Cryptography in Action

Cryptography in Action

Gerard Tel (Editor)

Universiteit Utrecht

Instituut voor Informatica en Informatiekunde
Universiteit Utrecht

Preface

This syllabus contains papers discussed at a small sympusium called *Cryptography in Action*, held on January 30, 2005, at Utrecht university. Presentation at the symposium and contribution to this syllabus were obligatory requirements in the course *Cryptography* of the Master of Algorithmic Systems.

Sixteen presentations were given on the Symposium, arranged in three tracks as indicated in the schedule:

Cryptografie in Actie

Minisymposium 31 januari 2005, BBL 4e verd.

Thema:	Fundamenten	Ontwerpen	Systemen
Locatie en Begeleider:	BBL-416, Erik Jan van Leeuwen	BBL-420, Thomas Wolle	BBL-426, Marjan van den Akker
13.00 - 13.05	Openingstoespraak door Gerard Tel in de hal van BBL4		
Slot 1: 13.05 - 13.30	Johnson Leow: Quantum cryptografie	Maarten van Steenbergen: Online betalen	Jan-Pieter vd Heuvel: C2000
Slot 2: 13.35 - 14.00		Michael Beye: Block cipher design	Meindert Kamphuis: Compressie en Cryptografie
Slot 3: 14.05 - 14.30	Johan van Rooij: Bilineaire functies	Jules van Kempen: Cryptografische modes	Nelleke Steendam: KARMA
Slot 4: 14.35 - 15.00	Petra Rutgers: Provable security	Teun Slijkerman: Steganografie	Jeffrey van Helden: Content Protection
Slot 5: 15.05 - 15.30	Marco Streng: Factorizatie	Wouter Slob: Elektronisch Patientendossier	Nico Munting: Blue Ray
Slot 6: 15.35 - 16.00	Eric Ho: Improving makes it worse		Gerben Oostra: Bluetooth
	Slottoespraak door Gerard Tel en afsluitende verversingen		

The track on *Foundations* was chaired by Erik Jan van Leeuwen and contained the presentations of Johnson Leow, Johan van Rooij, Petra Rutgers. Marco Streng, and Eric Ho (Chapters 1 trough 5). The track on *Designs* was chaired by Thomas Wolle and contained the presentations of Maarten van Steenbergen, Michael Beye, Jules van Kempen, Teun Slijkeran, and Wouter Slob (Chapters 6 through 10). The track on *Systems* was chaired by Marjan van den Akker

and contained the presentations of Jan-Pieter van den Heuvel, Meindert Kamphuis, Nelleke Steendam, Jeffrey van Helden, Nico Munting, and Gerben Oostra (Chapters 11 through 16).

At `http://www.cs.uu.nl/~gerard/FotoAlbum/F2005/Uithof1` a few photos of the symposium can be seen.

This photo shows most of the speakers. From left to right, they are Jeffrey van Helden, Nico Munting, Jan-Pieter van den Heuvel, Wouter Slob, Gerard Tel, Jules van Kempen, Nelleke Steendam, Eric Ho, Petra Rutgers, Teun Slijkerman, Michael Beye, Marco Streng, Gerben Oostra, Johan van Rooij, and Maarten van Steenbergen.

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

Gerard Tel, February 2005.
email: `gerard@cs.uu.nl`

Contents

Chapter 1

Introduction to Quantum Cryptography

Written by *Johnson Leow*

One of the most important problems of cryptography is the key distribution problem; how do sender and recipient agree upon a secret key while being sure that a third party cannot acquire any information about it. Although public key cryptography provide algorithms like RSA or ElGamal to solve this problem, there is still no mathematical proof of the non-existence of a polynomial algorithm for cracking the key of those algorithms. Another point of concern is that public key cryptography will become obsolete when it is possible to build a quantum computer, because there already exists quantum algorithms which do factorization and discrete logarithm calculations in polynomial time. However, the key distribution problem is provable secure if quantum communication is used. The procedure that is used to agree upon a secure key with the help of quantum communication is called "quantum key distribution" and is generally called "quantum cryptography". One of the most important features of quantum cryptography is that a third party cannot acquire any reliable information by eavesdropping the quantum channel and that any attempt to do so will be detectable. It is this last feature that makes quantum cryptography very special, because non of the protocols of classical cryptography has this property.

The origins of quantum cryptography dates back to the work of Stephen Wiesner in the seventies who proposed to use single-quantum states as counterfeit-proof money. Wiesner published his ideas in 1983 which Bennett and Brassard used in 1984 to create the first quantum cryptographic protocol what is know known as the **BB84** protocol . However, it wasn't until 1989 that it could be implemented in a laboratory environment where polarized photons were used to create a secure communication channel of 32 cm.

An advancement was made at the beginning of the nineties when Ekert proposed a new protocol based on the Einstein-Podolsky-Rosen (EPR) paradox [Eke91]. Around the same period, in 1992, Bennett published a protocol called the **B92** protocol which could be implemented with single photon interference [Ben92].

Although the technological difficulties are still a big issue in the present day context, there has been a rapid development in the area of quantum computing and quantum cryptography in solving these issues.

In this article i will first explain some principles of quantum mechanics and from thereon continue with the quantum cryptographic protocols BB84, B92 and the basis of EPR based protocols. Finally, i will conclude this paper with a few comments on eavesdropping strategies and address some technological issues which still need to be solved before quantum cryptography can be implemented in practice.

1.1 Principals of Quantum Mechanics

We begin our discussion on quantum mechanics with the introduction of the Quantum Bit, nicknamed the **Qubit** . The biggest difference between the classical computer bit and the qubit is that the qubit can be both 0 and 1 at the same time. Now to implement the qubit, we use the polarization states of a photon. For example, a photon can be in a vertically polarized quantum state $|\updownarrow\rangle$, which we assign as the one-bit representation, or a horizontally polarized state quantum $|\leftrightarrow\rangle$, which we assign as the zero-bit representation. The photon can also be in a linear combination of these states, which we call a **superposition**. In this case the resulting state represents both the zero and the one bit at the same time.

1.1.1 Dirac notation

The quantum states are elements of a **Hilbert Space H** and are generally called **kets**. The kets are denoted by $|label\rangle$ where the label stands for the name of the state we want to give. The Hilbert space is defined as a vector space over the complex numbers **C** with an inner product

$$\langle\ ,\ \rangle : \mathbf{H} \times \mathbf{H} \longrightarrow \mathbf{C}$$

which is complete with respect to the norm

$$\|u\| = \sqrt{\langle u, u\rangle}$$

(See [Zei99] for more info on Hilbert spaces). Therefore, the qubit is just a ket in a two dimensional Hilbert space. For example, if $|0\rangle$ and $|1\rangle$ denote a arbitrary orthonormal basis of a two dimensional Hilbert space **H**, then each qubit can be written as

$$|qubit\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \text{ with } \alpha, \beta \in \mathbf{C}$$

Because any scalar multiple of a ket represents the same state of a quantum system, we can assume that $|qubit\rangle$ is normalized to unit length. In other words, we can restrict ourselves to $\|\alpha\|^2 + \|\beta\|^2 = 1$.

Given a Hilbert space **H**, we can define the space $\mathbf{H}^*$ as $\mathbf{H}^* = Hom(H, C)$. $\mathbf{H}^*$ is a Hilbert space, called the **dual** space of **H** and denotes the set of all linear maps from **H** to **C**. The elements of **H** are called **bra's** and they are denoted by $\langle label|$. We can now define a bilinear map $\mathbf{H}^* \times \mathbf{H} \longrightarrow \mathbf{C}$ by $(\langle\psi|)(|\phi\rangle) = \langle\psi|\phi\rangle \in \mathbf{C}$, were we call the last expression a **bracket**. Furthermore, if we have a orthonormal basis $|\phi_i\rangle$ of a Hilbert space **H**, then $\langle\phi_i|$ is an orthonormal basis of the dual space $\mathbf{H}^*$. The two bases are related via

$$\langle\phi_i|\phi_j\rangle = \delta_{ij},\ \delta_{ij} = 1 \text{ if } i = j,\ 0 \text{ otherwise}$$

1.1.2 Observations in quantum mechanics

An **observable** A in quantum mechanics is a **Hermitian operator**, which is a linear transformation from a Hilbert space **H** to itself. If we represent a linear transformation as a matrix, then a Hermitian operator is a matrix A who is the same as his transposed conjugate which we denote with a dagger:

$$A^\dagger = A \text{ with } A^\dagger := \bar{A}^T$$

An observable A has a set of eigenvectors called **eigenkets** $|\phi_i\rangle$ and eigenvalues λ_i:

$$A\,|\phi_i\rangle = \lambda_i\,|\phi_i\rangle$$

In the cases i consider here, the eigenkets form a complete orthonormal basis of the Hilbert space **H** and therefore every state $|\psi\rangle \in \mathbf{H}$ can be written as a linear combination of those eigenkets:

$$|\psi\rangle = \sum_i a_i\,|\phi_i\rangle \;\text{ with }\; \sum_i |a_i|^2 = 1$$

We can also write this as

$$|\psi\rangle = \sum_i |\phi_i\rangle\,\langle\phi_i|\psi\rangle \;\text{ with }\; \langle\phi_i|\psi\rangle = a_i$$

From this last equation we can see that the completeness of A is expressed by

$$\sum_i |\phi_i\rangle\langle\phi_i| = 1$$

Therefore, we have

$$A = \sum_i \lambda_i |\phi_i\rangle\langle\phi_i|$$

In quantum mechanics we can only measure observables and if we perform a measurement of the observable A on a state $|\psi\rangle = \sum a_i\,|\phi_i\rangle$, it will return one of the eigenvalues λ_i of A with probability $|a_i|^2$. The measurement though, will alter the state $|\psi\rangle$. If the measurement returns the eigenvalue λ_i of A, then afterwards the state $|\psi\rangle$ will be $|\phi_i\rangle$.
In quantum mechanics however, there is a limitation of what we can observe due to the so called **Heisenberg Uncertainty Principle**. Suppose we have two observables A and B, and let us define the **commutator** $[A, B]$ as

$$[A, B] = AB - BA$$

The two observables A and B are then called **compatible** if they commute; $[A, B] = 0$ and **incompatible** if they don't commute. Finally, let $\triangle A = A - \langle A\rangle$, then **Heisenberg's Uncertainty Principle** states that

$$\langle(\triangle A)^2\rangle\langle(\triangle B)^2\rangle \geq \frac{1}{4}\|\langle[A, B]\rangle\|^2 \text{ with } \langle(\triangle A)^2\rangle = \big\langle\psi\big|(\triangle A)^2\big|\psi\big\rangle\,.$$

This means that if A and B are incompatible, then we cannot measure A and B both with unlimited precision. This property of quantum mechanics will be used a lot in the examples later on.

1.1.3 Polarization states of a photon

We consider the example of the polarization states of a photon in a two dimensional Hilbert space **H**. We take (for example) a orthonormal basis consisting of the ket states $|\updownarrow\rangle$ and $|\leftrightarrow\rangle$ and we take a second orthonormal basis consisting of the states $|\nearrow\rangle$ and $|\searrow\rangle$. The two bases are related via:

$$|\nearrow\rangle = \frac{1}{\sqrt{2}}(|\updownarrow\rangle + |\leftrightarrow\rangle) \text{ and } |\searrow\rangle = \frac{1}{\sqrt{2}}(|\updownarrow\rangle - |\leftrightarrow\rangle)$$

Let us define the observables A and B as

$$A = \lambda_{\updownarrow} |\updownarrow\rangle + \lambda_{\leftrightarrow} |\leftrightarrow\rangle \text{ and } B = \lambda_{\nearrow} |\nearrow\rangle + \lambda_{\searrow} |\searrow\rangle$$

and suppose we have a normalized state $|\psi\rangle = |\updownarrow\rangle$. According to the quantum measurement theory, the measurement of observable A of the state $|\psi\rangle$ will return the eigenvalue $\lambda_{\updownarrow}$ with probability

$$|\langle\updownarrow|\psi\rangle|^2 = |\langle\updownarrow|\updownarrow\rangle|^2 = 1$$

and the eigenvalue $\lambda_{\leftrightarrow}$ with probability

$$|\langle\leftrightarrow|\psi\rangle|^2 = |\langle\leftrightarrow|\updownarrow\rangle|^2 = 0$$

because of the orthonormality of the states. Suppose now that we want to measure observable B of the state $|\psi\rangle$. Because we know that

$$|\updownarrow\rangle = \frac{1}{\sqrt{2}}(|\nearrow\rangle + |\searrow\rangle)$$

the measurement of B will return the eigenvalue $\lambda_{\nearrow}$ with probability

$$|\langle\nearrow|\psi\rangle|^2 = |\langle\nearrow| \frac{1}{\sqrt{2}}(|\nearrow\rangle + |\searrow\rangle)|^2 = \frac{1}{2}$$

The second term is zero because of the orthonormality of the states. With the same argument, the measurement will return $\lambda_{\searrow}$ with probability $\frac{1}{2}$. If we denote the state $|\psi_1\rangle$ as the resulting state after measurement, then we see that $|\psi_1\rangle$ is either $|\nearrow\rangle$ or $|\searrow\rangle$, which we (ambiguously) write as

$$|\nearrow\rangle = \frac{1}{\sqrt{2}}(|\updownarrow\rangle \pm |\leftrightarrow\rangle)$$

If we measure observable A of the state $|\psi_1\rangle$, then we will get $\lambda_{\updownarrow}$ or $\lambda_{\leftrightarrow}$ with a probability $\frac{1}{2}$ for each. This illustrates Heisenberg's Uncertainty Principle and from this example we can see that measurements of observables can alter a state which effects future measurements. This principle will be used in the next section to detect unwelcome eavesdroppers.

1.2 The BB84 Protocol

The BB84 protocol was developed in 1984 by Bennett and Brassard and makes use of the polarization states of the photon. We have seen from the previous section that the polarization states of the photon lie in a two-dimensional Hilbert space. Before describing this protocol, we first need two orthonormal bases of this two-dimensional Hilbert space. For simplicity, we choose the two orthonormal bases as the bases of our example in the previous section. Therefore, the first basis consists of the states $|\updownarrow\rangle$ and $|\leftrightarrow\rangle$ and the second basis consists of the states $|\nearrow\rangle$ and $|\searrow\rangle$. The two bases are related via:

$$|\nearrow\rangle = \frac{1}{\sqrt{2}}(|\updownarrow\rangle + |\leftrightarrow\rangle) \text{ and } |\searrow\rangle = \frac{1}{\sqrt{2}}(|\updownarrow\rangle - |\leftrightarrow\rangle)$$

So we see that $|\langle\phi|\psi\rangle|^2 = \frac{1}{2}$ if ψ and ϕ come from different bases.
Secondly, the BB84 protocol uses two incompatible orthonormal quantum alphabets to communicate and therefore we define our first alphabet as

$$|\updownarrow\rangle = 1 \text{ and } |\leftrightarrow\rangle = 0$$

and our second alphabet as

$$|\nearrow\rangle = 1 \text{ en } |\searrow\rangle = 0$$

Explanation of the protocol

If Alice and Bob wish to establish a secret key, they can now start communicating over the quantum channel using the BB84 protocol as follows:

1. Alice generates a random bitsequence A which will be used as part of the secret key.
2. For each bit A_i of sequence A, Alice will choose a random bit a_i. The bit a_i will be used to decide which alphabet is used to transmit A_i.
3. With each photon that Bob receives, he will choose an alphabet randomly to perform his measurements. In 50% of the cases, the chosen alphabet will coincide with that of Alice and the result B_i of Bob's measurement will be the same as A_i. In all the other cases, the chosen alphabet will not be compatible with that of Alice and the result of Bob's measurement B_i will therefore only be the same as A_i in 50% of the cases. In total, this means that only 75% of Bob's measured sequence B will be the same as Alice's sequence A.

Following the BB84 protocol, Alice and Bob now continue their communication over a public channel:

1. Bob tells Alice which quantum alphabet he used for his measurement and Alice tells Bob if that is the right one or not. So a_i will be known to Alice and Bob (and maybe a third eavesdropping party).
2. Alice and Bob then delete all the bits A_i and B_i from A and B respectively, for which they used incompatible alphabets to communicate. The resulting sequences are called the **raw keys** and they coincide with each other if there hasn't been any eavesdropping over the quantum channel.
3. Finally, Alice and Bob compare small portions of their raw keys to estimate the error-rates and then delete those bits used for the comparison to produce their **tentative final key** . If they discover any errors, they will know that a third party has been eavesdropping on them. If this is the case, Alice and Bob can start over again. If not, Alice and Bob then have established a secret key.

1.2.1 Detecting Oscar in the BB84 protocol

Suppose that Oscar is an eavesdropping third party. We have seen that if the alphabets chosen by Alice and Bob respectively are the same, then A_i and B_i will also be the same. However, if Oscar decides to eavesdrop on the quantum channel, then some B_i differ from A_i. For example, suppose that Alice is using the alphabet $|\updownarrow\rangle$ to send the one bit to Bob while Oscar is performing measurements using the other alphabet. The resulting state after the measurement will be either a $|\nearrow\rangle$ or a $|\searrow\rangle$. In any case, if Bob performs a measurement using the same alphabet as Alice's used, he will find the state $|\leftrightarrow\rangle$ with probability $\frac{1}{2}$. This means that $B_i = 0$ while $A_i = 1$. From this example we see that Oscar's eavesdropping will generate errors in the key of Alice and Bob.

1.2.2 The BB84 protocol with noise

In the previous section we have assumed that communication via the quantum channel is noise-free. But in practice, there will always be noise present which influences the error rate of the sequences A and B. We therefore cannot tell whether the errors are caused by noise or by eavesdropping of a third party Oscar. The solution to this problem is that we assume that all the errors are caused by Oscar. Therefore, the final key will only be partially secret. To agree upon a secret key, Alice and Bob must delete all errors from their raw key. One way to do this is by dividing the raw key into blocks of a certain length and then perform parity checks on them. This is the process of **error correction**. If the parity check does not agree, they will perform a binary search for the error by bisecting the block into two subblocks and comparing the parities of those subblocks. After comparing all the blocks, the step is repeated by performing a random common permutation on their remanent raw key, dividing it into blocks and comparing parity checks. Afterwards, Alice and Bob publicly select a random subset from their remanent raw key to compare parities and employ the binary search strategy if an error is detected. If no error has been found, then Alice and Bob can declare their remanent key as a **reconciled key** . Finally, Alice and Bob select n random subsets of their reconciled key without revealing their content. Their final secret key is then defined as the parities of those subsets. Because Oscar is assumed to have caused all the errors, the number n is dependent of the error rate of the original raw key. This last step is called **privacy amplification** .

1.3 The B92 Protocol

The B92 protocol uses only one alphabet instead of two. Furthermore, the alphabet used in this protocol is non-orthonormal. So let us define our alphabet (for example) as

$$|\updownarrow\rangle = 1 \text{ and } |\nearrow\rangle = 0$$

In this section, we use the same relations between the bases as in the previous sections. Furthermore, we define two projection operators

$$P_{|\leftrightarrow\rangle} = |\leftrightarrow\rangle\langle\leftrightarrow| \text{ and } P_{|\searrow\rangle} = |\searrow\rangle\langle\searrow|$$

We see that

$$\langle\updownarrow|P_{|\leftrightarrow\rangle}|\updownarrow\rangle = \langle\nearrow|P_{|\searrow\rangle}|\nearrow\rangle = 0 \text{ and } \langle\nearrow|P_{|\leftrightarrow\rangle}|\nearrow\rangle = \langle\updownarrow|P_{|\searrow\rangle}|\updownarrow\rangle = \frac{1}{2}$$

Now suppose that Alice and Bob want to establish a secret key, they can then start communicating over the quantum channel using the B92 protocol as follows.

1. Alice and Bob generate a random bitsequence A and B which will be used as part of the secret key, like in the BB84 protocol.

2. Alice sends her bitsequence A using the quantum alphabet and Bob measures the received states according to his bitsequence B. If the bit B_i of B is 0 then Bob will make the measurement with $P_{|\leftrightarrow\rangle}$ and if B_i is 1 he will make the measurement with $P_{|\searrow\rangle}$.

3. If Bob measures the states Alice has sent, he will find one of the following results: the state $|\updownarrow\rangle$ which represents the one bit, the state $|\nearrow\rangle$ which represents the zero bit or a state

which isn't defined in our alphabet like $|\leftrightarrow\rangle$ or $|\searrow\rangle$. In the last case we will say that the measurement failed and produced an **erasure** . We see that if B_i is different from A_i, then Bob's measurement will fail because Bob's measurement operator is orthogonal to that of Alice's states. If B_i and A_i have the same bitvalues, then there will be a probability of $\frac{1}{2}$ that a $|\updownarrow\rangle$ or a $|\nearrow\rangle$ comes out of the measurement. So we see that only one out of four measurements will give a result. In all the other cases the measurement will give an erasure. The erasure rate therefore, is 75%.

4. Bob tells Alice which of his measurements succeeded and Alice only keeps the bits from A for which the measurement of Bob was a success. Alice and Bob now obtain a raw key.

5. Finally, Alice and Bob compare small portions of their raw key like in the BB84 protocol to estimate the error-rate. If they discover any errors, they will know that a third party has been eavesdropping on them.

1.3.1 Detecting Oscar in the B92 protocol

We have seen that if Bob's measurement passes, in other words, if result of the measurement delivers a state in our alphabet, then B_i and A_i are the same. However, if Oscar decides to eavesdrop by making a measurement, with for example, the $P_{|\nearrow\rangle}$ operator, then some passed measurements will have a B_i that differs from A_i. We can see this with the following example: if Alice sends a one-bit state $|\updownarrow\rangle$ to Bob while Oscar is in the middle, then Oscar will alter the state to a $|\nearrow\rangle$ or a $|\searrow\rangle$. This means that if Bob measures the resulting state he will get the zero-bit state $|\nearrow\rangle$ with probability $\left\langle \nearrow \left| P_{|\leftrightarrow\rangle} \right| \nearrow \right\rangle = \frac{1}{2}$, instead of the usual erasure result when Oscar wasn't present.

1.4 Other Protocols

There are many other quantum cryptographic protocols besides the BB84 and the B92 protocol and it would be too much to list them all here. Therefore, i have chosen a few examples which i will outline briefly

1.4.1 EPR Based Quantum Cryptography

This variation of the BB84 protocol uses the so called **Einstein Podolsky Rosen Paradox** (ERP), which Einstein, Podolsky and Rosen published in 1935 to challenge the foundations of quantum mechanics. The idea is due to Artur Ekert who published this in 1991 [Eke91]. The proposed scheme is to use two qubits instead of one. The two qubits will come from a common source where one qubit will go to Alice and one to Bob. The qubits are so called **EPR Pairs** , whose states are correlated in such a way that the measurement of a chosen observable A of one the states automatically determines the result of the measurement of A of the other state. This means that if Alice and Bob measure their own qubit with the same basis, they will get a result which is correlated to each other. Furthermore, they know what the correlation is, so they will be able agree upon a common key in this way. Because the ERP paradox reaches inside the fundamentals of quantum physics, i will not go any further in the explanation of ERP Based quantum cryptography.

1.4.2 Variations on the BB84 Protocol

There is a large collection of variations on the BB84 protocol. For example, one can assume that the two bases are not chosen with equal probability. This will lead to a bigger probability of Alice and Bob choosing the same basis, but it will also be easier for Oscar to correctly guess the used basis. Therefore, it is still not clear if the net result is better or not. There are many other variations which i will not mention here. The net result of some of those are still questionable, but there are other variations which makes the practical implementation easier.

1.5 Eavesdropping tactics

In the protocols that i have discussed in the previous sections, i have only considered opaque eavesdropping, which means that Oscar would intercept and observe Alice's photons after which he sends the measured states to Bob. However, Oscar can also use other techniques to eavesdrop. There reason that i didn't consider those in the previous sections is because the analysis of all eavesdropping schemes would become very long and technical. In this section, i will briefly enlist a few eavesdropping strategies that Oscar can use.

- **Translucent eavesdropping without entanglement**. When Alice sends a qubit to Bob, Oscar can let a system of her choice, called a **probe**, interact with the qubit and then let it proceed to Bob in the modified state. He can choose the probe freely as well as its initial state and the interaction, as long it satisfies the quantum rules, which means that the interaction is described by a unitary operator.

- **Translucent eavesdropping with entanglement**. In this approach, Oscar will entangle his probe to the state send by Alice, and then let it proceed to Bob. Because this is one the most sophisticated eavesdropping method, i will not say more things about it.

- **Eavesdropping based on implementation weakness**. In the next section we will see that quantum cryptography is still difficult to implement in practice because the technology of today has still not caught up with it yet. For example, the probability that a single photon laser of today produces more than one photon is $\frac{1}{200}$. Therefore, if Oscar has an eavesdropping device that detects multiple photons, he can divert one of the photons for measurement. In this way Oscar can read $\frac{1}{200}$ of Alice's states without being detected.

For all these eavesdropping schemes, there exist algorithms to detect intrusion. However, what we really want is an algorithm that can handle large classes of eavesdropping schemes instead of one specific detection algorithm for one specific eavesdropping scheme. Therefore, the analysis of eavesdropping schemes is still far from finished. (See [AEP94] for more info on this topic)

1.6 Technological Challenges of Quantum Cryptography

To construct a quantum cryptography device, we need to consider the production, propagation and detection of single photons. We first need a device that emits a single photon. Unfortunately, this is difficult to realize experimentally because of the Poisson statistics of available light sources. For example, with the use of faint laser pulses there is only a probability of 10% that

the pulse contains one photon. There are other ways to produce photons, but they still can't overcome the fact that the photon creation process is inefficient and that the devices are still too complicated to use for practical situations.

After a single photon is created, the photon is send through a quantum channel. The quantum channel can be a optical fiber or just the free space. Photon propagation using optical fibers would be the most logical medium to consider, since they are widely used in telecommunications and are of high quality. The wavelength used for photons in optical fibers is near 1300 or 1500 nm. For these wavelengths the attenuation of optical fibers is of the order of 0.35 and 0.20 dB/km, which means that half of the photons are lost after about 9 and 15 km respectively. However, one major drawback of using optical fibers is that there still aren't any detectors that can detect photons above the 1000 nm wavelength. The development of these detectors is still going on and i will come back later on this. Although there aren't any good detectors for photon wavelengths above 1000 nm, there are efficient photon detectors for wavelengths around 800 nm which are also commercially available. However, if the 800 nm wavelength would be used, the currently available optical fibers can't be used as a quantum channel because of the wavelength incompatibility. In that case, the photon propagation requires free space transmission or the use of special fibers. Unfortunately, the quality of any special fibers isn't as high as that of optical fibers. Besides the qualitative inferiority, special fibers also have a practical disadvantage compared to optical fibers, since there already exist optical fiber networks, while special fibers are not commonly used.
Besides optical fibers, we can also consider free space transmission which has some advantages compared to the use of optical fibers. The atmosphere has a high transmission window at a wavelength of around 770 nm where photons can be detected using commercial, high efficiency counting modules. Furthermore, the polarization state of a photon will not be altered by the atmosphere at these wavelengths. However, free space transmission has a few disadvantages as well. In contrast to optical fibers, the energy transmitted in free space will spread out, leading to varying transmission losses. Furthermore, ambient light can enter the receiver, leading to a higher error rate. Fortunately, these errors can be reduced by using spectral filters and timing discrimination. Finally, free space transmission depend on atmospheric conditions and is only possible with clear weather. Therefore, free space channels have to be improved before they can be used in practice.

Although the safest bet is to use optical fibers as our photon carriers, we have seen that a new detector has to be developed for this. The detection of photons can in principle be achieved using techniques like photon-multipliers, avalanche-photodiodes, multichannel plates or superconducting Josephson junctions. The ideal detector should fulfill the following requirements:

1. it should have a quantum detection efficiency over a large spectral range
2. the noise rate, that is a signal creation without a photon arriving, should be small
3. the time jitter, that is the time variation between photon detection and the electric signal generation, should be small.
4. the recovery time should be small to make high data rates possible.
5. the detector should be handy to allow commercial use

Unfortunately, it is impossible to meet all the mentioned criteria. The best choice currently, are silicon avalanche photodiodes. For photons below 1100 nm, there are commercial detectors which have quantum efficiencies of 70% at a wavelength 700 nm, a time jitter of around 300

psec, a maximum count rate of more than 5 MHz and noise rate of 50 Hz for temperatures of -20 C. For photons above 1100nm, the only available avalanche photodiodes are photodiodes made from Germanium or InGaAs. These detectors have a relative bad performance compared to photodiodes made of silicon. Unfortunately, no industrial effort has been done to optimize photodiodes to perform at wavelengths above 1100 nm. However, there is no physical reason why photodiodes working at wavelengths above 1100 nm should be more delicate than below. The practical reason for the lack of commercial products is that the bandgap of Silicon is too large and therefore not sensitive enough for photon counting. Furthermore, the market for photon counting is still not mature yet. But if these problems are solved, the future of quantum cryptography will make a great leap forward.

Summary and Conclusions

If we compare quantum cryptography with classical cryptography, we see that the unique contribution of quantum cryptography is that it provides a mechanism for eavesdropping detection. On the other hand, on of the biggest disadvantages of quantum cryptography is that it provides no mechanism for authentication. Therefore, quantum cryptography and classical cryptography can be best used as complementary tools. For example, a small authentication key is first exchanged over a secure channel using classical methods. Then that key can be amplified by quantum cryptographic methods to an arbitrary length.

However, much remains to be done before quantum cryptography can be truly implemented in practice. Firstly, quantum protocols need to be extended to a computer network setting and it needs a more sophisticated error correction and detection mechanism. Secondly, there is a greater need for understanding intrusion detection in the presence of noise. Furthermore, there is a need for better understanding of detection algorithms. Because most quantum intrusion detection algorithms are based on specific eavesdropping strategies, they won't work against other eavesdropping schemes. It is therefore important to develop intrusion detection algorithms that can handle large classes of eavesdropping strategies instead of one specific strategy. Finally, technology has to be improved in order to meet the standards of quantum cryptography. As we have seen, true single photon lasers and efficient photon detectors are still under development.

Although quantum cryptography still has a lot of issues to be solved, the area of quantum cryptography and quantum computing in general is still under full development. Therefore, in any event, we can look forward to a bright future for quantum cryptography.

Chapter 2

Cryptography Using Bilinear Maps

Written by *Johan van Rooij*

This chapter is about the use of bilinear maps in cryptography. But since bilinear maps that fulfill the security requirements are not very easy to comprehend I will focus on consequences and applications of these maps. In section 2.1 I will introduce the concept of a bilinear map, and the variant we use in cryptography: a pairing. Next I will demonstrate two applications of these maps in 2.2 namely identity based signatures and a tripartite Diffie-Hellman protocol. In order to introduce two concrete pairings in section 2.3.3 we will first need to introduce the mathematics of finite fields and elliptic curves. These are all discussed in section 2.3. Next I will show that these pairings have a great impact on the complexity of the Diffie-Hellman problems applied to two different groups (2.4). And finally I present a signature scheme for short signatures and proof it to be secure in 2.5.

2.1 Non-Degenerate Bilinear Maps

I will begin by stating the definition of a bilinear map. Hereafter I will show which bilinear maps are useful for cryptography.

Definition 2.1 *For additively written groups G_1, G_2 and a multiplicatively written group G_T a map $f : G_1 \times G_2 \to G_T$ is* bilinear *if:*

$$f(x_1 + x_2, y) = f(x_1, y)f(x_2, y) \text{ and } f(x, y_1 + y_2) = f(x, y_1)f(x, y_2)$$

It is important to remind yourself of the fact that in the above definition G_1 and G_2 are additively and G_T is multiplicatively written, for these are purely notational independent of the group structure.

In cryptography we use non-degenerate bilinear maps, which we call pairings.

