too, since the LSB modification technique can be used on sounds too. Except for this, there are several other things that can be detected. High, inaudible frequencies can be scanned for information and odd distortions or patterns in the sounds might point out the existence of a secret message. Also, differences in pitch, echo or background noise may raise suspicion.

Like implementing steganography using video files as cover sources, the methods of detecting hidden information are also a combination of techniques used for images and audio files. However, a different steganographic technique can be used that is especially effective when used in video films. The usage of special code signs or gestures is very difficult to detect with a computer system. This method was used in the Vietnam war so prisoners of war could

communicate messages secretly through the video films the enemy soldiers made to send to the homefront.

7.4 Defeating steganograms

While steganograms may not always be successfully detected, there are different ways of removing hidden messages from possible cover sources. Knowledge or certainty of the existence of a hidden message is not needed, since messages can even be destroyed without this. Although there will never be a 100 percent guarantee of success, the number of possible ways of sending hidden messages can easily be reduced using any combination of steganographic defeating techniques.

7.4.1 Text

The best way of removing hidden messages from a plain text might be rewriting and reformulating the contents. Rewriting it using different words and sentence constructions will most certainly remove all ways of reproducing a hidden message, since it will take care of almost every possible way data can be stored inside a plain text. The character position scheme will no longer work because the words have been changed, and the same is valid for the differentations in whitespacing, since the text will have a new layout.

The only method that will not be covered by this technique is the usage of a publicly available cover source. Since this source cannot easily be altered, there is no effective way of stopping this method, except for intercepting the secret key.

7.4.2 Images

Compressing an image using lossy compression will remove messages that are hidden using the LSB modification technique. This will also happen when the image is resized, the color palette is modified or the colors themselves are modified. Conversion to a different image format, which often uses a different type of compression, will also help in removing hidden messages. And altering the luminiscence for example, will remove watermarks in the visible part of an image.

7.4.3 Audio and video

Most of the techniques that can be used on images, can also be applied on audio files. Compressing an audio file with lossy compression will result in loss of the hidden message as it will change the whole structure of a file. Also, several lossy compression schemes use the limits of the human ear to their advantage by removing all frequencies that cannot be heard. This will also remove any frequencies that are used by a steganographic system which hides information in that part of the spectrum.

Another possible way of removing steganograms is lowering the bitrate of the audio file. In that case, there will be less available space to store hidden data and therefore, at least parts of it will get lost.

For video, once more again, the same methods as for images and audio files can be applied to remove hidden information. To defeat the use of signals or gestures however, human insight is still neccessary, as computer systems are not yet capable of detecting this with a reasonable rate of success.

7.5 Conclusion

Steganography, especially combined with cryptography, is a powerful tool which enables people to communicate without possible eavesdroppers even knowing there is a form of communication in the first place. The methods used in the science of steganography have advanced a lot over the past centuries, especially with the rise of the computer era. Although the techniques are still not used very often, the possibilities are endless. Many different techniques exist and continue to be developed, while the ways of detecting hidden messages also advance quickly.

Since detection can never give a guarantee of finding all hidden information, it can be used together with methods of defeating steganography, to minimize the chances of hidden communication taking place. Even then, perfect steganography, where the secret key will merely point out parts of a cover source which form the message, will pass undetected, because the cover source contains no information about the secret message at all.

In the near future, the most important use of steganographic techniques will probably be lying in the field of digital watermarking. Content providers are eager to protect their copyrighted works against illegal distribution and digital watermarks provide a way of tracking the owners of these materials. Although it will not prevent the distribution itself, it will enable the content provider to start legal actions against the violators of the copyrights, as they can now be tracked down.

Steganography might also become limited under laws, since governments already claimed that criminals use these techniques to communicate. More restrictions on the use of privacy-protecting technologies are not very unlikely, especially in this period of time with great anxiety of terrorist and other attacks.

Chapter 8

Online Voting

Written by *Dennis Leman*

Recent years have shown us declining turnout during government elections. This is not only the case in the Netherlands, but many countries have a low voter turnout during elections and see this as a serious problem. Many believe they could improve voter turnout by improving two major issues, namely convenience and mobility of the voting system. By making it easier to vote, it is hoped, to increase participation in elections.

With more and more internet applications being developed to simplify and improve various aspects of every day life, it is only natural the internet is a serious medium to be considered for large scale elections. Certain companies have already established electronic voting systems which they claim to be ready for such elections. The first thought of an online voting systems bring forth the obvious advantages for it being convenient for voters. "Voting in your pajamas" as it is called by some. Many expect simplifying the voting procedure like this will cause more people to cast their votes. Furthermore a system where users can vote remotely would also be cost efficient. All the voting is done at home and the actual work is done by the voter's computer and the main voting centre. In the ideal situation, when voting commences no personal is needed for counting, managing voting booths and such as is the case with traditional elections.

Although all the advantages are apparent, a lot has to be taken into account before such a voting system can be created. One has to make sure the voting system is a safe one. Otherwise even with all the advantages of an online system the voting system would be useless for a democracy. In the first section we will see what kind of properties are required for voting systems in general, it being a traditional election or an online election. After that we will look at some protocols used for online voting, starting at the most basic protocols and evolving to more complex and safer ones. Finally we will look at various social elements that appear with online voting schemes.

8.1 Voting in general

We will define voting schemes in general. All elections follow a certain scheme, whether an online voting or a traditional one. Each election consist of four phases:

- Announcement: The election is announced, the dates are selected.

- Registration: This is the stage in the elections process where a certain election authority creates the electoral roll and publishes it. This phase will sort out all issues regarding the eligibility of voters and it will be concluded by the publishing of the final electoral roll containing all eligible voters. During this phase, eligible voters may be given identification tags, which is nothing more than a unique number to distinguish the voters.

- Voting: During this phase, voters cast their ballot using the protocol used by the used voting scheme. The procedure here varies significantly on the basis of the chosen scheme. In some cases a single session with a ballot collecting authority would suffice, whereas in other cases, multiple sessions may be required.

- Tallying: At the end of voting the tabulating process is initiated and the final tally is published.

The announcement phase take place before the actual voting is done. Therefore his chapter will concentrate mainly on the other phases where the voting protocols take place.

8.1.1 Involved parties

With each election certain parties are involved. The participants can roughly be divided into two groups:

- Voters;

- Election Authorities: Validator and Tallier.

The form of the election authorities can highly differ depending on the voting protocol used for the election. In some cases the two subgroups (validator and tallier) are combined into one group. Generally when the election authorities are divided, the tallier will be committed with the handling of computing the results. The validator communicates between voters and the tallier and makes sure only eligible voters cast (valid) votes.

8.1.2 Properties of a voting scheme

For a voting system to become useful it has to meet certain requirements. These are as follows:

- Accuracy: Votes cannot be altered or removed afterwards. Also invalid votes will not be counted.

- Democracy: Only eligible voters can cast votes. An eligble voter can only cast one valid vote.

- Privacy: During and after voting it is not possible for either the election authorities or other voters to find out what a certain voter has voted for. Voters should not be able to prove what they have voted for, also called receipt-freeness. If they could do so it could lead to vote selling and coercion.

- Verifiability: Voters can indepentely verify all ballots are tabulated correctly.

- Convenience: Voters are able to cast their votes in one short sessions with minimal equipement.

Casus 8.1: ANDOS.

All-or Nothing Disclosure of Secrets (ANDOS) protocols address the following problem: a merchant has n secret bit-strings and wishes to sell one of them to a buyer who has the ability to choose which one he wants. There are two privacy requirements: the merchant does not want the buyer to obtain information about any other secret and the buyer does not want the merchant to learn anything about the string he has chosen.

- Flexibility: There is a varierity of ballot formats. (e.g., write-in candidates, survey questions, multiple languages); be compatible with a variety of standard platforms and technologies; and be accessible to people with disabilities;
- Mobility: Voters are not restricted to physical location from which they can cast their votes.
- Scalability: The size of the election does not drastically effect the performance of the voting-procedures.
- Reliability: Election systems should work robustly. When various failures occur there should not be a loss of any votes.

8.2 Voting Protocols

The voting protocols define how communicating runs between the election authorities and the voter. Preferably fulfilling the constrains mentioned in the previous subsection. Many papers have been written on this subject and many protocols have been developed. It would be impossible to name and discuss all of them, but the most known and used protocols will be discussed in the following subsections.

8.2.1 Simple protocol

We start off by mentioning the most simple protocol. This would be the protocol if one wouldn't think things through. Thus only considering the advantages of online voting, but neglecting the dangers and diffilculties. In this simple system voters would submit their vote along with a unique identification number (issued when they registered) to a validator who would then take their names off the allowed voters list. The validator would then strip off the id number and submit just the votes to the tallier who would count the votes.

Although this system has the advantages of being flexible, convenient and mobile, this system is far from secure. If the validator is compromised votes can easily be traced back to the voter or votes could be changed. Both privacy and accuracy, two important properties, lack with this protocol. Also there is no verifiability, voters have now way of knowing wether their vote is correctly counted. If the validator and the tallier collude then the voters also have no way of knowing about this either. Voters could also vote multiple times by using other voters id numbers which undermines democracy. So it easy to see that a simple protocol as this is far from sufficient to run a fair election and more complex methods have to be used.

Figure 8.2: The two way agency protocol

8.2.2 Two agency protocol

Nurmi, Salomaa, and Santean [NSS91] proposed an approach that solves many of the problems mentioned above. In this Two Agency Protocol as displayed Figure 8.2, the electronic validator distributes a secret identification tag to each voter just prior to the election. The validator then sends the tallier a list of all identification tags, with no record of the corresponding voters. Each voter sends the tallier his or her identification tag and an encrypted file containing a copy of the tag and the voted ballot. Encryption is done by public/secret key-methods such as ElGamal or RSA.

At this point the tallier can make sure the identification tag is valid, but the tallier has no way of examining the contents of the ballot. Also, since the tallier only received a list with identification tags, the tallier has no way of pairing a certain ballot to a voter. The tallier publishes the encrypted file (so that the voter has proof that the file was submitted on time), and the voter responds by sending the tallier the key necessary to decrypt it. When the election is over, the tallier publishes a list of all voted ballots and the corresponding encrypted files. At this point the voters can confirm that their votes were counted properly. Any voter who finds an error can protest by submitting the encrypted file and decryption key again. Because the encrypted file was published earlier, the tallier cannot deny having received it.

The Two Agency Protocol is verifiable by individual voters (unlike the simple protocol discussed earlier), however, it still has several problems. Most importantly, it does not protect voters' privacy if the tallier and validator collude. The authors note that having the election authorities divided in two subgroups do not contribute to an improved security and can therefor be combined to one party.

Therefore they devised another protocol which instead as the Two Way Agency Protocol uses only one party. Not surprising the protocol was called the One Agency Protocol. Generally the protocol is identical to its predecessor the Two Agency Protocol, except for the tag distribution procedure. In the One Agency Protocol, tags are distributed by the tallier (there is no validator) using an ANDOS (all-or-nothing disclosure of secrets, see 8.1) protocol for secret selling of secrets. The ANDOS protocol makes sure voters receive a secret identification tag, without the tallier knowing what it is. This does solve the problem of a colluding election authority, but the ANDOS protocol is computationally complex which makes it inpractible when used for elections with a large number of voters.

If we compare the Agency Way protocol with the simple protocol we see that the major privacy and accuracy problems are solved, without decreasing much of the convenience, mobililty and flexibility. Although the major problems which existed with the simple protocol are solved, both the One and Two way agency protocols fail to satisfy a part of the privacy property. The mechanism that allows voters to verify that their votes were counted correctly

Figure 8.3: The blind signature protocol

also allows them to prove that they voted in a particular way. Additionally the accuracy property is also not completely satisfied because the tallier may cast votes for all the voters who have been assigned tags but do not exercise their right to vote. These voters may discover this violation and report it, but they cannot prove that they did not actually vote.

8.2.3 Blind Signatures

The concept of blind signatures was first introduced by David Chaum in 1982, he suggested that blind signatures could be used for secret ballot elections. Ten years later, Fujioka, Okamoto, and Ohta developed a practical voting scheme that uses blind signatures to solve the collusion problem inherent in protocols like the Two Agency Protocol without significantly increasing the overall complexity of the protocol [FOO93].

Blind signatures are a class of digital signatures that allow a document to be signed without revealing its contents. The effect is similar to placing a document and a sheet of carbon paper inside an envelope. If somebody signs the outside of the envelope, they also sign the document on the inside of the envelope. The signature remains attached to the document, even when it is removed from the envelope. The concept and details of the techniques behind the uses of blind signatures can be found in [Tel02]

In the Fujioka, Okamoto, and Ohta protocol, shown in Figure 8.3, the voter prepares a voted ballot, encrypts it with a secret key, and blinds it. The voter then signs the ballot and sends it to the validator. The validator verifies that the signature belongs to a registered voter who has not yet voted. If the ballot is valid, the validator signs the ballot and returns it to the voter. The voter removes the blinding encryption layer, revealing an encrypted ballot signed by the validator. The voter then sends the resultant signed encrypted ballot to the tallier. The tallier checks the signature on the encrypted ballot. If the ballot is valid, the tallier places it on a list that is published after all voters vote. After the list has been published, voters verify that their ballots are on the list and send the tallier the decryption keys necessary to open their ballots. The tallier uses these keys to decrypt the ballots and add the votes to the election tally. After the election the tallier publishes the decryption keys along with the encrypted ballots so that voters may independently verify the election results.

8.2.4 Sensus

One of the drawbacks of the Blind Signature protocol is the voter has to wait till the voting has ended before the voter can verify the casted vote was the correct one, which is not in line with the property of flexibility. Cranor and Cytron developed and implemented a scheme, called the Sensus system, closely based on the Blind Signature protocol mentioned above. The major difference between the schemes emerges after the voter has submitted the encrypted ballot to the tallier. Instead of waiting till the voting ends the tallier sends a receipt to the voter when his/her ballot has been received. This receipt is no more than a confirmation the vote has been transferred to the tallier correctly. The voter may submit the decryption key immediately after receiving this receipt, completing the entire voting process in one session. The implemented Sensus system employs a pollster agent that performs all cryptographic functions and transactions with the election programs on the voter's behalf. Tests conducted with a prototype implementation of Sensus indicate that the entire voting process can be completed within a few minutes.

The Sensus protocol possesses most of the desirable characteristics; however, it fails to correct some of the problems inherent in the One and Two Agency protocols. Perhaps the most important problem is that the election authority (in this case the validator) can cast votes for abstaining voters. These invalid votes can be detected by the abstaining voters themselves or by an auditor who checks the signatures on all the validation requests submitted. However, there is no way to identify the invalid ballots and remove them from the tally – if significant numbers of invalid votes are detected, the election will have to be repeated. If voters who wish to abstain submit blank ballots, this problem can be avoided.

Even the Sensus protocol does not satisfy the verifiability property completely because the protocol cannot be verified by any interested party. The Sensus allows voters to verify that their own votes were counted correctly and thus satisfying the property to a large degree. However, a verification system that relies on voters taking action after the election is over is not likely to be thoroughly exercised. One way to encourage voters to verify their votes would be to implement the protocol to automatically verify the vote after the election and report back to the voter if any problems are detected.

8.3 Conclusion

During the primary elections for choosing the Democratic candidate in Arizona 2000 the first binding election on such a large scale was held which gave voters the possibility to vote online. Authorities claim the elections to have run smoothly without any security problems, but the online voting part of the election was burdened with some practical problems. Phonelines, which were used to provide information, were overloaded. The server was down for a period of several hours, which meant voters could not cast their vote during that period. It is hard to say whether votes were lost during that period of down-time, meaning one could not verify the reliability of the voting system. The same goes for other properties mentioned at the beginning of this chapter.

As said in the beginning there are numerous different protocols which can be used. This chapter essentially shows only one acceptable protocol (blind signature). Other protocols with good results have been devised, but none pass all requirements. One of these protocols is the homomorphic voting system which is proven to sustain the privacy property including being receipt-free, but is limited to yes/no questions. Mix-nets protocol is another acceptable scheme

but is inpractical when the number of voters is large and thus not well scalable.

To hold large elections at this stage might still be premature. Protocols could use further developing and although protocols are acceptable, protocols should also withstand practical problems. Furthermore the notion of more people would vote if elections were held online is an assumption yet to be proven. The elections in Arizona 2000 showed more voters compared to the years before, but one cannot base all conclusions on one occurence. Studies have shown that simplifying traditional elections to improve mobility and convenience had little consequence for the voter turnout. Some have explained the increase of voters during the Arizona 2000 elections due to the hype around the elections. It was the first election on large scale to be using an online voting system. The turnout, with the novelty of online voting disappearing, might decline back to what it was before.

Another social aspect of online voting which deserves some attention is the fact minorities and the less educated are less familiar with computers and the internet. Some believe holding an online election would scare of this part of the population from voting, creating a biased election. Besides the technical theories and practical implementations there are also social question marks to be considered before an election should be held which can be of great impact on a country's future.

Chapter 9

Montgomery Multiplication

Written by *Joost Verhoog*

There are many different cryptosystems available today, all with different characteristics and behaviours. However, there is one property that many cryptosystems share: they compute modulo a large number N. These include the RSA and ElGamal-systems, Diffie-Hellman key exchange, and Schnorr's signing scheme. This is because computing modulo N gives access to all the mathematical properties that the groups $\mathbb{Z}_N$ and $\mathbb{Z}_N^*$ possess.

The implementations of these computations is not at all trivial. In the past decades, many papers concerning fast multiplication and exponentiation have been published. Many different algorithms have been proposed, all with different characteristics with respect to speed and ease of implementation. Most of them are based on Montgomery's algorithm for fast modular multiplication.

We'll start with describing modular multiplication in section 9.1, give Montgomery's Algorithm in section 9.2, and its implementation in hardware in section 9.3. A concluding section 9.4 describes timing attacks, and an implementation on a processor that cannot be attacked in this way.

To prevent confusion between the modulus and congruence, we define:

Definition 9.1 *$x \,\%\, M$ is defined as the remainder of x when dividing by M*

Definition 9.2 *$x \equiv_M y$ iff $M|(x-y)$*

9.1 Definition of the implementation

The definition of multiplying modulo N is given in Algorithm 9.1.

This is a definition of the algorithm in the sense that every implementation must give the same result as this one. This definition could be used as an implementation, only the the `'div'` operation is, in general, very slow. Because modular multiplication is the basis of modular exponentiation, there are usually many multiplications involved, and having to do so many divisions makes the algorithm very slow.

It is clear that we need a better way to multiply modulo N; fortunately, there is.

```
function Mul(x, y)
  m ← x * y
  d ← m `div` N
  return m − d * N
```

Algorithm 9.1: Definition of multiplying modulo N

9.2 Montgomery's algorithm

Whereas division is slow in general, it is fast for some numbers. We'll try to find an algorithm that divides by an number R for which it is easy to divide. For example, if m is in binary format and $R = 2^n$, dividing m by R means chopping of n bits from m, which is very fast. Of course $m \,\%\, R$ means taking the least significant n bits of R.

In order to do this, we take R relatively prime to, but larger than N, and define:

Definition 9.3 $M : \mathbb{Z}_N \to \mathbb{Z}_N : x \mapsto xR \,\%\, N$.
$M(x)$ is called the Montgomery Representation [Mon85] of x.

We observe that $M(x) < N$. Because M is a one-on-one from $\mathbb{Z}_N$ to $\mathbb{Z}_N$, it has an inverse. If we define R' to be the inverse of R in $\mathbb{Z}_N$, then the inverse of M, $M^{-1} = xR' \,\%\, N$, because $(xR \,\%\, N)R' \,\%\, N = xRR' \,\%\, N = x \,\%\, N$.

Now take $N' = -N^{-1}$ in $\mathbb{Z}_R$, and define the function REDC as in Algorithm 9.2.

Lemma 9.4 *The division in line 2 of REDC gives an integer result.*

Proof. We need to proof that the division by R is possible. We observe that

$$\begin{array}{rcl} mN & = & ((x \,\%\, R)N' \,\%\, R)N \\ & \equiv_R & xNN' \\ & \equiv_R & -x \end{array}$$

Which means that

$$\begin{array}{rcl} x + nM & \equiv_R & x + -x \\ & \equiv_R & 0 \end{array}$$

So $x + nM$ is divisable by R. △

```
function REDC(x)
  m ← (x % R)N' % R
  t ← (x + mN)/R
  if t ≥ N then
    return t − N
  else
    return t
  end if
```

Algorithm 9.2: Montgomery's algorithm

This is a much faster algorithm than Algorithm 9.1. It divides by R and takes remainders by R, which are both very fast. The only operations that consume time are the two multiplications. Because of this, the following lemma may come as a surprise:

Lemma 9.5 *If $x < RN$, then REDC(x) computes $M^{-1}(x) = xR' \,\%\, N$*

Proof. First we observe that

$$\begin{array}{rcll} x + mN & \equiv_N & x & \\ tR & \equiv_N & x & \text{(because } t = (x + mN)/R\text{)} \\ t & \equiv_N & xR' & \text{(multiplying both sides by } R'\text{)} \end{array}$$

So t is congruent modulo N to the desired result. Now we see that

$$\begin{array}{rcll} m & < & R & \text{(because } m \text{ is computed modulo } R\text{)} \\ x + mN & < & RN + RN & \text{(because we assume } x < RN\text{)} \\ (x + mN)/R & < & 2N & \text{(dividing both sides by R)} \end{array}$$

So either t or $t - N$ is the desired result. In the algorithm, this is checked, so REDC gives the desired result. △

With this, computing $M(x)$ and doing multiplication becomes very easy:

1. **Converting to Montgomery Representation**

   $M(x) = REDC(xR^2)$, because

   $$\begin{array}{rcl} REDC(xR^2) & = & xR^2R' \,\%\, N \\ & = & xR \,\%\, N \\ & = & M(x) \end{array}$$

   R is fixed, so R^2 has to be computed only once. Thus, computing $M(x)$ takes three multiplications.

2. **Multiplication in Montgomery Representation**

   If we have two numbers in Montgomery Representation, we can multiply them using $REDC$. If $x' = M(x)$, $y' = M(y)$, then we compute their product modulo N in Montgomery Representation as $REDC(x'y')$, because

   $$\begin{array}{rcl} xyR \,\%\, N & = & xyRRR' \,\%\, N \\ & = & (xR \,\%\, N)(yR \,\%\, N)R' \,\%\, N \\ & = & M(x)M(y)R' \,\%\, N \\ & = & REDC(x'y') \end{array}$$

   Lemma 9.5 applies, because $x' < N$ and $y' < N$, so $x'y' < N^2 < RN$, and its premisse holds.

9.3 Implementation in Hardware

It it often necessary to implement cryptographic systems, including modular multiplications, directly into hardware. In the absence of a processor, only operations on bits are possible. Luckily, there is an easy translation of Montgomery's Algorithm (Algorithm 9.2) to a version that uses bit-operations and additions only. It is given in Algorithm 9.3.

In this algorithm, we take $R = 2^n$ to multiply n-bit numbers. We assume N is odd, and denote x in bits as $(x_{n-1} \dots x_0)$. There are n steps. In every step i, y is added to t if the i^{th}

```
function MultBit(x, y)
  t ← 0
  for i ← 0 to n − 1 do
    if x_i then
      t ← t + y
    end if
    if t_0 then
      t ← t + N
    end if
    t ← t/2
  end for
  if t < N then
    return t
  else
    return t − N
  end if
```

Algorithm 9.3: Montgomery's Algorithm on Hardware

bit of x is a 1. If, after that, t is odd (so $t_0 = 1$), N is added to make t even, so it can be divided by two.

This algorithm uses addition and division by two, both of which can be easily implemented in hardware.

Lemma 9.6 *Algorithm 9.3 returns* $xyR^{-1} \,\%\, n$

Proof. Define T_i as the value of t after i loops. Then $T_0 = 0$, $T_{i+1} = (T_i + x_i y)/2$ or $(T_i + x_i y + N)/2$, whichever is an integer.

By induction over i we prove that (1) $2^i T_i \equiv_N (x_i - 1 \ldots x_0)y$ and (2) $0 \leq T_i < N + y (< 2N)$, so that after the for-loop T_n or $T_n - N$ is the desired answer.

$T_0 = 0$, so (1) and (2) hold.

Now assume that (1) and (2) hold. (Induction Hypothesis)

To prove (1), we distinguish two cases:

1. Assume $(T_i + x_i y)/2$ is an integer.

$$\begin{array}{lcll} 2^{i+1}T_{i+1} & = & 2^{i+1}(T_i + x_i y)/2 & \text{(definition of } T_{i+1}\text{)} \\ & = & 2^i(T_i + x_i y) & \\ & \equiv_N & 2^i((x_{i-1} \ldots x_0)y + x_i y) & \text{(Induction Hypothesis)} \\ & = & 2^i(x_i \ldots x_0)y & \end{array}$$

2. Assume $(T_i + x_i y + N)/2$ is an integer.

$$\begin{array}{lcll} 2^{i+1}T_{i+1} & = & 2^{i+1}(T_i + x_i y + N)/2 & \text{(definition of } T_{i+1}\text{)} \\ & = & 2^i(T_i + x_i y + N) & \\ & \equiv_N & 2^i(T_i + x_i y) & \text{(substracting a multiple of N)} \\ & \equiv_N & 2^i((x_{i-1} \ldots x_0)y + x_i y) & \text{(Induction Hypothesis)} \\ & = & 2^i(x_i \ldots x_0)y & \end{array}$$

To prove (2), we observe that

```
function MontProc(x, y)
  t ← 0
  for i ← 0 to l − 1 do
    m ← (t_0 + x_i y_0)N′ %2^α
    t ← (t + x_i y + mN)/2^α
  end for
  if t < N then
    return t
  else
    return t − N
  end if
```

Algorithm 9.4: Montgomery's Algorithm on a processor

$$\begin{array}{rcll} T_{i+1} & < & (T_i + x_i y + N)/2 & \text{(because N is positive)} \\ & < & (N + y + x_i y + N)/2 & \text{(Induction Hypothesis)} \\ & = & (2N + (x_i + 1)y)/2 & \\ & = & N + (x_i + 1)/2 & \\ & < & N + y & \text{(because } x_i \in \{0, 1\}\text{)} \end{array}$$

△

Now that we have $MultBit(x, y)$, we can compute $M(x) = MultBit(x, R^2)$ and $M^{-1}(x) = MultBit(x, 1)$

9.4 Concluding Remarks

When implementing a cryptographic system, one has to be very careful. In an attack known as a *timing attack*, the opponent (Oscar) has the possibility to inspect the resource behaviour of the cryptographic system when a computation involving the private key is done. This is not an uncommon situation, especially on shared machine, like a terminal. When the resource behaviour of the cryptographic system depends on the private key that is used, Oscar can gather information about the private key in a timing attack. This is something we want to prevent. In the algorithms that we have seen this far, there is condition at the end. Thus, information about t is used to either substract or not. This will have an effect on the resource behaviour of the program. On such a small scale this is not noticable, but when you repeat the algorithm many times, a timing attack becomes possible because of it. That is why we'll try to prevent conditionals in the next algorithm.

When the processor supports word multiplication, there is a better way to implement Montgomery's Algorithm (see [BM02]). We assume that the processor has word size 2^α, and that x has l digits (x is written as l α-bit words $x_{l-1} \dots x_0$). We take $(R = 2^\alpha)^l$. The algorithm is given in Algorithm 9.4.

It it just Algorithm 9.2, applied for every digit of n. Surprisingly, there is no check condition of $t > N$ inside the loop. This is really something we want, and in Lemma 9.7 we prove that the only thing that we have to do is take R large enough (see [HQ00]).

Lemma 9.7 *The result of a Montgomery Multiplication is bounded by* $2N$ *if* $R > 4N$.

Proof. $t = \frac{XY+mN}{R} = \frac{XY}{R} + \frac{m}{R}N < \frac{4}{4}N + N = 2N$ △

Concluding, Montgomery's Algorithm gives a good way to implement modular multiplication, and can be easily implemented on both hardware and processors. It has been improved in many different ways to become faster, more precisely configured, etc. These implementations are too large to discuss, but the papers about them become relativly easy to read once there is basic understanding of Montgomery's algorithm.

Chapter 10

Vickrey auction protocols

Written by *Bart de Boer*

In this chapter, the Vickrey auction will be discussed. An online auction has certain requirements and the Vickrey auction has some interesting theoretical properties that may allow the Vickrey auction to meet these requirements. A protocol will be described that realizes these theoretical properties.