Definition 2.2 *$e : G_1 \times G_2 \to G_T$ is a* pairing *if:*

1. *f is bilinear.*
2. *f is non-degenerate: there exist $x \in G_1$, $y \in G_2$ s.t. $e(x, y) \neq 1$.*

For a pairing to be of any cryptographic use we also need it to be efficiently computable, while for any random $x \in G_2$ or $y \in G_2$, and a $z \in G_T$ it should be infeasible to solve $e(x, y) = z$.

This all seems to be pretty straightforward, but there are no 'easy' cryptographic pairings. We know bilinear maps from linear algebra, but applied to $\mathbb{Z}_p$ (p prime) this does not give use an appropriate pairing. For example in:

$$f : \mathbb{Z}_p \times \mathbb{Z}_p \to \mathbb{Z}_p : (x, y) \to cxy$$

We use the fact that addition in $\mathbb{Z}_p$ equals addition in the field $\mathbb{F}_p$ (see: 2.3.1), and therefore we can use the multiplication in $\mathbb{F}_p$ to define a function. But since we can also divide by anything except zero in $\mathbb{F}_p$ it does not give us a useful pairing. It is bilinear and non-degenerate, but also easy to invert and therefore useless to us.

The only suitable pairings found so far are derived from the Weil and Tate pairings, which are defined over subgroups of elliptic curves over finite fields which take values in finite fields. These are quite complex mathematical topics, which I will only discuss these superficially. Before going into these mathematical details I will first introduce some applications.

2.2 Applications in cryptography

Pairings can be used for a number of cryptographic applications. In this section I will introduce identity based signatures [Hes] and a tripartite Diffie-Hellman protocol [Jou04]. Another application, namely short signatures, will be discussed in section 2.5.

Both applications in this section exploit the fact that a pairing has the following property that can be trivially derived from the fact that it is bilinear.

$$\forall t \in \mathbb{Z} : f(tx, y) = f(x, ty)$$

Where once again we should remind ourselves that we have an additively written group where t does not belong to, so if it was multiplicatively written this would become x^t.

I will not prove these applications to be secure. Security relies on the security offered by the discrete logarithm on elliptic curves problem and the pairing being irreversible. The first of these can sometimes be reduced to the discrete logarithm problem on finite fields (the one we know), but in general is an even more complex problem. The proofs all follow the same principle: if an algorithm can break the encryption with a certain chance and within a certain amount of time, it can also solve a problem which should be impossible to solve.

2.2.1 Identity Based Signature Schemes

Traditional cryptographic signatures schemes use a public-private key pair, where the binding between the public key and the signer forms a new problem: How do we prevent an attacker from supplying the verifier with a false public key? In practice this is done using certificates from a Trust Authority. This is not only inefficient, but also still allows creation of false certificates if an attacker can supply the verifier with a false public key from the Trust Authority. An algorithm in which the public key can be derived from the signers identity would be useful indeed. Or more accurately we need a signature scheme in which the public key from a signer can easily be derived using a public deterministic algorithm from a string representing the signer's identity.

The first signature scheme that implements this and was proven to be secure was proposed by Hess [Hes] (Algorithm 2.1). It works on groups G_1, G_2 and G_T as in the previous section and uses two hash function $H : \{0, 1\}^* \to G_1 \backslash \{0\}$ and $h : \{0, 1\}^* \times G_2 \to \mathbb{Z}_p^*$ for a large prime p.

Setup: By Trust Authority
1. Pick random $P \in G_2 \backslash \{0\}$ and a secret$t \in \mathbb{Z}_p^*$.
2. $Q_{TA} := tP$.
3. Publish $(P, Q_{TA}) \in G_2 \times G_2$.
Extract: The Thrust Autohority constructs a key pair corresponding to ID.
Public key: $Q_{ID} := H(ID) \in G_1$
Private key: $S_{ID} := tQ_{ID} \in G_1$.
Sign: Message M,
1. Pick random $k \in \mathbb{Z}_p^*$.
2. $r := kP$.
3. $v := h(M, r)$.
4. $u := \frac{v}{k} S_{ID}$.
And the signature is the tuple $(u, r) \in G_1 \times G_2$.
Verify: Message M, and the signature (u, r).
1. $v := h(M, r)$.
The signature is accepted if and only if $e(u, r) = e(Q_{ID}, Q_{TA})^v$.

Algorithm 2.1: An identity based signature scheme

The verification holds for a valid signature since:

$$e(u, r) = e(\tfrac{v}{k} S_{ID}, kP) = e(\tfrac{vk}{k} S_{ID}, P) = e(S_{ID}, P)^v$$
$$= e(tQ_{ID}, P)^v = e(Q_{ID}, tP)^v = e(Q_{ID}, Q_{TA})^v$$

In the scheme the secret integer held by the Trust Authority, is the relation between the Trust Authority's two values and the relation between a signers public and private key. This value cannot be computed from these values, since this is an discrete logarithm problem on an elliptic curve. But this value can be used by the Trust Authority to give a signer his private key. During the evaluation of the bilinear map this value is not used, but the relation it imposes is implicitly transfered from the first argument to the second argument of the map. The rest of the scheme is pretty straightforward and nothing new compared to what we have already seen during our course.

An attacker who provides the verifier with a false public Trust Authority key still cannot sign a message. For the verifier still has the correct public key from the signer, and the attacker cannot create a false private key such that a forged signature will be accepted due to the false public Trust Authority key.

2.2.2 Tripartite Diffie-Hellman

The Diffie-Hellman protocol is an efficient way to give two participants a common key. It has been used as a building block for many complex cryptographic tools. A generalisation of this protocol would enable us construct many new and efficient protocols. The protocol I will introduced was introduced by Joux in [Jou04].

When dealing with more than two participants we can construct a protocol based on use the usual Diffie-Hellman protocol, but it will require multiple communication rounds and could require some participants to be on-line at the same time.

What we want to do is create protocol where three participants choose integers a, b, c from $\mathbb{Z}_p^*$, and publish $A = aP$, $B = bP$, $C = cP$ for a generator P of G_1. Next we need a map F

which allows us to calculate the key from the published values and the chosen integer. ($K = F(a,B,C) = F(A,b,C) = F(A,B,c)$).

A straightforward approach would be to use $F(x,Y,Z) = e(Y,Z)^x$ for this results in:

$$\left.\begin{array}{l} F(a,B,C) = e(bP,cP)^a \\ F(b,A,C) = e(aP,cP)^b \\ F(c,A,B) = e(aP,bP)^c \end{array}\right\} = e(P,P)^{abc}$$

The only problem is that both the Weil and Tate pairing can not be used in this context. The Weil paring has the property that when $G_1 = G_2$ for arbitrary $P \in G_1$: $e(P,P) = 1$, which results in all keys to equal 1 (See: 2.12). Even worse: every linear dependent P and λP cause $e(P,\lambda P) = e(P,P)^\lambda = 1$. Therefore we use two different subgroups of a larger one, with linearly independent generators P and Q. With $P_A = aP$, $Q_A = aQ$ etc. we now define $F(x,Y,Z) = e(Y,Z)^x$ and get:

$$\left.\begin{array}{l} F(a,P_B,Q_C) = e(bP,cQ)^a \\ F(b,P_A,Q_C) = e(aP,cQ)^b \\ F(c,P_A,Q_B) = e(aP,bQ)^c \end{array}\right\} = e(P,Q)^{abc} = \left\{\begin{array}{l} F(a,P_C,Q_B) = e(cP,bQ)^a \\ F(b,P_C,Q_A) = e(cP,aQ)^b \\ F(c,P_B,Q_A) = e(bP,aQ)^c \end{array}\right.$$

The Tate paring is defined somewhat differently, but also requires two elements in order to avoid possible divisions by zero. I will not go into these mathematical details.

What should be clear is that we can create new protocols for obtaining a common key by exploiting the properties of a pairing.

2.3 Concrete Pairings

In order to define the Weil and Tate pairings we will first need to introduce finite fields and elliptic curves over these finite fields. Thus I will do so first.

2.3.1 Finite Fields

Before introducing fields, I first want to formalise the concept of a group.

Definition 2.3 *An* Abelian group $(G,+)$ *is a set of elements* G *and a binary operation* $+ : G \times G \to G$ *with the following properties:*

1. *Associative: For each* $a,b,c \in G : (a+b)+c = a+(b+c)$.
2. *Identity: There is an element* $e \in G$ *such that for all* $a \in G : a+e = a = e+a$.
3. *Inverse: For each* $a \in G$ *there is an* $a^{-1} \in G$ *such that* $a + a^{-1} = e$.
4. *Commutative: For each* $a,b \in G : a+b = b+a$.

We often ommit the operation when refering to the group.

Using the above definition we can now define a field as follows.

Definition 2.4 *A* field $(F,+,\cdot)$ *is a set of elements* F *with two binary operations defined on it* $\cdot,+ : G \times G \to G$ *with the following properties:*

1. *$(G, +)$ is an Abelian group.*
2. *$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$ and $(a + b) \cdot c = (a \cdot c) + (b \cdot c)$.*
3. *if 0 is the identity element from the Abelian group $(G, +)$ then $(G \backslash \{0\}, \cdot)$ is an Abelian group as well.*

A field is nothing more than a set in which we can not only add as we do in an additive group, but multiply as well without violating a few basic rules. A few examples of fields are the rational and real numbers ($\mathbb{Q}$ and $\mathbb{R}$). Only these are of no good for cryptography, since we can only store elements from a finite set. Therefore we use finite fields.

Fact 2.5 *For a prime number p we can create the finite field $\mathbb{F}_p$ by combining the operations of addition and multiplication modulo p of $\mathbb{Z}_p$ and $\mathbb{Z}_p^*$.*

This can be done since $\mathbb{Z}_p^*$ (p prime) contains all the elements of $\mathbb{Z}_p$ except 0. An alternative notation is $GF(p)$ for $\mathbb{F}_p$, where GF stands for Galois Field. These are not the only existing finite fields, there is a generalisation of this concept, but we need some theory about polynomials first.

Definition 2.6 *$F[X]$ is the set of polynomials in X with coefficients from a field F.*

Definition 2.7 *A polynomial $p \in F[X]$ for some field F is* irreducible *if there are no polynomials $f, g \in F[X]$ such that $p = fg$ where f and g must be of degree at least 1.*

Not for every polynomial $f \in \mathbb{F}_p[X]$ there is an $r \in \mathbb{F}_p$ such that $f(r) = 0$. Such an r is called a *root* of f in $\mathbb{F}_p$. Polynomials have the property that for any root $r \in F$ of $f \in F[X]$ we can write $f = g(X - r)$ for some other polynomial $g \in F[X]$. Therefore we know than an irreducible polynomial $f \in \mathbb{F}_p[X]$ does not have any roots in $\mathbb{F}_p$.

For example $X^2 + 1 \in \mathbb{F}_3[X]$ is irreducible and does not have any roots, while $X^2 + 1 \in \mathbb{F}_2[X]$ does have a root: in $\mathbb{F}_2[X]$ we have $(X + 1)(X + 1) = X^2 + 1$ and $1^2 + 1 = 0$.

We can use this to extend a field F by adding the roots of a irreducible polynomial. This is done by calculating in $F[X]$ modulo our irreducible polynomial f. The arithmetic modulo a polynomial is not very different from modulo a number: if the degree of the polynomial is greater or equal to the degree of the modulus we use the remainder of division of the polynomial by the modulus. This new field is denoted as $F[X]/(f)$.

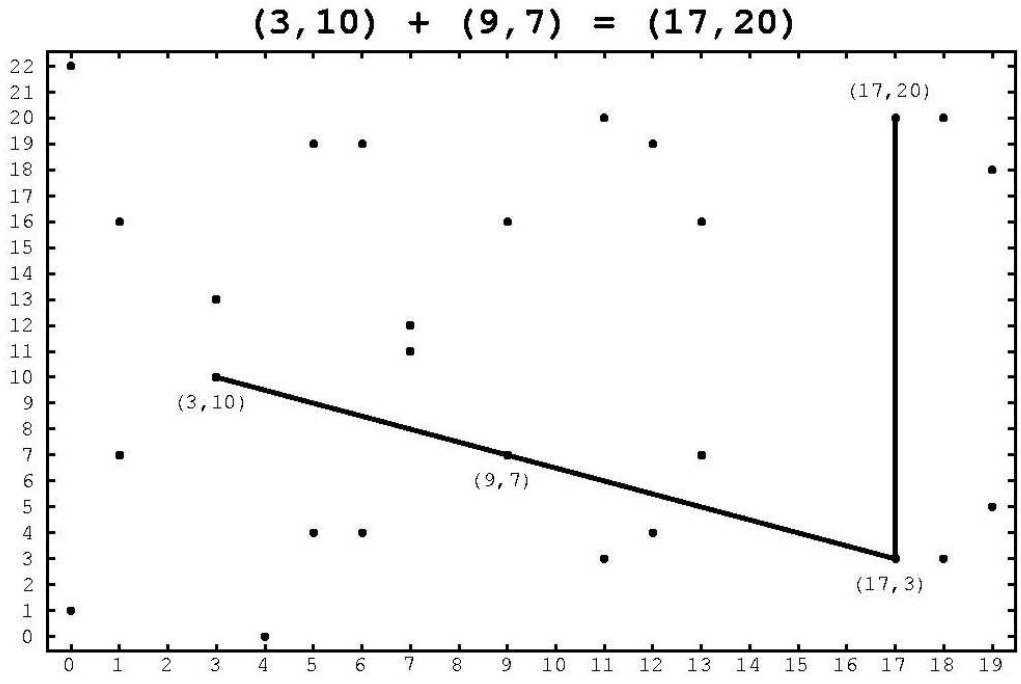
Following the previous example of $X^2 + 1 \in \mathbb{F}_3[X]$, we can form the field $\mathbb{F}_3[X]/(X^2 + 1)$. This field consists of the following elements: $\{0, 1, 2, X, X + 1, X + 2, 2X, 2X + 1, 2X + 2\}$, and by using the modulo $X^1 + 1$ arithmic and $f = X^2 + 1$, we get our root X: $f(X) = X^2 + 1 = 0 \pmod{X^2 + 1}$.

Fact 2.8 *The number of elements of a finite field always is a power k of a prime number p. We write $\mathbb{F}_{p^k}$ for such a field and it can be constructed by choosing a irreducible polynomial f of degree k over $\mathbb{F}_p$, and constructing $\mathbb{F}_{p^k} \cong \mathbb{F}_p[X]/(f)$.*

One can see that the X in the example above behaves just like the i in complex numbers. To keep thing insightful on can remember that for any finite field, the polynomial variable could be identified with some complex number.

Elements from these fields can be stored as a k-dimensional vector space over $\mathbb{F}_p$. When using $\mathbb{F}_{2^k}$ we can represent a single element from this field by a row of k bits.

Casus 2.2: An example of the group operation on an elliptic curve



The dots correspond to the points on the elliptic curve $y^2 = x^3 + x + 1$ over $\mathbb{F}_{23}$. We demonstrate the addition: (3,10) + (9,7), the line trough both intersects the curve at (17,3) and therefore the point opposite (17,3) namely (17,20) is the result.

2.3.2 Elliptic Curves

The groups the pairings work on are subgroups of elliptic curves. Therefore we will now introduce them.

Definition 2.9 *An* elliptic curve $E/\mathbb{F}_p$ *($p > 3$) is the set of solutions (x, y) to an equation of the form $y^2 = x^3 + ax + b$, where $a, b \in \mathbb{F}_p$ together with a point at infinity (∞) on which we define a group law in the following way:*

- ∞ *is the identity of the group:* $\forall (x, y) : (x, y) + \infty = (x, y) = \infty + (x, y)$, *and* $\infty + \infty = \infty$.
- $\forall (x, y) : (x, y)^{-1} = (x, -y)$ *i.e.* $(x, y) + (x, -y) = \infty$.
- $\forall x_1 \neq x_2 : (x_1, y_1) + (x_2, y_2) = (x_3, y_3) = (\lambda^2 - x_1 - x_2, \lambda(x_1 - x_3) - y_1)$ *with* $\lambda = \frac{y_2 - y_1}{x_2 - x_1}$.
- $(x_1, y_1) + (x_1, y_1) = (x_3, y_3) = (\lambda^2 - 2x_1, \lambda(x_1 - x_3) - y_1)$ *with* $\lambda = \frac{3x_1^2 + 1}{2y_1}$.

The group law in the definition appears to be counter intuitive, but it has a metric interpretation (see: casus 2.2). First of all observe that for each $x \neq 0$ there are two y such that $(x, y) \in E$. For two points (x_1, y_1), (x_2, y_2) we draw a line through both and find a third point intersecting the curve at (x', y'). Now $(x_3, y_3) = (x_1, y_1) + (x_2, y_2) = (x', -y')$ (the point opposite the third intersection point). If this point does not exist ($y_1 = -y_2$) we take ∞ as the resulting point. When doubling a point ($(x, y) + (x, y)$) the metric interpretation is only visible when taking an elliptic curve over a continuous field such as $\mathbb{R}$ or $\mathbb{Q}$. Since there are no two points to draw a

line through, we take the limit of two points moving to our single point, and obtain the line tangent to the curve at (x, y). The above definition contains these metric operations in the form of formulae.

Definition 2.10 *For an elliptic curve $E/\mathbb{F}_p$ we define $E(\mathbb{F}_{p^k})$ to be the set of points (x, y) on the curve with $x, y \in \mathbb{F}_{p^k}$.*

We often refer to $E(\mathbb{F}_{p^k})$ as being the elliptic curve, since the only difference with E is that it specifies the points allowed.

2.3.3 The Weil and Tate pairing

In this section we will define the Weil and Tate pairing and the subgroups of elliptic curves useful for constructing our required pairings. Since the exact formulas required to compute these pairings require a lot more mathematics and showing how to compute them efficiently requires even more math than I can tell in this chapter, I will keep things brief. For more information on the Weil paring and its computation see [Mil04] and for the Tate paring see [PSB04].

Definition 2.11 *We define $G[p]$ be the subgroup of G of elements of which order divides p.*

Fact 2.12 *Let p be a prime number, $E/\mathbb{F}_q$ an elliptic curve and α an integer so that $p \mid q^\alpha - 1$, and for all $k = 1 \ldots \alpha - 1 : p \not| q^k - 1$. The* Weil *paring is a map $e_p : E[p] \times E[p] \to \mathbb{F}_{q^\alpha}^\times$ with the following properties:*

- *Identity: for all $R \in E[p] : e_p(R, R) = 1$.*
- *Bilinear: for all $R_1, R_2 \in E[p]$ and $a, b \in \mathbb{Z}$ we have $e_p(aR_1, bR_2) = e_p(R_1, R_2)^{ab}$.*
- *Non-degenerate: If for an $R \in E[p]$ we have for all $R' \in E[p] : e_p(R, R') = 1$ then $R = \infty$. So if P,Q linearly independent: $e_p(P, Q) \neq 1$.*
- *Computable: for all $R_1, R_2 \in E[p]$ the pairing can be computed in polynomial time.*

We now use a results from Balasubramanian and Koblitz in [BK98] to construct the required groups.

Theorem 2.13 *Let $E/\mathbb{F}_q$ be an elliptic curve and $P \in E(\mathbb{F}_q)$ be a point of prime order p with $p \not| q$. Let α be so that $p \mid q^\alpha - 1$, and for all $k = 1 \ldots \alpha - 1 : p \not| q^k - 1$. If $\alpha > 1$ then there exists a $Q \in E(\mathbb{F}_{q^\alpha})$ of order p that is linearly independent of P.*

Using this theorem we can create groups $G_1 =< P > \subseteq E(\mathbb{F}_q)$, $G_2 =< Q > \subseteq E(\mathbb{F}_{q^\alpha})$ and $G_T = \mathbb{F}_{q^\alpha}^\times$ ($\mathbb{F}_{q^\alpha}$ with multiplication as the group operation) on which the Weil paring has the required properties.

In practical situations we mostly use the Tate pairing, since it is much more efficiently computable. The Tate pairing is a non-degenerate bilinear map:

$$t_p : E(\mathbb{F}_{q^\alpha})[p] \times E(\mathbb{F}_{q^\alpha})/pE(\mathbb{F}_{q^\alpha}) \to \mathbb{F}_{q^\alpha}^\times/(\mathbb{F}_{q^\alpha}^\times)^p$$

It is even more difficult to explain the groups on which this pairing works, therefore I will only use the Weil pairing.

What is missing in the above discussion is a claim or proof about the pairings being non-invertible. We will do so while dealing with the Diffie-Hellman problems on Elliptic Curves (lemma 2.15).

2.4 Pairings and the Elliptic Curve Diffie-Hellman problems

The traditional Diffie-Hellman problems involve two elements from a single group and their exponents. Our parings are defined on two different groups, which give rise to generalisations of the Diffie-Hellman problems. What we are about to see is that the existence of these pairings has great influence on the difficulty of these problems.

Definition 2.14 ***co-Diffie-Hellman problems***
The Decision co-Diffie-Hellman problem on (G_1, G_2)*:*
Given $g, g^a \in G_1$ *and* $h, h^b \in G_2$ *is* $a = b$ *true?*
The Computational co-Diffie-Hellman problem on (G_1, G_2)*:*
Given $g, \in G_1$ *and* $h, h^b \in G_2$ *compute* $g^b \in G_1$*?*

When $G_1 = G_2$ these problems reduce to the standard Diffie-Hellman problems.

By using the Weil paring $e_p : E[p] \times E[p] \to \mathbb{F}_{q^\alpha}^\times$ and different subgroups G_1 and G_2 of $E[p]$ as we have done before we can solve the Decision co-Diffie-Hellman for a tuple (g, g^a, h, h^b) problem quite easily:

$$a = b \bmod p \iff e(g, h^b) = e(g^a, h)$$

This appears to be a bad thing, but lets see first what happens to the Computational co-Diffie-Hellman problem. The best known algorithm for solving this problem is to compute the discrete logarithm in G_1. The Computational co-Diffie-Hellman problem and the discrete logarithm on G_1 can even shown to be equivalent given some extra information about G_1. In order to keep the Computational co-Diffie-Hellman problem intractable we therefore only need to make the discrete log problem on $G_1 = E(\mathbb{F}_q)$ intractable.

In order to defend ourselves from general discrete log algorithms like the ones we have seen during the course we need to ensure that the order p of G_1 is a sufficiently large prime. For our specific case a number of people ([FR94] and [AMV93]) have suggested reductions on $E(\mathbb{F}_q)$ to compute the discrete log more efficiently. They all use a specific function to map the discrete log problem on G_1 to some extension of $\mathbb{F}_q$ say $\mathbb{F}_{q^i}$. We now can solve the discrete log problem on $\mathbb{F}_{q^i}^\times$ using the Number Field Sieve Algorithm (see: 4). But since G_1 has order p the subgroup of $\mathbb{F}_{q^i}^\times$ the problem is mapped to must also have order p. We know that $\mathbb{F}_{q^i}^\times$ has $q^i - 1$ elements and the order of a subgroup always divides the order of the group: $p \mid (q^i - 1)$. If we remember the α from (fact 2.12) we always have $i \geq \alpha$, so we only need to make sure we are working on an elliptic curve with a large enough α (and q) to make the discrete log on $\mathbb{F}_{q^\alpha}^\times$ infeasible.

Lemma 2.15 *A paring defined on two groups* G_1*,* G_2 *for which co-Diffie-Hellman problem cannot be solved efficiently and for which there exists an efficiently computable isomorphism* $\phi : G_2 \to G_1$*, cannot be inverted effficiently.*

An isomorphism is a map that maps a generator g_2 of G_2 to a generator g_1 of G_1 and every element g_2^a to g_1^a.

Proof. Given a Computational co-Diffie-Hellman instance with $g, \in G_1$ and $h, h^b \in G_2$, it should be impossible compute $g^b \in G_1$. Now we assume our pairing to be invertible: for a random $x \in G_1$ or $y \in G_2$ and a given $z \in G_T$ we can solve $e(x, y) = z$.

But if we compute $v := e(g, h)$ and $v^b = e(g, h^b)$ we can now compute $x = g^b$ by solving $e(x, h) = v^b$. Therefore a pre-image for the first argument of the pairing should not be computable efficiently.

Setup:
Pick a random private key $x \in \mathbb{Z}_p$ and compute the public key $v := g_2^x$.
Sign: Message M.
Evaluate the hash function $h = H(M) \in G_1$ and compute the signature $\sigma = h^x$.
Verify:
Evaluate the hash function $h = H(M)$ and accept if the co-Diffie-Hellman relation goes for h, σ from G_1 and g_2, v from G_2.

Algorithm 2.3: A Gap Diffie-Hellman Signature Scheme

In order to prove that pre-images for the second argument should not be computable, we need our map ϕ. Let a be an integer such that $g = \phi(h)^a$. We compute a new v : $v := e(\phi(h), h)$, now $e(g, h^b) = e(\phi(h)^a, h^b) = v^{ab}$. If we can solve y in $e(\phi(h), y) = v^{ab}$ it will be $y = h^{ab}$ from which we can compute $g^b = \phi(h)^{ab} = \phi(h^{ab}) = \phi(y)$.

Thus the pairing cannot be efficiently inverted. △

The required isomorphism exists for our groups G_1 and G_2 but since it is constructed using Galois maps of $\mathbb{F}_{q^\alpha}$ over $\mathbb{F}_q$, I will spare you the details.

A pair of groups on which the Decision co-Diffie-Hellman is easy but the Computational co-Diffie-Hellman is computationally infeasible are called Gap Diffie Hellman groups. The (in)feasibility of the different Diffie-Hellman problems allow us to construct entirely new protocols one of which allows short signatures and will be the main topic of the next section.

2.5 Short Signatures

2.5.1 A secure signature algorithm

We can create a signature scheme based on the fact that Computational co-Diffie-Hellman is infeasible, but Decision co-Diffie-Hellman is easy. This was done by Boneh, Lynn and Shacham in [DBS04] (Algorithm 2.3). Before applying concrete groups and the Weil paring to the scheme, I will first prove it to be secure. We will prove that it is hard to create a existential forgery even the attacker is given a number of signatures on messages of his choice. The proof will use the Random Oracle Model [PS00], which means that we assume that the attacker cannot use any information about the hash function used: it is a random, but gives the same answers to identical queries (oracle). We will use the algorithm of the attacker to solve the Computational co-Diffie-Hellman problem by simulating the hash function so that it gives answers with some extra meaning, while we keep them uniformly distributed. This will become clear in the proof. For more information about the Random Oracle Model or provable security in general, see 3.3.3.

Theorem 2.16 *If G_1 and G_2 are Gap Diffie-Hellman groups on which we can only break the Computational co-Diffie-Hellman problem with chance smaller than δ, then an attacker can only construct an existential forgery with a chance smaller than $\epsilon = e(q+1)\delta$, even if it is given q signatures on messages of its choice. The forgery created by the attacker may of course not be a signature he asked for.*

Proof. Suppose we have an algorithm that constructs an existential forgery with chance greater than ϵ. I will now show how to construct an algorithm that solves the Computational co-Diffie-

Hellman problem on G_1 and G_2 with probability greater than δ. For g_2 a generator of G_2, we start with $h \in G_1$ and g_2, $u = g_2^a$ from G_2 and construct h^a with chance greater than δ as follows:

Setup We start by giving the attacker the public key $u \cdot g_2^r$ for a random $r \in \mathbb{Z}_p$, where p is the order of G_1 and G_2.
Queries to the hash function To keep our responses to the queries to the hash function consistent we maintain an initially empty list of tuples (M_j, w_j, b_j, c_j). If a asked for a hash value for a message M_i which is in the list we respond with the corresponding $w_i \in G_1$. If M_i is not in the list we pick a random $b_i \in \mathbb{Z}_p$ uniformly and an $c_i \in \{0, 1\}$ with $\Pr[c_i = 0] = \frac{1}{q+1}$. Next we respond with $w_i := h^{1-c_i} \cdot \phi(g_2)^{b_i} \in G_1$ and add (M_i, w_i, b_i, c_i) to the list.
Remember the isomorphism ϕ from our proof of the pairing being not invertible. And note that w_i is picked uniform in G_1 and is independent of the attackers view.
Requested signatures When constructing a signature for a message M_i we query our simulated hash function for the corresponding w_i and c_i. If $c_i = 0$ we abort our construction, otherwise we know $c_i = 1$ and so $w_i = \phi(g_2)^{b_i}$. We create the signature $\sigma_i = \phi(u)^{b_i} \cdot \phi(g_2)^{rb_i}$ which equals w_i^{a+r} and therefore is a valid signature over M_i for the public key $u \cdot g_2^r = g_2^{a+r}$. The signature is send to the attacker.
Solving Computational co-Diffie Hellman When the attacker succeeds and produces his existential forgery σ_f over message M_f, we once again query our own simulated hash function for a corresponding tuple (M_f, w_f, b_f, c_f). If $c_f = 1$ we abort our construction, otherwise $c_f = 0$ and therefore $w_f = h \cdot \phi(g_2)^b$ and $\sigma_f = h^{a+r} \cdot \phi(g_2)^{b(a+r)}$. We now construct $h^a = \sigma_f \cdot (h^r \cdot \phi(u)^b \cdot \phi(g_2)^{rb})^{-1}$.

We now only need to show that this algorithm solves the Computational co-Diffie-Hellman problem with the claimed chance. Therefore we introduce three claims, which together proof the theorem.

Claim 2.17 *The probability that our algorithm does not abort during the production of signatures as a results of the attackers requests is at least* $\frac{1}{e}$.

Proof. We assume without loss of generality that the attacker does not ask for the same signature twice. By induction we will prove that after the attacker requested i signatures the probability that our algorithm did not abort is at least $(1 - \frac{1}{q+1})^i$. For $i = 0$ this is true since we do not abort when no signatures are requested. When signature i is requested the value of c_i is independent of the information of the attacker so the chance that we will abort is $\frac{1}{q+1}$. Using the induction hypothesis we now know that our chance of not aborting is at least $(1 - \frac{1}{q+1})^i$. And since the attacker is only allowed to ask for q signatures our chance of not aborting is greater than $(1 - \frac{1}{q+1})^q \geq \frac{1}{e}$. △

Claim 2.18 *If our algorithm has not aborted during the production of signatures on the attackers requests, then the attacker can not know it is being used by our algorithm and hence it produces a valid signature with chance greater than* ϵ.

Proof. The public key given to the attacker is from the same distribution as a public key produced by the signature scheme. Responses the the hash function are also uniformly distributed in G_1 and all responses to requests for signatures are valid. Thus there is no difference the attacker can notice and he will produce a valid signature with a chance greater than ϵ. △

Claim 2.19 *The probability that our algorithm aborts after the attacker gives us a valid forgery is* $\frac{1}{q+1}$.

Proof. Since the attacker did not ask for the signature he forged, he can only have some information about c_f from the simulated hash function evaluation. But the w_f has the same distribution whether $c_f = 0$ or $c_f = 1$ thus he cannot know whether our algorithm will abort or not due to its forgery. Hence the chance of abortion is $\frac{1}{q+1}$. △

You might have noticed that each claim only gives us any information if the event in the foregoing claim happened. So if we did not abort claim 2.18 says something about our chance of getting a forgery and if this happened as well claim 2.19 says something about our chance of solving the Computational co-Diffie-Hellman problem.

Therefore the chance on solving the problem (not aborting) C:

$$C > \frac{1}{e}\,[2.17] \times \epsilon\,[2.18] \times \frac{1}{q+1}\,[2.19] = \frac{\epsilon}{e(q+1)}$$

When δ is the upper bound on the chance of solving the Computational co-Diffie-Hellman problem ($\delta \geq C \geq \frac{\epsilon}{e(q+1)}$) then the attacker can never construct this forgery with a chance greater than $\epsilon = e(q+1)\delta$. △

If we truly want to proof this theorem nicely we also have to keep the time it takes to run our algorithm into account, for it be infeasible to run. If one would do so on can see that our algorithm takes only linear amount of time in the number of times we have to simulate the hash function or supply the attacker with queries and the linear factor is approximated by the time required for an exponentiation in G_1.