10.1 Auctions

In the last years there has been a growing interest in the area of multi-agent systems. Multi-agent systems are composed of technical entities called agents that interact. These agents can in some sense be said to be intelligent and autonomous. In the agent community topics such as electronic commerce, automated resource allocation and task assignment are of great importance. Auctions are very flexible and efficient tools that are often applied to these problems. For instance, using an auction for selling goods or recourse allocation might work as follows: each bidder makes a bid expressing the amount he is willing to pay and the bidder with the highest bid wins the auction. For task assignment scenarios the auction works exactly the other way around: each bidder willing to execute a task bids the amount he wants to be paid for executing the task and the bidder with the lowest bids wins the auction and gets to execute the task.

10.1.1 Auction requirements

To be able to have a fair auction that functions well, there are some requirements to be met:

- Economic design: the auction should be designed on solid economic principles and the participants in the auction should bid their true value of the item. If bidders have some reason to bid less than their true value, it is possible that the winning bid may be artificially low.
- Fast execution: The auction must run in reasonable time, even for a large number of participants.

- Privacy: The auction must be private. Not only should outsiders not be able to see the bids, but the participants of the auction should also not be able to see each other's bids. We don't even want the auctioneer to be able to know the bids. One necessary exception to this is of course that the final selling price of the item needs to be public. Keeping all the bids private is a very useful requirement, since especially for resource allocation and task assignment, the same things may be auctioned many times. If after the first round all the true values are known, participants can adjust their price to their competitors and thus reduce or increase the selling price. This is something that is not allowed by the requirement of economic design.
- Anonymity: We don't want the identity of the participants to be revealed. This is something different than privacy, which protects the values of the bids, but not the identity of the bidders.
- Public verifiability: All bidders must be able to verify the outcome of the auction.

10.1.2 Auction types

Which auction types fit our requirements best? We consider the following broad categories of auctions:

- English auction. The English auction is a so-called 'increasing-price auction'. In this type of auction a good or commodity is offered at an increasing price. Initially it is offered at x tokens, at successive points of i tokens it is bid at $x+i$ tokens. At each time a bidder can make clear if he is still willing to pay the price and the auction ends when there is only one participant left who is willing to pay that price. This sort of auction can be found at Sotheby's and Christie's. The English auction has many disadvantages:
  - The time it takes to execute the auction is proportional to the price at which the item is sold.
  - The communication costs are very high, since they too depend on the selling price. Also, at lower prices large numbers of bidders may simultaneously bid for the item, which may require much communication.
  - An enormous amount of information is leaked. By observing the auction carefully, much information about the good's valuation by the participants is obtained.
- Dutch auction. The Dutch auction is a so-called 'decreasing price auction'. This type of auction is similar to the English auction, except for the fact that the price decreases over time instead of increases. Initially, the price is again set to be x. At time i, the price will be $x - i$. The first participant to bid is the one who takes the item. Like the English auction, the execution time of the Dutch auction is dependent on the selling price of the item. The Dutch auction preserves a lot of privacy, since only the highest bid is revealed. Unfortunately, the winning bid is often just the bid that we want to keep secret the most. The Dutch auction also doesn't fit our requirement of economic design.
- Vickrey auction. The Vickrey auction is a 'second-price' sealed-bid auction. The participants all send a sealed bid to the auctioneer who is the only one who is able to open the bids. The auctioneer determines which bid is the highest and the participant who made this bid wins. However, the price the winner pays is the price of the second highest bid. The Vickrey auction protocol has some qualities that make it particularly interesting:

- Because there is only one round of bidding, it requires very little communication (and thus low bandwidth) and has low time consumption.
- It possesses a dominant strategy, namely for each participant to bid his true valuation of the good.
- By sealing the bids with some method of encryption, it is theoretically possible that the bids expressing the private values remain secret.

10.2 The Vickrey auction

In the previous section we set the requirements we want an auction to meet. Clearly, the first two types of auctions mentioned will not fit the requirements. The Vickrey auction however has the necessary theoretical properties. The fact that it has only one round of bidding makes fast execution possible. The dominant strategy meets our requirement of economic design and sealing the bids should allow the bidders privacy and anonymity. We will see later on that the last requirement of public verifiability will also be met.

10.2.1 The dominant strategy

We mentioned that the Vickrey auction has a dominant strategy. This means that if an agent applies this strategy, he is guaranteed to receive the highest possible payoff, no matter which strategies are used by the other bidders. The dominant strategy for the Vickrey auction is to bid one's true valuation of the good or task that is being auctioned. Even if an agent knows all the other bids in advance (which is something that should not be possible if the requirements of the auction are met), then he would still be best of if he just bids his own true value. The only conditions for this dominant strategy are that the bidders are symmetric (and thus cannot be distinguished) and that their valuations don't depend on each other. We prove this by using the example of bidding for task assignment with two agents (after all, the outcome of the auction is only determined by two bids: the highest bid to determine who won and the second-highest to set the price). The true values of the agents are in this case the costs they make when doing the task.

Given two agents A and B with corresponding true values v_a and v_b and corresponding bids b_a and b_b. We recall that in order to win the auction for a certain task, an agent must bid the lowest value. By doing this, he claims to be able to do the cheapest. The profit for agent A is now defined by the following equation:

$profit_a(b_a,b_b) =$

- $b_b - v_a$ if $b_a \leq b_b$
- 0 if $b_a \geq b_b$

In other words, if agent A makes a bid that is lower than the bid of agent B, he wins the auction and he can do the task. For this task, he gets paid the second-lowest bid, namely the value of agent B's bid. His profit is now the pay minus his own costs: $b_b - v_a$. If he does not win the auction, his profit is obviously 0. Now it can be easily seen why bidding the true valuation of the task is the optimal strategy. We consider agent A and look at the possible profits that could be made by not bidding his true value. If A bids more than his true value, maybe hoping to so get paid more for the job, there are three possible outcomes:

- $b_b < v_a < b_a$: B wins the auction and gets paid b_a for the job. So agent B receives more money than if A had bid v_a. Agent A however does not profit.

- $v_a < b_b < b_a$: A loses the auction, instead of winning it by bidding v_a. His profit is 0.

- $v_a < b_a < b_b$: A still wins the auction, but does not gain anything, because the task price still remains b_b.

If agent A decides to bid a value lower than his true value, one of the following cases describe the result:

- $b_a < v_a < b_b$: A wins, but is paid the same amount of money as if he bid v_a.

- $b_a < b_b < v_a$: A wins, but gets less money than his own costs and he loses v_a-v_b.

- $b_b < b_a < v_a$: A still loses and reduces B's payoff by v_a-b_a.

Concluding, bidding anything else than the true value v_a cannot yield more profit than just bidding v_a. This meets the requirement of economic design and simplifies the bid preparation. It makes speculation about other agents' true value's useless as long as an agent's only motivation is maximizing his profit.

10.2.2 The antisocial attitude

In the previous section, we said that the dominant strategy was a strategy with which the bidder was always best off if he only wanted to maximize his own profit. However, in the real-world applications, it is not very realistic to assume that agents are only interested in their own profits. It is more likely that there are also agents present who try to gain as much money *relative* to others (their competitors). We should therefore also consider the possibility of the so-called *antisocial agents*: agents who accept small losses if they can inflict great losses to other agents. Now how could an antisocial agent harm his competitors in the Vickrey auction? We go back to the example with two agents, A and B, and the auctioning of a certain task. Let's assume that agent B can do the task cheaper than agent A, implying that B's costs and thus his true value are lower than A's. Now agent A cannot win the auction effectively, but agent B's profit depends solely on what agent A bids. If agent A were to know agent B's true value, he could reduce his bid to a value lower than his own true value and thus bring agent B's profit down to a minimum. On the other hand, if agent B were to know agent A's true value then if the same task if being auctioned round after round, agent B would know that agent A is bidding below his own true value. He might decide that he doesn't care about gaining the small profit A left him and would rather see A lose a lot. He increases his bid just enough to let A win the auction, but this would mean that A suffers a great loss, namely the difference between A's true value and B's bid (Casus 10.1).

We see that the possibility of antisocial agents could have a great effect on the Vickrey auction. After all, when we think of the dominant strategy to be one of the advantages of the Vickrey auction, we'd better make sure that it really is the dominant strategy. This brings us back to the requirements we made. Privacy and anonymity are needed to make sure that antisocial agents do not get a chance to disrupt the auction. In the next section we will see how this is achieved.

Casus 10.1: Antisocial bidding

In the figure, we see that agent B's costs are lower than those of agent A. He wins the auction if the dominant strategy is used and gets paid v_a-v_b. If agent A knows the bid and true value of agent B, he can lower his bid to be b_a and reduce B's profit to b_a-v_b, which is almost nothing. If agent B knows agent A will make this bid, he can discard this small profit and inflict a great loss to agent A by bidding b_b. In this case, agent A will win the auction and get paid b_b. Since his costs v_a are much higher than b_b, he will lose a lot of money.

10.3 Privacy and anonymity

The previous section explained that we need privacy and anonymity to be able to say that having a dominant strategy is one of the properties of the Vickrey auction. However, once the dominant strategy is truly used, it is even more important that the bidders don't know each other's bids. In auctions where the bidders don't bid their true value, this is not as important because the bidding prices are not the bidder's honest price anyway. However, in the Vickrey auction, smart bidders use their true value and most bidders consider this true value to be information they do not want to share with other bidders.

10.3.1 Sealing the bids

Sealed-bid auctions are an ideal way to distribute merchandise. It is very suitable for a network environment and is therefore attracting a lot of attention in the research of e-auction. Encryption sealing is thought to be able to implement fairness and bid-privacy. The Vickrey auction is a sealed-bid auction. This means that all bidders are required to seal their bids before they submit it. After that time all the bids are opened and the winning bid is determined. To seal the bids, they are encrypted, usually with public key algorithms, such as RSA. This method is used to implement bid privacy. All the bidders have their own private key, which can be dealt by the group manager. This is the case in the example of group signatures, which is described below. This group manager has his own key to decrypt the bids and since he is the only one with this key, he is the only one who can decrypt the messages signed by the participants of the auction. The bidders encrypt the bids with their private key, and also sign the bid. Since they are the only one with that particular key, they can be identified by it.

There are a couple of different approaches to encrypt the bids. Below two of these approaches, the use of group signatures and distributed keys, are described. There are of course more approaches than just these two, but all these approaches are comparable. They all use some sort of signature scheme and they all suffer from the same drawback: the bid confidentiality and privacy are conditional. Why this is the case will be seen shortly. In the next section, a different method will be shown, which will eliminate the conditional aspect.

Group signatures

A group signature scheme allows members of the group (in this case the bidders) to sign messages anonymously on behalf of the group. The group manager is the only one who can open the signed messages and reveal the signer's identity. Group signature schemes are provably secure under the strong RSA and decisional Diffie Hellman assumptions. They can be used for electronic banking systems and electronic voting, but also in auctions. In the latter case, the bidders are the only ones who can place bids and the auctioneer is the only one who can open these bids and determine the bidder's identity.

Privacy and anonymity is so achieved among the bidders. However, the same does not go for the auctioneer. He still knows all the bidders' true values. Bidders are usually highly reluctant to show anybody their true valuation of the good that is sold, even if it is only the auctioneer. The correctness of the auction is also based on the assumption that the auctioneer or the group manager is truthful. The highest bidder has to trust the auctioneer that he doesn't overstate the price of the second-highest bid in order to get more money for himself or for his client. An auctioneer that is not trustworthy could also declare a fake winner to have won the auction if he or his client finds the bids too low. An auctioneer might also form a collusion with one of the bidders. He could for instance influence the selling price if he were to collude with the second-highest bidder without having to lie about the second-highest price himself.

Secret sharing

It has become clear that achieving privacy and anonymity does not guarantee a correct auction if the auctioneer or group manager has full responsibility, but we cannot be sure that he is trustworthy. One way to reduce the knowledge of the auctioneer is by using a so-called secret sharing scheme and distributing the knowledge of the true values over a number of auctioneers. We then use a protocol in which they combine their knowledge in such a way that the outcome of the auction can be determined, without the auctioneers finding out more than partial information about the true values of the participants. How this works is described below.

We assume that all bids are drawn from some ordered set: $S = \{d_1, \ldots, d_V\}$. V is now the number of possible bids. There are n bidders and m auctioneers. We use a prime number p of sufficient size. All calculations will be made modulo p.

Each bidder composes his bid by creating V lists of m random integers modulo p. The rule for creating these lists is that for a price d_l the m integers should sum up to a non-zero number if the bidder is willing to pay d_l, whereas when he does not want to pay d_l for the item, the m integers should sum up to 0. Each auctioneer is sent V numbers by each bidder. The ith auctioneer is sent the ith number from each list. The submissions are signed separately by the bidders so they can be opened one at a time and opening one of them will not reveal bids at other values.

To determine the winning bid, the auctioneers compute for each d_l the sum of the n numbers received for that d. Now each of the auctioneers has a number that is totally useless, but when they all combine these numbers, they have a number that is non-zero (with high probability) when there is at least one bidder who is still willing to pay the price. As soon as they find a bid for which the sum of all the bidders' numbers is zero, they know what the highest bid was. Once the highest bid is known, the auctioneers can use the signatures on the bids to prove which bidder is the winner by reassembling all bids for that winning d. The second highest bid can be determined in the same way, after disregarding the numbers given by the highest bidder.

One can see that this protocol is an improvement to the group signature scheme. Without combining their knowledge, the auctioneers have very little information on the true values of the bidders. There are also some weaknesses though. The auctioneers know the exact value of the highest bid, which as was mentioned before is often the bid with the highest level of secrecy. Another weakness is that determining the winner of the auction requires a number of rounds linear in the number of bidding points, which is unacceptably slow, especially for auctions with high prices.

10.3.2 Minimizing the auctioneer's knowledge

We have seen that even when using signature schemes and distributing the information of the bidders' true values over several auctioneers, the correct outcome of the auction is still vulnerable to collusion between auctioneers. There is also still information unnecessarily known to auctioneers and the execution time of the auction becomes unacceptable. It is also not possible for the bidder that won the auction to determine whether or not the price he was told to be the second-price was actually the right price. An untrustworthy auctioneer could name any price between the highest bid and the second-highest bid. This would increase the seller's revenue and since auctioneers usually get a percentage of the selling price, he would benefit from it too. Since no bids are published, it is never clear whether or not the given price was a real bid.

The following will describe a way to minimize the auctioneer's knowledge by going back to the signature scheme with just one auctioneer. We will see that it is in fact possible to let nothing but partial information of the bids be known to anybody, including the auctioneer, except for the second-highest bid, which can also be made known to all the bidders without revealing vital information.

10.3.3 Two bidding phases

As stated above, a bidder that won the auction is incapable to detect whether the given price is really one of the submitted bids. Cryptographic signatures can prevent a completely fake second-highest bid, but this does not stop an auctioneer from working together with an agent and accepting his bid after he has already evaluated the bids and knows which bid the auctioneer can make to increase his profit.