2.5.2 Truly Short Signatures

Short signatures allow new applications while keeping the security level high enough. They can for example be used for product registration systems where a user must enter the key, or they can be used when we want to encode a signature in a bar code.

In the previous section we have introduced a secure signature scheme based on Gap co-Diffie-Hellman groups. We will now use the elliptic curves over finite fields and the Weil paring to construct such a scheme while slightly modifing it so that it allows short signatures. The result is also from Boneh, Lynn and Shacham [DBS04] and is shown in algorithm 2.4.

We use an elliptic curve $E/\mathbb{F}_q$ and $P \in E(\mathbb{F}_q)$ a point of prime order p. And we use the result of Balasubramian and Koblitz (theorem 2.13) to obtain a $P \in E(\mathbb{F}_q)$ and $Q \in E(\mathbb{F}_{q^\alpha})$, both of order p that are linearly independent and remind ourselves of the involved α. And once again we take $G_1 =< P >$ and $G_2 =< Q >$. Notice that we use only a single element from $\mathbb{F}_q$ for a signature and reconstruct the elliptic curve ellement upon verification.

Definition 2.20 *The integer α so that $p \mid q^\alpha - 1$, and for all $k = 1 \ldots \alpha - 1 : p \not| q^k - 1$ if called the* security multiplier *of our signature scheme.*

This due to the fact that this α causes a smart way of reducing the elliptic curve discrete log problem to a discrete log problem in $\mathbb{F}^\times_{q^\alpha}$: a field of q^α elements (see 2.4).

By theorem 2.16 we know this signature scheme is secure if the Computational co-Diffie-Hellman problem is hard enough. We can remind ourselves the requirements from 2.4: p and q must be large enough to withstand generic discrete logarithm algorithm attacks, and the security multiplier α must be large enough to defend against reductions to $\mathbb{F}_{q^\alpha}$ to witch the Number Field Sieve algorithm can be applied.

A family of elliptic curves found by Miyaji et at. in [AMT01] allow for a security multiplier of $\alpha = 6$. It remains an open question whether higher security multipliers can be reached. There for

Setup:
Pick a random private key $x \in \mathbb{Z}_p$ and compute the public key $V := xQ \in E(\mathbb{F}_{Q^\alpha})$.
Sign: Message M.
Evaluate the hash function $R = H(M) \in G_1$ and compute $\sigma = xR \in E(\mathbb{F}_Q)$.
The signature s is the x-coordinate of σ: $s \in \mathbb{F}_q$.
Verify:
Find a $y \in \mathbb{F}_q$ such that $\sigma = (s, y)$ is a point on $E(\mathbb{F}_Q)$.
If no such y exist or σ does not have order p the signature is invalid.
Evaluate the hash function $R = H(M)$ and accept if and only if
either $e(\sigma, Q) = e(R, V)$ or $e(\sigma, Q)^{-1} = e(R, V)$

Algorithm 2.4: Short Signatures by Boneh, Lynn and Shacham

example exists an elliptic curve with $\alpha = 6$ over $\mathbb{F}_q$ with q a 168-bits prime. This allows 168-bits signatures with a 166-bit prime p, so 166-bit security against a generic discrete log attack and 1008-bit (6×168) protection against the Number Field Sieve algorithm (see: 4).

Chapter 3

Provable Security in practical situations

Written by *Petra Rutgers*

Joke An elderly Frenchman rises every morning at 5, goes out to the street in front of his house, and sprinkles a white powder up and down the street. One day, a neighbour, who has watched his routine for many years, confronts him. 'What is this powder you sprinkle on the street every morning, Pierre?' 'It is elephant powder, mon ami,' the gentleman says. 'It keeps the elephants away.' 'But Pierre,' says the neighbour. 'Everybody knows that there are no elephants in France.' Says the older man matter-of-factly, 'I guess it must be working, then.'

The joke has a serious message. Cryptography is like elephant powder. If it seems to work, and keeps the attackers, our elephants, away, then we trust it. This isn't really very satisfactory. [Mil04]

Since the beginning of public-key cryptography, many suitable algorithmic problems for crypthography have been proposed. Then, many crypthographic schemes have been designed, together with more or less heuristic proofs of their security. However, most of those schemes have thereafter been broken.

The simple fact that a cryptographic algorithm withstands cryptanalytic attacks for several years is often considered as a kind of validation procedure, but some schemes take a long time before being broken. Therefore, the lack of attacks at some time should never be considered as a security validation of any proposal.

Organisation of the chapter In the next section, I will explain what provable and practical security is. In section 2 I will recall some basics we have seen in class, which are nessesarry for provable security, namely the classical assuptions on which the security may rely. In section 3 I will describe several signature and encryption schemes with their security results. All the information comes from [Poi02]

3.1 Provable security and practical security

A lot of research had tried to provide proofs in the framework of complexity theory: the proofs provide reduction from a well-studied problem to an attack against a cryptographic protocol, you named it *provable security*.

First, people definied the security notions required by actual cryptographic schemes, and then designed protocols which achieve these notions. The techniques were directly derived from the

complexity theory, providing polynomial reductions. However, their aim was mainly theoretical, and thus they tried to minimize the required assumptions on the primitives (one-way functions, etc). Therefore, they just needed to exhibit polynomial reductions from the basic assumption on the primitive into an attack of the security notion. However, such a result has no practical impact on actual security of proposed schemes. Indeed, even with a polynomial reduction, one may be able to break the cryptographic protocol within few hours, whereas the reduction just leads to an algorithm against the underlying problem which requires many years. Therefore, those reductions only prove the security when very huge parameters are used, under the assumption that no polynomial time algorithm exists to solve the underlying problem.

For a few years, more effcient reductions have been expected, which provide more practical security results. The perfect situation is reached when one is able to prove that, from an attack, one can describe an algorithm against the underlying problem, with almost the same success probability within almost the same amount of time. We have then achieved *practical security* . Unfortunately, in many cases, provable security is at the cost of an important loss in terms of efficiency for the cryptographic protocol, or relies on a weaker computational problem. Therefore, a classical way to give some evidences about the security of an efficient scheme relative to a strong computational problem is to make some hypotheses on the adversary's behavior: the attack is generic, independent of the actual implementation of some objects:
- of the hash function, in the "random oracle model";
- of the group, in the "generic (group) model". (I will skip this part)

3.2 Needed basics from class

We assume that the algorithms are public and that only a short parameter (the secret key) can be kept secret. The question is: can a scheme be secure?

An **encryption scheme** consists of three algorithms , namely:

- **G** - key generation
- **E** - encryption
- **D** - decryption

The ciphertext comes from $c = E_{k_e}(M, r)$. The encryption key k_e is public and a unique message M satisfies the relation (with possibly several random r). At least an exhaustive search on M and r can lead to M, but maybe a better attack is possible. So, unconditional secrecy is impossible and we needed algorithmic assumptions.

3.2.1 Computational Assumptions

As seen, for asymmetric cryptography, no security can be unconditionally guarenteed. Therefore, for any cryptographic protocol, security relies on a computational assumption: the existence of one-way functions, or permutations, possibly trapdoor. I will now mention them.

Integer factoring

The most famous intractable problem is factorization of integers : while it is easy to multiply two prime integers p and q to get the product $N = p \cdot q$, it is not simple to decompose N into its

prime factor p and q. Unfortunately, it just provides a one-way function, without any possibility to invert the process.

RSA problem : let $N = pq$ be the product of two large primes of simular sizes and e an integer relactive prime to $\phi(N)$. For a given $y \in \mathbf{Z}_N^*$, find $x \in \mathbf{Z}_N^*$ such that $x^e = y \ mod \ N$. The RSA assumption then says that this problem is intractable for any modulus $N = pq$, large enough: the success probability $Succ^{RSA}(\mathbf{A})$ of any adversary $\mathbf{A}$ within a reaonable running time is small.

Discrete logarithm

Some other classical problems are related to the discrete logarithm . The setting is quite general: one is given a finite cyclic group $\wp$ of prime order q and a generator g (i.e. $\wp = < g >$). In such a group, one consider the following problem (using the additive notations):

The *Discrete logarithm* problem (DL): given $y \in \wp$, compute $x \in \mathbf{Z}_q$ such that $y = g^x$, then one writes $x = log_g \ y$.

Trapdoor

The Discrete Logarithm is difficult and no information can help. We have the *Computational Diffie-Hellman* problem (CDH): given two elements in the group $\wp$, $\mathbf{a} = g^a$ and $\mathbf{b} = g^b$, compute $\mathbf{c} = g^{ab}$. Then one writes $\mathbf{c} = \mathbf{DH(a,b)}$. The textbfDecisional Diffie-Hellman problem (DDH) : given three elements in the group $\wp$, $\mathbf{a} = g^a$, $\mathbf{b} = g^b$ and $\mathbf{c} = g^c$, decider whether $\mathbf{c} = \mathbf{DH(a,b)}$.

3.3 Security proofs

For security proofs are algorithmic assumptions necessary, take as example the RSA problem:

$n = pq$: public modulus	RSA Encryption
e: public exponent	$E(M) = M^e mod \ N$
$d = e^{-1} mod \ \phi(N)$: private	$D(C) = C^d mod \ N$

If the RSA problem is easy, secrecy is not satisfied: anybody could recover M from C.

Security proofs are not only necessary, they also give the guarantee that the assumption is enough for secrecy: if an adversary can break the secrecy then one can break the assumption, this is named a "reductionist" proof.

A proof by reduction means a reduction of a problem P to an attack Atk: Let A be an adversary that breaks the scheme P then A can be used to solve P. If P is interactable then the scheme is unbreakable.

Provably secure scheme exist to prove the security of a cryptographic scheme. Therefore one has to make precise the algorithmic assumptions as well as the security notions to be guaranteed. Further we need a reduction, because an adversary can help to break the assumption. So we defines according to this requirements the security notions: the goals of an adversary and the availalble information of the adversary.

3.3.1 Public-key encryption

Definitions: A *public-key encryption scheme* is defined by the three following algorithms :

- The *key generation algorithm* K: the algorithm K produces a pair (x, y) of matching public and private keys. Algorithm K is probablistic.

- The *encryption algorithm* E: given a message M and a public key y, E produces a ciphertext C of M. This algorithm may be probabilistic. In this latter case, we can write $E(y, M; r)$ where r is the random type.
- The *decryption algorithm* D: given a ciphertext C and the private key x, D gives back the plaintext M. This algorithm is necessarily deterministic.

Security notions. The goals of the adversary may be various. There a three security notions :

- The first common security notion that one would like for an encryption scheme is *one-wayness* (OW) : with just public data, an attacker cannot get back the whole plaintext of a given ciphertext. More formally, this means that for any adversary A, her success in inverting E without the private key should be nigligible over the message space $\mathbf{M}$.
- However, many applications require more, namely the *semantic security* . This security notion means computational impossibility to distinguish between two messages, chosen by the adversary, one of which has been encrypted, with a probabality better than one half: her advantage should be negligible.
- A later notion is *non-malleability* (NM) . To break it, given a ciphertext, the adversary tries to produce a new ciphertext such that the plaintext are meaningfully related.

On the other hand, an attacker can play many kinds of attacks, according to the available information:

- Since we are considering asymmetric encryption, the adversary can encrypt and plaintext of her choice, granted the public key, hence the *chosen-plaintext attack* (CPA) .
- She may furthermore have access to more information, modeled by partial of full access to some oracles:
  - a plaintext-checking oracle which, on input (M, C), answers whether C encrypt the message M. This attack has been named the *plaintext-checking attack* (PCA)
  - or the decryption oracle itself, which on any ciphertext, except the challenge ciphertext, answers the corresponding plaintext. The scenario which allows adaptively chosen ciphertexts as queries to the decryption oracle is the strongest attack, and is named the *chosen-ciphertext attack* (CCA).

In public-key encryption is it difficult to reach CCA security. Maybe it is possible, but with inefficient schemes. However, inefficient schemes are not useful in practice: everybody wants security, but only if it is transparent.

Basic Encryption Schemes

The Plain-RSA Encryption . The RSA primitive can also be used for encryption: the key generation algorithm produces a large composite number $N = pq$, a public key e, and a private key d such that $e \cdot d = 1 \ mod \ \varphi(N)$. The encryption of a message M, encoded as an element in $\mathbf{Z}_N$, is $C = M^e \ mod \ N$. This cipher text can be decrypted thanks to the knowledge of d, $M = C^d \ mod \ N$. Clearly, this encryption is OW-CPA, relative to the RSA problem. The determinism makes a plaintext-checking oracle useless. Indeed, the encryption of a message M, under a public key k_p is always the same, and thus it is easy to check whether a ciphertext C really encrypts M, by re-encrypting this latter. Therefore the RSA encryption scheme is OW-PCA relative to the

RSA problem as well. Because of this determinism, it cannot be semantically secure: given the encryption C of either M_0 or M_1, the adversary simply computes $C_0 = M_0^e \ mod \ N$ and checks whether $C_0 = C$ or not to make the decision.

The El Gamal Encryption Scheme . In 1985, El Gamal also designed a DL-based public-key encryption scheme, inspired by the Diffie-Hellman key exchange protocol: given a cyclic group $\wp$ of prime order q and a generator g, the generation algorithm produces a random element $x \in \mathbf{Z}_q^*$ as private key, and a public key $y = g^x$. The encryption of a message M, encoded as an element M in $\wp$, is a pair $(c = g^a; d = y^a + M)$. This ciphertext can be easily decrypted thanks to the knowledge of x, since $y^a = g^{xa} = c^x$, and thus $M = d - c^x$. This encryption scheme is well-known to be OW-CPA relative to the CDH problem. It is also IND-CPA relative to the DDH problem. About OW-PCA, it relies on the new GDH problem. However, it does not prevent adaptive chosen-ciphertext attacks because of the homomorphic property. As we have seen above, the expected security level is IND-CCA. We wonder if we can achieve this strong security with practical encryption schemes.

3.3.2 Digital signature schemes

Definitions: A *signature scheme* is defined by the three following algorithms :

- The *key generation algorithm* K: on input 1^k, the algorithm K produces a pair (x, y) of matching public and private keys. Algorithm K is probablistic. The input k is called the security parameter.
- The *signing algorithm Sig*: Given a message M and a pair of matching public and private keys (x, y), Sig produces a signature S. The signing algorithm might be probabilistic.
- The Verification algorithm Ver: Given a signature S, a message M and a public key y, Ver tests whether S is a valid signature of M with respect to y.

Forgeries and attacks. We now formalize some security notions which capture the main practical situations. On the one hand, the goals of the adversary may be various:

- Disclosing the private key x of the signer. It is the most serieus attack. This attack is termed *Total break*.
- Constructing an efficient algorithm which is able to sign messages with good probability of success is called *universal forgery*.
- Providing a new message-signature pair is called *existential forgery*.

On the other hand, various means can be made available to the adversary, helping her into the forgery. We focus on two specific kinds of attacks against signature schemes:

- *no-message attacks*: the attacker only knows the public key of the signer.:
- *known-message attacks*: the attacker has access to a list of valid message-signature pairs . One of the subclasses is the *adaptive chosen-message attack* (CMA), where the attacker can ask the signer to sign any message of her choice.

When one design a signature scheme, one wants to computationally rule out existential forgeries even under adaptive chosen-message attacks. More formally, one wants that the success probability of any adversary $\mathbf{A}$ with a reasonable amount of time is small, where

$$Succ^{CMA}(\mathbf{A}) = Pr[(x, y) \leftarrow K(1^k), (M, S) \leftarrow \mathbf{A}^{Sig_x}(y) : Ver(y, M, S) = 1]$$

In the signaturen scheme is it difficult to obtain security against existential forgeries. As in public-key encryption, maybe security is possible, but with inefficient schemes.

Basic signature schemes

The plain-RSA signature. Rivest, Shamir and Adleman proposed the first signature based on the "trapdoor one-way permutation paradigm", using the RSA function:

- the key generation algorithm produces a large composite number $N = pq$, a public key e, and a private key d such that $e \cdot d = 1 \ mod \ \phi(N)$.
- The signature of a message M, encoded as an element in $\mathbf{Z}_N$, is $S = M^d \ mod \ N$.
- The verification algorithm simply checks whether $M = S^e \ mod \ N$.

However, the RSA scheme is not secure by itself since it is subject to existential forgery: it is easy to create a valid message-signature pair, without any help of the signer, first randomly choosing a certificate S and getting the signed message M from the public verification relation, $M = S^e \ mod \ N$.

The ElGamal signature scheme. In 1985, ElGamal proposed the first digital signature scheme based on the DL problem, but with no formal security analyses:

- The key generation algorithm produces a large prime p, as well as an element $g \in \mathbf{Z}_p^*$ of large order. It also creates a pair of keys, the private key $x \in \mathbf{Z}_{p-1}^*$ and the public key $y = g^x \ mod \ p$.
- The signature of message M is a pair (r, s) where $r = g^k \ mod \ p$ with a random $k \in \mathbf{Z}_{p-1}^*$, and $s = (M - xr)k^{-1} \ mod \ (p-1)$. This pair satisfies $g^m = y^r r^s \ mod \ p$,
- The verification algorithm checks $g^m = y^r r^s \ mod \ p$,

Unfortunately, as above, existential forgeries are easy.

3.3.3 The random oracle model

As already marked, one often has to made some assumptions about the adversary's behaviour. I will present a classical model, the **random oracle model**: this model was the first to be introduced in the cryptographic community: the hash function is formalized by an oracle which produces a truly random value for each new query. Ofcourse, if the same query is asked twice, identical answers are obtained.

This model has been strongly accepted by the community, and it considered as a good one, in which proofs of security give a good taste of the actual security level. Even if it does not provide a formal proof of security it is argued that proofs in this model ensure security of the overall design of the scheme provided that the hash function has no weakness.

More formally, this model can also be seen as a restriction on the advarsary's capabilities. Indeed, it simply means that the attack is generic without considering any particular instantiation of the hash function. Here is a proof in the random oracle model. 2.16

Summary and Conclusions

Provable Security is originated in the late 80's, applied to enryption and signature schemes. The applicability increased due to the use of ideal substitutes like random oracles vs. hash functions. Provable security is now required by cryptographic standards.

The limits of provable security are the following: first does provable security not provide proofs in the mathematical sense. Which means that proofs are relative (to computational assumptions), that proofs often use ideal models and further that proofs are not formal objects (time is needed for acceptance). But still has provable security provided some good guarantee that a scheme is not flawed.

Chapter 4

Factorization

Written by *Marco Streng*

The RSA algorithm is based on the assumption that factoring large integers is impossible to do in a reasonable amount of time. In this chapter, we will start with an introduction to factorization. After that, we will turn our attention to the two most important factoring algorithms for RSA numbers, the quadratic sieve and the number field sieve. Due to the technical nature of the number field sieve, we will focus mostly on the quadratic sieve.

4.1 Factorization

A *factor* of an integer N is a number d which divides N. We call such a factor *trivial* if it is either 1 or N itself.

Definition 4.1 *A factoring algorithm is an algorithm which finds a nontrivial factor m of N for any composite integer N.*

In the RSA algorithm, we are dealing with an integer N of the form $N = pq$ for two primes p and q. A factoring algorithm is exactly what we need in order to find p and q, because they are the nontrivial factors of N.

A factoring algorithm is also sufficient for finding the prime factorization of any integer, because we can use it together with a prime test and repeat it on the found factor m and on N/m until we only have prime factors left. We refer to [Beu03] and [Tel02, Sec. 5.2] for more about prime testing.

4.1.1 Trial division

Now suppose we are given a number N which we want to factor. As a first attempt, we could look at the numbers $2, 3, 4, 5, \ldots$ and try to divide N by them. This method is called *trial division*. In the worst case, N is the product of two big primes and we would have to try to divide N by all numbers up to $\sqrt{N}$ to find those primes. This costs $\sqrt{N}$ division steps, which is exponential in the length $\log(N)$ of our input N. Because of this huge runtime, this method cannot be used to factor big numbers.

Still, trial division can be very useful to find small factors of N. It takes only B divisions to find all factors up to B, so it takes only polynomial time to find all factors up to $\log(N)$. Therefore, when designing a factoring algorithm, it is safe to assume that the input has no prime factors smaller than its logarithm.

4.1.2 Greatest Common Divisor (GCD)

Instead of using division, we could try using the *greatest common divisor* (GCD). Given two integers a and b, the greatest common divisor $\gcd(a, b)$ of a and b is defined to be the greatest integer which divides both a and b. The greatest common divisor can be calculated in quadratic time with Stein's algorithm [Tel02, Sec. 5.2].

For any integer d, $\gcd(d, N)$ is a factor of N. Now if d has at least one prime factor in common with N, then we know that $\gcd(d, N)$ is greater than one. Also, if N does not divide d, then we know that $\gcd(d, N)$ is not equal to N (because $\gcd(d, N)|d$). So the trick to factoring a number N now seems to be: finding a number d which is not a multiple of N, but does have at least one prime factor in common with N.

4.1.3 Congruent squares

Suppose we know two integers a and b whose squares are congruent modulo N. We then have $N|(a^2 - b^2) = (a - b)(a + b)$, which means that each prime factor of N divides either $a - b$ or $a + b$, so the number $a - b$ might just be what were looking for in the previous section. There were two conditions which that number should satisfy. Firstly, N should not divide it, which is the same as saying that a and b should not be congruent mod N. Secondly, we need $a - b$ to have at least one prime factor in common with N. The last condition is satisfied as soon as N does not divide $a + b$, so we want a not to be congruent to $-b \bmod N$.

Conclusion: In order to find a nontrivial factor of N, all we need is a pair of integers (a, b) with $a^2 \equiv b^2 \bmod N$ and $a \not\equiv \pm b \bmod N$. From this conclusion we can derive the following theorem.

Theorem 4.2 *Suppose N is an odd integer with at least two different prime factors and $a \not\equiv 0 \bmod N$. And suppose b is a random integer with $a^2 \equiv b^2 \bmod N$. Then (a, b) gives rise to a nontrivial factor of N with probability at least $\frac{1}{2}$.*

Proof. If $\gcd(a, N)$ is not equal to one, then it is itself a nontrivial factor of N and we are done, so suppose $\gcd(a, N) = 1$. Then both a and b are in the unit group $\mathbb{Z}_N^*$ of the ring of integers modulo N. Write N as $N = p^k m$ with $\gcd(p, m) = 1$ and p prime. From the conditions of the theorem, we know that p exists and that p and m are at least 3. By the Chinese remainder theorem, we know $\mathbb{Z}_N^* \cong \mathbb{Z}_{p^k}^* \times \mathbb{Z}_m^*$. Now any b with $b^2 \equiv a^2$ modulo both p^k and m will have $b^2 \equiv a^2 \bmod N$. There are the two solutions a and $-a$ modulo p^k and at least the two solutions a and $-a$ modulo m, so there are at least 4 solutions modulo N. Only two of those solutions are the trivial ones $b \equiv \pm a \bmod N$, so at least half of the solutions give rise to a nontrivial factor of N. △

Now if we can generate "random" pairs of integers with congruent squares, then the expected number of pairs needed for a nontrivial factor is two. Many factoring algorithms (including the quadratic sieve and the number field sieve) are ways to generate such "random" pairs.

Casus 4.1: FACTORIZATION BY HAND.

Suppose we want to factor some integer, let's say $N = 491568$, but we don't have a calculator ready. Then how do we proceed? An easy thing to do is just look at the last digit and see if it is a multiple of two and/or five, then divide it out. By repeating this, our example quickly becomes $2^4 \cdot 30723$ and we know 2 and 5 do not divide 3, so they do not divide 30723. Another trick is summing the digits and checking if three or nine divides the sum of the digits. $3 + 0 + 7 + 2 + 3 = 15$ is a multiple of three, but not of nine, so three divides 30723, but nine does not. Now we have $N = 2^4 \cdot 3 \cdot 10241$. Are there any more tricks? Yes, we could take the alternating sum of the digits: $1 - 0 + 2 - 4 + 1 = 0$ is a multiple of 11, so 10241 is a multiple of 11. Dividing by 11 gives $10241 = 11 \cdot 931$. Next: $931 = 1001 - 70$ and 7 divides 70, so 7 divides 931. So we divide 931 by 7 and only have 133 left. We already know that it has no divisors $2, 3$ or 5, so let's look at the next prime: 7. 133 is near $140 = 20 \cdot 7$. In fact, the difference is 7, so $133 = 7 \cdot 19$ and we know $491568 = 2^4 \cdot 3 \cdot 7^2 \cdot 11 \cdot 19$.
How did all these tricks work? Suppose $d_r \cdots d_1 d_0$ is the decimal expansion of N, in other words, suppose $N = \sum_{k=0}^{r} d_k 10^k$. Then $N \equiv d_0 \bmod 10$, so to see if two or five divides N, we only have to look at the last digit of N. When checking for factors three and nine, we summed the digits. Why? Because $10 \equiv 1 \bmod 9$, the sum of the digits of N is $s(N) = \sum_{k=0}^{r} d_k \equiv \sum_{k=0}^{r} d_k \cdot 10^k = N \bmod 9$, so to see if three or nine divides N we only have to look at the sum of the digits. In the same way, because $10 \equiv -1 \bmod 11$, we only have to look at the alternating sum to see if 11 divides N. Because $1001 = 7 \cdot 11 \cdot 13$, we could look at $931 - 1001 = -70$ and see immediately that 7 did divide 931 and 11 and 13 didn't. Other tricks include: recognizing the prime 101 by taking the alternating sum of pairs of digits $\sum_{k=0}^{r/2} (-1)^k (10 d_{2k+1} + d_{2k}) \equiv \sum_{k=0}^{r/2} 100^k (10 d_{2k+1} + d_{2k}) = N \bmod 101$ and using $11 \mid 99$, $37 \mid 999$ and $1001 = 7 \cdot 11 \cdot 13$ in a similar way.

4.1.4 Powers

Before we spend time on finding such squares, let us first look at the conditions in the above theorem. By trial division, we can assume N to be odd, but what if N has only one prime factor? That would mean that it is of the form $N = p^k$, where k is at least two. To eliminate this problem, we will check if N is a power. For $k = 2, 3, 4, \ldots$, calculate the integer nearest to $\sqrt[k]{N}$ and call it x. If N is a k-th power, then $N = x^k$, so x is a nontrivial factor of N and we are done. If we find no nontrivial factors among the x's, then we know that N is not a power. We only have to check this for k up to $\log_2 N$, because $N = x^k$ implies $2^k \leq x^k = N$, so $k \leq \log_2 N$. Because we only check this for $\log_2 N$ values of k and because we can use a fast numerical root-algorithm to calculate the integer-precision k-th roots, the complete power check will only take polynomial time.

4.2 The Quadratic Sieve

The *quadratic sieve* factoring algorithm was introduced in 1982 by Pomerance. The next paragraph tries to reason towards the quadratic sieve in the same natural way as [Pom05].

4.2.1 Square roots

All we had to do to factor a number N, was creating an algorithm which generates random pairs of congruent squares. So we take a number a, square it, and then all we need is a random square root of $(a^2 \bmod N)$ in $\mathbb{Z}_N$.

For example, suppose we wanted to factor $N = 493$ and we have selected the integer $a = 23$. Then $a^2 = 529$. We already know the square root $a = 23$ of 529, and taking the same square root we started with isn't really that random. However, we also have $529 - 493 = 36$ and $\sqrt{36} = 6$, which tells us $23^2 \equiv 6^2 \bmod 493$, giving us the factorization $493 = 17 \cdot 29$. In general, when trying to take the square root of a congruence class $(y \bmod N) \in \mathbb{Z}_N$, we could look at representatives $y \in \mathbb{Z}$ and take their square roots. This doesn't always work. If we square $(24 \bmod 493)$ for example, we get $(83 \bmod 493)$ and $\sqrt{83} = 9.110\ldots$, which is not an integer. What we want is a good chance that the representative y we take is a square. For that, we use the following lemma.

Lemma 4.3 *The smaller a number in $\mathbb{Z}$ is, the greater the chance that it is a square.*

Proof. The squares below an integer m are exactly the squares of the integers below $\sqrt{m}$. Therefore, there are $\sqrt{m}$ squares below m and because the derivative of the square root function decreases when m increases, so does the probability that m is a square. △

Now in order to have a good chance of finding a square root of $(a^2 \bmod N)$, we want a^2 to be congruent to a small positive integer. The trick is now to take a number a, then take the smallest integer $k \equiv a^2 \bmod N$ and then try to extract the square root of k. When a is smaller than $\sqrt{N}$, we just get $k = a^2$ and $\sqrt{k} = a$, which is not useful, but when a is greater than $\sqrt{N}$, we get a number k which is a bit more "random". Therefore, what we will do is look for squares in the sequence $a^2 - N$ for a at least $\sqrt{N}$.

Let's do an example. Suppose we are given the number $N = 943$ to factor. We extract the square root of 943 with some fast and imprecise floating point algorithm and get 30.7, so we calculate

$$\begin{aligned} 31^2 - 943 &= 18 \\ 32^2 - 943 &= 81 = 9^2 \quad \text{a square!} \end{aligned}$$

Now we have $32^2 \equiv 9^2 \bmod N$ and it is clear that $32 \not\equiv \pm 9 \bmod N$. Therefore we calculate $\gcd(N, 32 - 9) = \gcd(943, 23) = 23$ and conclude $943 = 23 \cdot (943/23) = 23 \cdot 41$.

4.2.2 Smooth numbers

Let's try this again with another number to factor. Suppose that this time we are given $N = 1649$ (example from [Pom05]). Now we should look for squares in the sequence

$$\begin{aligned} 41^2 - 1649 &= 32 \\ 42^2 - 1649 &= 115 \\ 43^2 - 1649 &= 200 \\ 44^2 - 1649 &= 287. \\ &\vdots \end{aligned}$$

None of the above numbers are squares, and this list will go on for a while before the first square appears. In fact, finding a square in this list may just take more time than doing trial division

on N. Nevertheless, the number $32 \cdot 200 = 6400 = 80^2$ is a square, so from

$$\begin{aligned} 41^2 &\equiv 32 \pmod{N} \\ 43^2 &\equiv 200 \pmod{N}, \end{aligned}$$

we can conclude $(41 \cdot 43)^2 \equiv 32 \cdot 200 = 80^2 \bmod N$, which gives us a pair of congruent squares which we can use to factor N.

The last example shows that we do not have to go through the complete list of numbers of the form $a^2 - n$: we can multiply some numbers from the list with each other to create squares.

To find subsets of the list with product a square, we use the following characterization of squares: A number y is a square if and only if every prime p has an even index in its factorization. For example, factoring the above list gives

$$\begin{aligned} 41^2 - 1649 &= 32 = 2^5 \\ 42^2 - 1649 &= 115 = 5 \cdot 23 \\ 43^2 - 1649 &= 200 = 2^3 \cdot 5^2 \\ 44^2 - 1649 &= 287 = 7 \cdot 41 \end{aligned}$$

and we can see that $32 \cdot 200$ is $2^8 \cdot 5^2$, so it is a square.

How does this work in general? How do we find a subsequence with product a square and how many squares do we have to calculate before we can get such a subsequence? To answer these questions we will use *smooth numbers*. An integer is called *smooth* if all its prime factors are small. Specifically, a number is called B-smooth if all its prime factors are at most B. Now suppose we have chosen some *smoothness bound* B and have selected only the B-smooth numbers from the above list. Then we should try to find some subsequence with product a square. For that, we will use linear algebra for vector spaces over the field $\mathbb{F}_2 = \mathbb{Z}_2 = \{0, 1\}$ of integers modulo 2.