We therefore divide the auction process into two parts. In the first part the auctioneers receive all the encrypted bids and publishes these anonymized bids on a electronic blackboard. This electronic blackboard is a popular tool to detect intrusion and it works essentially in the same way as a real blackboard would. The blackboard is controlled by the manager, but all the participants can see everything that is on the blackboard, which in this case are the encrypted bids. Any intrusion can be detected, so after publishing the bids on the blackboard, the outcome of the auction cannot be affected anymore. Not even by the auction manager. In this first phase, the auctioneer is not yet capable of opening the bids because he does not yet have the necessary keys.

The second phase starts after the submission deadline. Each bidder can view the encrypted bids on the blackboard. The auctioneer cannot alter bids or add fake bids. Now the bidders all (securely) send their keys. The auctioneer opens the bids and determines the winner and the second-highest bid.

This procedure makes sure that the auctioneer is incapable of changing bids or adding fake bids. He is also incapable of understating the selling price as long as the bidders are not collaborating with him. For instance, if the auctioneer declares the third highest bid to be the

Casus 10.2: The bidding matrix

The n bidders submit k binary values, denoting their bid. Each of these values is encrypted with a different key K_{ij}. The function $e(b, K)$ encodes the bid.

selling price, the second highest bidder can prove that this is not true by supplying his key. However, if this bidder does not want to clarify the auction outcome, there is no way to detect the understated selling price. A collusion of an auctioneer and one or more bidders can thus result in an incorrect outcome of the auction.

Another problem is that the protocol is interactive. Agents can refuse to give their keys and so slow down or even stop the auction process. To assure a smooth auction process, uncooperative bidders need to be fined. If fines are not feasible due to anonymity, a default bid can be assigned to these bidders. The problem of the auctioneer collaborating with bidders is solved in the next section.

Binary bid search on a bidding matrix

In the previous section it was shown that the auctioneer is unable to alter the outcome of the auction by making up fake bids. We still need to prove though that the price he declares is actually the second-highest bid. We also still want to limit the amount of information the auctioneer possesses. This is done as follows. The procedure looks a lot like what was seen in the distributed keys protocol. We make the bid interval discreet, creating a bidding list with k possible bids: $S = \{p_1, \ldots, p_k\}$. Each bidder submits a bidding list that consists of k binary values denoting whether he is willing to pay a given price or not. Each bit in the list is encrypted using a different, arbitrary key K_{ij} generated by the bidder. These bidding lists are put together to form the so-called bid matrix (see fig. 10.2). This bid matrix is published on the blackboard. The functions $e(b, K)$ and $d(b, K)$ encode and decode a bid b with key K. The bid columns are presented on the blackboard in random order to assure privacy of the bidders.

The goal is now to find an opening sequence that works in reasonable time and determines the winner and the amount of the second-highest bid, without revealing more information than necessary. While applying the rules of this sequence, the auctioneer requests key after key from the bidders and uses these to open elements of the bid-matrix until the outcome of the auction is clear. The auctioneer opens elements from the bid matrix one by one. After determining the outcome of the auction, the auctioneer publishes the keys necessary to prove to the participants that the outcome is correct (see fig. 10.3).

We will now see that there is a sequence that determines the outcome of the auction in reasonable time, while revealing nothing more than partial information about all bids, except

Casus 10.3: Graphical representation of the bidding matrix

The bidding interval runs from 0 to 75, with $k = 15$, so 15 possible bids. Each column represents a bid. The highlighted elements of the bidding matrix are the only elements that need to be published to prove the result of the auction. For n bidders, only $n + 1$ keys need to be published.

the second-highest one. Like a normal binary search, the binary bid search begins in the middle of the bidding interval. It opens consecutive bids until it has found two positive ones. In that case the row is finished and the binary bid search is called recursively on the upper half of the interval. If none of the bids was positive, the binary bid search is called over the lower half of the interval. If exactly one positive bid is found, the highest bidder was found and bids are opened in a downward search, to determine the second-highest bid, without revealing any unnecessary information (see fig. 10.4).

Casus 10.4: The binary bid search algorithm

The search starts in the middle of the interval, opening consecutive elements of the bidding matrix. After opening 6 elements, two positive values are found and the search is continued in the upper half of the interval. Only bids that are unknown or might still be positive are considered. After 12 opened elements, the highest bid is determined and the search continues downwards linearly to determine the second highest bid without revealing unnecessary information.

By using this method, only partial information is revealed with a low precision. The execution time is acceptable and could be further decreased by starting at the expected value of the second-highest bid instead of in the middle of the bid interval. We have now achieved that the auctioneer cannot alter the outcome of an auction without being detected. The needed trust in the auctioneer was reduced enormously by only revealing partial information. The auctioneer is so reduced to a mediator that has no means of influencing the outcome of the auction, which is the way it should be. A downside of the given protocol is that is does require extensive (secure and anonymous) communication between the auctioneer and the participants.

10.4 Conclusion

It was seen that when we use the described protocol, the requirements to the auction are met:

- The dominant strategy assures that the price at which the bid was sold is the true value. Since the bidders have no reason to bid anything else than their true value, the price at which the good is sold is not artificially high or low.
- Even though the binary bidding search requires a time that depends on the price of the sold good, the execution of the auction still runs in reasonable time.
- The privacy and anonymity of the bidders is protected by sealing the bids, randomizing the columns on the blackboard and preventing the auctioneer from acquiring vital information about the bids.
- All bidders can verify the outcome of the auction, without the privacy or the anonymity being compromised.

It seems that on paper, the Vickrey auction is the best auction to use for the topics mentioned earlier: electronic commerce, automated resource allocation and task assignment. In practice however, this is not the case. Even though the set requirements can be met, other auctions are often preferred over the Vickrey auction. This is caused by the following reasons:

- The Vickrey auction is thought to lead to a lower revenue than for instance the English auction. In the Vickrey auction protocol, the bidders are cannot be distinguished from each other and are independent. In the English auction however, the bidders are not independent because they are actively bidding against each other. This results in a higher selling price, which is obviously preferred by the sellers, even though it does not fit the requirement of economic design, since the selling price is artificially high.
- In practice, a good is not just sold once, but it often occurs that the same good or task is sold a number of times. This makes the Vickrey auction vulnerable to collusions between bidders. Bidders can collude to find out that they are the highest bidders. In the next round of bidding they will then both make a lower bid, which results in the good or task being sold for a lower price than true value.
- The sequential rounds of bidding on the same good or task have another result. Antisocial agents can do their evil work once more. The second-highest bidder can with each round of bidding slowly bring his bid closer to the winning bid. In the example of task assignment, this would have as a result that an antisocial bidder slowly decreases the

profit of his competitor to nothing. Another approach would be to bid 0 in the first round of bidding. In the example of task assignment, the antisocial bidder would still be paid the second-highest bid thus lose only a limited amount of money, but he would also find out the second-highest value, which in this case would have been the value that would have won the auction normally. He now knows the true value of the winning bidder and can use this information to destroy his competitor's profit, as described earlier.

We have seen that the properties of the Vickrey auction allow an implementation of a fair auction, which executes in reasonable time. The goods are sold for their true valuation and privacy and anonymity exists for the bidders. These are the reasons for the Vickrey auction's popularity on paper. In practice however, the people controlling the auction care more for a higher profit than whether the price is right or not. The bidders in the auction aren't perfect themselves either. Their goal often is not just their own benefit, but also the ruin of their competitors. Until these disadvantages can be prevented, the Vickrey auction's popularity is doomed to exist only on paper.

Chapter 11

Electronic banking

Written by *Els Maes*

Who could have thought hundreds of years ago that money could be taken out of a wall? And that there would be things called computers that would take over the services of banks? And this even through a sort of rope that is connected with another computer that is said to be the bank? Through history a lot of things have changed.

Beginning of existance:	Goods are bartered for other goods
650 B.C.:	Existance of "money"
17th century:	First real banks are established
1967:	First cash dispenser becomes operational
End of 20th century:	First public electronic banking systems originate
Now and future:	New inventions

Both banking and technology have made a big evolution throughout the centuries, and they are still evolving. Nowadays banking fused with technology and brings forth electronic banking. No-one knows where it will end. One important factor of this electronic banking is the security factor. This is what will be discussed in this chapter. This discussion will be kept within the bounds of internet banking.

In Section 11.1 an informal description of e-banking is given, together with some characteristics. In Section 11.2 the pros en cons are considered of electronic banking. In Section ref-Maessecu the wishes concerning security are listed. In Section 11.4 the maintained security aspects are described. The comparison between the expectations and the reality will be made in Section 11.5. Section 11.6 will contain the conclusions.

11.1 Characteristics

Electronic banking is a feature to do banking via internet, telephone or mobile phone. E-banking is a service offered by banks. It is used to check balances, retrieve account history, do money transfers to an own account or to another account, etcetera. All actions that we used to do personally at the bank office can be done with the computer, even the printing of a bank account statement. Only getting cash and doing deposits is not possible over the internet (yet).

At this moment banks are offering more and more products over the internet, like insurances and stock exchanges. But it must be said that these parts are mostly used to obtain information on these products and not for the real purchase, this is done personally at the bank's office.

E-banking is becoming a common property: at this moment for about 20% of the Europeans do internet banking and the growth of this number is estimated to 37% yearly. Lots of people find it convenient to do banking via the internet: it means that they shouldn't leave the house anymore, and they can do it in between: for instance with one eye on the television and the other on the computer. They can also do their e-banking from somewhere else, for instance when working abroad. There are people though, who don't have any affection with computers, think of elderly people but also of people who aren't familiar with computers. They won't do their banking via the internet, but they might nevertheless use the telephone for it.

There are different implementations for electronic banking. Some implementations run only online and directly in a browser, with or without an applet. Some implementations can be run offline, where one can enter one's data without being connected to the bank's computer (possibly via the Internet) the whole time. And some implementations use a stand-alone application that handles the data-transfer with the bank its own way. This means that the bank needs to provide software to the user. The advantage with an applet is that the software is on the bank's computer and that it is automatically downloaded to the user's computer. If there are any updates, they are automatically installed. With a stand-alone application, the software has to be distributed to the users, which costs more time and money.

11.2 Pros en cons of electronic banking

Of course also this feature has advantages and disadvantages. There are always some positive and negative factors concerning electronic banking.

An advantage of e-banking is that it can be done from all over the world: when you're on holiday in China, and you think that you forgot to pay the phone bill, you can do it from over there. But you have to have the right tokens with you.

E-banking is faster than paper-banking. If you do paper banking, you have to bring the papers to the bank office or send them by mail, and the bank has to input them. That takes a lot of time. Sending them over the internet is faster, because the data from the user is directly inputed into the bank's system. Also the checking of the account is done automatically. Because the bank receives all transfers on computer they can check everything immediately and automatically without interference of humans.

What is thought to be a big disadvantage of e-banking is the security factor. It is not known if it is really a disadvantage, because it cannot be proven to be perfectly secure or to be not secure at all. People who don't use e-banking often take this as the major reason.

For the bank itself there are also some pros and cons. A drawback is the extra cost. Client software should be programmed and provided. Security aspects need to be fulfilled with the software. Helpdesks should be created for the users. Also a big risk exists for the bank. Banks are not keen to have their systems hacked which is a risk when going online. But on the other hand there can be personnel reductions in the bank offices. Less people will come to hand over transfers. There is no personnel needed to type in the transfers, this goes automatically from the users computer to the bank's computer. The account numbers and the amount of money can also be checked immediately. This means that the pros and cons for the bank are quite in balance.

This chapter handles the security of the electronic banking. As will be seen there are quite a few requirements to be fulfilled before a system can be labelled secure. What these requirements are and in which views they are of aren't fulfilled will be considered in the next sections.

11.3 Security wishes and expectations

As security is really important in this matter, all people want their money to be safe, the wishes and expectations concerning security are enumerated next. It gives a brief overview what is needed to make an electronic banking system secure. These requirements are also stated in [CDC+01].

- Authentication: A bank account should only be accessed by its true and rightful owner, so the bank has to be sure that the person who logs on really is who he says he is. On the other side, the user should be able to verify the identity of the counterparty. There has to be the possibility to check the authentication: people need to be able to check the identity of the bank and vice versa.
- Data authentication: the data that is received has to be the same as the data that is sent. The bank and the client should be able to detect all kinds of changes: insertion, deletion, substitution, etcetera. This requires a secure connection: if the connection cannot be tapped off, it is less likely that the data has been changed.
- Privacy and confidentiality: People want no-one else to be able to see their balance, except for the bank servant and other authorized persons.
- Non-repudiation: The bank has to be able to prove to a third party that a client has done a certain transaction. Due to an individual verification code a client cannot deny to have done that transaction.
- Protection of computers: The bank computer and the user's computer need to be protected from intruders. It may not be possible to hack those computers.

These are the requirements for a safe banking system. Banks as well as users would like to have all these requirements fulfilled entirely, but this is almost impossible considering the costs involved. In reality security is provided in the best possible way.

11.4 Security in reality

Of course we want to know what security there is implemented in reality. To learn this some research has to be done, for different banks the security levels for electronic banking are examined. There are two phases in the communication with the bank. The first one is the handshake part, and then data transfer takes place. At first there will be agreed upon the algorithms that will be used. These algorithms are used to protect data, to agree on cryptographic keys and to authenticate each other. In the second step the private and public keys that will be used for the purpose of data protection need to be set. Thirdly

the authentication between client and bank is fulfilled. After the handshake, data can be transferred between the two parties. All data will be encrypted.

The connection between the user and the bank should be secure. For this matter, SSL is used. SSL stands for Secure Socket Layer, it ensures that the connection between the bank and the user is secure. The SSL-protocol ensures the user that he is really communicating with a bank computer. SSL can only be used if there is a certificate, signed by a Certification Authority. Verisign [Veri] is such a CA. At Verisign it is possible to buy certificates. A CA will always check a company from top to bottom before they assign a certificate. They investigate the identity and the reliability of the requestor. A certificate can be seen as a digital passport and has a validity period. As a user receives information that is secured with a certificate then he can be sure of the identity of the sender. With SSL the data that is sent between the two parties is encrypted, on an asymmetric base. The bank has a public key that is provided in the certificate. The public key is a 1024 bit RSA key. So the data is encrypted using RSA over the secure connection.