Suppose we have some sequence $m_1, m_2, \ldots, m_k$ of B-smooth integers. If we denote by $\pi(B)$ the number of primes which are at most B, then the smooth numbers are exactly the numbers m which can be decomposed as $m = \prod_{i=1}^{\pi(B)} p_i^{v_i}$, where p_i is the ith prime. This gives us a unique *exponent vector* $\vec{v}(m) \in \mathbb{Z}^{\pi(B)}$ for each B-smooth number m. A smooth number m is a square if and only if its exponent vector has only even components, so we reduce the exponent vectors modulo 2 and get, for each smooth number m_i, some vector $\vec{w}(m_i)$ in the vector space $\mathbb{F}_2^{\pi(B)}$. A subsequence of m_i with product a square is equivalent to a subsequence of the vectors $w(m_i)$ in $\mathbb{F}_2^{\pi(B)}$ with sum 0, which, for a vector space over the field $\mathbb{F}_2$, is the same as a linear combination of the vectors, which exists as soon as $k > \pi(B)$. We can then use algorithms from linear algebra to find such a linear dependency in the $k \times \pi(B)$-matrix of exponent vectors mod 2. This is called the *matrix step* of the quadratic sieve. More on that in 4.2.5.

4.2.3 Sieving

We now know that the sequence of smooth numbers of the form $n^2 - N$ that we generate, should be $\pi(B) + 1$ integers long. But how do we recognize the smooth numbers? We could use trial division, because after B divisions, we know if a number is B-smooth, and if it is, then we also know the factorization (and the exponent vector).

A better way of recognizing smooth numbers is based on the *sieve of Eratosthenes*, which is an algorithm for generating a list of primes. The sieve of Eratosthenes starts with a list of the integers from 2 to some bound Y. In each step, we take the first unmarked number p and mark every p-th number after p. The numbers we mark are all multiples of p, hence not prime. So we

start with $p = 2$ and mark every second number, starting with 4. Then we sieve with the number 3, marking every third number starting with 6. The number 4 is already marked, so we skip it and sieve with the next unmarked number, 5. When we have done this for every unmarked number up to $\sqrt{Y}$, all unmarked numbers must be prime.

Now, contrarily to the sieve of Eratosthenes, we are not interested in the unmarked numbers. We are interested in numbers with small prime factors, so we want the numbers which are marked a lot of times in the early stage of the sieve. Instead of marking numbers, we will divide them by the prime we are sieving with. In addition, we sieve by higher powers of these primes as well. After sieving with all primes up to B, all B-smooth numbers will be turned into ones. As an example, let us sieve for 5-smooth numbers up to 20. We start with the list

$$1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20$$

and sieve for even numbers including 2 itself, dividing them by 2, which gives

$$1, \underline{1}, 3, \underline{2}, 5, \underline{3}, 7, \underline{4}, 9, \underline{5}, 11, \underline{6}, 13, \underline{7}, 15, \underline{8}, 17, \underline{9}, 19, \underline{10}.$$

Now we sieve for multiples of $2^2 = 4$, and divide every fourth number by 2 as well:

$$1, 1, 3, \underline{1}, 5, 3, 7, \underline{2}, 9, 5, 11, \underline{3}, 13, 7, 15, \underline{4}, 17, 9, 19, \underline{5}.$$

The next power of 2 is 8, so we divide every 8-th number by 2. The last power of 2 below 20 is 16, so we divide the 16-th number by 2 again.

$$1, 1, 3, 1, 5, 3, 7, \underline{1}, 9, 5, 11, 3, 13, 7, 15, \underline{1}, 17, 9, 19, 5.$$

Then we go to the next prime, which is 3 and do the same:

$$1, 1, \underline{1}, 1, 5, \underline{1}, 7, 1, \underline{3}, 5, 11, \underline{1}, 13, 7, \underline{5}, 1, 17, \underline{3}, 19, 5,$$
$$1, 1, 1, 1, 5, 1, 7, 1, \underline{1}, 5, 11, 1, 13, 7, 5, 1, 17, \underline{1}, 19, 5.$$

The last prime which is at most 5 is 5 itself, so

$$1, 1, 1, 1, \underline{1}, 1, 7, 1, 1, \underline{1}, 11, 1, 13, 7, \underline{1}, 1, 17, 1, 19, \underline{1}.$$

The numbers that are turned into ones are exactly the 5-smooth numbers up to 20. If we remember the factors by which we divide the numbers, then we can create the exponent vectors simultaneously without extra cost.

So how much time does this cost? When sieving for multiples of p^k, we are going through the list of integers up to Y with steps of p^k. This takes Y/p^k steps, so the time takes to find all B-smooth numbers up to Y is proportional to

$$\sum_{p \leq B} \sum_{k=1}^{\infty} Y/p^k,$$

which can be proved to be about $Y \log \log B$ ([Pom05]).

This way, we checked the integers $1, 2, \ldots, Y$ for smoothness. What we wanted to check was the first part of the quadratic sequence $a^2 - N$, so we have to adapt the sieve. What made sieving work was the fact that integers which are divisible by p^k occur regularly in the list $1, 2, \ldots, Y$. Fortunately they occur just as regularly in the list of integers of the form $a^2 - N$. Therefore, we first solve $x^2 - N \equiv 0 \bmod p$. This quadratic congruence has at most 2 solutions mod p, and they can all be found using only elementary number theory. If there are two solutions, say a_1, a_2, then we have $p | a^2 - N$ if and only if $a \equiv a_j \bmod N$ for some j. Therefore, we can first sieve for integers that are a_1 mod p, and then for integers that are a_2 mod p. Similar methods can be used for powers of primes. We call this sieve, when adapted to the quadratic case, a "quadratic sieve".

4.2.4 The smoothness bound

There now remains one important optimization problem: How should we choose our smoothness bound B? On the one hand, if B is chosen to be small, then we need to find only a few B-smooth numbers and the matrix will be small. On the other hand, the greater B is, the greater the chance for a random number to be B-smooth. Therefore, we need to choose B optimally.

We will now sketch the reasoning which leads to the optimal choice of B as well as an estimate of the time needed for sieving. Details can be found in [Pom05].

If a runs in the interval $[n^{1/2}, n^{1/2}+n^{\epsilon}]$, then a^2-n is smaller than $(n^{1/2}+n^{\epsilon})^2-n \approx 2n^{1/2+\epsilon}$. Call this bound X. Let $\psi(X,B)$ be the number of B-smooth integers in the interval $[1,X]$. Then $\psi(X,B)/X$ is the probability that a random integer in $[1,X]$ is B-smooth, so we need approximately $X/\psi(X,B)$ random numbers for each B-smooth one. We needed a bit more than $\pi(B)$ random numbers, so we should create about $\pi(B)X/\psi(X,B)$ numbers of the form a^2-N, assuming that they are random enough. According to the previous section, it then takes time $\pi(B)\log\log BX/\psi(X,B)$ to check all these integers for B-smoothness. Now suppose $B=X^{1/u}$ for some fixed u. Then we can use the following estimate from [Pom05],

$$\frac{\psi(X,X^{1/u})}{X} \approx u^u.$$

If we use this estimate together with the rough approximation $\pi(B)\log\log B \approx B$, we approximate the sieving time by

$$X^{1/u}u^u.$$

Now we need to choose u to minimize this expression. [Pom05] gives as optimal value

$$u \sim (2\log X/\log\log X)^{1/2},$$

and uses this to calculate an optimal smoothness bound

$$B = \exp((1/2+o(1))(\log N\log\log N)^{1/2}),$$

and a conjectural asymptotic complexity of

$$X^{1/u}u^u = \exp((1+o(1))(\log N\log\log N)^{1/2}). \tag{4.1}$$

4.2.5 Matrix step

Now let's get back to the matrix step. We have a binary matrix which is about $B\times B$ and all entries are zero or one. In fact most entries are zero. There are numerous algorithms from linear algebra for finding a dependency in such a matrix and we will not discuss them here. Instead, we refer to [Pom05] for more details and note that, asymptotically, it is faster than sieving.

4.3 Overview

4.3.1 Complexity

In order to express the asymptotic complexity of current factoring methods, one usually uses the function

$$L_x[v,\lambda] = \exp(\lambda(\log x)^v(\log\log x)^{1-v})$$

with parameters v, λ. For $v = 0$, this function is just $L_x[0, \lambda] = \exp(\lambda(\log\log x)) = (\log x)^\lambda$, which is polynomial in the length of x. But for $v = 1$, it is $L_x[1, \lambda] = \exp(\lambda(\log x))$, which is exponential. It is common to write $L_x[v, \lambda]$ instead of $L_x[v, \lambda + o(1)]$. Now for the quadratic sieve, from (4.1), we have a conjectured complexity of $L_N[\frac{1}{2}, 1]$, which seems to be somewhere in between the quadratic and the exponential with $v = \frac{1}{2}$. Many other factoring algorithms also have $v = \frac{1}{2}$, and this seemed to be a lower bound for the runtime of factoring algorithms. That was until the introduction of the number field sieve.

4.3.2 Algorithm outline

Now before we get to the number field sieve, let us first look at the big picture so far. Many algorithms, including the quadratic sieve and the number field sieve, work in a similar way. By theorem 4.2, it suffices to be able to construct some solutions to $x^2 \equiv 1 \bmod N$ in an apparently random manner. To construct solutions like that, many algorithms follow these three steps:

Step 1. *Select a factor base.* Select a finite set of non-zero elements $a_1, a_2, \ldots, a_I \in \mathbb{Z}_N$. We call this set the factor base. In the quadratic sieve algorithm, the factor base was formed by the congruence classes of the primes $\leq B$.

Step 2. *Collect relations.* Collect relations between the elements of the factor base. By a relation, we mean a vector $v = (v_1, \ldots, v_I) \in \mathbb{Z}^I$ for which

$$\prod_{i=1}^{I} a_i^{v_i} = 1. \tag{4.2}$$

Stop as soon as the collection V of found relations contains a bit more than I elements.

Step 3. *Find dependencies.* Find dependencies modulo 2 among the elements of V. A dependency is a subset W of V such that $\sum_W v = 2 \cdot w$ for a vector $w \in \mathbb{Z}^I$. If we take x with $(x \bmod N) = \prod_{i=1}^{I} a_i^{w_i}$, we get

$$\begin{aligned}(x^2 \bmod N) &= \prod_{i=1}^{I} a_i^{2w_i} = \prod_{i=1}^{I} a_i^{\sum_W v_i} = \prod_{i=1}^{I}\prod_{W} a_i^{v_i} \\ &= \prod_{W}\prod_{i=1}^{I} a_i^{v_i} = 1.\end{aligned}$$

This step is also called the matrix step. See 4.2.5.

In the quadratic sieve algorithm, relations were of the slightly different form

$$\prod_{i=1}^{I} a_i^{v_i} = (x^2 \bmod N),$$

where the vector v is the exponent vector of $x^2 - N$. A dependency from step 3 then gives a solution to $a^2 \equiv b^2 \bmod N$ instead of $x^2 \equiv 1 \bmod N$, but aside from that, the algorithm follows the same principle.

Because the number of elements of V is greater than I, we can always find a dependency in step 3. From the way the relations are generated in a good algorithm, we expect x to be random and thus have probability $\frac{1}{2}$ to yield a nontrivial factor of N. This assumption is backed up by experiments for both algorithms mentioned in this text. Now if x does not give a nontrivial factor of N, we can repeat step 3, finding another dependency in the matrix, without repeating step 2. To make this possible, we created not just $I + 1$, but a little more relations in step 2.

Lots of factoring algorithms consist of choosing a factor base of I elements, followed by collecting a bit more than I relations.

4.4 The Number Field Sieve

The *number field sieve* also follows this outline. It is the first factoring algorithm to break the $v = \frac{1}{2}$ barrier from 4.3.1 and it is the fastest known algorithm for factoring large hard composite numbers. In order to understand the details of the number field sieve, one has to know something about algebraic number theory, so we will not go through any of the details of this algorithm. Instead, we will tell something about number fields, motivate the main idea of the algorithm and finish with some references for further reading.

4.4.1 Number Fields

In this section, we will try to give a basic notion of fields and number fields. For details or proofs, we refer to any introduction to algebra or number theory.

By a *ring*, we mean a set with addition, subtraction and multiplication that satisfy some axioms which we will not discuss here. Examples are the rings $\mathbb{Z}$ of integers, $\mathbb{Q}$ of rationals and $\mathbb{R}$ of reals, and the ring of integers modulo a number n, $\mathbb{Z}_n$. A *field* is a ring in which multiplication has an inverse, ie. a ring in which division is possible. Examples are the fields $\mathbb{Q}$ and $\mathbb{R}$ and the finite field $\mathbb{F}_p = \mathbb{Z}_p$ for p prime.

By a field extension of a field K, we mean a field L which contains K. We call the extension finite when L is a finite-dimensional vector space over K.

Definition 4.4 *A number field is a finite extension of the field of rational numbers, $\mathbb{Q}$.*

Example 1. $\mathbb{Q}$ is a number field.

Example 2. Let $\mathbb{Q}(i) = \{a + bi \in \mathbb{C} : a, b \in \mathbb{Q}\}$. This is a field which contains $\mathbb{Q}$ and is a two dimensional vector space over $\mathbb{Q}$ with basis $\{1, i\}$. Therefore, it is a number field.

Example 3. For any irreducible polynomial $f = a_n X^n + \cdots + a_0$ over $\mathbb{Q}$, we can take a zero α of f in $\mathbb{C}$. We can then create a field $\mathbb{Q}(\alpha)$ which contains $\mathbb{Q}$ and α. In this field, we have $0 = f(\alpha) = a_n\alpha^n + \cdots + a_0$, hence $\alpha^n = -1/a_n(a_{n-1}\alpha^{n-1} + \cdots + a_0)$. The field $\mathbb{Q}(\alpha)$ is an n dimensional vector space with basis $\{1, \alpha, \alpha^2, \ldots, \alpha^{n-1}\}$, so it is a number field. For example, the polynomial $f = X^2 + 1$ is irreducible and has a zero $i \in \mathbb{C}$. This gives us the field $\mathbb{Q}(i)$ from the previous example.

Definition 4.5 *A number ring is a subring of a number field.*

Example 1. $\mathbb{Z}$ is a number ring.

Example 2. $\mathbb{Z}[i] = \{a + bi \in \mathbb{C} : a, b \in \mathbb{Z}\}$ is a subring of the field $\mathbb{Q}(i)$.

Example 3. For any irreducible polynomial over $\mathbb{Z}$ of the form $f = X^n + a_{n-1}X^{n-1} + \cdots + a_0$, take a zero α of f in $\mathbb{C}$. We can then create a ring $\mathbb{Z}[\alpha]$ which contains $\mathbb{Z}$ and α. In this ring, we have $0 = f(\alpha) = \alpha^n + a_{n-1}\alpha^{n-1} + \cdots + a_0$, hence $\alpha^n = -(a_{n-1}\alpha^{n-1} + \cdots + a_0)$. This ring is a number ring, because it is a subring of $\mathbb{Q}(\alpha)$.

4.4.2 The idea of the number field sieve

The number field sieve uses smooth numbers in number rings different from $\mathbb{Z}$, an idea that was proposed by Pollard in 1988. The complete algorithm was introduced in 1990 by Lenstra, Lenstra, Manasse and Pollard. Since then, many people have contributed to theoretical and practical improvements.

Casus 4.2: Non-unique Factorization.

In the number ring $\mathbb{Z}$, any element n has a factorization into irreducible elements of the ring. This factorization is unique, up to the order of the factors and multiplication of the factors by ± 1. For example, the factorizations $12 = 2 \cdot 2 \cdot 3$ and $12 = 3 \cdot (-2) \cdot 2$ are the same.
In general, however, this is not always true. There are number rings in which we don't have unique factorization. Take the ring $\mathbb{Z}[\sqrt{-19}] = \{a + b\sqrt{-19} : a, b \in \mathbb{Z}\}$ for example. In this ring, 20 factors as $2 \cdot 2 \cdot 5$, where both 2 and 5 are irreducible. On the other hand, $(1 + \sqrt{-19})(1 - \sqrt{-19}) = 1^2 - \sqrt{-19}^2 = 1 + 19 = 20$ gives another factorization of 20.
In number theory, this problem is solved in two steps. The first step is working with so called *ideals* of the ring, instead of elements. This is called ideal arithmetic. The second step is enlarging the ring by adding some more elements to it. After enlarging the ring to the so called *ring of integers* $\mathcal{O}$ of $\mathbb{Q}(\alpha)$, we have unique factorization of ideals into *prime ideals*. Finding the ring of integers is a nontrivial topic in algebraic number theory.

The idea of the number field sieve is to construct a number field $K = \mathbb{Q}(\alpha)$ and a map ϕ from the ring $\mathbb{Z}[\alpha]$ to the ring $\mathbb{Z}_N$. This map should be a ring homomorphism, which means that it preserves addition and multiplication: $\phi(x + y) = \phi(x) + \phi(y)$, $\phi(xy) = \phi(x)\phi(y)$, $\phi(0) = 0$ and $\phi(1) = 1$.

If $\phi(\alpha) = (m \bmod N)$ for a small number m, then we have $\phi(a + b\alpha) = (a + bm \bmod N)$. The idea is to find coprime a, b where both $a + b\alpha \in K$ and $a + bm \in \mathbb{Z}$ are smooth. (We will leave a definition of smoothness in K to more advanced texts). Then we can factor $a + b\alpha$ in K and $a + bm$ in $\mathbb{Z}$. If we apply ϕ to the factorization of $a + b\alpha$, we get a product of classes in $\mathbb{Z}_N$. On the other hand, if we reduce the factorization of $a + bm$ modulo N, we get another product of classes in $\mathbb{Z}_N$. Now we have two products in $\mathbb{Z}_N$ that are the same. If we divide those products, we get a relation like (4.2).

The above suggests that we take a factor base containing $\phi(\pi)$ for all "small" primes π of $\mathbb{Z}[\alpha]$ and $(p \bmod N)$ for all small primes p of $\mathbb{Z}$.

The first step of the number field sieve is choosing a polynomial f. Then we need the function ϕ, for which there must be a class $(m \bmod N) = \phi(\alpha)$ for which $(f(m) \bmod N) = \phi(f(\alpha)) = \phi(0) = (0 \bmod N)$, ie. $f(m) \equiv 0 \bmod N$. This means that we should choose f in such a way that we know a small zero m of $(f \bmod N)$. Now there are already some questions that need to be asked, like "how do we choose a polynomial?", "what if the number ring does not have unique factorization?" (casus 4.2) and "what are the primes of the number ring?" The answers to those questions, as well as the actual sieving procedure, are too advanced for this text and a course in algebraic number theory is recommended.

The number field sieve has the advantage over the quadratic sieve that it generates much smaller numbers, which therefore have a much greater probability to be smooth. This means that it needs to generate less numbers and is much faster. The number field sieve has a conjectured asymptotic complexity of $L_N[\frac{1}{3}, 1.9019]$ for arbitrary integers and $L_N[\frac{1}{3}, 1.5263]$ for integers of the special form for which it was originally created. The fact that it has $v = \frac{1}{3}$, where the quadratic sieve and other algorithms have $v = \frac{1}{2}$, makes it the fastest algorithm for factoring integers of more than 120 digits.

For an introduction to the number field sieve, we refer to both [LHL93] and [Ste05], which are both very clear and provide different approaches to explaining the ideas behind the algorithm. The first also provides some enhancements of the sieve and also contains the implementation that was used for the record factorization of $2^{523} - 1$.

Summary and Conclusions

The table on the right shows the records of the RSA Challenge, a factorization race organized by RSA Laboratories (see also [Tel02, Kader 14.1]). So far, the largest of these challenges to be factored was RSA-576, a 576-bit integer, which is a clear sign that 512-bit RSA is no longer secure.

Year	Number	Digits
1991	RSA-100	100
1992	RSA-110	110
1993	RSA-120	120
1994	RSA-129	129
1996	RSA-130	130
1999	RSA-140	140
1999	RSA-155	155
2003	RSA-160	160
2003	RSA-576	174

Until the factorization of the 129-digit number RSA-129 in 1994, all challenges were won with the quadratic sieve. All greater RSA-challenges were won using the number field sieve. Because of enhancements made to the algorithm and improvements of hardware, the size of the numbers we can factor increases a few digits a year, so it will take a long time before the number field sieve will be able to crack 1024-bit RSA.

A totally different kind of threat to RSA may lie in the development of quantum computers. A quantum computer algorithm called *Shor's algorithm* ([Sho97]), factors integers in polynomial time and space. This means that the development of a large-scale quantum computer would mean the end of RSA. There are, however, a number of practical difficulties in constructing a quantum computer. The successful factorization of the integer $15 = 3 \times 5$ by IBM in 2001 gives a good indication of what is currently possible with Shor's algorithm.

Therefore, it seems that 1024-bit RSA will be secure for a long time. We cannot, however, make any predictions about the development of totally new factoring algorithms and new branches of mathematics that are yet to be uncovered.

Chapter 5

How Improving makes it Worse

Written by *Eric Ka Yan Ho*

The main goal of this chapter is to raise the reader's awareness when he or she is confronted with (theoretical) cryptographic improvements that claim to lead to a more reliable, secure and robust cryptosystem. As we will see, these claims are not always correct as the suggested improvements may even harm the overall system security in different ways. We will discuss several kinds of these enhancements and present some methodological ways to analyse the security of cryptosystems.

5.1 Introduction

The notion of security in a public cryptosystem is usually based upon some mathematical and computational assumption. In order to verify the alledged amount of security that is offered by a specific implementation of a cryptographic scheme, it is neccesary to investigate all the (computationally) possible attacks that can be applied to this system. Identifying these attacks allow us to suggest improvements that could seal off the security gaps exploited by these attacks. Since the introduction of public cryptographic systems, this process of detecting gaps and repairing them has been and always will be a neverending story. It is therefore needless to say that up till now countless improvements have been formulated for innumerable cryptographic mechanisms in order to obtain even better security, reliability and useability than before.

However, contrary to the common belief that improvements always provide a better system, some enhancements do not neccesarily imply that the level of security has increased. It may even be the case that some of these improvements that appear profitable at first sight, can damage the system in such a way that other attacks, more insidious and more difficult to identify, can be unleashed.

There are many kinds of improvements thinkable. Some might be based upon some theoretical ground while others might not even be related to the cryptosystem itself but, for instance, applied to the hull of the system which envelops the specific cryptographic scheme. The article of [LN02] is a good example of the second kind of improvement where we see that a cryptosystem's security is not only based on the cryptographic scheme itself but also on its interaction with the working enviroment. Although there many more of such cases to think of, we will limit our scope to the first kind of improvements.

This chapter is primarily based upon [JQYY02] and the first part (5.2) of this chapter deals with the content of this paper. The authors of this article confront us with several security paradoxes that show us how a seemingly beneficial improvement to a cryptosystem may in fact weaken it when used in practice. Their research findings are based upon the well known RSA system due to its widely use and acceptability as a public key cryptosystem. Also, we would like to focus your attention towards methodologies that can help us identify such paradoxes more easily before they have any chance of being considered or implemented into public cryptosystems. We will discuss this topic in the second part (5.3) of this chapter, which is based upon [Nao03] and [SSSW98].

5.2 Paradoxical Improvements based on RSA

Because of its popularity, the RSA system is presently one of the best analyzed schemes and many optimisation measures have been suggested for it. These measures could be designed, for instance, with the premisse of offering a better error detection, a more robust security against active attacks or an increase in the size of the security parameters. In this section we wish to illustrate a simple example of how an improved RSA scheme may allow adversaries to succesfully initiate attacks with weaker assumptions needed than was possible with the original RSA scheme.

5.2.1 Attack Model

Our enviroment consists of different RSA versions, each being an universal improvement over the previous version namely RSA in standard mode, RSA enhanced with CRT (Chinese Remaindering Theory) and RSA enhanced with CRT and a Fault Analysis Prevention method. To show that a specific improvement may allow attacks with weaker assumptions than before we need to define the capabilities and resources of our adversary:

1. *Induced Faults* - Our adversary may inject faults into the system. To be more specific, she is able to change bits to their complementary value.

2. *Limited Feedback Channel* - Our adversary can only receive limited feedback from the system. Adversaries may thus receive indications about the kind of fault due to a change of behaviour of the overall system but they don't get actual outputs from the decryption device.

Using this profile of our adversary, we will devise an attack on the three RSA systems. The goal of our attacker will naturally be to uncover the value of the secret decryption key. We will see that with each improvement, the attacker can use weaker assumptions to reach his goal.

5.2.2 RSA (Standard Mode)

Given the RSA system in standard mode, the encryption and decryption process are resp. defined as $c = m^e \ mod \ n$ and $m = c^d \ mod \ n$, where c, m are resp. the cipher - and plaintext; e, d are resp. the secret decryption key and public encryption key; n is the public RSA modulus whose factors are the secret primes p and q.

Suppose the adversary introduces an error during the decryption process. More specifically, the adversary flips a random chosen bit, d_i for instance, to its complementary value. The decryption process will then output the corrupted value $m' = c^{d'} \ mod \ n$. Now, let us rewrite the decryption key d as its binary expansion. We get:

$d = \sum_{i=0}^{k-1} d_i \ 2^i$

This allows us to rewrite the corrupted decryption key d' as:

$d' = \sum_{i=0,(i\neq j)}^{k-1} d_i \ 2^i + \overline{d}_j \ 2^j = d + (\overline{d}_j - d_i) \ 2^j$

We derive the following from this:

$$\frac{(m')^e}{c} \equiv c^{e(\overline{d}_j - d_i)2^j} \equiv \begin{cases} c^{e2^j} (\text{mod n}) & \text{if } d_i = 0 \\ 1/c^{e2^j} (\text{mod n}) & \text{if } d_i = 1 \end{cases} \tag{5.1}$$

So, by exhaustively checking whether $\frac{(m')^e}{c} \equiv c^{e2^j} (\text{mod n})$ holds for some $0 \leq j \leq k-1$, our adversary can retrieve the value of d_j and this attack is easily generalised to the case where several bits of d have 'flipped'. Note that we do not need to know the original value of d_i. Of course, this attack is only possible if and only if our adversary can get access to the value of m'.

5.2.3 RSA - CRT

A RSA system enhanced with CRT allows the decryption process to be speeded up by a factor of 4 because we are able to spread the computations over the secret primes p and q. The decryption process is defined as $m = m_p + p[p^{-1}(m_q - m_p) \ mod \ q]$ where $m_p = c^{d_p} \ mod \ p$, $m_p = c^{d_p} \ mod \ p$, $d_p = d \ mod \ (p-1)$ and $d_q = d \ mod \ (q-1)$.

Suppose we initiate the same kind of attack on the RSA - CRT scheme as we did in the previous section. This time, such an attack may even have worse effects than in previous case! Let us assume that our adversary has corrupted a value in the computation of m_p but not in m_q. The CRT calculation will then be $m' = m'_p + p[p^{-1}(m_q - m'_p) \ mod \ q]$ instead of m. Since $m \not\equiv m' \ (mod \ p)$ and $m \equiv m' \ (mod \ q)$, it follows that $gcd(m'_e - c(mod \ n), n)$ gives us the secret prime factor q.

This attack can be considered stronger than the one in the previous section because there is no assumption about the kind of (induced) error: Any error during computation of the exponentatio modulo one prime factor will result in success for our adversary. Given the secret prime factors p and q, it is possible to derive the secret decryption key d. Note that the attack is still possible if and only if our adversary can get access to the value of m'.

5.2.4 RSA - CRT with Fault Analysis Prevention

In order to solve the problem of the previous subsection, Shamir has suggested an elegant method to protect against failure models: Given a random small integer r, let the decryption algorithm compute $m_{rp} = c^{d_{rp}} \ mod \ rp$ and $m_{rq} = c^{d_{rq}} \ mod \ rq$, where $d_{rp} = d \ mod \ \phi(rp)$ and $d_{rq} = d \ mod \ \phi(rq)$. Now, if $m_{rp} \not\equiv m_{rq} \ (mod \ r)$ then there is an error in the calculation and instead of outputting the corrupted value m', the system outputs an error message. If this is not the case, then the computation was faultless and the plaintext m is recovered by applying CRT on $m_p = m_{rp} \ mod \ p$ and $m_q = m_{rq} \ mod \ q$. Note that the probability of an error being undetected is equal to $1/r$ so if r is a 20 - bit integer then the probability of this happening is smaller than 10^{-6}.

Input: $c,\ d = (d_{k-1}, .., d_0)_2,\ n$; Output: $A = c^d\ mod\ n$;

$A \leftarrow 1$; $B \leftarrow c$;
for i from 0 to (k - 1)
    if $(d_i = 1)$ then $A \leftarrow AB\ mod\ n$; //Go to 5.2
    $B \leftarrow B^2\ mod\ n$;
end;

Algorithm 5.1: RSA Exponention with Square - and - Multiply, step 1.

Given this suggestion we could conclude that RSA - CRT enhanced with the Fault Analysis Prevention method of Shamir would be a logical choice. This is entirely not the case as we will see next. This time, our adversary is capable of inducing faults as before but the feedback is only limited to observing whether a ciphertext decrypts correctly or not. The attack is based on the fact that RSA exponentiation is usually implemented with the square - and - multiply technique where multiplication and modular reduction are interleaved to fit the word - size $\Omega = 2^\omega$. These two parts are depicted in 5.1 and 5.2.

Suppose an adversary wants to guess the value of bit d_i and that this value is 1. Also, let's assume that one or several bits of errors have been introduced into some words A_j of register A where $j > j_\tau$ and j_τ is the current value of the counter j when the faults are introduced. Because the words containing the errors are no longer required by the next iteration, the computation of $R = AB\ mod\ n$ will still be correct when you traceback to the output of 5.2. This result is then restored into $A = c^d\ mod\ n$ of 5.1 where the output will be correct too. However, if $d_i = 0$ then the if - statement to the interleaved multiplication will be bypassed and A will still contain the error(s) while the calculation of B continues. This will result in an incorrect value of the final result $c^d\ mod\ n$.

Given the assumption that our adversary knows when a ciphertext decrypts correctly or not, she will know the value of d_i dependent on whether the system returns an error message or not. Note that this attack does not require our adversary to know the value of m' and is therefore considered to be an attack that uses weaker assumptions than the attacks mentioned in earler subsections.

Input: $A = (A_{t-1}, .., A_0)_{2^\omega},\ B,\ n$;
Output: $R = AB\ mod\ n$;

$R \leftarrow 0$;
for j from (t - 1) downto 0
    $R \leftarrow (R2^\omega + A_j B)\ mod\ n$;
end;

Algorithm 5.2: RSA Exponention with Square - and - Multiply, step 2.

5.2.5 Observability of the System

We have seen in the previous subsection how improving the reliability of a system may weaken it. In the paper of [JQYY02] we are confronted with two other examples comparable with the one outlined above. These other two examples show us how:

1. a system (for example, 'OAEP'), which is more secure against active attacks may, in fact, be weaker.

2. a system (for example, 'RSA for Paranoids') with increased length in its parameters (which possibly make the underlying problem harder) may, in fact, be weaker.

As the authors state, these paradoxes are mostly the result of the fact that the observability of the system has increased. The enhanced system with Fault Analysis Prevention acts as an oracle in all the cases, from which an adversary may collect secret information. One might remark that we could recompute the answer instead of returning an error message but this will also be noticed by our adversary since the computation will take longer to complete. This fact can then be revealed by a timing analysis, leading to the same sort of attack.

5.3 Methodologies towards Secure System Design

The mentioned paradoxes are just a few examples of improvements that decrease the system's security. Even though one might question whether these cases of decreasing security levels are out of the extraordinary, we are able to find cases where improving doesn't result in any higher level of security. For instance, the Xinmei scheme discussed in [XDT03] is an example of a signature scheme based on error - correcting codes whose improvement, the Alabbadi - Wicker scheme, did not result in an increase of security (although it did defeat the attacks on the original scheme): Both methods still do not have the important property that it should be infeasble for an attacker to find a signature scheme that passes the verification step given only this verification algorithm. The difficulty thus lies in identifying the security gaps in these cases. This can be illustrated with the following situation:

Suppose there are two experienced cryptographers namely Alice and Bob. One day, Alice barges into Bob's office and tells him that she has designed a new signature scheme with an 120 bits long public key and where the signatures are 160 bits long. Bob sceptically asks Alice what kind of computational assumption this scheme is based on. Alice then explains that it is based on a new trapdoor permutation f_k and a new hash function h and the assumption that after given f_k (and not the trapdoor information) and many pairs $(m_i, f_k^{-1}(h(m_i)))$, it is still hard to come up with a new pair $(m, f_k^{-1}(h(m)))$. The difficult question that Bob should ask himself now is whether the assumption of Alice is actually the same as the assumption "This signature scheme is secure".