Public key of Rabobank [Rabo]
3081 8902 8181 00ED A3EC 69CB 5496 3A27 FE3D 94FB 115D D8CE
F7BB 1DB6 DB2F 5FF2 9863 97E8 AED7 2A05 3519 229F 9C00 CBDD
B101 4B7C 2105 7AEB 0834 2251 DBF7 8582 7598 068D 654B 8DDF
1BFC 26D1 1CDA 2C1C 57F5 E054 9C6F 8B59 A6B1 9AA8 9FED 9420
60F9 BEE3 3A89 834D 2DF9 F5B4 43CB 7847 E26C C7C6 B31F E93C
D0FE 29C6 7462 1A45 C1A0 D961 9500 0B02 0301 0001

A certification authority also has a key-id. This ID is used for other parties to make sure that the Certification Authority is a party that provides legal certificates. If such an ID wouldn't be available, then any crook could create certificates for a bank, and then forge transfers.

Next it will be discussed what kind of security is provided for the other requirements, like authentication and non-repudiation. There are many different banks that use different strategies. The first case is the case of a key pair [KBC]. To log on to the system a fixed password is used. With the installation of the user, a personal private key pair is created. This key pair, currently a 1024 bit RSA key, is saved on a disk. It can only be used in combination with the personal password. When a transaction is made, the transaction has to be signed, like a paper transaction. So as alternative for a written signature a digital signature is created. When sending a transaction the personal password must be typed in, and for really sending it over the connection the signature is calculated using the key pair.

In the next case, there is the use of hardware tokens: a digipass and a random reader [Rabo]. A digipass is a little calculator that can be used to create a signature. This calculator has some extra buttons: I for identification and S for sending. When logging on to the system the I-button is used. With this I-button a one-time password is calculated. This password has a limited validity, and expires after 36 seconds. This password is also bounded to the owner of the account and the digipass. With another digipass there will be calculated another password, even if it is calculated at exactly the same moment. This implies that in the digipass a personal key is hidden. When transferring money to another account, the send-button is used. When pushing the send-button the personal password is asked. For the calculation of the signature a zero-knowledge proof is used. After typing the password a challenge is given which is used to compute the signature. It is made sure that it takes quite

some time to do a transfer: a challenge is given and you have to wait for the calculation to finish; this way no-one will quickly do a transfer from someone else's account.

The random reader has the same features. For this token also the pin card is needed, that is put in the random reader and the pin code is formed to create a digital signature. With the random reader also the amount on the chipknip can be checked.

These signatures are bounded to time, so they are different every moment. The signature that is created is a one-time password. The use of such a password increases the security because if someone can see this password, he cannot do anything with it, because it changes every moment.

With the key pair and the tokens the signature is created using a hash algorithm called Secure Hash Algorithm. A hash algorithm is characterized by its irreversibility: the password or the private key cannot be calculated with the result of the hashing, and this is exactly what is needed with a signature.

There are also banks which use a list of passwords [Post]. The passwords are on a paper list. Every time a user wants to send a transfer, they take the next password on the list. This password itself is a digital signature. For logging on a fixed password is used.

Type of password	Logging on	Sending transfers
Key pair	ID-number, Fixed password	Digital signature (Fixed password + key pair + hash algorithm)
Digipass / random reader	ID-number, Digital signature	Digital Signature (Token + password + challenge + hash algorithm)
List of passwords	ID-number, Fixed password	Password of list

An additional form of data protection is that the bank's computer is protected by a firewall, so hackers get no chances. A firewall is a program that checks and logs all data entering the computer. If someone tries to enter the computer the firewall makes sure all doors are shut. This is not enough though; the user's computer should be protected by a firewall also. This is not that difficult to establish because most browsers have the possibility to use a firewall.

Although many users think of security as a matter for the banks, they do have a part in security either. This is one of the issues which makes theoretical security different from practical security.

11.5 Is it really secure?

For authentication the user needs to be sure of the identity of the bank. This is true if the certificate is checked every time when electronic banking is used. But most people don't even look once at the certificate, some because of ignorance, and others due to lack of interest. Lots of people don't know what a certificate is about and how to check the correctness. Some

people on the other hand, see the message concerning the certificate and think at that moment that it probably will be correct.

The bank wants to be sure about the identity of the sender, but this can be doubted to be true. If someone is able to figure out passwords then this might not be true. Intruders can make a look-a-like of the log-on-screen of the bank, and get the password this way. People will type in the password and an error message is given. For instance with the list of passwords: if a user types his password on such a screen, then the intruder can use this password to forge a transfer. This is only a risk if the password is no one-time password, with a one-time password the signature is changed with the time, and the signature is only valid for a certain amount of time, often 36 seconds is used as a time period.

In the case of a key pair, an intruder needs the key pair to do any harm. This key pair should be a file on a disk, and this disk should be hidden every time after the e-banking is finished. But most of the time, this file is on the hard disk, or the disk is near the computer. This makes it for intruders even easier. They only have to figure out the password and that can be done by looking over someone's shoulder or just search or look around well. People are very careless with their passwords. They stick a note on their computer screen, or write the password on the disk which contains the key pair.

In case of the digipass, the password of the user should be known: this password can be retrieved by looking over someone's shoulder. In case of the random reader, beside the password also the pin card is needed, so cheating gets more difficult.

The banks follow up on security issues, and if there are better algorithms on the market they will implement it if they feel it's necessary. Banks also attract specialized firms to try to hack their computers. This way they can find leaks in there security and fix them before anything serious happens.

Banks also help the users. By establishing some preventative rules they can protect their clients by getting into problems. If clients are online but there are no actions for a certain period of time, the online session will be closed automatically by the bank's computer. If the session stays open, intruders can easily get into the system. For instance if someone is in a public place and leaves the computer but the bank session is still open, an intruder can search through the user's bank accounts.

If the digipass or random reader is used with an incorrect password and this is done a number of times, the token is blocked. If a thief finds a digipass, he can do a brute-force attack by trying all passwords he can think off. By blocking the digipass he has only a very small number (often this number is set to 10) of attempts. The password of the digipass has a limited validity, because of the time-dependence. If a bank client is typing his password in at the digipass, and someone with bad intentions can see what the password is, he cannot use it later, because it has expired at that moment. Also every digipass is bound to a certain user, so a digipass cannot be used to log on to an arbitrary e-banking system. A random reader on the contrary can be used by any client, but with the pin card and code of that client.

Banks try to cover themselves against liabilities by mentioning in small characters in the contract that they are not liable for any sustained damage caused by problems in the security of the e-banking system.

In the past some problems have occured. Most problems are now solved because of the improvement of the technological aspects. It must be said also, that it is difficult to find many problems because banks are not very keen to spread them out in the open. The banks don't want to denigrate themselves.

11.5.1 Known problems and possible attacks

Early 2004, the Dutch Postbank [Post] had a problem: hackers sent mail to clients in which they asked clients to go to a certain webpage, and type there username and password in that webpage. This way crooks could achieve passwords of clients and they could get into their systems.

Also the stealing of the list of passwords is a problem. If someone breaks in and can find a list of passwords, he can do transfers from someone's account. At least if he also knows the username and the password. But this is often not so difficult. There are people who, for instance, always use their birthdate as their password.

There's of course always a risk of a brute force attack. A crook can try all passwords and that way find one that works. This takes though a lot of time. With one-time passwords it becomes a lot tougher to break the password, because every few seconds the password changes.

Attacks caused by impersonification are also a risk. Crooks can impersonate the bank and get information like passwords that way.

11.5.2 User responsability

Users have responsabilities too. Always the bank is aimed at when discussing security, but the user also has to make sure he's acting safely.

The user's computer has to be protected against viruses and hackers, by using virus protectors and firewalls. The user only can make sure this all happens, but the bank has the responsability to inform clients what risks there are and what can be done to prevent for those risks.

User's shouldn't open attachments that appear in e-mails from strangers. This attachments could contain viruses, trojan horses or worms that could make electronic banking unsafe. Years ago viruses are detected that send confidential informations like passwords to hackers, and unfortunately such viruses still exist.

It is also not a good idea to perform electronic banking when visiting an internet cafe or another public place. It is very easy to look over one's shoulder when he is typing a password. There is also the risk that the logging off might go wrong and that the bank account can still be accessed. A good advice is to never let the computer unguarded while e-banking.

The user has to be careful with passwords, tokens, lists of passwords, etcetera. Those should be kept private and not exchanged with other persons. People also have to be attentive that passwords should not be easy to guess: the names of beloved people or pets and birth dates are not the most safe passwords.

People have to check the secure connection. Make sure that the connection is safe and that a legal certificate is used.

Users also have to take notice of changes in their account history or balance that are weird. For instance that the balance has decreased with 1000 euros. Then they have to notify the bank and ask them to reverse the transfer and sort things out.

Security is a matter of all parties. The bank as well as the user have the responsability to increase the security to a high level.

11.6 Conclusion

Electronic banking is used world wide. As from all over the world bank accounts can be accessed, e-banking has to be secure. This security has to be provided in the means of passwords or other safety measures. There are quite some possibilities to protect data that is transferred from being eavesdropped. There are secure connections but also the aspects of authentication, privacy, confidentiality and non-repudiation. This can be provided using digital signatures that can be calculated in many different ways. The security of the connection can be obtained by using the SSL protocol which, in combination with a certificate, can make sure to a user that he is communicating with the bank.

The bank has quite some responsibilities in providing security, but not all responsability can be shifted upon the bank. Users can do some actions to to increase security. Also the banks make a consideration between best security and lowest cost. If they secure the electronic banking system thoroughly then it will cost a lot of money. Or they can forget all about security and that will cost less money. They choose a central aisle: they secure the most important risks, and such a way that the cost is realistic.

Electronic banking contains risks, but this is also the case with paper-banking. None of both is perfectly secure and they never will be. But at this moment the security of the electronic banking systems is reasonably well. I think it can be said that the use of a hardware token in combination with a challenge is quite safe. This system is really difficult to forge by crooks.

Banks do quite some effort to make it secure, and if users do the same, then a big step in the good direction is made.

Chapter 12

Group Signatures

Written by *Jaap Jan Nagel*

Group signatures are a variation to the theme of signatures. The difference is that with group signatures instead of a single user that can sign a document, a group of users can sign a document independently. The problems that group signatures give, and some methods to create a group signature are discribed in this chapter

12.1 Introduction

When we talk about group signatures we have a group with numerous members and a single group manager. For the group there is a single group public key (*gpk*), and each group member i of the group has his own secret signing key which he can use to produce a signature relative to *gpk*. The group manager also has a secret key (*gmsk*) which he can use to extract the identity of a group member given a signature.

There is no relation between group signatures and group cryptography. In group cryptography the entire or part of the group is needed to create a signature. This could also be reversed to create a group signature where a threshold or veto scheme is needed, so to sign a message we would need several members to combine their keys to create a key that can be used to sign a message.

This is however not what we want in the topic of group signatures. We want one group member to be able to sign a message on behalf of the group, which means that several methods have to be devised to be able to deal with traitors. In group cryptography the traitor could be pinpointed because he supplied a false key, which resulted in a false group key. In group signatures the traitor has to be pinpointed by a receiver of the signatures, because he determines that the message that was signed was incorrect, which means that the signer of the message is probably corrupt.

Group signatures were first introduced in [CH91], and several methods were described to create group signatures. Several methods to determine the quality of a group signature followed, and some of the results of these studies are summarized in this chapter.

In section 12.2 the basic properties of group signatures, and the requirements a group signature has to fulfil are described. In section 12.3 some methods of creating group signature

Casus 12.1: Group signatures for print servers

A company has several computers, and each computer is connected to a local network. Each department of the company has its own printer which is also connected to the local network. Only persons of a particular department are allowed to use the printer of that department. This means that before starting to print a job the printer has to be convinced that the user demanding the print job actually works for the department.
At the same time the company also demands privacy, which means that the name of the user that is demanding the print job cannot be revealed. If however at some point it is discovered that the printer has been used to often, the director of the company must be able to discover who misused the printer.

schemes are described, and the duality between Group Signatures and Traitor Tracing is described.

12.2 Basics

There are some basic properties that every group signature has to have. De properties that have to be fulfilled for a group signature were informally described in the previous section. These properties can be formally described in the following way:

- Messages can only be signed by members of the group, and only one group member is needed to sign a message.
- A receiver of a signed message can verify if the signature is a valid signature for that group, but cannot discover which person signed the message
- If necessary the signature can be opened by the group manager to reveal the identity of the person that signed the message

These properties define a group signature, but how can we define a good group signature. Over the years several requirements were described that a good group signature has to fulfil. There are properties like unforgeability, exculpability, traceability, coalition-resistance, no-framing, anonymity, and unlinkability. The problem with all these requirements is that they are redundant or overlapping. In [BMW03] it was attempted to bring this list down to as few as possible requirements, and this resulted in the following two requirements:

Full-Anonymity: It is computationally infeasible for an adversary who is not in possession of the group manager's secret key for opening, to recover the identity of the signer from a group signature, even if the adversary has access to the secret keys of all group members. The last part of course means that the signing functions have to be one-way.

Full-Tracability: There is no subset of cooperation group members (even consisting of the entire group, and even in possession of the group manager's secret key for opening) that can create valid signatures that cannot be opened, or signatures that cannot be traced back to some member of the coalition. Which means that in case of misuse the anonymity of a signer can always be revoked by the group manager.

These two requirements are much stronger then the known requirements of anonymity and tracability. Tracability just states that a corrupted group member can be identified, and anonymity just states that the identity of a member cannot be compromised by anyone outside the group. It is easy to see that the other requirements as stated above can also be fitted into these two requirements. Unlinkability for example states that it is impossible for anyone outside the group to link two messages to the same group member (even if he doesn't know who that group member is).

It was believed that these requirements were complete for group signatures, but just recently in [DTX03] it was described that these two properties aren't enough, and that two additional requirements are needed, namely leak-freedom and immediate-revocation. Formally these properties can be described in the following way:

Leak-freedom: It is impossible for a group member who signed a message to convince anyone that the signature was his, even if the signer is in possession of all other signers' secrets, except the group manager's opening key.

Immediate-revocation: It is impossible for a valid group member who is revoked at time t to generate valid signatures at any time $t' > t$.

One could say that leak-freedom is a property that should be included into full-anonymity, because leak-freedom only describes that not even the group member itself can revoke his own anonymity. Perhaps the definition of full-anonymity has to be expanded in this way, but in [DTX03] it was stated that the definition of full-anonymity is already too complex. Immediate-revocation is a property that could be seen as an addition to the Full-Tracability property, because it describes that the revocation by the group manager should be instant. These four requirements define a good group signature scheme, but finding such a scheme is difficult given these properties.