If identifying 'dangerous' improvements is already hard to do then it also means that designing a secure cryptosystem is a very difficult task and it remains doubtful if there can ever be a single, uniformed way of engineering such a cryptosystem. Because security is context - sensitive we must take into account the relation between the basic design with the implementation, operational environment and the potential adversaries at each stage of the engineering scheme. Still, if we are able to define methodologies that help us quantify the overall security level of cryptographic schemes, we would hopefully stumble lesser upon 'dangerous' improvements as illustrated in the previous section. We will now introduce two of these methodologies, each operating on another system level.

5.3.1 On the Analysis of Cryptographic Assumptions

This subsection deals with the content of [Nao03]. The author proposes a methodology which allows us to classify computational assumptions based on the complexity of being able to falsify them. This is done by creating a certain challenge to their validity.

When we want to evaluate the validity of an assumption then we would like to be able to falsify it in a constructive manner. In this case, we should also consider the complexity of this task with respect to the assumption in order to say something about the level of acceptability of the assumption. The methodology describes the use of a public challenge, published by a system designer, which is solvable in a time related to the time of breaking the assumption when the assumption is false. The idea underlying this public challenge is based upon the principle of public evaluation. Another important entity in this methodology is the *falsifier* whose objective is to solve the challenges posed. What the adversary is to the assumption, the falsifier is to the challenge. One particular difference between these two is that if an assumption false, the probability of success might still be rather small, whereas a falsifier needs a much higher probability of success, preferably close to 1.

Three levels of falsifiability are introduced in this paper namely *efficiently falsifiable*, *falsifiable* and *somewhat falsifiable*. The differences between them are mainly the time it takes to verify a solution to the challenge. An adversary α to assumption A with parameters (n, t, ε) means that α working in time at most t can break the assumption on instances of size n with probability better than ε. Another parameter we need is δ which is an upperbound on the failure of the challenge (i.e. even if the assumption is false, there might be a chance, bounded by δ that the challenge is not solvable).

Definition 5.1 *Efficiently Falsifiable. An (n, t, ε) assumption A is efficiently falsifiable if there is a distribution on challenges D_n and a verification procedure $V : \{0,1\}^* \times \{0,1\}^* \longmapsto \{accept, reject\}$ such that*

- *Sampling from D_n can be done efficiently, in time polynomial in n, log $1/\varepsilon$ and log $1/\delta$.*
- *The run time of V is polynomial in n, log$1/\varepsilon$ and log $1/\delta$.*
- *If the assumption A is false then there exists a falsifier β that for a $d \in_R D_n$ finds a solution y for which $V(d,\ y)$ is 'accept' with probability at least $1 - \delta$ and the runtime of β is t' which is polynomial in the runtime of α als well as n, log $1/\varepsilon$ and log $1/\delta$.*
- *If there is a falsifier β that solves random challenges $d \in_R D_n$ in time t with probability at least γ then there is a $(t',\ \varepsilon')$ adversary α for breaking the original assumption A where t' and ε' are polynomially related to t and γ.*

Definition 5.2 *Falsifiable. An (n, t, ε) assumption A is falsifiable if everthing is as in 5.1, except that the runtime of sampling from D_n and V may depend on $1/\varepsilon$ (rather than log $1/\varepsilon$).*

Definition 5.3 *Somewhat Falsifiable. An (n, t, ε) assumption A is somewhat falsifiable if everthing is as in 5.1, except that the runtime of sampling from V may depend on $1/\varepsilon$ (rather than log $1/\varepsilon$) and the runtime of β. In particular this means that V may simulate β.*

As an example, we discuss one - way permutations and ask ourselves in what class the assumption "this scheme is a secure implementation of one - way permutation" lies. Given a function$f : \{0,1\}^n \longmapsto \{0,1\}^n$, is it a one - way permutation, i.e. no t time algorithm can find x for a random $y = f(x)$ with probability better than ε. The authors claim that the assumption

"'f is a one - way permutation" is efficiently falsifiable. This is true for similar reasons as factoring: Choose a set of challenges specified as the range of a function. Let H be a family of pairwise independent hash functions, for $h \in H$ we have $h : \{0,1\}^n \longmapsto 0,1^{n-log\ n+2log\ \varepsilon+log\ \delta}$. The challenge is specified by $h \in_R H$ and $c \in \{0,1\}^{n-log\ n+2log\ \varepsilon+log\ \delta}$. A solution x is such that $f(x) = y$ and $h(y) = c$ holds. Note that we are not checking the permutation part i.e. f is indeed 1 - 1 and length preserving.

There are still many open and challenging problems that, at the moment, are not handled by this methodology. An example of a challenging problem is physical security: We have dealt so far with the usual mathematical abstraction of a distributed system and ignored the physical implementation. However, as we have seen in 5.2, many adversaries will probably take advantage of aspects like power consumption, tamper resistance and observability which make the system vulnerable. Ideally, we would like to tackle this problem somewhat in the same manner as we did with computational assumptions. The proposed classification scheme also has a few drawbacks in the sense that it cannot yet handle some rather important issues. An example of such an issue is that implication is not preserved: If assumption A implies assumption B and assumption A is efficiently falsifiable, then we cannot conclude that B is also efficiently falsifiable due to the reason that we cannot use the challenges of assumption A for assumption B. What's more, this classification scheme does not yet highlight all major results in cryptography like the result of HILL, stating that it is possible to construct a pseudo - random sequence generator from any one - way function.

5.3.2 On the Analysis of Overall System Security

Looking closer at the physical security is done in [SSSW98]. In this paper we take a software engineering analysis approach towards discovering weaknesses in a system, to prioritize weaknesses in terms of relative dangers to the system and to determine cost - effective countermeasures. These cost - effective countermeasures could be technical or non - technical of nature (for instance, education). In order to do so we need models which describe our adversaries and the vulnerability of the system during its lifespan.

Adversary Model

This model characterises adversaries in terms of resources, access, risk tolerance and objectives. In other words, this model describes what the means / tools of the adversary are, what the adversary is willing to do and what the goal is of the adversary. This model is dependent on the system, of course. There exists many adversary variants but we assume that a rational adversary follows the path of least resistance i.e. he chooses an attack that maximises his expected return given his budget constraints of resources, access and risk. Note that financial resources are not directly modelled in this representation of the adversary since they are used to buy either capabilities or access.

Vulnerability Model

To map the vulnerabilities of the sytem, the authors use the metaphor of a vulnerability landscape where the valleys and peaks represent respectively the vulnerabilities and countermeasures. To understand this vulnerability landscape, the designer maps out all things related to the target system, which includes the system itself, physical facilities, personnel and their responsibilities, equipment and procedures (protocols). As the system evolves over time, the landscape also evolves accordingly in response. The changes in this landscape are accumulative in the sense

that changes in the system do not lead to the recreation of the entire landscape but can be added to the existing landscape.

Any succesful attack has three steps:

1. Diagnosis of the system to identify a possible attack.
2. Gaining the neccesary access.
3. Executing the attack.

One might reason that in order to protect the system, one needs to block only one of these steps. The goal of the defender or the system designer is then to find the most cost - effective way to block any potential attack.

The vulnerability model sorts out the weak points of the system by access. To identify all the opportunities for adversaries to attack, the designer must consider the entire life cycle of each component of the security environment. Assume for instance, an adversary who decides to bug the telephones destined for a company's offices. There are many attack opportunities and shows that office equipment is vulnerable during its entire lifespan: From the drawingboard to the manufacturing plant and from the loading docks to the work place. Even after disposal we cannot guarantee that the level of vulnerability is gone with the system. This example extends to the virtual working environment: Designers can leave exploitable bugs in the system and adversaries can write virusses that are contained within electronic mails as an executable.

There are two access categories namely physical security and the trust model. Physical security speaks of the problem of enforcing the notion of ownership and protecting your assets, in physical as well as virtual sense. The trust model represents how an organisation determines whom to trust with its assets, like employees. These two aspects are primarily handled by indentification and signature schemes.

Methodology Overview

Given the above models, we can characterise attacks and choose rational countermeasures. This methodology is based upon an "attack tree model" which is an visualisation tool to enumerate and weigh different attacks against the system and consists of five steps:

1. Create the attack trees for the system The system engineer creates an attack tree by replicating the work of an adversary to find weak points in the system. The root of the tree is the component that prompted the analysis. The child nodes are then created by decomposing the parent node into its life cyle. Each phase in the lifecycke breaks down into the two access categories of physical security and the trust model. This process can be repeated and the analysis terminates in a series of vulnerability leaves.
2. Apply weights to the leaves For each leaf in the tree, quantitative values are assigned for risk, access and cost to the adversary. For example, a possible node for encrypted messages would be passive collection of brute force decryption of the ciphertext. Here, the risk and access are low and the costs depend on the cryptographic algorithm.
3. Prune the tree so that only exploitable leaves remain Executing the attack.
4. Generate corresponding countermeasures This is entirely determined by the system engineer.
5. Optimise countermeasure options Countermeasures are ranked according to five attributes:

(a) the cost to purchase and run

(b) the ease of use

(c) the compatibility with in - place technology and ability to interoperate with other communities of interest

(d) its overhead on system resources

(e) time to market or availability

Finally, the desinger attempts to deploy the most cost - effective set of countermeasures. An extensive example is discussed in [SSSW98]. Of course, this methodology only provides a means for enhancing the creativity of the analyst in finding vulnerabilities and countermeasures. Achieving optimal coverage is still very hard to do.

Summary and Conclusions

We have shown that a straightforward improvement of a cryptosystem that may look beneficial at first sight may decrease the level of security even more than in the case when the improvement was not implemented. We then introduced two methodologies on two different levels who, with their combined effort, can assist us in identifying these paradoxes more easily. Still, much work remains to be done in this area.

Chapter 6

E-Cash

Written by *Maarten van Steenbergen*

E-Cash is a collective name for all sorts of electronic money. A small amount of data can be used to buy things on the internet, or be passed along to other consumers. Smart encoding is needed to ensure privacy of the user and protection against fraud.

6.1 Potential problems with Off-Line payments

Electronic cash cryptographic systems can roughly be divided in On-Line and Off-Line systems. In almost all systems, we identify three parties: A user (or payer) who wants to buy something, a shop which wants to sell and receive money, and a bank which looks after the money in the meantime and makes a huge profit along the way.

With On-Line systems, a user is identified the moment she makes the payment. First, her balance is checked, and if it is sufficient, the payment will be done and the balance is decreased with the same amount. Examples of this type of e-cash system include online creditcard payments, Way2Pay and PayPal. If the user makes use of one of these systems, her identity is known to the bank and often also to the shop. If a user uses a prepaid system like the "wallie-card", the identity of the user is not known to the bank or the shop, but different payments can be tracked down to the same card. As far as the cryptographic system is concerned this is the same as a creditcard, where the identity is known.

With Off-Line systems which can still be used on the Internet of course, a payment is not checked the moment the puchase is done. Also, because the is no account which needs to be checked for a balance, the identity of the user could remain private. A potential problem however is that the user tries to spend the same money twice, which is not possible if the account balance is checked before the transaction is processed. In this paper we discuss a Off-Line e-cash system in which the identity of the user is protected, except when the user tries to practise fraud.

If we would use standard cryptographic systems, there would be some extra problems. For example, if the user is required to sign the e-cash with his private key in order to be able to track her down in case of double-spending, the shop as well as the bank can track her spendings. Ideally, the user's identity is kept private as long as the user doesn't double spent. And if she does, the bank should be able to recognise the double-spending and identify the user.

Casus 6.1: The need for user-signed e-cash

One of the problems with online internet payments is that three parties need to be online: the user, the shop and the bank. There is a fundamental reason the first two need to be online in order to do a purchase. But often a user and a shop can't complete a transaction because a bank or payment system is not available. Also, if the bank or payment system makes an error, it could be difficult for a user to get her money back, because payments don't need to be cryptographically authenticated by the user in many systems. Websites like `http://nopaypal.com/` collect hundreds of such complaints. Such experiences from users greatly holp up acceptance of e-cash and internet payments.

6.2 The Brands system for Off-Line e-cash

In [Bra93], Stefan Brands describes an off-line payment system in which a users identity is not reveiled to the shop or the bank, until the user tries to double spend the money. This system can be extended with an extra protection (called an observer) to make it difficult to spend the same money in the first place.

The system differentiates roughly five stages: the establishment of a bank, the registration of a user, the withdrawel of money from the users account to an electronic banknote, payment of the banknote to a shop, and the shop returning the banknote to the bank and (hopefully) receiving real money. Before we can give a detailed description of these different stages of the system, we must discuss some mathematical foundations.

The Brands system is based on the Discrete Log problem and/or the Representation Problem which is computationally as hard as the Discrete Log problem. The following definitions of the *Discrete Log problem*, *generators*, *representations* and the *Representation Problem* are taken from [Bra93]. In these definitions, we assume q to be a large prime and G_q a subgroup of $\mathbb{Z}_p^*$, with $p = kq + 1$ also prime for some $k \in \mathbb{N}$. In practise, the length of p would be at least 512 bits, and that of q at least 140 bits.

Definition 6.1 *Finding the unique index* $\log_g h \in \mathbb{Z}_q$ *with respect to* $g \in G_q \setminus \{1\}$ *is the Discrete Log problem. An algorithm is said to solve the Discrete Log problem is, for inputs* $g \neq 1$*,* h *generated uniformly at random, it outputs* $\log_g h$ *with nonnegligible probability of succes. The Discrete Log assumption states that there is no polynomial-time algorithm which solves the Discrete Log problem.*

Definition 6.2 *Let* $k \leq 2$ *be a constant. A generator-tuple of length* k *is a* k*-tuple* $(g_1, \ldots, g_k)$ *with, for all* $i, j \in \{1, \ldots, k\}, g_i \in G_q \setminus \{1\}$ *and* $g_i \neq g_j$ *if* $i \neq j$*. An index-tuple of length* k *is a* k*-tuple* $(a_1, \ldots, a_k)$ *with* $a_i \in \mathbb{Z}_q$ *for all* $i \in \{1, \ldots, k\}$*. For any* $h \in G_q$*, a representing index-tuple (also called representation) of* h *with respect to a generator-tuple* $(g_1, \ldots, g_k)$ *is an index-tuple* $(a_1, \ldots, a_k)$ *such that* $\prod_{i=1}^{k} g_i^{a_i} = h$*.*

Definition 6.3 *Again, let* q *be prime and* G_q *a subgroup of* $\mathbb{Z}_p^*$*. Let* $g_1, \ldots, g_k, h \in G$*. The representation problem is to find a representation of* h *with resprect to* $(g_1, \ldots, g_k)$*.*

6.2.1 Establishment of a bank

A we said before, the first step of the Brands system is setting up a bank. The bank needs to choose a large prime p, and a large but smaller prime q which divides $p - 1$. The bank also

chooses three generators g, g_1, g_2 from G_q. With q being prime every element of G_q is a generator, so this shouldn't be a problem. The bank then selects a secret key x, and publishes its public key $h = g^x \bmod p$. Next, the bank needs two collision-free hash functions H and H_0 which are defined as follows:

$$H(v_1, v_2, v_3, v_4, v_5) = k, \text{ with } v_1, v_2, v_3, v_4, v_5, k \in G_q \tag{6.1}$$

$$H_0(v_6, v_7, \text{shopid}, \text{datetime}) = k' \text{ with } v_6, v_7, k' \in G_q \tag{6.2}$$

The variables p, q, g, g_1, g_2, h, H, and H_0 are public, and x remains private to the bank.

6.2.2 Opening an account: the user's information

The user also chooses a random number u_1 from G_q, and shares the public key $h_u = g_1^{u_1} \bmod p$ with the bank. The bank stores h_u along with other user identification information, and both the bank and the user will use a number z in future calculations:

$$z = (h_u g_2)^x \bmod q \tag{6.3}$$

Using definition 6.2, the representation of z in terms of (g_1, g_2) is $(u_1 x, x)$ because $z = (g_1^{u_1} g_2)^x = g_1^{u_1 x} g_2^x$.

6.2.3 Cash withdrawal

In the Brands system, money is transferred to the user in "coins" which are worth a fixed amount of money. This is a problem, because a user should be able to pay any amount of money. Ideally, the system would be altered to allow for arbitrary payments. Such a system does not yet exist as far as I know, so we'll have to live with this limitation at the moment.

The coin should contain information which only the bank can generate; no shop would accept a coin which can be generated by anyone. The shop owner wants to be sure it can trade in the electonic coin for real money at the bank, so the shop has to be able to proof the bank generated the coin.

First, the bank chooses a number w from G_q, and sends the following to the user:

$$a = g^w \bmod p \tag{6.4}$$

$$b = (h_u g_2)^w \bmod p \tag{6.5}$$

Then the user generates random numbers s, x_1 and x_2 from G_q to calculate:

$$A = (h_u g_2)^s \bmod p \tag{6.6}$$

$$B = g_1^{x_1} g_2^{x_2} \bmod p \tag{6.7}$$

$$z' = z^s \bmod p \tag{6.8}$$

z' is constructed in such a way, that it is a signature on A: $z' = z^s = ((h_u g_2)^x)^s = ((h_u g_2)^s)^x = A^x$. The user also generates two random numbers $u, v \in G_q$ and then calculates:

$$a' = a^u g^v \bmod p \tag{6.9}$$

$$b' = b^{su} A^v \bmod p \tag{6.10}$$

The user then computes c' and c as follows:

$$c' = H(A, B, z', a', b') \tag{6.11}$$

$$c = c'/u \bmod q \tag{6.12}$$

The result c is sent to the bank, which responds with r:

$$r = w + cx \bmod q \tag{6.13}$$

The user wants to check if she has received a valid e-cash coin, and checks if indeed:

$$g^r = ah^c \bmod q \tag{6.14}$$

$$(h_u g_2)^r = bz^c \bmod q \tag{6.15}$$

This should be the case, because $g_r = g^w g^{x^c} = ah^c$ and $(h_u g_2)^r = (h_u g_2)^{w+cx} = (h_u g_2)^w (h_u g_2)^{x^c} = bz^c$. Finding such a r is the same as solving the Discrete Log problem. So only the bank, who knows x and w, can calculate this r and the user can be rest assured that her electronic coin is provably covered by real money. The user now calculates $r' = v + ru \bmod q$ and stores the coin as the ordered set of numbers $A, B, (z', a', b', r')$.

Note that at this point, the user hasn't added private information to the coin, i.e. u_1 is not used in the calculations. Therefore, for the bank to be able to track down the user in a case of fraud, u_1 will have to be used in some way in the payment protocol. This means that it's no problem if the coin is copied by anyone, because no-one can spend it without knowing what u_1 is.

6.2.4 Payment protocol

When the user want to buy something from the shop, she sends the coin $\{A, B, (z', a', b', r')\}$ to the shop. The shop returns a challenge d:

$$d = H_0(A, B, \text{ shopid, datetime}) \tag{6.16}$$

The shopid is included in the hash to prevent theft of the coin after payment. The bank will only accept a coin signed with this hash from this particular shop. Optionally, the list of purchased items could be included in the hash funtion. This would enable both the user to proof which items are bought in the transaction.

Next, user calculates a response r_1, r_2 which enables the shop owner to check if the coin is valid and if it belongs to the user who is trying to used it. Also, the response r_1, r_2 is calculated using u_1, so the shop owner is able to proof that the user validated the purchase.

$$r_1 = du_1 s + x_1 \bmod q \tag{6.17}$$

$$r_2 = ds + x_2 \bmod q \tag{6.18}$$

Now the shop will have to check two things. They will want to check if the coin is valid and signed by the bank and also if the user has signed it, which is a requirement from the bank, before they pay real money in return for the e-coin. The shop checks to following equations:

$$A^d B = g_1^{r_1} g_2^{r_2} \bmod p \tag{6.19}$$

$$g^{r'} = a' h^{c'} \bmod p \tag{6.20}$$

$$Ar' = b'z'^{c'} \bmod p \tag{6.21}$$

These equations can be rewritten as:

$$\begin{aligned} A^d B &= (h_u g_2)^{sd} g_1^{x_1} g_2^{x_2} \bmod p \\ &= g_1^{u_1 sd + x_1} g_2^{sd + x_2} \bmod p \\ &= g_1^{r_1} g_2^{r_2} \bmod p \end{aligned}$$

$$\begin{aligned} g^{r'} &= g^{v+ru} \bmod p \\ &= g^{v+(w+cx)u} g^{xcu} \bmod p \\ &= a^u g^v h^{cu} \bmod p \\ &= a' h^{c'} \bmod p \end{aligned}$$

$$\begin{aligned} A^{r'} &= (h_u g_2)^{s(v+ru)} \bmod p \\ &= (h_u g_2)^{sv+us(w+cx)} \bmod p \\ &= (h_u g_2)^{swu} (h_u g_2)^{sxcu+sv} \bmod p \\ &= b^{su} A^{xcu+v} \bmod p = b^{su} z'^{cu} A^v \bmod p \\ &= [b^{su} A^v] z'^{cu} \bmod p = b' z'^{c'} \bmod p \end{aligned}$$

6.2.5 Deposit protocol

The shop now has enough information to return the coin to the bank, and receive real money in return on its banking account. The coin cannot be used to pay another shop. However, an extension to this system does exists that allows a coin the be transferred to another user.

The shop now has the coin $\{A, B, (z', a', b', r')\}$ and the user's signature (r_1, r_2) for this specific purchase. The shop gives this information along with its shopid and the datetime of the purchase to the bank. The bank can use equations 6.19, 6.20 and 6.21 to check if this is a valid coin. The bank checks a database to see if this coin was previously spent. If it's not in the database, the shop gets paid and the coin is stored in the database as $\{A, B,\ \text{datetime}, r_1, r_2\}$.

Ik the coin is already in the database and the datetime is the same, the the shop is trying the cash the same coin twice, and the deposit is rejected. If the datetime is different, then the user has spent the same coin twice. The bank now has the following information about the two deposits: $\{A, B,\ \text{datetime}, r_1, r_2\}$ and $\{A, B,\ \text{datetime}', r_1', r_2'\}$. Now the bank can extract the identity of the user as follows:

$$(r_1 - r_1')/(r_2 - r_2') = (du_1 s - d' u_1 s)/(ds - d's) = u_1 \bmod q \tag{6.22}$$

The bank checks the user database for $h_u = g_1^{u_1}$ mod$p$ to find the user who double-spent the coin. Knowing the private key of this user serves as a proof that a coin was double-spent by this user.

6.3 Advantages of the Brands system

The Brands system has some benefits that are not found in other off-line cash systems:

Cryptographically secure The system is built on the Discrete Log problem. And while there is no proof that the DL is computationally secure, it is regarded as save, as long as the primes p, q are long enough.

Privacy Every party knows exactly what it needs to know:

- **The bank** knows the identity of the user when the coin is given to her, because the bank has to know what account to get the money from. But the bank doesn't know the identity of the user when the coin is returned, except when the user has double-spent the coin. Also, the shop can't link different payments by the same user.
- **The user** doensn't need to know the shopid that the bank and shop agreed upon, so the user can't pretend to be the shop and try to make the bank think the shop is trying to cash the same coin twice.
- **The shop** can't link different payments to eachother or to the user. However, the shop can find out the identity of the user if the user signs two different challenges d (See formula 6.16). It would be better if only the bank is able to establish the identity of the user in such a case.

Fraud prevention In every step, messages are signed either by the user, or by the bank. Therefore, the bank can't debit the account of the user without supplying a real coin; every withdrawal must be signed by the user. Therefore a user cannot deny having made a withdrawal; the bank has the user's signature on the withdrawal. Furthermore, if for example in a criminal investigation the police want to prove that the user has made a particular payment, they can use the user's private key to check if the signature matches the signature that the shop received earlier. So the user cannot deny having made a particular purchase. Of course, the user will have to be forced to give up her private key of sign the message with her private key, but the police can check that the private key is authentic by comparing a newly generated public key to the known public key of the user.

Extensible There are several extensions to this system:

- **Multiple spending** In the standard system, a coin can only be spent once. The system can be extended to allow for coins that can be spended k times in such a way that only after spending it $k+1$ times, the identity of the user is revealed. In this extension however, the k payments can be linked to eachother.
- **Observer protocol** The system can be extended with a fourth party, the observer O. In practise, this could be a chip on the user's banking card or a device that is attached to the user's computer. Every coin is also signed by the Observer chip, and the Observer stores spent coins in permanent memory. This chip can prevent double-spending. Of course, if the chip's secret key is somehow discovered by the user, the user can still be tracked down by the bank after double spending.
- **Checks** In the system explained in this chapter, it's only possible to pay with coins of one fixed amount. The system is extendible to allow the use of checks, of which an arbitraty amount can be used for one payment. However, different payments of the same check can be linked together by the shop or the bank. But as long as the user doesn't overspend the check, her identity remains conceiled.
- **Transferability** Another extension exists which allows a user to transfer money to another user without intervention of the bank.

Analougues in RSA The system can be modified to make use of multiplicative groups, like in the RSA system. Only the withdrawal protocol is not adaptable to the RSA system.

6.4 Why aren't Off-Line payments widely used

As we can see in the previous section, off-line cash comes with many advantages. Why then, is such a system not in widespread use? We will try to answer that question in this section.

6.4.1 Standards

The acceptance of standards is much more important in off-line cash than it is with online cash. For example, when used in a standard webbrowsing environment many online systems work with standard http techniques: the shop links to a "bank", for example the `way2pay.nl` website. The user validates the payment on this more or less trusted site, and way2pay links the user back to the shop website, so the shop knows the payment has been made.

With Off-line cash, the transaction needs to take place between the user and the shop directly. So the browser needs an extention which can do the calculations. Standard Java of Javascript solutions don't work, because to do the calculations these programs should be able to read the user's private key, but it's not secure for an applet supplied by a shop to be able to read to user's private key.

Installing an extra extension to the browser in order to make the transactions more secure, is a hurdle most users wont take until offline cash is in wide acceptance and the benefits are large enough to overcome this hurdle. This presents a chicken-and-egg problem. Also, if additional software needs to be installed, there has to be a worldwide standard on what this software should do exactly.

6.4.2 Patents

As explained in the previous subsection, online payment systems are relatively easy to implement; every CS student can come up with a system such as way2pay or paypal. Most of the effert companies like this have to do lies in the marketing and client support. But marketing and management techniques can't be patented. Software techniques can be patented in the US, and maybe in the future also in the EU.

Patents may hinder the acceptance of offline cash. Suppose the inventors of for example the Brands system (which relies in turn in previous inventions) choose not to ask money for their invention, and establish the FreE-Cash foundation. If they themselves are threatened by patent claims from other companies, they have to be able to pay lawyers, even if the claims are incorrect.

So if a patent system on software exists, the inventors are almost forced to patent and ask money for the system in order to be able to defend themselves legally. This in turn can hinder acceptance by shops and software companies because they have to pay to the inventors of this particular e-cash system.

6.4.3 Public opinion

Even if an offline e-cash system is cryptographically safe, the public will still have serious doubts about accepting some bits as a payment. Even though the Brands system is safer in many ways than a creditcard for example, the creditcard is something people are used to and can understand.

And even if fraud can only occur when a user is sloppy with her private key u_1, stories will emerge about insecurity of the system, and hinder its acceptance

6.4.4 Possibility of huge over-spending

If a coin is double-spent, the user for whom the coin was made can be tracked. But it is quite possible the user's private key u_1 is stolen. Then the thief can theoretically use the same coin thousands of times on the internet before the fraud is detected. And even then, a system should be in place which distributes this information to all shops, so that shop owners can reject coins that are already double-spent.

The damage is much larger than with online systems: if creditcard information is stolen, the thief can max out the creditcard, but she can not spent thousand times as much as the card limit.

Also, if a thief tries to pay with a creditcard, the payment information is passed directly to the creditcard company. They can track suspicious behaviour, and trace a thief quite fast.

6.4.5 Secure implementation needs hardware

As said before, an offline system when used for payments on the internet, needs a software addition on the client side, unlike many online payment systems. But this software addition is not enough. When a computer which stores a private key u_1 gets cracked, the thief has full access to the legitimate user's money.

A safer implementation needs a hardware device attached to the computer which stores the private key u_1, and does the calculations without sharing u_1 with the software on the computer. This device could for example have a display and a button, to ensure no payments can be made without consent of the user.

Every user of the system must first have such a device before being able to make use of the system securely. This could be yet another hurdle for acceptance of off-line cash.

Summary and Conclusions

Cryptography offers huge possibilities for secure payment systems, if used correctly. Consumers don't have to rely on bank being online 24/7 in order to be able to pay shops or each other. Also, users don't have to put all their trust in the bank; if a user says not to have made a withdrawal, he can ask the bank to prove it by showing the signatures.

An other advantage is that the same off-line e-cash can be used on a chipcard the user can put in his wallet and for payments on the internet. For example, the hardware device described in subsection 6.4.5 could be made to read chipcards. As an extra advantage, if the secret key is only stored on the chipcard, the computer cannot be used to make payments if the user is not present. Even if the thief has physical access to the computer.

I think that, despite of the advantages, offline cash will not be used for a very long time to come. Each of the five objections from section 6.4 is in itself enough for offline cash not being able to be widely accepted as a payment method on the internet. For things like chipcards however, the Brands could be a more secure and flexible alternative to the current systems.

Chapter 7

Design of Blockciphers

Written by *M.R.T. Beye*

Blockciphers have become a popular way to encrypt sensitive information and have remained so for decades. A multitude of implementations have been developed in recent years and some of these algorithms have become widely used. However, a number of successful attacks have been devised, to attack inherent weaknesses present in many of these ciphers. This paper will attempt to give insight in the design of cryptographically strong blockciphers.

First of all some basics regarding blockciphers will be introduced (and for some revisited). The focus will be on ciphers that use the so-called Feistel structure, which will be decomposed into its components. The most common and successful attacks and exploits will be identified and related to weaknesses within these components.

A definition of desired properties, such as confusion and diffusion, will be given, after which it will be made clear how these properties work to protect an algorithm against aforementioned attacks. Finally, some requirements to guarantee the desired properties are given, and the CAST approach will receive some attention as a good means of building robust blockciphers that satisfy these requirements.

7.1 Introduction to blockciphers

7.1.1 History

Blockciphers have been around for years and are a cheap and efficient way to encrypt data for secure communication on networks. Back in 1949 Shannon [Sha49] introduced his notion of secrecy and proved that the Vernam cipher, which used very large keys and needs synchronization between sender and receiver, is an algorithm which provides unconditional secrecy, although it is not very practical. Shannon also showed that unconditional secrecy cannot be achieved in a cheaper or more efficient fashion.

The implication of his results is that the blockciphers created since have been aiming for empirical secrecy in stead of the unconditional kind. This means that these ciphers are not provably immune to any and all attacks and that it has been a matter of waiting for cryptanalysts to devise a way to attack and defeat the principles behind them. Still, we would like to continue to use blockciphers, for the fact that they are much faster that asymmetric (i.e. public key)

algorithms and can be implemented in hardware with limited capabilities. A good example of a blockcipher is DES, which has been very widely used and is well documented and studied.

7.1.2 The mother of all blockciphers: DES

In 1972 the National Bureau of Standards called for algorithms to be submitted, in order to establish a new Data Encryption Standard. IBM responded with the winning algorithm in 1974, after which the National Security Agency revised its design. This resulted in a version with a reduced blocksize and keylength and possibly also a revision of the S-boxes that are used.