12.2.1 Adding members

In the field of group signatures we speak of static and dynamic groups . Static groups are groups that don't allow expansion of revocation, which makes them easier to define. The problem is however that when misuse is determined the group becomes useless, because we cannot expell the corrupted member from the group. This is why we want the groups to be dynamic. For a group to be fully dynamic it also has to be able to cope with the adding of new group members. The modifications can be separated in two steps. The first step is to make it possible to add members to the group. And the second step is to make it possible to remove members from the group.

To make it possible to add members to the group we first have to change some properties of the group manager. In the fixed situation the group manager is just one person, but to be able to add members we have to spit the tasks of the manager to two persons. These splits follow the two requirements that have to be fulfilled for a group signature scheme, full-tracability and full anonymity. The group manager key *gmsk* is split into two different keys: *gmsk* becomes the opening key and *gmik* is the issuing key. The opening key *gmsk* can only be used to open a signature to identify the signer, while the issuing key *gmik* can only be used to add members to the group.

The reason for this split is that otherwise if the manager has false intentions, he can generate new keys for non-existing members, and can use them unpunished. When there is a dispute, and a key is found that cannot be linked to a member of the group by the manager

with the opening key, he can now link the message to the issuing manager , so the property of full-tracability is kept in this way.

The property of full-anonymity must also be retained in this situation. This means that the issuing manager must not be able to determine who signed a specific message.

One could say that the problem of adding members to a group is now solved, but the reality is slightly more complicated. Suppose a member is added to the group at time t and he signs a message from time $t-1$. This problem can be solved by adding a timestamp to a message. Which means we have to have a function that can update a secret key so that it can be used in a timestamp following the current timestamp.

12.2.2 Removing members

The method of timestamping a signature can also be used when we want to remove members from a group, but this is again more complicated. It means that the use of the update function has to be restricted, which means that this function must not be public. The easiest solution is to put this function in the hands of the (issuing) manager. The consequence is that every member of the group has to make contact with the manager from time to time to update his key. If the group membership of a member is revoked his key will not be updated, and will be useless from that time on.

The problem in this situation is that we have fulfilled the property of revocation, but the property immediate-revocation is not fulfilled. Even if the keys are updated every day, which is already very time-consuming for the group manager, it still is not possible to immediately revoke the membership of a corrupt group member.

Because the removing of members is very costly because of the reissuing of keys, fully dynamic groups should only be used if it is absolutely nessessary for the application, and in all other situations it should probably be enough that the group is just incremental. This means that the *Update* operation stays property of every group member, but the consequence is that in case of revokation of a group membership the entire group has to be reset.

12.2.3 Main Procedures

In the process of defining group signatures different procedures can be defined. A group signature scheme is made of seven procedures as stated by [KY03], that are executed by the active participants of the system, which are the Group Manager and the users. These steps can be identified as follows:

Setup: The initialisation of the group signature system by the group manager. A public key (*gpk*) is produced, and a private string (*gmsk*) used for the user key generation.

Join: The Group Manager uses his *gmsk* to create a secret key sk_i for user i.

Sign: Given the private key sk_i of user i and a message, a signature is generated.

Verify: Given the public key *gpk* and the signature of a message, the validity of a signature kan by verified.

Open: Given a signature and the *gmsk* of the Group Manager the identity of the signer can be found.

Revoke: The Group Manager uses the id i of a group member to revoke his membership.

Update: This method is used to update a secret key sk_i to the current timestamp.

Casus 12.2: Group signatures in a company

For another possible application of group signatures we can look at a company where employees have to validate price lists, press releases, or digital contracts on behalf of the entire company. In this case, the boss can set up a group signature scheme, and act as the group manager. Then the employees can sign or validate various documents on behalf of the entire company. By using this approach, the boss will conceal the company's internal structure, and the customers of the company would only have to know a single company public key to verify the signatures on your documents. Moreover, only the boss can determine which employee signed which document.

Initially the *Setup* and the *Join* procedure are executed at the same time, to create a group with members. This is no problem if the group size and content never change, but when new group members have to be added or group members leave the group (in case of misuse for instance) this will not suffice. This is why there is an update procedure that is invoked by the *Join* and the *Revoke* procedure or by a user, either to renew the membership of a user, or to make sure that the *Immediate-revocation* requirement is met.

12.3 Methods

There are several methods to create a group signature scheme. In this section we are going to discuss some of these methods. We start with a method that uses normal signature schemes to create a group signature scheme. This is a method for creating static groups. Next it is described how we can create group signatures using traitor tracing. Besides the method using ordinary signatures, no algorithms will be discribed, because most of the algorithms for group signatures are quite difficult, and giving a summary of different methods is not very instructive. Over the years several methods have been divised, and some of these methods are described in the mentioned articles: [CH91], [KY03],[BMW03] and [DTX03]. To create a bit of insight in the development of group signatures two methods are described.

12.3.1 Using Signatures

This method is based on a public key system. The group manager chooses a system, and gives each member of the group a disjunct list of secret keys generated with the algorithm. Next he publishes all the public keys corresponding with the secret keys.

Now each person in the group can sign a message using a secret key from his personal list, the recipient can verify the signature using the corresponding public key. If each key is used once, then it is impossible to link two messages to the same user. Also if necessary the group manager can open a signature, because he knows in whose possession a particular secret key is, which means that if a message is signed using that secret key, the group manager can easily verify who signed it.

There are several problems with this solution. The first problem is that a member can only sign a fixed amount of messages, because a secret key can only be used once to prevent linking of messages. This can be solved by distributing new key pairs from time tot time, but this makes managing the group very labour intensive.

The biggest problem is that the complete scheme becomes useless when a corrupted group member is identified. Because there is no way to remove members from a group, the only way to revoke a group membership is to disable all public keys and setup a new scheme without the imposter.

When we look at the basic properties we see that some of them cannot be fulfilled. The requirement of full-anonymity cannot be fully fulfilled because an adversary in possession of all the secret keys of the group members can sign the signed message with all the keys and can compare to find out which key was used. This can only be made computationally infeasible if the amount of distributed keys is very big. Because every pair of keys is created independently it is infeasable for group members to work together to create a key whose public key is on the distributed list, but is not a key of one of the cooperating members. This means that the requirement of full-tracability is fulfilled in this scheme.

Leak-freedom cannot be fulfilled in this scheme, because a group member can simply give the key he used to sign the message. The receiver can then verify if the key he distributed is indeed the key that was used to sign the message. On the other hand the receiver cannot be entirely sure that the member indeed signed the message because the group member could have obtained the key in an illegal way. immediate-revocation is also difficult because as stated above the entire scheme is worthless when a traitor is identified.

12.3.2 Group Signatures using Traitor Tracing

Defining a good group signature system is not easy and it could be made easier if we could find a system which we can use to derive a Group Signature system. In the field of ordinary signatures this was easy, because there exists a dual system for signatures which we can use. With ordinary signatures we could use public key encryption in the opposite way. We took the message M we wanted to sign, and use the decryption function with a private key to obtain $Sign(M)$. Everyone who has the public key can now use the encryption function on $Sign(M)$ and if the result of the encryption function gives the same message as M we can confirm that the message is properly signed. The process of defining a group signature can be simplified if we can find a similar dual for group signatures. Traitor tracing schemes can be used for this purpose.

When you tell a secret to one person, and the secret is compromised, then it is not difficult to determine who is guilty of compromising the secret. If the secret however is told to several persons this is complex. The same problem arises when we have data that should be available only to a specific group. An example of such use is a subscription to a movie channel like *Canal+*. The person who is guilty of compromising the "secret" is often called a traitor, so the objective is to trace who the traitor was when we find out that the secret has been compromised.

In a traitor tracing scheme three types of users can be defined: the authority, the users and the senders. In an earlier section we described that a group signature scheme also consists seven procedures, a traitor tracing scheme exists of seven procedures, as stated by [KY03], which are defined as follows:

Setup: The initialisation of the traitor tracing system by the authority. A "public" key (gpk) is produced, and a private string ($gmsk$) used for the user key generation.

Join: The authority uses the $gmsk$ to create a secret key sk_i for user i.

Encrypt: Given the public key gpk of the senders the message is encrypted.

Casus 12.3: Example use of traitor tracing

On the internet several websites can be found where mp3 music can legally be downloaded (after paying a fee). The music companies are more and more encouraging this method of distributing their music, but there is a problem. A person who legally downloaded a song can now share this song using illegal sharing communities like Kazaa. This is why the music companies want to be able to pinpoint who distributed the song, so traitor tracing has to be used.
We watermark an mp3 with a different watermark per each transaction with a user. So each user gets a unique mp3 of the same song. When we find an instance of our mp3 on a P2P network we can check it for watermarks. If any of the watermarks we retrieve match the watermarks we assigned users, we will be able to determine who the original traitor of the program was.

Decrypt: Given the private key sk_i of user i and a the encrypted message, the message can be decrypted

Trace: Given the contents of a pirate decoder and the *gmsk* of the authority the identity of the traitor can be found.

Revoke: The authority uses the id i of a subscriber to revoke his membership, and all pirates that have been using the key of the user are also revoked

Update: This method is used to update a secret key sk_i to the current timestamp.

When we compare the seven procedures of both schemes, we see that they have a lot in common. The *Setup*, the *Join*, the *Revoke* and the *Update* are exactly the same. The decryption can be used to sign a message, and the encryption can be used to verify a message. In the same way the *Trace* procedure can be used to *Open* the scheme to identify who misused the system.

This means that every scheme that is used for Traitor Tracing can easily be adapted to be used as a Group Signature scheme. The advantages are obvious, but some attention has to be given to the requirements of a group signature that were stated earlier. Full-tracability and immidiate-revokation are just as important for group signatures as for traitor tracing. The requirements of full-anonimity and leak-freedom are irrelevant for traitor tracing, but very important for group signatures, so when adapting a Traitor Tracing scheme first some attention has to be given to these two properties, to see if they are fulfilled. Full-anonimity is irrelevant for traitor tracing because a user only uses his own secret key sk_i to decrypt encrypted messages for his own use, so an adversery would have no use in learning which i corresponds with a sk_i.

12.4 Summary and Conclusion

Group signatures are used if a group of persons want to sign messages, where it has to look to the outside that they signed it as a collective, meaning that not a certain employee (or group member) of a company (or group) signed the message, but that it was signed by a company employee. However the (group) manager of the company would in some cases want to know who signed a particular message, so he has to be able to *open* a signature.

There are some basic requirements that a group signature has to fulfil: full-anonimity and full-tracability. The requirement of full-anonimity has the purpose to make sure that nobody from the outside a group can link a message to a member of the group. Full-tracability has the purpose to make sure that no group member (or group of colluding group members) can create signatures that are valid but cannot be traced to the actual signer.

There is a relation between public key encryption and signatures, which makes it easy to transform algorithms that can be used for encryption to an algorithm that can be used for signatures. This duality also exists between group signatures and traitor tracing, most of the procedures used for traitor-tracing can be transformed one-on-one to a procedure for group signatures.

Many articles have been writen on the subject of group signatures, but the use of group signatures is not very common yet. The degree of anonimity that a group signature takes care of is not nessesary in most situations. The four requirements that were stated for group signatures have made the creation of a scheme that can fulfil all these properties very difficult, and maybe some time has to be spent to determine if the requirements are in fact as important as they are stated to be. A lightened down version of the requirements, might not have a too big impact on the security, but will make the creation of a group signature scheme much easier.

Chapter 13

Micropayments

Written by *Shay Uzery*

In this chapter we will present the concept of *micropayments*, their special characteristics and the motivation that stands behind their creation. We will also describe several micropayment schemes, and show how they handle the different problems that face payment schemes in general and micropayments schemes in particular.

A *payment scheme* is a set of protocols that enables transactions and payments. In the general case, it involves three sides: a buyer, a merchant and a bank.[1] Current payment schemes use cryptographic signature methods in order to achieve a high security level, and rely on some centralized bank that has to be involved in all of the transactions and computes or verifies these signatures. Most of these schemes also rely on credit card charging. The *SET* [SET] (Secure Electronic Transaction) protocol is an example for such a scheme. This protocol was produced by the credit card companies *Visa* and *MasterCard*, and its purpose is to handle transactions over networks. In this protocol, when a buyer wishes to use his credit card in order to pay a merchant for his merchandise, the merchant will use *SET* in order to get an on-line authorization for this transaction.

As a simple example of payment scheme we can name the electronic checks. In this sort of payment scheme, when the buyer decides to make a transaction with some merchant, he will pay the merchant by digitally signing a check that contains all the relevant details that describe the transaction. The merchant will send the check to the bank to be deposited. The bank has to verify the authenticity of the check and that it is the first time that it is deposited. If this is the case, the bank will credit the merchant with the amount the check carries and will charge the buyer with the same amount.

Micropayment schemes are payment schemes that deal with payments of small amounts. Typical micropayments are worth one to ten cents. Because of these small amounts that micropayments carry, the schemes that implement micropayments need to be of a very low cost. Therefore, we must abandon the approach of using a centralized bank that has to compute cryptographic signatures for every transaction, and adopt a new approach. As we will see in the following sections, different micropayment schemes adopt different approaches, which are usually tightly connected to the application and environment in which these schemes are used. We will see that some schemes under some assumptions choose to use symmetric key

[1] We would like to mention that in the term "bank" we refer to any financial entity that is responsible for issuing and cashing of the electronic money involved in the scheme.

cryptography, while other schemes use cryptographic signatures only for a negligible number of payments and for the other payments use some other kind of cryptographic mechanisms. There are schemes that will continue using cryptographic signatures, but on the other hand require involvement of the bank only in a small part of the transactions. Nevertheless, all of these schemes offer a security level which is lower than the standard payment schemes which rely on a centralized bank and cryptographic signatures, and therefore it is not likely that micropayments schemes will replace standard schemes when it comes to high payment values.

It is expected that micropayments will have a great importance in the development of electronic commerce. Applications in which micropayments can be used include paying for each web page visited, paying for each minute of music or video, pay-per-view TV, etc.

Micropayments could be implemented by electronic checks, since the merchant and the buyer can digitally sign the check during the transaction. The reasons for not doing so are as follows:

- As we already mentioned, standard electronic payment schemes complexity is high due to complex cryptographic protocols, like digital signatures. These protocols are used in order to achieve a high security level. In this context it is important to keep in mind that micropayments are payments of small amounts. Therefore, extreme security is not required, and we would like our payment mechanism to be fast and simple.

- It is likely to be the case that the bank's processing cost of the payment will be larger than the value of the micropayment value itself.