In 1975 the algorithm was publicly released, including details of its inner workings, to the displeasure of the NSA. The fact that the NSA had made adjustments, made cryptanalysts curious and gave rise to an increase in cryptographical research. In 1977 DES was officially taken in use as a standard and has remained so for decades.

Criticism was mounting however, as both the NSA and other parties claimed that DES was no longer safe, this because of the limited keylength. Unfortunately, for years, no safe replacements were available and DES remained in use, albeit in the form of 3DES. The problem with the other candidates seemed to be their vulnerability to so-called differential attacks, as introduced by Biham and Shamir [BS91] in the early 90s. It turned out that the NSA and IBM had known of this type of attack, but prepared DES against it and kept the information secret.

Matsui [Mat94] was inspired by the idea of differential cryptanalysis and devised his own attack: linear cryptanalysis. This attack is able to break the 16 rounds of DES faster than exhaustive key search, in contrast to differential attacks. The methods of both of these attacks will be explained in detail in section 7.3, as will some other weaknesses and attacks.

7.2 Inner workings of Feistel based ciphers

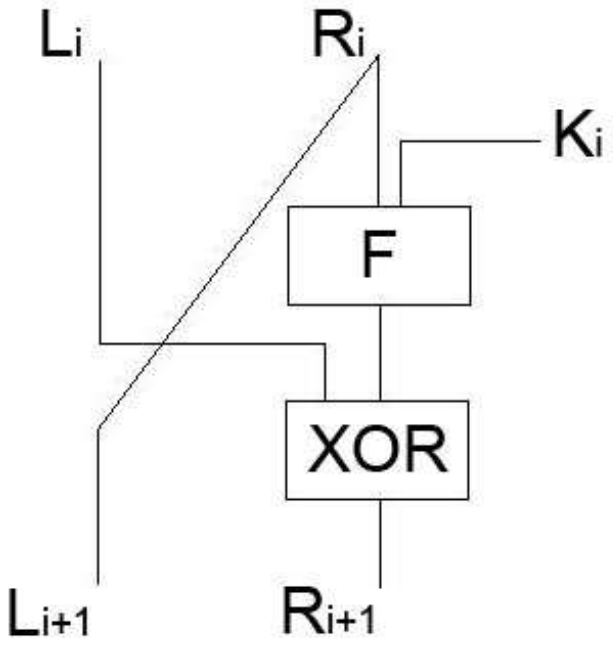
7.2.1 The Feistel network

In this paper we will focus on ciphers that use the Feistel structure, because much relevant research has been done in this field and many popular ciphers use this structure.

A Feistel network in general takes a piece of data with a length equal to half the blocksize and combines it with another piece of data of the same length. This is usually done by means of a simple XOR ($\oplus$) operation, but other bitwise functions can be used equally well. The other piece of data that serves as input to the XOR function is derived from the remaining half of the data in the block and the round key.

Blockciphers iterate this type of function a number of rounds, in order to ensure good intermixing of the plaintext and the keys. The number of iterations will prove to be an important aspect in protection against attacks, as shown in sections 7.3 and 7.4.

A graphical representation of a single iteration of a Feistel network is given in the figure below:



7.2.2 Breakdown into components

Ciphers of the Feistel type can be decomposed into four main components:

- S-boxes
- Framework
- Key Schedule
- Round Function

The function which mixes half of the data with the key is called the round function F. Usually, this function uses S-boxes (S for substitution), which we would like to resemble a random function as closely as possible, in order to correctly mask patterns in the plaintext. S-boxes are usually implemented using lookup tables, whose rows and columns represent part of the input, thus computing the output. To make sure that S-boxes in round i are influenced by multiple S-boxes in round $i-1$, a P-box (P for permutation) is used to shuffle bits between rounds.

The key schedule determines what round keys are generated, based on the primary (randomly chosen) key. This generating of round keys is usually implemented using shift registers.

Aforementioned components must fulfill some demands separately and as a whole in order to make the resulting cipher cryptographically strong. Next we will show where the faults can lie that make a cipher vulnerable to attacks.

7.3 Attacks and weaknesses

7.3.1 Exhaustive Key Search and DESX

Exhaustive key search is the most straightforward attack that can be launched on a blockcipher. Although it can easily be discouraged by increasing the keylength, the existing DES algorithm

for instance is no longer safe against this type of brute forcing. If one needs to run a cipher on a piece of hardware that is designed for DES, increasing keylength is no option. One solution that was found is 3DES, where the plaintext is encrypted three times, using normal DES. This effectively increases the keylength and the properties of DES make 3DES a good form of encryption, without pitfalls. However, it is also quite expensive in terms of computational workload. A good alternative to protect DES against exhaustive key search has been advocated by Kilian and Rogaway [KR96] in the form of DESX.

The principle behind this approach is defined by the following equation:

$$DESX_{k,k1,k2}(x) = k2 \oplus DES_k(k1 \oplus x) \tag{7.1}$$

Note that the overhead this has on the original DES algorithm is minimal and that DESX can easily be implemented on existing DES hardware. Kilian and Rogaway suggest that this cipher is harder to break by a 'generic key-search adversary', who attempts to decide if a ciphertext was outputted by the cipher under study. This is a generalized way of looking at key search. They state that the fact that decision becomes harder, so does determining the specific key used for encryption. Their result states that the effective key length of DESX would be $k + n - 1 - \log m$, where k is the keylength, n is the blocksize and m is the number of plaintext/ciphertext pairs the adversary possesses.

However, DES is still vulnerable to linear attacks, and no mention is made as to whether DESX makes any kind of improvement when it comes to linear attacks in stead of exhaustive key search.

7.3.2 Linear Attacks

Linear attacks are known plaintext attacks that look for special qualities in the behavior of the cipher. The principle underlying linear attacks is to find a linear approximation that relates subsets of plaintext, ciphertext and key bits in the following way:

$$P_{i1} \oplus P_{i2} \oplus \cdots \oplus P_{ia} \oplus C_{j1} \oplus C_{j2} \oplus \cdots \oplus C_{jb} = K_{k1} \oplus K_{k2} \oplus \cdots \oplus K_{kc} \tag{7.2}$$

Where $i1, i2 \cdots ia$, etc denote fixed positions in plaintext P, ciphertext C and key K.

If Pl is the probability that equation 7.2 holds. The effectiveness of the linear approximation is related to the magnitude of $|Pl - 1/2|$. If $|Pl - 1/2|$ is large enough and sufficient plaintexts-ciphertext pairs are available, one can determine one key bit that is equal to the XOR of the key bits in the righthandside of equation 7.2 as the value that most often satisfies this equation. Such a linear approximation is made by combining a number of S-box linear approximations in different rounds, so that all terms that do not involve plaintext, ciphertext or key bits are cancelled out.

The number of plaintexts in a linear attack is inversely related to $|Pl - 1/2|$ so an increase in the needed number of plaintexts can be attained by selecting S-boxes so that $|Pl - 1/2|$ is minimized.

The probability of finding a linear approximation is the product of the probabilities of finding all the required S-box linear approximations. This probability can be decreased by either making sure more S-box linear approximations are needed (by using more rounds in the algorithm) or by increasing the non-linearity of the S-boxes, thereby decreasing the odds of finding the needed S-box linear approximations. The latter approach is preferred, because adding more rounds makes the algorithm slower in both encryption and decryption, while merely patching the problem, whereas improving the design on S-boxes solves the problem at the root (and leaves the addition of further rounds as an extra option). In section 7.4, the concept of non-linearity will be further

explained and it will become apparent how to ensure the S-boxes are sufficiently non-linear. Also, section 7.5 will provide a comparison of $|Pl - 1/2|$ between CAST and DES ciphers.

7.3.3 Differential Attacks

Differential attacks are chosen plaintext attacks that exploit the inability of a cipher to map input to output differences in a statistically uniform way. This fact leaks key bit information into the ciphertext. An example of a differential attack (in this case against DES) is given by Tel [Tel02]. Section 7.4 will show how a property called diffusion can prevent this type of attack from being successful.

7.3.4 Related Key Attacks

Related key attacks are attacks where one makes use of 'related keys', in the sense that keys that are derived in a deterministic way based purely on the primary key and previous round keys are 'related'. If one knows one round key, one can use this information to find out information about other round keys, or even the primary key.

7.3.5 Weak and semi-weak keys

In blockciphers, a key K has an inverse key I if it holds that:

$$ENC_I(p) = ENC_K^{-1}(p) = DEC_K(p) \tag{7.3}$$

for any plaintext vector p.

Fixed points of key K are defined to be plaintext vectors p such that $ENC_K(p) = p$. Anti-fixed points are plaintext vectors p' such that $ENC_K(p')$ is the compliment of p'.

Weak keys are keys that are their own inverse. In DES, these generate a palindromic set of round keys and have 2^{32} fixed points. Semi-weak keys are keys that are not weak, but of which the inverse can easily be found. In literature, semi-weak anti-palindromic and semi-weak non-anti-palindromic keys are distinguished, but we will not go into details here. DES possesses a total of 4 weak and 12 semi-weak keys. As these keys, although small in number, possess unwanted qualities, care should be taken to avoid the existence of weak keys in the design of key schedules for new blockciphers.

7.4 Further defenses against attacks

7.4.1 Desired properties

We have seen how simple exhaustive key search can be made more difficult, but now we will define the properties that provide a defense against the more refined attacks.

As we have seen, the attacks described in section 7.3 make clever use of intrinsic properties (or faults) in the S-boxes or key schedule of the cipher. If we want to create a cipher that does not possess these properties, we must have a definition of what the desired properties are. The concepts of confusion and diffusion, as introduced by Shannon [Sha49], provide us with such a definition, as does non-linearity. Other terms one comes across in literature are avalanche, Strict Avalanche Criterion (SAC) and Bit Independence Criterion (BIC). We will discuss the first three of these properties, which are related to the inner workings of the S-boxes and the Round Function that employs them.

7.4.2 Confusion

Confusion measures the degree to which a cipher masks input bits in its output by means of the chance that an inversion in an input bit causes an inversion in an output bit. In an ideal case, this probability should be 1/2 for any input / output bit pair. Another way of looking at it is that the cipher is not biased towards ones or zeros in its mapping of input to output bits.

7.4.3 Diffusion

Diffusion is the property that states that a given plaintext bit can affect output bits. The higher the diffusion, the more output bits have a chance to be affected by a certain input bit. Katos [Kat05] presents a way of measuring diffusion by means of diffusion instances, which are matrices that can be statistically analyzed, to reveal the degree of diffusion of the corresponding cipher. The statistical tests include, among others, equality of the number of ones and zeros in the matrices, as well as the randomness of their distribution.

7.4.4 Non-linearity

Nonlinearity of an m-bit boolean function is defined as:

$$NL(s) = \min_{t \in A} d(s,t) \tag{7.4}$$

where A is the set of all m-bit affine boolean functions.

When we extend this definition to $m \times n$ S-boxes, we obtain:

$$NL(S) = \min_{a_1,a_2,\cdots,a_n \in \{0,1\};\exists i,a_i \neq 0} NL(\oplus_{i=1}^{n} a_i s_i) \tag{7.5}$$

where s_i is the m-bit boolean function of the i^{th} output bit of S-box S. This means that the non-linearity of an S-box is the minimum non-linearity of over all non-zero linear combinations of the m-bit boolean functions it employs.

Lee, Hays and Tavares [LHT97] support the claim that a non-linearity of 2^{m-2} is sufficient for $m \times n$ S-boxes. The examples in the following sections will use 8×32 S-boxes, so this would lead to a minimum required non-linearity of 64.

Lee, Hays and Tavares proceed to prove that if we require a non-linearity of at least 2^{m-2}, we have a good chance of randomly selecting an acceptable S-box. They show that for 8×32 S-boxes the chance of selecting a faulty S-box (so with non-linearity less than 64) is smaller than 2^{-11}. By testing candidate S-boxes for non-linearity, certainty can be given that the selected S-boxes really meet the criteria. Experiments have apparently not found any randomly generated S-boxes with a non-linearity smaller than 72.

7.5 Correct design of components

If we are to find components that are resistant to attacks, we need to be able to determine if they meet the requirements stated in the previous section. We will now look at a design paradigm called CAST, which aims to provide a way of creating a family of solid blockciphers, ensuring they possess these qualities in a theoretically provable way.

7.5.1 S-boxes in CAST

The CAST system, as explained by Adams [Ada97], shows us a way to find S-boxes that possess the required amount of non-linearity, in order to make differential cryptanalysis as hard as exhaustive key search. The choice of S-boxes in CAST focuses on these being highly non-linear. This is done by choosing linearly independent binary bent vectors, that together make the matrix of an S-box. The Hamming weight and Hamming distances in this matrix are a measure for the success of chosen set of vectors. If they are highly non-linear and close to SAC-fulfilling, they also guarantee Output Bit Independence Criterion. Adams states that a good set can be found in a few weeks of running time. When we compare the performance of S-boxes of this type to those used by DES, we arrive at the following table:

no of rounds	$\lvert P_l - 1/2\rvert$ CAST	$\lvert P_l - 1/2\rvert$ DES
8	2^{-17}	1.22×2^{-11}
12	2^{-25}	1.19×2^{-17}
16	2^{-33}	1.49×2^{-24}

The author would like to remind the reader that the procedure described in section 7.4.4 also yields S-boxes with high degrees of non-linearity and does this much faster. Random selection and tests for non-linearity can be done fast, and with a chance of encountering a bad S-box in the magnitude of 2^{-11}, not much time is likely to be wasted. It is not clear in how far these will be SAC-fulfilling. If this does not follow from the high degree of non-linearity, the proof given by Adams cannot be applied to randomly selected S-boxes. However, the work of Lee, Heys and Tavares is also aimed towards use in CAST-like ciphers, even published in the same journal as Adams' work, suggesting good compatibility.

7.5.2 Framework in CAST

CAST uses the same type of framework as DES does, namely the Subsitution-Permutation Network (SPN), as suggested by Shannon. CAST uses the Feistel structure to implement this scheme, which is well studied and does not seem to possess inherent structural weaknesses, like the "tree structure" implementation does seem to have. CAST leaves the parameters for blocksize and number of rounds open to the preference of the developer of individual implementations. It must be noted that any SPN system has its security increased, by increasing the number of rounds used. Adams also notes that experimentation has found taught us that simply choosing S-boxes that are resistant to differential attacks in isolation do not make for a resistant cipher if the framework is not chosen in such a way that it makes specific use of the properties of the S-boxes. For instance, when putting S-boxes with relatively flat output XOR distributions (good confusion) into DES, the resulting cipher becomes more vulnerable to differential cryptanalysis than the original DES algorithm. To ensure equal treatment of both data halves, an even number of rounds is needed. Regarding the number of rounds, a comparison with DES might be in order. Looking at both differential and linear attacks, DES would now seem to need 18-20 rounds in order to be as strong as its keysize. Section 7.5.4 will show the advantages of CAST over for instance DES when it comes to the number of rounds needed.

7.5.3 Key Schedule in CAST

Although CAST uses a key schedule typical for Feistel networks (generating round keys from a primary key), it pays attention to additional constraints that make for a good resistance to key clustering attacks. Without going into details, these are known as key/ciphertext Strict Avalanche Criterion and Bit Independence Criterion (BIC). It is necessary to use different subsets of primary key bits to generate round key i and $i+1$. For attaining good avalanche it is also wise to use all partial keys to generate the round key for round $N/2$ and reuse them in the remaining rounds. The novelty that CAST adds is the fact that round keys are not generated using a complex function, such as DES's slide registers. CAST makes use of a simple function and a dependency on a set of key schedule S-boxes. This makes recovering round key bits by means of linear attacks less useful, because there is no known clear way of using round key bits to rediscover primary key bits, as can be done with other ciphers like DES. Adams gives the following example for an 8 round cipher:

Let the primary key be $KEY = k_1k_2k_3k_4k_5k_6k_7k_8$, where k_i represents the i^{th} byte.
Partial keys K'_i are selected as follows:
$K'_1 = k_1k_2$
$K'_2 = k_3k_4$
$K'_3 = k_5k_6$
$K'_4 = k_7k_8$

After round 4, KEY is transformed into $KEY' = k'_1k'_2k'_3k'_4k'_5k'_6k'_7k'_8$
by means of the following transformation step:
$k'_1k'_2k'_3k'_4 = k_1k_2k_3k_4 \oplus S_1[k_5] \oplus S_2[k_7]$
$k'_5k'_6k'_7k'_8 = k_5k_6k_7k_8 \oplus S_1[k'_2] \oplus S_2[k'_4]$

After this transformation, the remaining partial keys are generated:
$K'_5 = k'_4k'_3$
$K'_6 = k'_2k'_1$
$K'_7 = k'_8k'_7$
$K'_8 = k'_6k'_5$

One can verify that the partial keys are influenced by all primary key bits from round 4 and up. Having constructed the partial keys, they are used as input to the key schedule S-boxes, in order to create the round keys: $K_i = S_1[K'_{i.1}] \oplus S_2[K'_{i.2}]$, where $K'_{i.j}$ denotes the j^{th} byte of key K'_i.

Key schedules for systems with a different number of rounds or a different blocksize can be constructed in similar ways. Adams proves that this way of generating keys rules out the existence of weak and semi-weak keys, because no key in this scheme can have an inverse key. One step in his proof depends on the specific properties of the S-boxes used. Therefor, the proof is not general and cannot be blindly applied to specific implementations, because the choice of S-boxes is not fixed in CAST.

7.5.4 Round Function in CAST

One data half of length n is combined with the round key by a function, usually XOR addition. By splitting the result into pieces of size m and feeding these into separate S-boxes, thereafter combing them through another binary operation (again, usually $\oplus$), the input and output length of the round function will be equal, even though the $m \times n$ S-boxes themselves result in data expansion. This makes iterating the function over multiple rounds easy to implement. The terms being so vague, it is hard to formally prove confusion, diffusion and avalanche properties of the

round function. Adams acknowledges this, yet makes the claim that CAST ciphers seem to possess good statistical properties after only 2-3 rounds, where DES needs 5-6 rounds to attain similar properties. This supports the claims that diffusion, confusion and avalanche are better, resulting in a better security / number of rounds ratio. What can be proven is the fact that the round function, when chosen to specifications, exhibits both highest-order SAC and highest-order BIC.

Adams states that the use of a simple XOR operation in the round function is already good, but that some improvements are possible. The most important demands on the operation of choice 'a', are that it maps input to output in a non-linear fashion and that it must hide information about the keys used, using an operation on a single input, or pair of inputs. These demands defends the round function against differential attacks. The demand that the operation be such that subset sum operation must not be distributive over $a(x, y)$, provides protection against linear attacks. A suitable candidate is suggested by Adams: If the round function uses $\oplus$ for all operations a, b, c and d, it can be seen as:
$F(R, K) = S_1(B^{(1)}) \oplus \cdots \oplus S_4(B^{(4)})$, where $B = R \oplus K$ and $B^{(j)}$ is the j^{th} byte of B.
The alternative suggestion is similar, yet computes B as:
$B = ((R \oplus K_1) * K2) \,\%\, n$

This should require only a factor 2 more computation compared to the version that utilizes $\oplus$ for all its operations. Apparently providing intrinsic immunity against linear and differential attacks, the alternative method appears the right choice. However, there is evidence that any CAST blockcipher with a blocksize and keysize of 64 bits, four 8×32 S-boxes and 32-bit round keys would require more chosen/known plaintext for successful differential and linear attacks, than exist for a 64 bit blocksize when 12 rounds or more are used. If this turns out to be true, the choice offered above loses most of its importance.

Summary and Conclusions

In the past, the creation of new ciphers has inadvertently led to the invention of new types of attacks. Now it would appear that the attacks invented against DES have spawned a new generation of ciphers that are (more) resistant to these attacks. Evidence suggests that some ciphers (i.e. CAST) can be made immune to linear and differential attacks. This does not make them safe by an absolute definition however. The results of Shannon [Sha49] still apply, telling us that ciphers that can be executed this efficiently, cannot provide us with unconditional secrecy. This means that we cannot guarantee that the ciphers produced by CAST will remain safe from attacks yet to be invented. No certainty can be given if such an attack exists or not or when it might be found, but undoubtedly, the race will be on to attempt and find it.

Chapter 8

DES modes

Written by *Jules van Kempen*

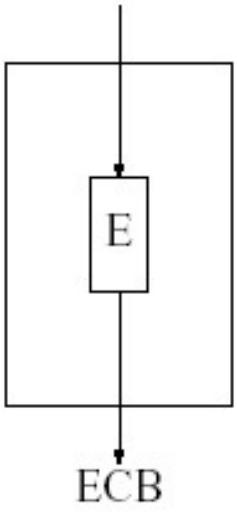
Block ciphers like DES, 3DES, AES, FEAL-8 and many more are well-known. They encipher blocks of data of a particular size (DES: 64b, AES: 128b, FEAL8: 16b) into blocks of equal size. However, normally one wants to encipher a text which is a lot larger than this block size. Then more needs to be done to successfully encipher this data. This is were modes of operation come into play. They define a method to map large plaintext files into ciphertext by using the block cipher. Typically these modes achieve this by chopping the plaintext into blocks of the desired size and then applying the block cipher directly on each part. However, just applying the block cipher directly on these blocks gives serious security problems. Therefore, modes use interaction with the previous ciphertext and/or plaintext to garble the encryption.

In this chapter we shall mainly focus on modes of operation for DES, but these modes can actually be applied to any block cipher. First we will discuss the 4 standard modes of operation as proposed by the introduction of DES, these modes are still used often and not only in combination with DES. Each of them has its own advantages and disadvantages. We shall then also discuss the so called *counter* mode (standard in AES) and hybrid forms between these modes. Finally, we will end with showing how DES modes can be applied to 3DES, since DES offcourse is not secure anymore.

8.1 The 4 standard modes of operation for DES

Together with the introduction of DES 4 modes of operation were proposed as standards to use with DES. They are **E**lectronic **C**ode**B**ook (ECB), **C**ipher **B**lock **C**haining (CBC), **O**utput **F**eed**B**ack (OFB) and **C**ipher **F**eed**B**ack (CFB). It is doubtful whether ECB is really a mode, as we shall discuss later on, since it is in our view more the absence of a mode. Each of the other 3 modes has its own benefits, but also its own limitations and/or risks. In this section we will take a closer look at each of these 4 modes of operation. All of them are still regularly used to encrypt messages, but offcourse no longer in combination with DES, since the DES key is too weak. In the following sections we will still use DES as the block cipher for the discussed modes, but one can actually replace this with any desired block cipher.

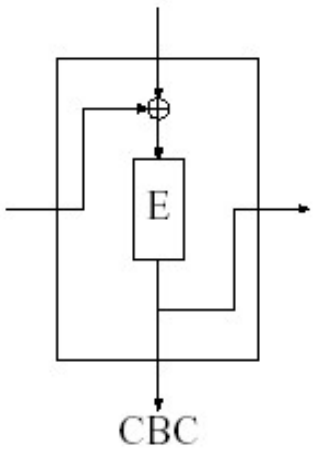
8.1.1 ECB



Electronic **C**ode**B**ook or ECB as we will call it from here on is the direct way of applying the DES block cipher on the plaintext. The plaintext is chopped into blocks of 64 bits and each of these blocks is directly encrypted by DES. This method gives rise to serious security threats. Since each block is encrypted in exactly the same way, 2 blocks with the same plaintext will be encrypted to the same ciphertext. This gives an attacker the ability to swap blocks, or worse, to insert blocks of plaintext into a message, given that he knows the encryption of these blocks of plaintext. Consider this mode when Alice wants to send the bank an order to pay 500 euro to Bob, Oscar intercepts the message, knows where Bob is named in the ciphertext and knows how his own name is to be encrypted by Alice's block cipher. He then takes out the block with Bob's name and inserts the block containing his own name. The bank will receive the tampered message, decrypt it and see that Alice wants to transfer 500 euro to Oscar. This is offcourse unacceptable, thus one wants to make sure that it is not possible to just insert blocks into a message without loosing feasibility of the message. The way this is achieved is by creating interaction between different blocks in the encryption mode. Therefore, all well-known modes have interaction between the encryption of subsequent blocks. As a result of this interaction, changing one block will result in garbling neighboring blocks. This is also why we stated earlier that ECB is more the absence of a mode than a mode itself in our view. Since a mode should be a method which defines how plaintext should be enciphered in combination with previously calculated results to provide safety. Whereas ECB just applies DES directly onto the plaintext, without specifying any interaction between results. In formula the encryption using ECB is the following:

$$cipher(n) = DES(plain(n))$$

8.1.2 CBC



Cipher **B**lock **C**haining is one the most used and popular modes. It solves the problem as described above with the insertion of blocks by creating interaction with the previous ciphertext during encryption. More formally, before the plaintext is encrypted by DES it is first XOR-ed with the previous ciphertext. As a result of this an attacker cannot modify the ciphertext without garbling the decryption of the next plaintext and therefore these attacks will be discovered. For encryption of the first block an Initialization Vector (IV) is needed to act as the previous ciphertext. This vector is a random 64 bit block which is sent in the clear to the receiver. In formula this mode is described in the following way:

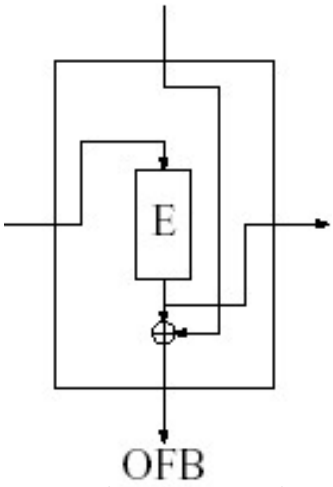
$$cipher(n) = DES(cipher(n-1) \oplus plain(n))$$

Because the ciphertext in almost all cases needs to be sent across a connection there is a high probability of transmission-errors. Therefore, transmission errors in one block should not result in decryption errors in the whole message, this could occur since each ciphertext is propagated forward in the encryption process. One of the goals of a mode should be to minimize the number of corrupted blocks when a transmission-error takes place. Obviously, 1 corrupted block is the minimum we can hope for since the block containing the transmission-error will be corrupted.

Using CBC, transmission errors will only corrupt 2 blocks, namely the block containing the error and the next block. This can easily been seen by looking at the formula of this mode: $cipher(n) = DES(cipher(n-1) \oplus plain(n))$. Here we can see that the n-th plaintext is completely determined by the n-th and $n-1$-th cipher and therefore only transmission-errors in these blocks will cause corruption.

8.1.3 OFB

Output **F**eed**B**ack also uses an initialization vector which is sent in the clear. This vector is then repeatedly encrypted with DES and the resulting blocks are used as a keystream to XOR the plaintext with. In Formula:



$$cipher(n) = DES^n(IV) \oplus plain(n)$$

An advantage of this method is that it enables the user to precompute most of the work needed for the encryption since there is no interaction between the successive DES operations and the plaintext or ciphertext. Just a keystream can be calculated and then later, one only needs to XOR the plaintext with this.

However, this mode is vulnerable to a different kind of attack, named the *bit-flipping* attack . If an attacker knows the plaintext corresponding to a ciphertext he is able to change the ciphertext so that the decryption of that block will result in a plaintext of his free choice. He does this by flipping the same bits in the ciphertext as he wants to see flipped in the plaintext. As a result of this, the same bits in the plaintext will be flipped in the decryption, this occurs since the plaintext is only subjected to a XOR in the enciphering.

Theorem 8.1 *In OFB, if for a block the plaintext is p and the ciphertext is c, then for this block, the plaintext p' corresponds to the ciphertext $c' = c \oplus (p \oplus p')$.*

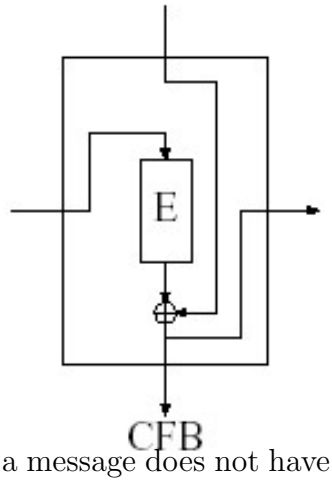
Proof. Clearly $p = c \oplus X$, for some fixed value X (this is the result of $DES^n(IV)$). Now if one feeds the decrypter c' instead of c in this block he will calculate the corresponding plaintext by $c' \oplus X$. Which is equal to $c \oplus (p \oplus p') \oplus X$, by transitivity of the XOR we rewrite into $(c \oplus X) \oplus p \oplus p'$, which is equal to $p \oplus p \oplus p'$, which is the same as p'. △

Furthermore, changing the ciphertext of a block doesn't influence the decryption of other blocks, since there is no interaction between ciphertexts and plaintexts, just a keystream is used.

Another disadvantage of the OFB method is that there is a chance, albeit very small, that one might accidentally choose a key and initialization vector that will cause the keystream to repeat in a short loop.

8.1.4 CFB

Cipher **F**eed**B**ack is very similar to OFB, but also uses interaction with the previous ciphertext during the encryption. More specific, the plaintext is XOR-ed with the DES encryption of the previous ciphertext. In formula:



$$cipher(n) = DES(cipher(n-1)) \oplus plain(n)$$

Since this mode also only uses a XOR to encipher the plaintext one might think that the same bit-flipping attack as described by the OFB mode could happen here. But this is not true, one is able to successfully change the plaintext of a block, but then the next block will be totally garbled, since this uses the previous ciphertext. So these attacks will be discovered in this mode. However, since the last block of a message does not have a successor, an attacker could perform this attack on the last block of a message without being discovered. So, one needs to make sure that the message ends with a block containing redundant information, i.e. only 0's, then there is no threat of a bit-flipping attack.

Since ciphertext is being propagated through the encryption, we need to look closer at the consequences of a transmission-error. When CFB suffers a transmission-error only 2 blocks will be corrupted, just as the CBC mode, as can be seen in the formula of this mode: $cipher(n) = DES(cipher(n-1)) \oplus plain(n)$. So the plaintext solely depends on the last 2 ciphertexts.

Also, one does not need to be afraid of cycling as is possible with OFB, since in CFB there is interaction with the previous ciphertext in generating the encryption.

8.2 Other modes of operation

After the founding of DES a lot more modes were proposed for use, but most of these seemed to do no better than the 4 basic modes of operation. Therefore we shall look into only one other mode namely the *counter* mode, since this mode does provide some additional safety aspects with regard to the 4 basic modes. But this mode has some problems of its own as well. Therefore, we will finish with looking at hybrid forms between the counter and the basic modes and show how they get the best of both worlds to create the best available modes of operation.

8.2.1 Counter mode

Counter mode is actually a mode associated with AES, it is the proposed standard mode for it. But it is offcourse also applicable on other modes, such as DES. Counter mode is quite similar to OFB, the DES block cipher is used to generate a keystream based on an initial value. Then this keystream is used to XOR the plaintext with. However, in OFB the block was repeatedly encrypted with DES to generate the stream, whereas in this mode, a counter function is used to generate a sequence of blocks and each of these blocks is then encrypted by DES just once. This makes sure that there is no risk of the keystream falling into a short loop, as there was in OFB. In Formula:

$$cipher(n) = plain(n) \oplus DES(counter(n))$$

However, the bit-flipping attack, as discussed with the OFB mode, is also applicable here. That is because the use of a pre-generated keystream. Using message authentication codes thus

is necessary to increase the security against such attacks. But it would be better to make these attacks impossible by proper construction of the mode of operation.

Another potential danger in the counter mode is the possible reuse of a counter with the same key. This will result in two identical keystream segments, a so-called *two-time pad* situation. This is a very dangerous situation since such a two-time pad situation reveals information about the plaintexts, as is shown in the NSA's VENONA project [Cro]. Therefore, it is necessary to guarantee counter uniqueness. As a result of this the used initialization vector is not random as it was in previous modes. Since random IV is often not completely random as a result of the system random generator not being completely random. They often use pseudo-random methods which could lead to the discussed two-time pad situation. So, to avoid potential failing counter mode does not use a random IV.

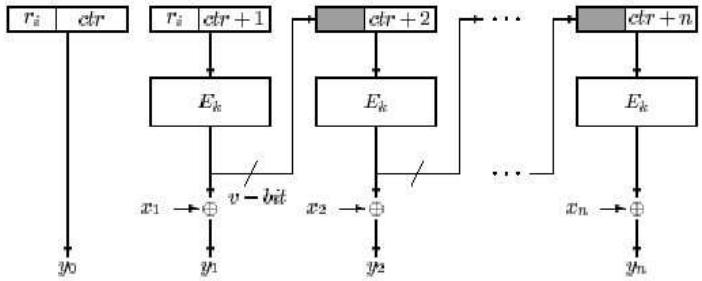
One might then wonder what the advantage of the counter mode is. The advantage is the proven amount of security of the counter mode in contrast with the proven amount of security of the 4 basic modes, the *concrete security* as it is named. Bellare et. al. [BDJR00] have proven tighter bounds on the security of the counter mode than the security of the 4 basic modes. They do so by assuming that the used block cipher is indistinguishable from a random permutation function, the so called *random oracle model*. Then they look at the *advantage* an attacker can still achieve based on looking at results from the encipherment. They define this advantage as the capability of an attacker to distinguish a cipher from a random permutation. This is a standard definition of confidentiality in theoretical cryptography. It is used since the user will not be able to tell anything from a cipher if it is just a random number to him. Therefore, the user will need much more encrypted data in counter mode before he could *potentially* gain the same *advantage* as in the basic modes.

8.2.2 CTR-OFB

To get the best of two worlds hybrid modes of operation have been developed, they have the same *concrete security* as the counter mode in the model of Bellare et. al. and they do not have the problems that they might accidentally create a short cycle. The first of two of these hybrid modes we shall discuss is CTR-OFB for Counter - Output Feedback. During the encryption the plaintext is XOR-ed with a keystream as in OFB, but in contrast with OFB were the new keystream block was created by applying DES directly on the previous keystream block, here we let a part of the block consist out of a counter and let the other part consist of part of the previous keystream block. This way accidentally choosing the same counter does not have to lead to a two-time pad situation since part of the block consists of a random value, which will create different keystreams. In formula:

$$keystream(n) = DES(firstPartOf(keystream(n-1)) + +counter(n))$$
$$cipher(n) = plain(n) \oplus keystream(n)$$

Sung et. al. [SLL$^+$] proved the same security for CTR-OFB as the counter mode, by applying the same proof for CTR-OFB as Bellare et. al. [BDJR00] did for the counter mode. This way this mode inherits the best qualities of both modes. However, this mode is also potentially weak, since a keystream is used to XOR the plaintext with, which gives an attacker the possibility of a bit-flipping attack as discussed before. Using a message authentication code can help with this problem, but it would be nicer if the mode did not have this weakness at all. An advantage of this mode is that it can be precomputed, since the keystream generation has no interaction with either ciphertext or plaintext.

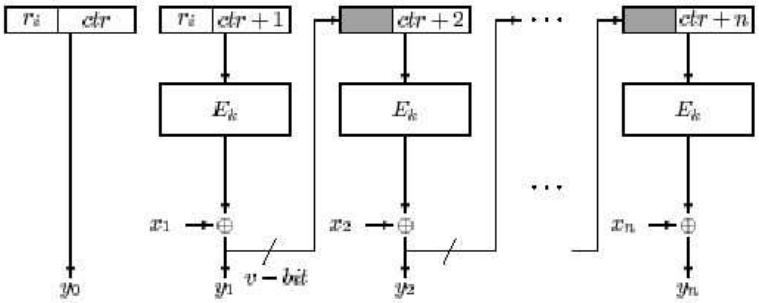


8.2.3 CTR-CFB

The second hybrid mode that we will discuss will be the CTR-CFB for Counter - Cipher Feedback mode. It is very similar to the CTR-OFB mode, but instead uses interaction with the previous ciphertext while encrypting a block. The difference between CTR-OFB and CTR-CFB is exactly similar to the difference between OFB and CFB. Here the DES block cipher is executed on the previous cipher to generate part of the new block. In formula:

$$cipher(n) = plain(n) \oplus DES(firstPartOf(cipher(n-1)) + +counter(n))$$

Sung et. al. [SLL$^+$] proved the same security for CTR-CFB as for CTR-OFB in similar fashion. Also, there is no risk here of a two-time pad as with the counter mode. Therefore, this mode also gets the best of both worlds. In comparison to CTR-OFB this method provides better security, since there is no chance for a bit-flipping attack as a result of the feedback of the previous cipher. But this also makes that the encryption can not be pre-computed, as was possible with CTR-OFB. Also, with CTR-OFB there is still a very slim chance to generate a two-time pad, in the very rare case that the same key and random value are used. CTR-CFB has no such problems at all, since it interacts with the previous ciphertext. Therefore, to our opinion, the CTR-CFB mode provides the best characteristics of all modes and can be used best. If there however is little time, but there is the chance of doing pre-computing, then CTR-OFB provides the best security.



8.3 Modes for 3DES

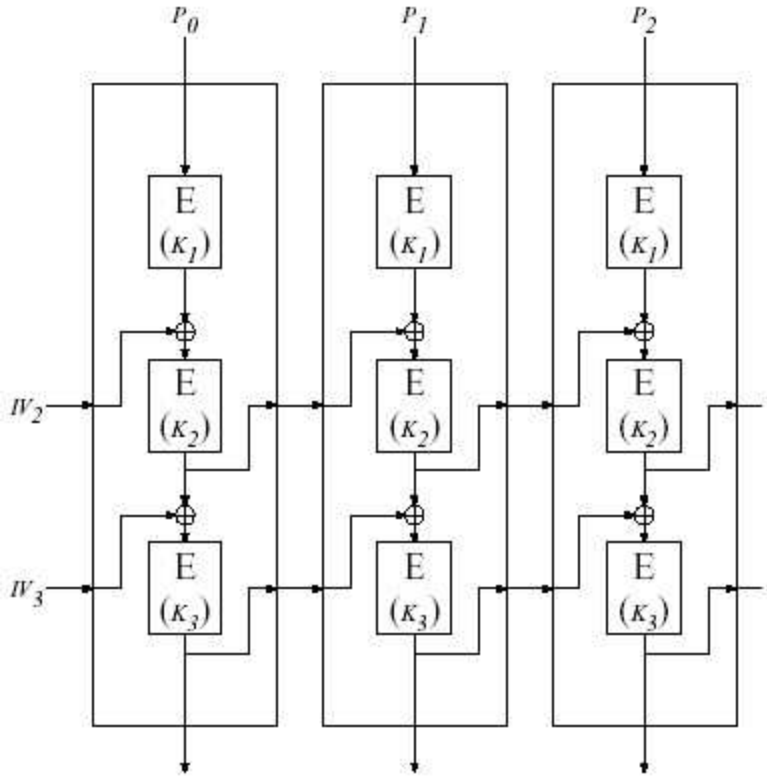
So far we have only discussed modes in the context of the DES block cipher, however, DES is not used anymore since its key is to weak, therefore we will look at the replacement of DES, 3DES and see how the modes are applied with regard to this block cipher.

The most simple way to use modes for 3DES is by using the DES modes of operation and replacing the DES block cipher by the 3DES block cipher. However, suggestions have been made

that by applying multiple modes to 3DES the effective keysize could be increased. In 3DES the effective key is only 112 bits (out of 168 bits), as a result of the meet-in-the-middle attack. Therefore, it is justified to further go into this possibility. Applying multiple modes to 3DES is mostly done by defining three modes which should be used, the first mode is built around the first DES encryption, the second mode is built around the second DES encryption and the third mode is used around the third DES encryption. As a result of this it is believed by some that effective keysizes of up to 168 bits can be achieved, a significant increase! However, until this moment there is no proof of this 168 bits effective keysize being achieved, even worse, many of the multiple modes of operation for 3DES have been proven to weaken the key to the level of single DES! Which means that they are not safe. We will give an example of such a multiple modes system for 3DES and show how it can be broken under Chosen Ciphertext Attack (CCA). The question which we would like to see answered in the end is in which way one could apply modes to 3DES best.

8.3.1 Breaking multiple modes of operation for 3DES

In this section we will give an example of how applying multiple modes of operation to 3DES can result in the key being broken. We will do this for the example where the three modes used are ECB, CBC and CBC, in that order. We will perform a Chosen Ciphertext Attack on the first key with the use of differential cryptanalysis. First, we will explain this mode. A block of plaintext $p(n)$ is first enciphered directly by DES, giving the intermediate ciphertext $c_{i1}(n)$ then this ciphertext is used as input for the CBC mode, so it is first XOR-ed with the previous ciphertext $c_{i2}(n-1)$ and then the second DES cipher is applied onto it. This gives the second intermediate ciphertext $c_{i2}(n)$, then CBC is applied onto this cipher as well. This results in the final ciphertext $c(n)$.



Under Chosen Ciphertext Attack the attacker will be able to choose the cipherblocks and can get the plaintext corresponding to this ciphertext. He thus is able to choose tuples of blocks (c_0, c_1, c_2) and get the corresponding plaintexts (p_0, p_1, p_2). He will use this ability to form pairs of

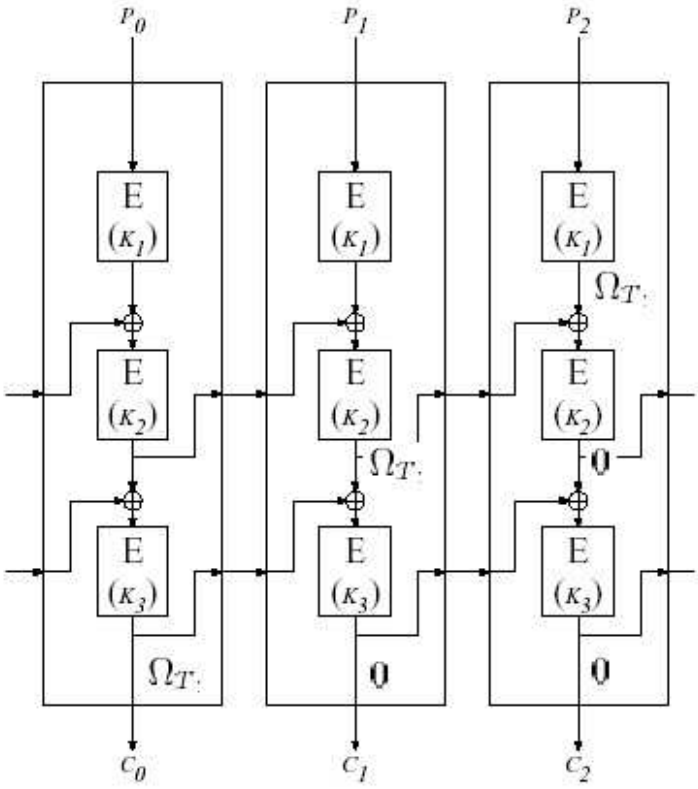


Figure 8.1: Flow of the difference between the 2 decryptions through the system if (c_0, c_1, c_2) and $(c_0 \oplus \Omega_T, c_1, c_2)$ are entered

these tuples (c_0, c_1, c_2) and $(c_0 \oplus \Omega_T, c_1, c_2)$ and receive the corresponding plaintexts (p_{00}, p_{01}, p_{02}) and (p_{10}, p_{11}, p_{12}). One should be aware that the chosen ciphertexts thus only differ in their first block, and the difference between these first blocks is Ω_T. As we now look at how the data will flow through the system in the decryption we see that the first intermediate value of block 3 will differ by the Ω_T we had chosen in the beginning. Also, we know the plaintexts corresponding to block 3. With this information a differential cryptanalysis can be performed on the first key.

However, some details need to be taken care of in order to apply the standard differential cryptanalysis to the full 16-round version of DES, since in this analysis it is assumed that both the plaintexts and ciphertexts are known, whereas we only know the plaintexts and the difference of the ciphertexts (we have no idea what the actual ciphertexts at the end of round 1 look like, we only know their difference). But with some additional calculation Eli Biham [Bih98] claims that this can be done in complexity lower than single DES, and practically achievable. So we can break the first key, next we need to break another key, which can also be done in complexity of single DES, we will omit the details about it, since the most difficult and interesting task is to find the first key as one can probably imagine. So, concluding, instead of giving us a larger effective key, this mode gives us a far smaller key than 3DES, even worse, the mode can be broken in practice!

In his paper [Bih98] Eli Biham further shows how the 3DES modes CBC—ECB—CBC, CBC—CBC—ECB and CBC—CBC—CBC can be broken under Chosen Ciphertext Attack in complexity of 2^{58}, which is far too close for comfort. Therefore these modes are not safe and should not be used.

8.3.2 The best mode for 3DES

As we have shown some multiple modes of operation for 3DES are not safe and the other modes of operation are not known to be strong or weak (they have an effective keysize between 56 and 168 bits). In contrast, using 3DES with one mode of operation is known to give an effective keysize of 112 bits, which is secure. Therefore, to us there seems to be no reason on this moment to risk weakening the key by applying multiple modes of operation, since this might result in an unsafe encryption. As for the best mode to use on 3DES we would say that this is the same as for single DES, since they are similar systems.

Summary and Conclusions

In this chapter we have seen how modes of operation work and which modes there are. For each mode both strong and weak facets are present. However, the hybrid forms CTR-CFB and CTR-OFB seem to have the best qualities, with a small favor for CTR-CFB as it makes bit-flipping attacks impossible. For to the more practical question which mode of operation is best to use with 3DES, we argued the best option is to use a single CTR-CFB mode around the 3DES block cipher.

Chapter 9

Steganography

Written by *Teun Slijkerman*

In the year 2000 the *USA Today* reported that terrorist groups were using big public websites such as eBay to give orders, distribute plans etc. The idea was that the information was hidden in images posted on these websites.

In ancient times messages were written on the back of wax writing tables or tatooed on the heads of slaves.

The preceding examples are two different forms of steganography. This chapter aims to give an introduction to the field of steganography.

9.1 Introduction

The goal of steganography[1] is to send a message to someone (over a public communication channel) without anyone else knowing the message was sent. In contrast with in cryptography, were the goal is to hide the content of a message from an outsider, steganography tries to hide the very existence of such a message.

One of the standard problems in steganography is the *prisoners problem*. Suppose there are two prisoners, Alice and Bob, who want to plan an escape. For this escape to be successful they have to communicate. The only problem is that they can only communicate via the warden Ward, who will read all messages. If the warden gets suspicious, both Alice and Bob are placed in solitairy confinement. So how can Alice and Bob send eachother messages coordinating the excape without raising the suspicion of Ward. They can't use any cryptography since Ward will see a message wich he can't read and knows immediately something is going on. The sollution is to hide the actual messages within seemingly innocent messages.

This chapter will be devided in two parts, sections 9.2 and 9.3 will explain some basic steganography methods, section 9.4 will illustrate an attack on the two steganographic systems described in section 9.3.

[1] $\sigma\tau\epsilon\gamma\alpha\nu\circ\varsigma\ \gamma\rho\alpha\phi\epsilon\iota\nu$: covered writing

9.2 Null Cyphers

Throughout the history, one of the most commonly used steganography methods are *Null Cyphers*. These null cyphers used a very simple rule instead of an immensly complicated algorithm. A classic example is shown here:

```
PRESIDENT'S EMBARGO RULING SHOULD HAVE IMMEDIATE NOTICE. GRAVE
SITUATION AFFECTING INTERNATIONAL LAW. STATEMENT FORESHADOWS RUIN
OF MANY NEUTRALS. YELLOW JOURNALS UNIFYING NATIONAL EXCITEMENT
IMMENSELY.

APPARENTLY NEUTRAL'S PROTEST IS THOROUGHLY DISCOUNTED AND IGNORED.
ISMAN HARD HIT. BLOCKADE ISSUE AFFECTS PRETEXT FOR EMBARGO ON
BYPRODUCTS, EJECTING SUETS AND VEGETABLE OILS.
```

These two messages were sent during the 1st world war by the German embassy in the United States. If you take the fist letter of every word in the first message or the seccond letter of ervey word in the second message. You can read the actual message which is:

```
PERSHING SAILS FROM N.Y. JUNE 1
```

When you need to send an image to someone, it is possible to create a powerpoint presentation and hide the image behind another image. It is also possible to hide bits in the source code of a webpage by encoding a 0 as a space and a 1 as a tab at the end of a line.

All of these methods are extremely simple, but can be very effective as long as nobody suspects anything. As long as you send enough messages/powerpoint files to someone, one message with extra content will not raise any suspicion.

9.3 Least Significant Bit steganography

As we have seen in the previous section, there is no need to use complicated algorithms to hide a message. As long as there is enough 'normal' communication going on, one more message will not raise any suspicion. This assumption is the basis for the steganography method described here.

9.3.1 Introduction

In this digital era allmost all media is stored digitally. This has resulted in enormous amounts of images, audio files and videos which are stored on the world wide web and every day new files are added. This results in a gargantuan amount of possible files to hide our message in. To be able to hide something in a file, it is imperitive to know the structure of the file. Otherwise you could end up with an image that won't work.

For the remainder of this chapter, we will focus on the BMP and GIF file-formats for the simple reason that hiding something in a these types of files is easy. In principle all file-formats have a potential for hiding a message is, but the algorithms can become very complicated. Take for instance the JPEG format. These files do not just describe the colors of every pixel, but the image is stored as a sequence of parameters for discrete cosine transformations (DCT's).

Figure 9.1: An image containing airport blueprints

9.3.2 S-Tools

In a BMP file every pixel is encoded by 8 (Greytone) or 24 (RGB color) bits. Let's take a random pixel with 24-bit encoding:

```
10010110 11010011 00100101
```

This pixel has a Red value of `10010110` (150), a Green value of `11010011` (211), and a Blue value of `00100101` (37). Suppose we want to hide a bit in the green value of this pixel but change the value as little as possible. The obvious way to do it is to use the least significant bit (LSB) to encode our bit as it will possibly change the value to `10010111` (151).

S-Tools uses this technique to store up to three bits of data per pixel. The potential size of the secret message for a 1024∗768 Bitmap with 24-bit color encoding is 1.024∗768∗3 = 2.359.296 bits (3 bits per pixel), which is 294.912 bytes.

To provide some extra security, S-Tools uses a semi-random generator to generate the order to hide the data. The only things needed by the receiver are the image with the message embeded and the password used by the semi-random generator. Figure 9.1 shows an example of blueprints of an airport hidden by S-Tools in an inconspicuous image.

This one of the simplest ways to hide data in an image. When S-Tools has to hide data in a GIF-image, things get a little more complicated. The used algorithm and the problems raising are described by Bailey and Curran [CB].

9.3.3 EzStego

When creating a GIF image, every used color is stored in a palet which can hold up to 256 different colors (Figure 9.2). Instead of storing the RGB information of every pixel, every pixel is described by a pointer to the palet. This method results in the use of 8 bits per pixel instead of the 24 needed in a BMP.

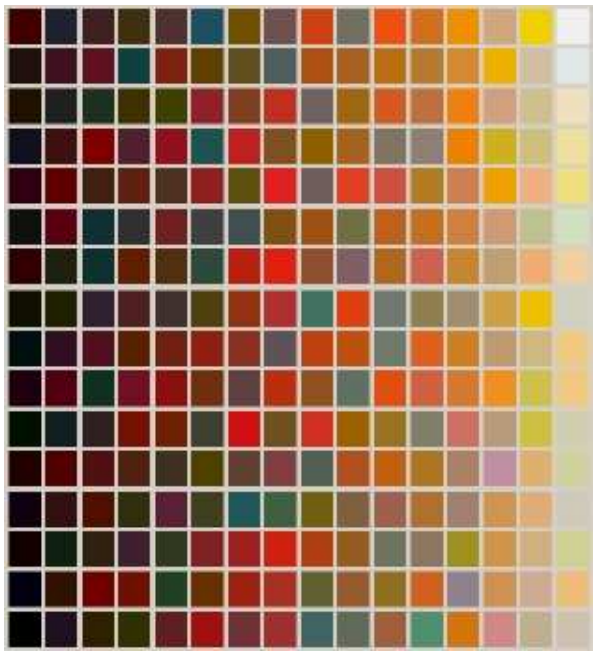


Figure 9.2: A palet of a gif image

The embedding function of EzStego uses a copy of the palet of the image which is sorted by luminance. This results in a palet where the human eye can hardly distinguish between two adjacent colors. The embedding function works line by line, starting from top left and ending at the bottom right of the picture.

For every pixel of the image the function matches the LSB of that pixel with the bit to be embedded, and replaces the colour by its neighbour in the sorted palette if necessary.

Figure 9.3 shows a simplified example of the embedding function of EzStego. Suppose we have a pixel with color 1. If we want to embed a `0` we change the color to color 4, if we want to embed a `1` we do nothing.

Extracting the hidden message from the image is pretty simple. All we have to do is get the LSB of the index in the sorted palet for every pixel.

9.4 Steganalysis: Visual attack

This section will describe a simple attack on the two steganographic systems described in the preceding section. The two methods described above are based on the assumption that the human eye cannot distinguish between two similar colors. This may be true, but when someone suspects an image contains a hidden message this assumption often doesn't hold. The most basic attack possible, the visual attack, will be descibed in this section.

9.4.1 Introduction

The idea behind a visual attack at LSB steganography is that the LSB's do reflect the main features of the image to some extend. When data is hidden in a file, the first thing to do is

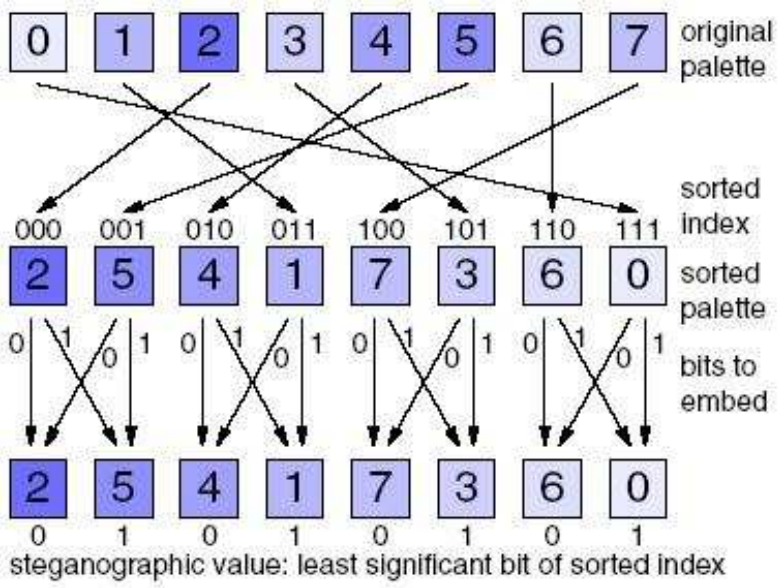


Figure 9.3: The embedding function of EzStego

compress the data because of the limited storage space. You may want to encrypt the data also if it is important to keep the contents hidden even if the steganography is broken.

Both of these processes (compressing and encrypting) result in an allmost equal distribution of 0's and 1's closely resembling random data. When embedding these bits into the carrier image, the main features represented in the LSB's get overwritten with 'random' data.

9.4.2 Attacking S-Tools

To execute a visual attack on an image possibly containing a hidden message, the first thing we have to do is to create a filtered image. The filter should only take the LSB's of the original image. Figure 9.4 shows the result of such a filter used on a picture with no hidden data and with a message with the maximum length embeded.

9.4.3 Attacking EzStego

To execute a visual attack on EzStego the filter function has to be modified a bit. Figure 9.5 shows the way the filter function works. First of all the palet is sorted the same way EzStego does when embedding a message. Then the colors of the palet will be changed to only black and white. This is done is such a way that the LSB of the index of a color determines the color.

Figure 9.4 shows the result if EzStego has embedded a small message in an image. Because EzStego stops embedding data after the message ends, there is a clear line indicating that there is something embedded.

9.4.4 Conclusions

The visual attack seems to work quite nicely on the examples provided above. Unfortunately this is not allways the case. Take figure 9.6 for example. Can you determine whether there is something embedded in this picture by looking at the filtered image?

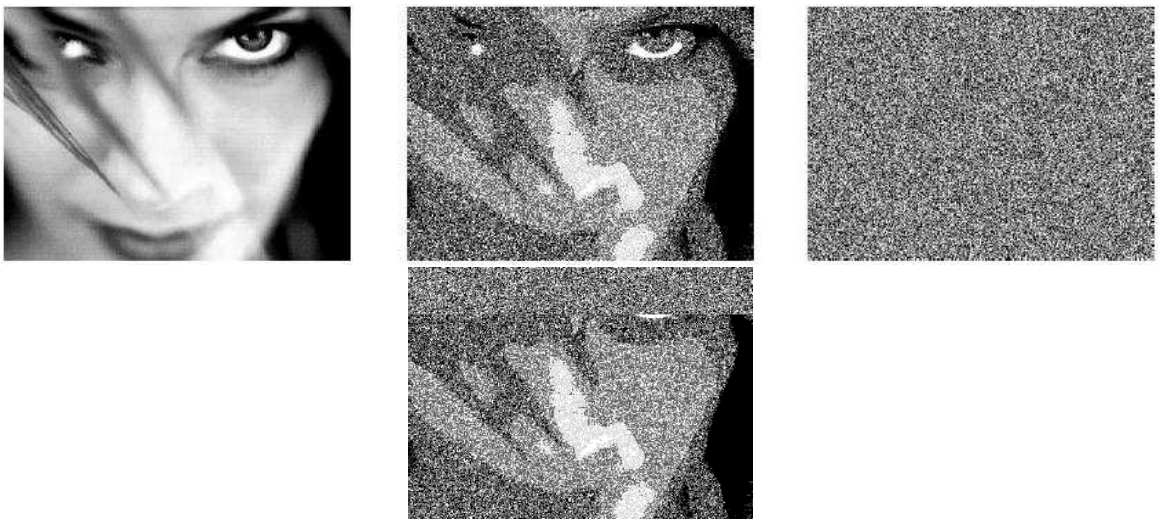


Figure 9.4: A BMP image (l), the same image with only the LSB's (m), the image with a maximum message embedded(r) and the image with a small message embedded by EzStego (bot)

Niels Provos and Peter Honeyman from CITI [PH] have done some research on the use of steganography on eBay. The method they used was a statistical attack. When used on the image in figure 9.6 the result is that there is a message embedded in the image.

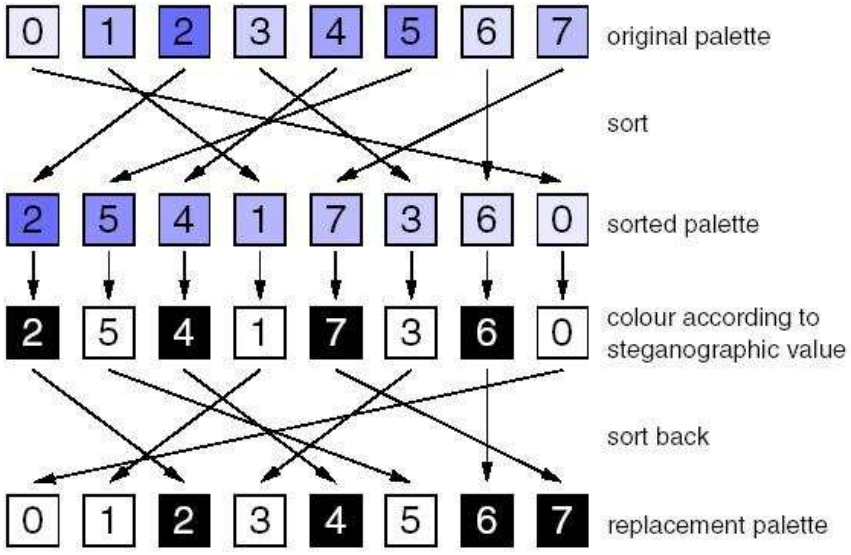


Figure 9.5: The filter function used to attack EzStego

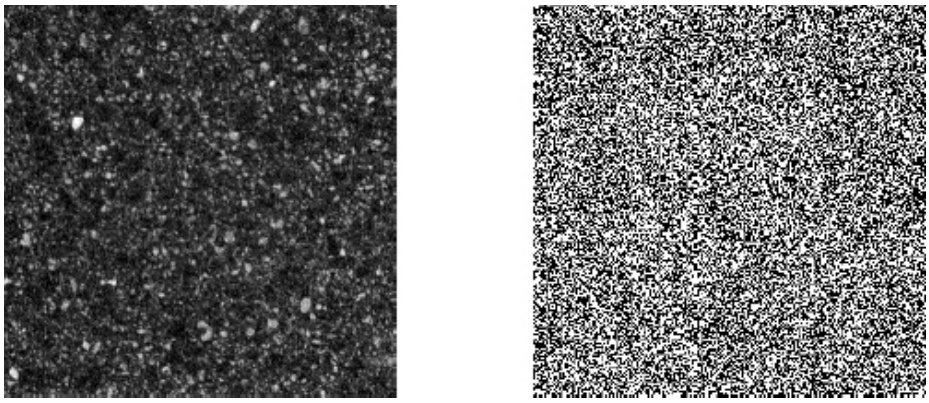


Figure 9.6: Is there something embedded?

Summary and Conclusions

The field of steganography is a relatively young topic for research. As a result of that the quality of the used methods is not that high yet. It is to be expected that there will be big improvements in the future.

Chapter 10

EHR: Electronic Health Record

Written by *Wouter Slob*

Inadequate communication between health workers is the cause of many medical mistakes. That is shown as result of a research done by TNS NIPO by order of NICTIZ (Dutch National Health ICT-Institute). The consequences of these mistakes can be very serious for the patient: remaining physical problems, emotional problems and even impossibility to work. About 50,000 people can't work anymore because of medical communication problems. An EHR can prevent many problems.

About 800,000 people above 18 years have experienced medical communication problems. They received the wrong medicines, a wrong surgery or treatment, or couldn't be treated because of insufficient medical information. The research estimated the total financial costs per year at 1.4 billion euro.

An Electronic Health Record (EHR) can prevent many problems. The NICTIZ is trying to create such an EHR. Some hospitals are already experimenting with an EHR. The goal is to have a national medication record at the end of 2005. The final goal is to record and make available all treatment-relevant, medical history of all inhabitants of The Netherlands.

10.1 History of Medical Records

10.1.1 Paper Medical Records

All current versions of the electronic records of the care provided to patients are based on the traditional medical record, which is the paper version. The phrase "medical record" may be applied in differing ways based on the actual healthcare practitioners providing the care, but most persons understand the medical record to be a history of the care they received from various clinicians. An important characteristic to keep in mind is that EACH practitioner keeps its own medical record for each of its patients. There is no integration of the data from the various clinicians treating the patient. Therefore, as a patient, you most likely have many different medical records.

Casus 10.1: Medical Errors

Research has shown that many people encounter medical errors caused by communication problems.
Example:
A patient with a serious form of intestine cancer wants to be treated in a specialized hospital. The local hospital, which has detected the cancer, sends an incomplete medical record of the patient to the other hospital. The research is incomplete and essential photographs are missing along with blood information. The patient got the impression that the information must be somewhere in the local hospital, but not on the right place. The consequences are that several researches must be repeated.

10.1.2 Computerization of Patient Records

The original objective for computerization of an individual's health record can be found in the Computerized Patient Record (CPR), which was defined as a computer-based record that includes all clinical and administrative information about a patient's care throughout his or her lifetime. The documentation of any practitioner ever involved in a person's healthcare would be included in the CPR, extending from prenatal to postmortem information. One of the expectations for the CPR included its role in decision support. Over a decade ago, the term CPR was used to distinguish the concept from the more traditional medical record to incorporate administrative and financial data often excluded from the patient medical record.