There are two main assumptions that are common to all micropayment schemes and guide their design. First, it is assumed that merchants do not have an economical incentive to commit fraud since the micropayments worth only several cents, while the merchant's reputation and the economical value for him to continue using the scheme for trade also in the future, is worth much more. In addition, in most of the schemes it is assumed that the bank demands from every participant in the scheme, whether it is a buyer or a merchant, a deposit when he first joins the system. This participant will get back his deposit when he leaves the system, but only after proving that he did not commit any frauds and that he does not leave debts behind.

Second, in some systems people are willing to take the risk of having some portion of fraud and uncollected transactions, if it costs less than trying to prevent these fraud cases by using security measurements which are heavier and more expensive. Banks and other big financial institutes are used to managing these kind of risks. An example for such a system is the credit card system.

The implications of these two assumptions on the design of micropayment schemes is in the security level that these schemes will consider to offer. Micropayment schemes do not try to prevent all fraud by using very secure cryptographic mechanism, which are expensive and slow, but rather try to make fraud economically unprofitable.

We will turn now to describe several micropayment schemes. We will start (section 13.1) with some background about two basic micropayment schemes, *hash-based micropayments* and *Rivest's electronic lottery scheme*. These schemes are used by some of the advanced schemes we will present in later sections. Then (section 13.2) we will discuss the *MR1 scheme*, which relies on *Rivest's electronic lottery scheme*. In section 13.3 we will examine *PPay*, a micropayment scheme for peer-to-peer applications. This scheme also makes use of *Rivest's electronic lottery scheme*. In section 13.4 we will present Park, Boyd and Dawson micropayment scheme for wireless systems, which builds on and improves the *SVP* scheme (section 13.4.1), and makes use of *hash-based micropayment* scheme. Finally, in section 13.5 we will give a summary of the schemes presented in this chapter and some conclusions. We will also present some general

arguments that stand against micropayment schemes, which have to do with non-technological issues.

13.1 Basic Micropayment Schemes

We will start our presentation of micropayment schemes by introducing two basic schemes that some of the later schemes use or build on their ideas. The first is *hash-based micropayments*, and the second is *Rivest's electronic lottery scheme*.

13.1.1 Hash-Based Micropayments

As we already mentioned, digital signature schemes are computational expensive and create a burden on the system. In order to reduce this computational complexity, some of the micropayment schemes use hash functions, which require much less computation power. For example, on a typical workstation it may take half a second to compute an *RSA* signature, while in this time 10,000 hash functions can be computed.

Therefore, by using hash functions the merchant can get high-rate of verification of micropayments without needing heavy computing resources. This property is of great importance if we want to keep the costs of collecting and processing huge number of micropayments as low as possible, so the whole scheme is kept profitable. In addition, using hash functions instead of digital signatures makes it possible to implement these schemes in smart cards, which are already a common mean of payment, but which do not have the computational capability to use digital signature schemes.

Hash-based micropayment schemes are an example of schemes that still use cryptographic signatures, but only for a small portion of all transactions (the root of the *H-chain*), and then use some other cryptographic mechanism, which is computationally cheaper (secure one-way hash function), in order to provide security for all other payments.

We will turn now to the description of the Hash-based micropayment scheme.

Let H be a computationally secure one-way hash function (i.e. easy to compute and hard to invert).

The buyer decides on a value X, which is called the *root* of the *H-chain* $T_n, T_{n-1}, ..., T_0$, that is computed according to the following rule: $T_n = X$ and $T_i = H(T_{i+1})$, for every $0 \leq i \leq n-1$. The values $T_1, ..., T_n$ are called *coupons* and they all have the same value v. The buyer can use these coupons in order to make (at most) n micropayments with the same merchant. When the buyer decides to make the first payment to a certain merchant, he sends the merchant T_0 with his digital signature on it. By that the buyer commits himself to the entire chain. The buyer will make the next payments (to the same merchant) by sending the merchant the following coupons, one coupon for each payment. The merchant verifies the validity of these payments by checking that $H(T_i) = T_{i-1}$.

13.1.2 Rivest's Electronic Lottery Scheme

Rivest's Electronic Lottery Scheme [Riv97] is an example to micropayment schemes that continue using cryptographic signatures, but require involvement of the bank only in a small part of the transactions. In the *Lottery scheme*, the buyer and the merchant decide for each micropayment if it is selected or discarded according to some selection rate. Micropayments are selected with probability $s, (0 \leq s \leq 1)$, and are worth $1/s$ times more the original amount.

The implementation of this scheme uses *H-chains* both for the merchant and buyer. The merchant sends the buyer the coupon $w = w_0$ of his *H-chain*: $w_0, w_1, ..., w_n$, where $w_i = H(w_{i+1})$, for $0 \leq i \leq n-1$.

The buyer signs both w and x_0, and by that commits himself to both *H-chains*. Micropayment x_i is chosen if $(x_i \bmod s)$ is equal to $(w_i \bmod s)$.

There are two main drawbacks that we would like to mention. The first drawback of the *Lottery scheme* is that the buyer and merchant are obliged to interact every time that a micropayment is done, which slows down the scheme. This problem increases as the system grows in size, and the merchant has to handle more and more buyers simultaneously. Another drawback of this scheme is that it might be that a certain buyer has to pay more than the value of transactions he actually did. This situation can happen when more than one payment is selected out of $1/s$ payments of a certain buyer. Because every selected micropayment is worth $1/s$ times more than the original amount, this buyer has to pay for more than the $1/s$ payments that he actually had. The probability that such a situation occurs decreases as the buyer makes more payments, but still this problem can deter buyers from joining such a scheme.

13.2 The MR1 Scheme

The *MR1 scheme* is due to Rivest and Micali [MR02]. This scheme keeps some of the ideas of *Rivest's Lottery scheme*, which we presented in the previous section, and fixes some of its drawbacks.

In the *MR1 scheme* we again have a buyer that wants to buy some service or merchandise from a certain merchant by sending him a check C. Check C is computed by using the variable T, which contains all the relevant details that describe the transaction. Like in the *Lottery scheme*, a micropayment is chosen with probability $s, (0 \leq s \leq 1)$. In the *Lottery scheme* we saw that the buyer and merchant had to interact for every micropayment in order to decide whether to select it or not. The *MR1 scheme* will avoid this by letting the merchant to compute some value from the check C, which only he can compute. This value will be used to decide whether a certain micropayment is selected or not. Because only the merchant can compute this value, the buyer cannot know if a certain check will be selected or not, and therefore he cannot avoid sending checks that are supposed to be deposited.

We will turn now to describe the scheme. The scheme will make use of a function $F, F : \{0,1\}^* \rightarrow [0,1]$, which is a fixed public function.

Every new buyer or merchant that joins the system is given a (public key, secret key) pair, that will be used for digital signing.

When a certain buyer B decides to make a transaction T with a merchant M, he will send M a check $C = SIG_B(T)$. This check is selected if the inequation $F(SIG_M(C)) < s$ holds. If C is a selected check, then it is worth $1/s$ times the amount written on the check, and M can deposit it by sending C and $SIG_M(C)$ to the bank BA. The bank BA has to verify the validity of B's and M's signatures. BA has also to check that C is deposited for the first time. If this is the case, BA will update B's and M's accounts accordingly.

As we can see, the only set-up that is needed in the scheme is when a new party is joining the system. From this moment on, this party does not need to go through any other set-up steps in order to participate in transactions. This makes this scheme fast and efficient. In addition, and like we already mentioned, there is no interaction between the buyer and the merchant during the payment. This property again adds to speed and efficiency of the scheme,

and makes it more scalable since the merchant is no longer a bottleneck in this sense. Another observation that we can make is that B cannot avoid sending checks that will be selected. This is because he cannot compute $SIG_M(C)$. Since $SIG_M(C)$ is a random number we also get that the selection rate of the scheme is s. Finally, Bank B is involved only in a fraction s of the payments, which then have a macro value.

However, the *MR1* scheme still does not solve the second problem we mentioned in *Rivest's Lottery scheme*, meaning it is still possible that a certain buyer has to pay more than the value of transactions he actually did. In their article, [MR02], Rivest and Micali describe two other micropayment schemes, *MR2* and *MR3*, which solve this problem.

The *MR2* scheme shifts the risk of excessive payment from the buyer to the bank. One reason for that is that banks are much more accustomed to managing risks. The second reason is that the probability that such a situation occurs decreases as the number of payments grows. Since the bank handles not only one buyer but many, this problem becomes negligible.

The *MR3* scheme also shifts the risk of excessive payment from the buyer to the bank, in the same way that it is done in the *MR2*, and in addition it shifts the decision about the selection of payments from the merchant to the bank.

We refer the readers to [MR02] for a more detailed description of these two schemes.

13.3 Micropayments in Peer-to-Peer Systems

Peer-to-peer (P2P) applications enable sharing of all kinds of data - music, video, software, etc.,- distributed computation, data bases etc., by connecting large amounts of resources at the edges of the network, instead of using expensive centralized resources. *P2P* applications have become very popular in the last years, and maybe the most known of them is the *KaZaA* application that reported over 4.5 million users sharing a total of 7 petabytes of data.

As a result of legal pressure from the recording industry, which demands to get the royalties for the trade in its merchandise, new *P2P* applications for sharing multimedia files are developed, which consist mechanisms that enable peers to trade between themselves in an efficient and secure way. In this sense, peers behave both as merchants, who sell goods, and buyers, who purchase goods. Therefore, it is wise to use in these applications a transferable coin that can participate in many transactions before it has to be cashed by the bank. Another observation regarding *P2P* applications is that the peers offer a massive computational resource that can be used instead of having the bank carrying all the load of work.

On the other hand, using transferable coins delays the detection of fraud, and sharing the security operations with peers, which we do not know and therefore cannot say anything about the level of trust we give them, also raises some security problems and questions.

In this section we will present *PPay* [YGM03], a micropayment scheme that addresses the mentioned above problems. *PPay* limits the bank involvement only for specific cases, for instance when peers open or close accounts, while taking care that every case of fraud will be detected and that the misbehaving peer is found.

PPay is using the concept of *floating, self-managed currency. Floating currency* means that the currency can directly move from one peer to another while the bank is completely not involved in this process. In addition, *PPay* is using transferable coins that keep the same size regardless the number of times they are transferred.

Self-managed currency means that the peer who owns the coin is responsible for all security matters related to the trade using this coin. Pay attention that this property makes use of the computational resources available at the peers, and by that spreads the computational burden

that has to do with the security of the system, among all peers, rather than being dependent on a centralized bank.

As it is clearly seen, the concept of *floating, self-managed currency* greatly reduces bank involvement and burden.

As we already mentioned, peers behave both as buyers and merchants under *P2P* applications. Therefore, in this section we will name the peers users and not buyers or merchants. We will turn now to the description of the scheme's implementation.

A user U must have digital coins under his possession in order to be able to pay during transactions. If he does not hold digital coins, he can turn to the bank BA in order to buy new *raw coins*. These coins have the form: $C = \{U, sn\}SK_{BA}$, where sn is the unique serial number that identifies the coin. The user U is called the *owner* of the coin C.

U can use his coins in order to pay for a service or a merchandise he gets from an other user V. He does so by sending V an *assignment* to the coin C: $A_{UV} = \{V, seq_1, C\}SK_U$, where seq_1 is the sequence number of the assignment, and has to be greater than the sequence number of the previous assignment of this coin. V is now called the *holder* of the coin.

V can cash his assigned coins at the bank for some constant commission. This commission is intended to deter users from cashing coins too often since it damages the advantages of floating currency. On the other hand, V can use this coin in order to pay another user X. V will do so by sending the *reassignment request* to the *owner* U of the coin: $R_{UVX} = \{X, A_{UV}\}SK_V$.

U is obliged to keep a record of the reassignment request since it is the proof that V gave up on his assignment of the coin. U will then send to V and X the new assignment: $A_{UX} = \{X, seq_2, C\}SK_U$.

It is the responsibility of V and X to verify that $seq_2 > seq_1$.

There are several observations concerning this scheme that are worth mentioning. First, the computational burden on the bank is now reduced to $O(m)$, where m is the number of coins issued. This is a dramatic improvement to standard payment schemes, where the bank's burden is $O(n)$, where n is the total number of transactions made in the system. This improvement is achieved due to the concept of *floating currency*, which means that the bank is no longer involved in the transactions between two users, and is only involved when one of the users wishes to cash his coins.

Another observation is regarding the reassigning process of a coin, which seems to be quite expensive since it involves two cryptographic signature (one of V and one of U), four signature verifications, and three network messages. We cannot reduce the computational burden of each reassignment process, but we can reduce the total number of reassignments. This can be done by using, for instance, *Rivest's electronic lottery tickets* (see section 13.1.2), which will reduce the number of reassignments processed to a portion s of the total number of transactions. Of course now every selected transaction is worth $1/s$ times the original value.

We will address now the issue of security in *PPay*. *PPay* guarantees that every fraud case and the user that committed it will be discovered. To achieve this property, the scheme requires that the following invariants will hold at all time:

1. An owner U of a coin C cannot deny that an other user, H, is the holder of this coin in case that H is the a holder of C. Thus, when H tries to reassign or cash C, U cannot refuse this request. However, if H is not a valid holder of C, U does have a proof that H gave up on C. A proof in these two cases is the signed reassignment: $R_{UVX} = \{X, A_{UV}\}SK_V$.

2. The sequence number given to a coin increases with every reassignment of this coin.

PPay is using the mentioned above invariants in order to fight frauds. For instance, if a holder H of a coin C tries to spend it twice, first with user X and then with user Y, then according to the second part of the first invariant, the owner U of this coin will have a proof that H gave up on this coin, and U will refuse the second reassignment. Another possible fraud is when an owner U of a coin C tries to spend C twice. In this case, when this coin will arrive to the bank to be cashed for the second time, it is guaranteed that the bank will notice this irregularity. This is because the bank holds a record of every issued coin with its number and owner. In this case, U will have to give a proof for the invalidity of one of the reassignments.

13.4 Micropayments in Wireless Communication

In this section we will discuss another area in which micropayments can be used and implemented which is the wireless communication area. In the last years we can clearly see how wireless communication devices are strongly linked and make use of computing services, like the internet, messaging applications, E-commerce etc.

Public-key based security protocols for wireless communication users are not yet available, and it seems that it will take a couple of years before we see them in the market.

As an answer to this lack of technology, Stern and Vaudenay [SV97] came up with the *Small Value Payment (SVP)* scheme, which relies on tamper resistant devices at each buyer and merchant side, together with symmetric key cryptography. As long as public-key infrastructure is not available for wireless communication users, this assumption seems reasonable. Current mobile phones consist smart cards, which are considered as a tamper resistant device, and hardware that implements symmetric key cryptography, like the *A5* protocol.

In this section we will present a micropayment scheme by Park, Boyd and Dawson [PBD00], which improves the *SVP* scheme. We will start by giving a description of the *SVP* scheme.

13.4.1 Small Value Payment (SVP) Scheme

In the initialisation phase of the scheme, bank BA has to compute the following values:

- His own secret key k_{BA}. This value will be sent in a secure manner to the tamper-proof devices of all the merchants.
- Random value t.
- A spending key $s = H_{k_{BA}}(B, t)$ for every buyer B, where s has to be unique for every buyer, and H is a computationally secure one-way hash function. BA will send s and t in a secure manner to the corresponding buyer.

When buyer B wishes to perform a transaction with merchant M, he sends the message (B, t, r) to the merchant M, where r is some random number chosen by B. In return, M sends a random challenge, q, to B. B computes: $v = h(B, M, c, q, r, s)$, where h is a one-way hash function and c is a micropayment amount. B sends the value v to M. Now, all M has to do before accepting the payment is to verify that $v = h(B, M, c, q, r, H_{k_{BA}}(B, t))$. This process is a *zero knowledge proof (ZKP)* in which B proves to M his knowledge of the spending key s.

The *SVP* scheme has a number of drawbacks which make it impractical. First, because the buyer does not provide any signature the scheme does not provide non-repudiation. Second,

the secret key k_{BA} is shared among all the merchants, which is a weak point from a security point of view. And last, the buyer does authenticate himself to the merchant, but the merchant does not authenticate himself to the buyer.

13.4.2 Improved SVP Scheme

The *improved SVP* scheme builds on the *SVP* scheme and improves it by using *Hash chains*. In the new scheme we will have four secret keys, instead of only one we had in the *SVP* scheme:

- K_{BA} is a secret key that is known only to the bank.
- $K_{ID(B),BA}$ is a secret key between the buyer and the bank and is unique for every buyer. The bank computes this key as: $K_{B,BA} = H(K_{BA}, ID(B))$, and delivers it to the buyer.
- $K_{M,BA}$ is a secret key that is common between all the merchants and the bank.
- $K_{ID(B),M}$ is a key between the buyer and the merchant and is unique for every buyer. This key is computed by the bank as: $K_{ID(B),M} = H(K_{M,BA}, ID(B))$, and is delivered to the buyer. The merchant will compute this key in the beginning of every transaction, since its value changes from buyer to buyer.

The payment protocol starts with an initialisation phase, in which the buyer B sends a merchant M his identity, $ID(B)$, and a random challenge, r_B. When M receives this message he computes the common shared secret key $K_{ID(B),M} = H(K_{M,BA}, ID(B))$, chooses a random challenge, r_M and computes $h(r_B, r_M, K_{ID(B),M})$. M will then send to B the message: $(r_M, h(r_B, r_M, K_{ID(B),M}), ch\_data, TS_M)$, where *ch_data* is the fee that M charges for his merchandise and TS_M is a time stamp.

When B receives this message, he computes $h(r_B, r_M, K_{ID(B),M})$, and compares it with the value he received from M. By that M supplies a *ZKP* to B about the knowledge of the secret key $K_{M,BA}$. In the case that the two values are equal, B computes 2 commitments: $commitment_M = h(ID(B), M, K_{ID(B),M}, r_B, r_M, TS_M, ch\_data, c_0)$, and $commitment_B A = h(commitment_M, K_{M,BA})$. B sends M a message containing the two commitments and c_0, which is the root of B's *H-chain*.

The merchant computes himself the first commitment, and if it is equal to the value he received from B then it gives him a *ZKP* that the buyer knows the secret key $K_{ID(B),M}$.

After the payment initialisation protocol is done, the user can start making his purchases from the merchant. This part is done using *H-chains*, as we explained in section 13.1.1.

The merchant stores the payment data for later billing. When he turns to the broker to collect his payment, the broker verifies the $commitment_S$ field.

13.5 Summary and Conclusions

In the previous sections we described several micropayment schemes with the technological issues that each of them has to deal with. In section 13.2 we described some general micropayment schemes, which are practical and can be efficiently implemented. In sections 13.3 and 13.4 we described two micropayment schemes, which are specific to certain areas and applications, one for *P2P* applications and the other to wireless communication devices. From this discussion it seems that micropayment schemes are practical, and that they have an economical importance.

Nevertheless, there are still some arguments against micropayments that do not have to do with technology, but rather with economics, sociology, and psychology. We will present some of these arguments that were discussed by Odlyzko in [Odl03].

One of the claims against micropayments is that competing payment systems enjoy the same advances in computing and communications as micropayment schemes. These advances make competing payment systems economically valid for usage with smaller payment values. As a consequence, the market slice in which only micropayments schemes can be used is growing smaller.

In addition it is claimed, that changes in payment systems are quite slow. An example for that is the long process that took for credit cards to become popular. This slow process does not mean that micropayment schemes will not be adopted or used, but it does go against the expectations of fast acceptance of micropayment schemes.

In economics is known already for a long time that *bundling* approaches[2] are more profitable to merchants. In addition, one of the findings of behavioral economics is that customers prefer flat-rate plans on metered ones. This reasons contradicts again the ideas behind micropayments and the market to which micropayment schemes are intended.

Another claim against micropayments is that in an environment of low marginal costs, merchants are motivated to increase usage of their merchandise, and therefore all obstacles in the way of usage should be removed. The best way to increase usage is by using flat-rate prices or programs, and therefore it is likely that flat-rate programs will remain dominant.

[2] A bundling approach means offering several products as one combined product for a lower price than the accumulated price of all of these products bought separately.

Bibliography

[ABHL03] Ahn, L. von, Blum, M., Hopper, N., and Langford, J. CAPTCHA: Using hard AI problems for security. In Biham [Bih03], pp. 294–311.

[AHL00] Ahn, L. von, Hopper, N. J., and Langford, J. The CAPTCHA web page, 2000.
`http://www.captcha.net`.

[Bac] Back, A.
`http://www.cypherspace.org/adam/rsa/rc4.html`.

[Bih03] Biham, E. (ed.). *Advances in Cryptology – EUROCRYPT2003* (Berlin, 2003), vol. 2656 of *Lecture Notes in Computer Science*, Springer-Verlag.

[BK00] Biham, E. and Keller, N. Cryptanalysis of reduced variants of RIJNDAEL. In proc. *Proceedings of the Third Advanced Encryption Standard Conference* (2000), NIST.

[BM02] Batina, L. and Muurling, G. Montgomery in practice: How to do it more efficiently in hardware. In Preneel [Pre02], pp. 40–52.

[BMW03] Bellare, M., Micciancio, D., and Warinschi, B. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In Biham [Bih03], pp. 614–629.

[BS91] Biham, E. and Shamir, A. Differential cryptanalysis of Snefru, Khafre, REDOC-II, LOKI and Lucifer (extended abstract). In proc. *Advances in Cryptology – CRYPTO-91* (Berlin, 1991), J. Feigenbaum (ed.), vol. 576 of *Lecture Notes in Computer Science*, Springer-Verlag, p. 156.

[BS03] Blömer, J. and Seifert, J. Fault based cryptanalysis of the advanced encryption standard (AES). In Wright [Wri03], pp. 162–181.

[CDC+01] Claessens, J., Dem, V., Cock, D. D., Preneel, B., and Vandewalle, J. On the security of today's on-line electronic banking systems. Tech. rep., Department of Electrical Engineering, University of Leuven, 2001.

[CH91] Chaum, D. and Heijst, E. van. Group signatures. In proc. *Advances in Cryptology – EUROCRYPT* (Berlin, 1991), D. W. Davies (ed.), vol. 547 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 257–265.

[COR00] Information society, 2000.
`http://www.cordis.lu/ist/`.

[CP02] Courtois, N. and Pieprzyk, J. Cryptanalysis of block ciphers with overdefined systems of equations. In proc. *ASIACRYPT2002: Advances in Cryptology* (Berlin, 2002), Y. Zheng (ed.), vol. 2501 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 267–287.

[DR99] Daemen, J. and Rijmen, V. AES proposal: Rijndael, 1999.
`http://csrc.nist.gov/encryption/aes/rijndael/Rijndael.pdf`.

[DTX03] Ding, X., Tsudik, G., and Xu, S. Leak-free group signatures with immediate revocation, 2003.

[FMS01] Fluhrer, S., Mantin, I., and Shamir, A. Weaknesses in the key scheduling algorithm of RC4. In Vaudenay and Youssef [VY01], pp. 1–24.

[FOO93] Fujioka, A., Okamoto, T., and Ohta, K. A practical secret voting scheme for large scale elections. In proc. *Advances in Cryptology - AUSCRYPT '92* (1993), pp. 244–251.

[FS03] Ferguson, N. and Schneier, B. *Practical Cryptography*. John Wiley and Sons, 2003.

[FWS01] Ferguson, N., Whiting, D., and Schroeppel, R. A simple algebraic representation of Rijndael. In Vaudenay and Youssef [VY01], pp. 103–111.

[GW00] Grosul, A. L. and Wallach, D. S. A related-key cryptanalysis of RC4, 2000.

[Hey] Heys, H. A tutorial on linear and differential cryptanalysis.
`citeseer.nj.nec.com/443539.html`.

[How] Howard, M. RC4 usage errors leave your data exposed.
`http://archive.devx.com/security/bestdefense/2001/mh0201/mh0201-1.asp`.

[HQ00] Hachez, G. and Quisquater, J.-J. Montgomery exponentiation with no final subtractions: Improved results. In proc. *Cryptographic Hardware and Embedded Systems – CHES* (Berlin, 2000), Çetin K. Koç and C. Paar (eds.), vol. 1965 of *Lecture Notes in Computer Science*, Springer-Verlag, p. 293.

[KBC] KBC bank en verzekering.
`www.kbc.be`.

[KLS02] Kochanski, G., Lopresti, D., and Shih, C. A reverse Turing test using speech, 2002.

[KSF99] Kelsey, J., Schneier, B., and Ferguson, N. Yarrow-160: Notes on the design and analysis of the Yarrow cryptographic pseudorandom number generator. In proc. *Selected Areas in Cryptography* (Berlin, 1999), H. M. Heys and C. M. Adams (eds.), vol. 1758 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 13–33.

[KSWH98] Kelsey, J., Schneier, B., Wagner, D., and Hall, C. Cryptanalytic attacks on pseudorandom number generators. In proc. *Fast Software Encryption* (Berlin, 1998), S. Vaudenay (ed.), vol. 1372 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 168–188.

[KY03] Kiayias, A. and Yung, M. Extracting group signatures from traitor tracing schemes. In Biham [Bih03], pp. 630–648.

[Man01] Mantin, I. Analysis of the stream cipher RC4. Master's thesis, The Weizmann Institute of Science, 2001.

[MM03] Mori, G. and Malik, J. Recognizing objects in adversarial clutter: Breaking a visual captcha, 2003.

[Mon85] Montgomery, P. L. Modular multiplication without trial division. *Mathematics of computation* **44** (1985), 519–521.

[MR02] Micali, S. and Rivest, R. L. Micropayments revisited. In Preneel [Pre02], pp. 126–134.

[Nes00] New european schemes for signatures, integrity and encryption, 2000. `http://www.cryptonessie.org`.

[NSS91] Nurmi, H., Salomaa, A., and Santean, L. Secret ballot elections in computer networks. *Computers and Security* **10** (1991), 553–560.

[Odl03] Odlyzko, A. The case against micropayments. In Wright [Wri03], pp. 77–83.

[ODR02] Oswald, E., Daemen, J., and Rijmen, V. AES - the state of the art of Rijndael's security, 2002. `www.a-sit.at/technologieb/evaluation/aes_report_e.pdf`.

[PBD00] Park, D., Boyd, C., and Dawson, E. Micropayments for wireless communications. In proc. *Information Security and Cryptology* (Berlin, 2000), D. Won (ed.), vol. 2015 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 192–205.

[Post] Postbank. `www.postbank.nl`.

[Pre02] Preneel, B. (ed.). *Topics in Cryptology – CT-RSA2002* (Berlin, 2002), vol. 2271 of *Lecture Notes in Computer Science*, Springer-Verlag.

[Rabo] Rabobank. `www.rabobank.nl`.

[Riv] Rivest, R. RSA Security response to weaknesses in key scheduling algorithm of RC4. `http://www.rsasecurity.com/rsalabs/technotes/wep.html`.

[Riv97] Rivest, R. L. Electronic lottery tickets as micropayments, 1997.

[Rob94] Robshaw, M. Block ciphers. Tech. rep. TR - 601, RSA Laboratories, 1994. `citeseer.nj.nec.com/article/robshaw95block.html`.

[Roo95] Roos, A. A class of weak keys in the RC4 stream cipher, 1995.

[RSA] RSA Laboratories' frequently asked questions about today's cryptography. `http://www.rsasecurity.com/rsalabs/faq/index.html`.

[SET] Secure electronic transactions.
`http://www.setco.org`.

[SIR01] Stubblefield, A., Ioannidis, J., and Rubin, A. Using the Fluhrer, Mantin, and Shamir attack to break WEP, 2001.
`citeseer.nj.nec.com/stubblefield01using.html`.

[SV97] Stern, J. and Vaudenay, S. SVP: a flexible micropayment scheme, 1997.

[Tel02] Tel, G. *Cryptografie: Bescherming van de Digitale Maatschappij*. Addison-Wesley, 2002 (363 pp.).

[Tur50] Turing, A. Computing machinery and intelligence. *Mind* **59**, 236 (1950), 433–460.

[Veri] Verisign.
`www.verisign.com`.

[VY01] Vaudenay, S. and Youssef, A. M. (eds.). *Selected Areas in Cryptography* (Berlin, 2001), vol. 2259 of *Lecture Notes in Computer Science*, Springer-Verlag.

[Wik] Wikipedia. Orders of magnitude (numbers).
`http://www.wikipedia.com`.

[Wri03] Wright, R. N. (ed.). *Financial Cryptography* (Berlin, 2003), vol. 2742 of *Lecture Notes in Computer Science*, Springer-Verlag.

[XH02] Xiao, L. and Heys, H. Hardware design and analysis of block cipher components, 2002.
`citeseer.nj.nec.com/xiao02hardware.html`.

[YGM03] Yang, B. and Garcia-Molina, H. Ppay: Micropayments for peer-to-peer systems, 2003.

Index