10.2 The users

In Health Care we globally consider 3 different users of the EHR:

1. **The Patient**
2. **The Health Worker** (the person or institute that threads the patient)
3. **The Health Insurance** (those with whom the patient is insured with)

10.2.1 Patient

The patient is a person who is in need of medical treatment and visits the health worker. The patient wants to be treated as fast and as well as posible. Furthermore he wants to be treated in confidentiality with maximum security to his personal data. In accordance with law he must have the right to view his personal file and have the opportunity to erase (parts of) it.

10.2.2 Health Worker

The health worker is the person or institute who treats the patient. To treat the patient he will often need access to the medical history of the patient (of course only the relevant parts). In case of emergency a doctor may need immediately access to the patient's history, without having the time to ask permission.

The health worker is responsible for documenting all the treatments (dossier duty) and he is *also* responsible for the security and accuracy of this data.

10.2.3 Insurance

The Health Insurance wants information about the performed treatments, so they can verify that the compensation the patient claims is covered in his insurance and needs to be paid out.

10.3 Relations and Demands on EHR

In the EHR we have to deal with 1 patient, 1 Health Insurance and 1 or more Health Workers. All these parties want / need access to the EHR, but not everyone is permitted to see all data. And this is very important, because physicians has sworn that they will 'keep secret everything a patient tells them in confidence'. An EHR must be completely secured before a physician will enter data into it, because he can get into big trouble if anyone could access that data without authorization.

On the other side, in case of an emergency a doctor might need access to the data, without having the time to ask permission.

A patient also needs access to his complete EHR to view, add or delete parts of it.

Because a lot people might access the EHR and because someone's health depends on it we need a very good logging system, which records all activity according the EHR. (so that no one could change things in case of an insurance claim or a judicial investigation).

Roughly we can define a couple of demands on an EHR-system:

- A patient need access to his own EHR and must be able to allow access for physicians.
- A health worker must be able to access new data to the EHR this data must be accessible only for the Health Worker, the patient and people authorized by one of them.
- In case of an emergency it has to be possible to ignore the rule above.
- Every action according the EHR must be logged and must be verifiable .

10.4 Content of the EHR

Because an EHR lasts a lifetime, we can divide it into 2 parts:

1. A personal overview, with personal information like address-, family-, insurance-data.
2. A list of so called patientparts. Every patientpart contains information about a certain treatment. (The patientpart will be explained in more details below)

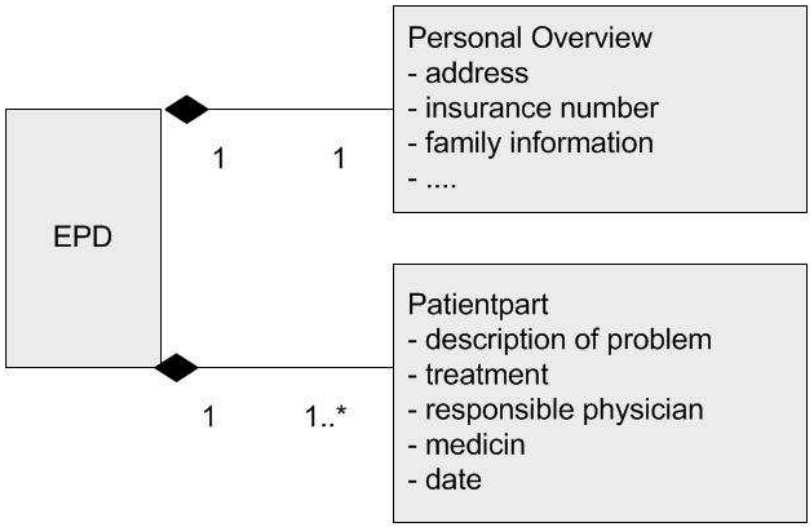


Figure 10.2: EHR UML

10.4.1 Personal Overview

The data in the personal overview contains information all parties often need access to. It contains administrative information about the person. This information is not medical related and is just needed to handle all the administrative parts. Because everyone needs access to this information it is not needed to secure this per user. We can say that everyone who has access to some part of the EHR also has access to the personal overview. (NOTE: a patient must still have the opportunity to hide information in this part whenever he wants to)

The Personal Overview is initialized when the patient visits a Medical Care for the first time. The Personal Overview is stored at that institute. Whenever the personal data changes in the future, the up to data version will be stored at the institute which made the changes (and so becomes responsible for the data).

10.4.2 Patientpart

Every time a patient visits a health worker, a new patientpart is created. The patient part contains all information about the treatment. (like: kind of treatment, medicines, responsible health worker, date, etc.) It is, of course, very important to secure the data in the patientparts. Not all users need access to all the data in all the records. If anyone can access the data, the health worker can't control the information the patient has given him in trust.

10.4.3 Storage

But where do we store the EHR? There are 4 concepts for a system:

1. Central Database
The first idea is to store all the data into a single database. The advantages are that we've got all the data together and immediate access to a patient's medical history. The disadvantages are that if an attacker hacked into the database he has access to all the data. And another

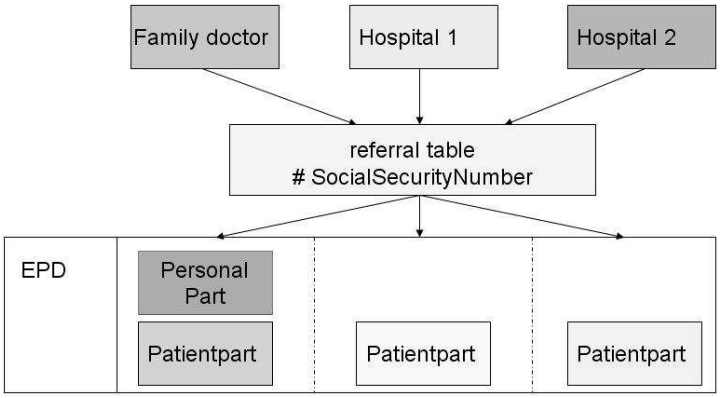


Figure 10.3: EHR Structure Example

disadvantage is that the healthworker might forget to update his record and, as a result, we get an incomplete record in the database.

2. Chipcard
The second idea is to store all data onto a kind of chipcard. The advantage is that the patient has great supervision about the people who access his data. The disadvantage is that he can forget / lose his card and so nobody has medical information.

3. Online storage
The third idea is to store the data online and give the patient control over it. The advantage is that the data is always available, but the disadvantage is that it isn't very safe on the internet and because a patient can edit his own medical record, a health worker hasn't got the certainty that the data is correct.

4. Decentral storage
The fourth and final idea is to store the data where they are created: with the health workers. In that case the health worker can be sure of the security of the data and whenever the health worker is hacked, the attacker has only the information of 1 treatement. The only thing we need is a referral table with information about where all the parts of a EHR are stored.

10.5 EHR-security with Biometrics

The EHR is based on the idea of supplying relevant medical data to authorized health workers. The data are characterized by highly medical relevance and highly privacy-sensitivity. It is important to control the access to the data, to guarantee the quality of the health data. Because incorrect data can lead to health damage or even life threatening situations for the patient it is very important to take good care of it Considering all this security risks it is easy to understand that it is necessary for a user to login before getting access (identification). Secondly we must verify the persons identity. And finally we must check if the person is authorized to access the data he is requesting.

10.5.1 How does identification with biometrics work?

For the identification we need a system which satisfies the following demands:

- **Identification:** Every patient must be able to identify himself.
- **Verification:** After the identification of a user it is important to verify that id. It is easy to tell a physician that you're name is 'Alice'. If a physician believes everyone immediately it is very easy for Oscar to say that his name is 'Alice'. It also important to check it because you might accidentally give a wrong identification to the physician and he might give you the wrong treatment on base of another person's EHR.
- **Authentication:** Authentication is the actual verification of a person. This is done by a special authority. If some wants to verify an ID he sends that ID to the authenticator and the authenticator replies whether the person is who he claims to be. We verify the ID of a person by checking random biometric data.

Biometric comprehends analyzing biometric data with mathematical and statistical methods. The system to verify the biometric data exists of:

- a sensor which records the human characteristics.
- an application which verifies the identity of the user.
- a place to store the reference material ('template')

1. Template
The template T is created by measuring body characteristics B with a sensor. We first encrypt the sensor readings into digital data and store the data into a template.

Definition 10.1 *The template is the encrypted sensor data:* $T = Enc(B)$.

There are two kinds of templates: the 'personalized template', which is unique for every person and the 'anonymous template' which contains only the last k digits of $Enc(B)$ (the same specification as used with bankcards).

A 'personalized template' can also been as an unique identifier for a certain person. And because privacy is very important in the EHR we prefer to use the 'anonymous template'.

The idea behind the 'anonymous template' is that it is almost impossible to fake all biometric parts of a human (remember that we take random biometric data to verify). And because the template is unique it is also impossible for hackers to match an EHR with a person once they hacked into the EHR-system. So the change of abuse with an 'anonymous template' is very small.

2. Storage: Healthcard (chipknip)
The use of a kind of Health Card (chipcard), to store to the template on, expands the space to allow a low error-reject degree because the high error-accept degree can be compensated because the information carrier must be property. Another option to store the templates is a central database. Because of security reasons there is chosen for the chipcard. Implementation of a central database is too complex and is quite easy with a chipcard.

3. Verification
In the operational stage we use the stored biometric template to verify the identity of a user.

body characteristic	eye		hand		face	behavior		
test condition	iris	retina	hand	finger print		voice	sig-na-ture	tests
Reliability	++	±	±	+	?	-	±/-	-
User-friendliness	-	-	+	±	?	±	+	±
Public acceptance	-	-	-	-	-	?	?	?
Costs	-	-	+	+	-	+	+	+

Figure 10.4: Table with summary of Biometric systems

With a sensor we measure the body characteristics again and we create a new template T^*. The verification application checks if there is a match: $T \approx T*$

We use a margin in the progress, cause there is always some deviation in two measurements of the same person and a exact match is almost impossible.

10.5.2 Biometrics to use

Human characteristics which can be used for biometrics can be divided into two categories: the physical characteristics and the behavior characteristics. With physical characteristics we mean data like fingerprint, blood vessel patron on the retina, iris-, finger-, hand and face geometry. With behavior characteristics we mean data from analyzing signature drawing dynamics and voice signals. The choice between on or more human characteristics for biometrics uses comes with a couple of assessments. We consider the reliability of the readings, the user-friendliness and public acceptance of the scanning procedure, the storage limitations of the Healt Card, the implementation costs and whether a scanning of characteristics needs a special act of the user.

10.5.3 Reliability of Biometrics

Biometric access-security is generally reliable. Most of the used systems are hard to mislead; for example they can differ a living finger from a not-living finger. One of the most trustworthy ways to identify and verify a person is by the use of iris recognition; the details of the eye are unique and stable. An other reliable way is realizable with fingerprint recognition. Other, less reliable, ways are using signatures or palm of the hand patterns. Voice recognition is the least reliable (and some people also believe a signature can be faked easily.

Because we want maximum security for the EHR, we choose biometric identification with measuring the iris or the fingerprint of a user. (Or, even more preferable, a combination of those two).

10.5.4 User-friendliness of the Sensors

Although manufacturers say the chance of physical damage nil, people don't fancy looking into a scanning device for an iris scan. Contact less methods, like taking a picture from some distance and using it for face recognition, are preferable, but unfortunately not very reliable. That's one of the main reasons a complete system with biometric security will take a while.

10.5.5 Storage of the template

In the operational-fase, the stored biometric template is used to verifiy the identity of a user. With a sensor we rescan the body characteristic and create a new template. The verification application compares the new template with the previous stored one. If the value of the new template matches within a certain margin with the stored one, the user is allowed. (We use a certain margin, because there are always some deviations when measuring biometrics)
We can store the template in two different ways:

Decentralized storage
If we store the template decentralized, the template is stored on a chipcard (Health Card). And we use the chipcard to verify the identity of a person. In this case we can still verify a person when the rest of the system is down.

Central storage
In this case the templates are stored into a central database. To verify a person we can query the database. We've got two kinds of searches onto the database: the direct and the indirect search. With an indirect search we compare the template of the user with all templates stored into the database (personalized template). With a direct search we compare the template of the user with a specific template (anonymous template).

10.5.6 Errors and security

Owing to certain circumstances verification with biometrics might fail: the new encrypted template don't match with the original stored template, even though it is the right person (false rejection). There also might be a template made from a person who isn't in the database, but the system finds a match (false acceptance). False rejection or acceptance can be caused by certain physical paramaters (temperature, dust) or by certain biological circumstances (sweat).

If the chance on false rejection is low, the chance on false acceptance might be high. That's why we want to combine biometrics (identification based on 'being') with the use of a loose information carrier (identification based on 'having'). The change that an attacker can create a card to access someone else's data is very small.

10.6 Resources

- http://www.nictiz.nl
- http://www.hl7.org/
- http://www.iom.edu
- http://www.hipaadvisory.com/action/ehealth/
- http://www.epddag.nl/
- http://www.health.tno.nl/
- http://www.memag.com/memag/article/articleDetail.jsp?id=143144

Chapter 11

C2000

Written by *Jan-Pieter van den Heuvel*

In this chapter the security and cryptographic aspects of the Dutch C2000 communication system will be discussed.

11.1 Introduction

11.1.1 What is C2000?

Communicatie 2000 (C2000) is a mobile communications system used by the Dutch public safety services. To serve the public better and to respond to serious incidents faster, emergency services in more and more regions are sharing a single emergency centre. C2000 is used primarily by fire department corps, ambulance services, police and the military police. Other parties may use this network for public safety but they have to apply first at the Ministry of Interior. The network replaced over 100 individual analog radio systems used by the public safety services.

The system was built on the order of the Ministry of Interior and Kingdom Relations, the Ministries of Defense and Public Health, Welfare and Sports. The Ministry of Interior is responsible for the contruction and the management of the whole infrastructure (base stations, fibre glass network, link stations, etc.). The users of the C2000 system are responsible for the purchase and the management of the user equipment (mobile stations, alarm receivers, etc.). The actual execution of the project will be performed by the ICT Service Cooperation Police, Justice and Safety (ISC). ISC is an agency of the Ministry of Interior and Kingdom Relations.

11.1.2 Network infrastructure

The C2000 network consists of 400 base stations, 15 link stations, a fibre glass network and equipment in a central control station. The fibre glass network used is owned by the Dutch Department of Defense. That is the only network that could guarantee the operation of the system during environmental disasters, war and terrorist attacks. A gateway is present to connect the C2000 system to the telephone network. Eventually 70.000 mobile stations will be connected with the network.

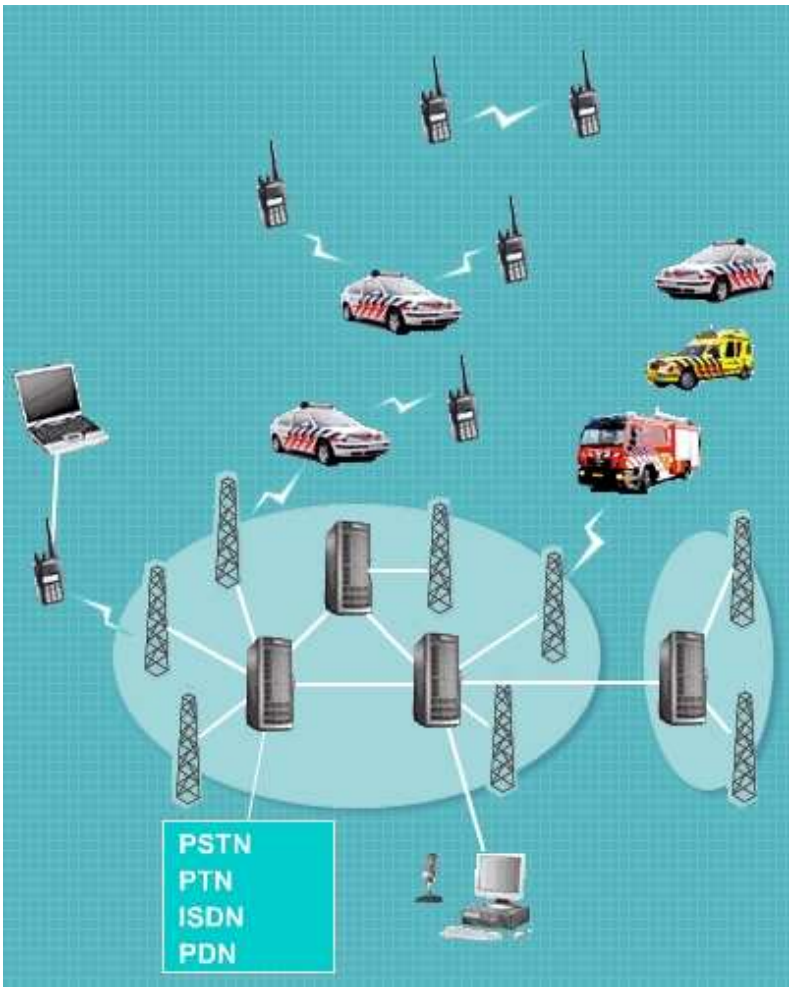


Figure 11.1: The C2000 network infrastructure

11.1.3 TETRA

C2000 is based on the European TETRA standard. As GSM is the standard for mobile telephony, TETRA is the standard for mobile communication of the public safety services. TETRA is the only official open European standard for these services. An open standard is developed in cooperation with users and industry by an official institute, in this case The European Telecommunications Standards Institute (ETSI). Because TETRA is an open standard, all manufacturers can produce equipment for a TETRA network. This is positive for the technical development of the equipment and, because of competition, costs will be lower. Because TETRA is a European standard, cross-country communication between public safety services is possible. The TETRA standard describes the use of one of the TEA1, TEA2, TEA3 and TEA4 sets of encryption algorithms developed by the Security Algorithms Group of Experts (SAGE) and are not public property. In the C2000 system TEA2 is used for the Air Interface encryption.

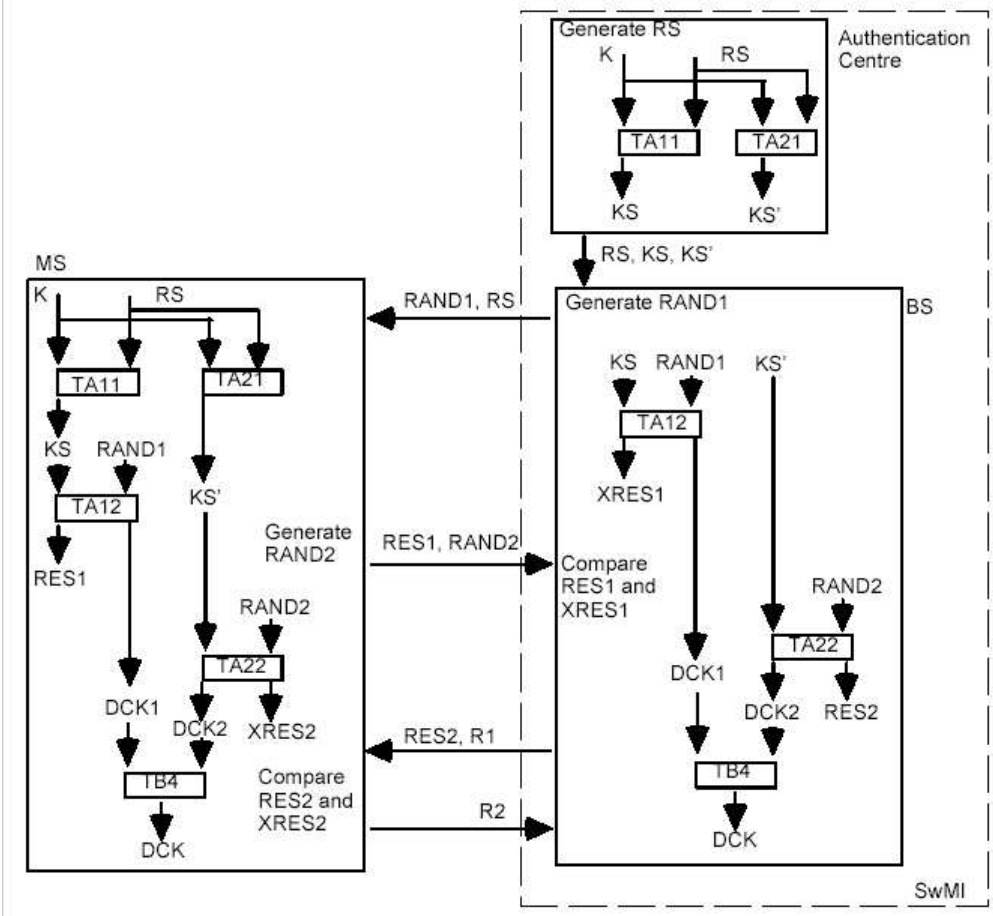


Figure 11.2: The TETRA authentication procedure

11.2 Air Interface Encryption

The air interface is the wireless connection between the Mobile Station (MS) and the Base Station (BS). This section describes the authentication and encryption of this interface.

11.2.1 Authentication

Authentication is neccessary to prevent illicit use of the system. The authentication is mutual, which means that the Mobile Station (MS) will prove to the network that it has knowledge of a shared secret key and the network will prove to the MS that it too has knowledge of the shared secret key. Authentication and provision of keys for use at the air interface shall be linked by a common set of algorithms. The algorithm set shall include a means of providing cipher keys over the air interface. The algorithm set used in the C2000 system is TAA-1/Dimitra System developed by Motorola. The secret key has a length of 128 bits. Authentication will be repeated every 24 hours to change the Derived Cipher Key used for encryption.

The authentication procedure is shown in Figure11.2. The Switching and Management Infrastructure (SwMI) shall use the shared secret key K and a random seed RS with algorithms TA11 and TA21 to generate the pair of session keys KS and KS'. It shall then send arandom

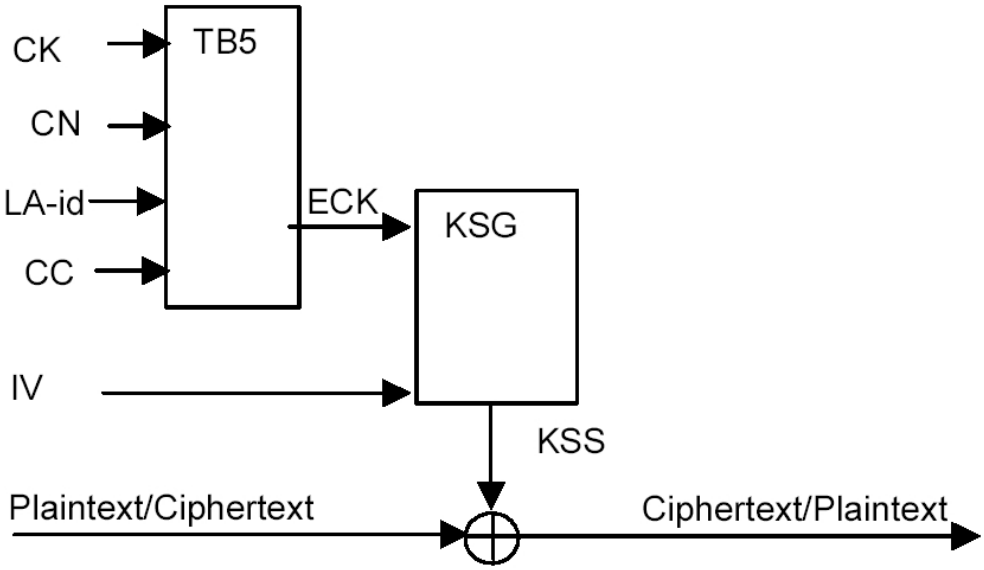


Figure 11.3: The construction and use of the ECK

challenge RAND1 to the MS together with random seed RS. The MS shall run TA11 to generate session key KS, and because the authentication is mutual it shall also run algorithm TA21 to generate a second session key KS'. Both MS and SwMI shall run algorithm TA12; the MS then sends response RES1 back to the SwMI. The MS also sends its mutual challenge RAND2 to the SwMI at the same time. The SwMI shall compare the response from the MS RES1 with its expected response XRES1. The SwMI shall then run TA22 to produce its response to the MS's challenge RES2. RES2 is sent to the MS, which shall also run TA22 to produce expected response XRES2. The MS shall compare RES2 with XRES2 and if the same, mutual authentication will have been achieved.

Algorithms TA12 and TA22 produce DCK1 and DCK2 respectively; these shall be combined in TB4 by both MS and SwMI to produce a DCK which has therefore been created as a result of challenges by both parties.

If the first authentication (of the SwMI) fails, the second authentication shall be abandoned. Because the shared secret key K is never transmitted and the complete process can be generated by one party, this protocol is a Zero Knowledge Protocol.

11.2.2 Keys

Communication is being encrypted with the Encryption Cipher Key (ECK) using a KeyStream-Generator (KSG). The ECK shall be derived using algorithm TB5 from a selected Cipher Key (CK). The CK shall be one of DCK, CCK, MGCK and SCK. TB5 combines CK with Carrier Number (CN), Colour Code (CC) and Location Area (LA) identifier to produce ECK. This is to prevent attacks on the encryption process by replaying cipher text to eliminate the keystream, and to prevent keystream replay within the repeat period of the frame numbering system. The whole procedure is shown in Figure 11.3.

Four types of keys are being used to encrypt data that is being sent over the air interface:

- Derived Cipher Key (DCK)

- Common Cipher Key (CCK)
- Group Cipher Key (GCK)
- Static Cipher Key (SCK)

All keys have a length of 80 bits.

Derived Cipher Key (DCK)

The DCK is derived from its two parts DCK1 and DCK2 in the authentication sequence using the algorithm TB4. The DCK will be used to encrypt data sent from the Mobile Station to the Base Station and from the Base Station to a single Mobile Station. Because every 24 hours re-authentication will be performed, the DCK has a lifetime of max. 24 hours.

Common Cipher Key (CCK)

The CCK is known to the infrastucture and is distributed to the Mobile Stations using Over The Air Rekeying (OTAR). With OTAR, the keys itself are encrypted with the shared secret key and the DCK. The CCK will be used to encrypt data sent from a Base Station to all Mobile Stations in a Location Area (similar to a GSM cell).

Group Cipher Key (GCK)

The GCK is known to the infrastucture and is distributed to the Mobile Stations using OTAR. The GCK shall not be used directly as an encryption key, but shall be modified with the CCK to provide a Modified GCK (MGCK) for use on the air interface. The GCK will be used to encrypt data sent from a Base Station to a (predefined) group of Mobile Stations. A Mobile Station can be a part of more than one group.

Static Cipher Key (SCK)

The SCK is transferred together with the shared secret key used for authentication and will have a long lifetime. The SCK is used to encrypt direct communication between Mobile Stations and to encrypt communication between a Mobile Station and a Base Station when the Authentication Centre is not available.

11.3 End-to-End Encryption

11.3.1 Use

End-to-End encryption is supported by TETRA and is mainly of interest to specialist users, for example drug squads and the military, who have very demanding security requirements. End-to-End encryption is used in combination with Air Interface encryption, because only the voice and data are being encrypted with End-to-End encryption and not the control information. TETRA signalling and user IDs remains protected to AI encryption standard. To ensure the decrypting engine in the receiver maintains synchronism with the encrypting engine in the transmitter, a synchronization vector is sent periodically. TETRA provides a mechanism known as frame stealing to facilitate this.

11.3.2 Algorithms

National security and export considerations often restrict enhanced grade encryption algorithms to a particular country. Therefore ETSI has not included a standard algorithm in the TETRA specification, but they reccommend the use of IDEA. An agreement has been made with Ascom Systec AG (Switzerland) to operate IDEA licences at a reasonable terms. The algorithm can simply be replaced by other algorithms (for example 3DES).

11.3.3 Keys

There are two keys involved in End-to-End encryption:

- Key Encryption Key (KEK)
- Traffic Encryption Key (TEK)

Traffic Encryption Keys (128-bit) are used for the encryption of the data and are being sent over the network encrypted with the Key Encryption Key. The Key Encryption Key cannot be changed over the network and has to be changed manually. As some users may not be in range of the infrastructure when the keys are initially sent, the management proposals include the use of a regime that uses previous, current and future key sets. This improves the interoperability of remote units (e.g. specialist undercover units) who may not want or be able to operate within the infrastructure.

11.4 Security Threats

Threats to communications systems are not just those caused by illicit eavesdropping but fall into three areas:

- Confidentiality
- Availability
- Integrity

11.4.1 Confidentiality

Confidentiality is the ability of the system to keep user data and traffic secret. The Air Interface encryption prevents eavesdropping by hackers, criminals and terrorists and the End-to-End encryption prevents eavesdropping by hostile government agencies and System Administrators collaborating with an attacker.

11.4.2 Availability

Availability is the continuance of the service reliably and without interruption. Denial of Service attacks (preventing the system from working by attempting to use up capacity) are stopped with the ability to disable Mobile Stations over the air. Jamming (Using RF energy to swamp receiver sites) can not stopped immediately, but until the source is located the Mobile Stations are able to connect directly to other Mobile Stations and traffic can be routed thru a Mobile Station to another Base Station.

11.4.3 Integrity

Integrity is the systems strength in ensuring user traffic and data is not altered. Masquerading as a legitimate user may occur often if terminals can be cloned and the subscriber identity copied. To prevent this, the shared secret key may be placed on a smartcard and including a chip on the smartcard to perform computations on the shared secret key. Manipulation of data may occur if an intermediary can capture the message and change it. This is prevented by encryption.

Summary and Conclusions

The C2000 system is an enormous improvement over the old analog radio systems that were used before. However the strong claims on the systems security are not completely true. The shared secret key is only 128 bits long which is quite short. Also, when a shared secret key is known, all other keys can be derived when the set of algorithms is known. The secrecy around the TEA encryption algorithms does not make the system more secure, but less secure. The more experts look at an encryption algoritm, the more secure that algorithm will be (if they can not find any way of breaking the algoritm, of course :-). As seen with the A5/1 encryption algorithm, the TEA algorithms will eventually be known by everybody. Is the C2000 system insecure? No, I don't think so. At least, it's much more secure than the old unencrypted analog radio systems used before.

Chapter 12

The importance of being random

Written by *Meindert Kamphuis*

All encryption depends on the availability of random keys. The best example for this would be the one-time pad encryption method [Tel02]: Every bit, from the information that needs to be encrypted, has to be XOR-ed with a corresponding bit from the key. This means that the length of the key is as long as the length of the information. One-time pad is a strong encryption as long as the key is truly random. If patterns were to arise, while investigating part of the key, an attack on the rest of the encrypted information would be less difficult. It is of the utmost importance that no pattern can be found within keys and thus be truly random, if possible. This section tries to give a deeper understanding of randomness and which types of generators/verifiers are interesting for this getting this understanding. For in most cases true random numbers are not at hand.

12.1 Defining random

In an essay so concerned with random a whole section is used for defining the concept. It will show that different definitions are necessary to completely understand this culprit. To get a better understanding of a basic fault when thinking of random, here a common definition of random will be discussed first:

12.1.1 Incoherent

When one would ask a program to return eight random bits and it would output a byte containing only zeros, most people would at least be a little suspicious. It is never wrong to be suspicious, but why would any other string be more random? No strings shown in casus 12.1 would qualify being random, using incoherent as the definition.

Of course, eight fewer strings will not make that much a difference as there are 248 ($= 2^8 - 8 = 256 - 8$) strings left to choose from.

Truly satisfying this surely is not and why would a seemingly random string `01001011` be incoherent. Every random string can have a description that shows the coherence of the bits. For this particular string, the coherence could be: the binary depiction of the number 75, which shows that the last statement stands.

The better definition for random would be:

Casus 12.1: Coherent strings

string	coherence
1. `00000000`	$x_n = 0$
2. `11111111`	inverse of 1.
3. `01010101`	if x_n is ODD then 0 else 1
4. `10101010`	inverse of 3.
5. `00001111`	if $n < length(x) \div 2$ then 0 else 1
6. `11110000`	inverse of 5.
7. `11001100`	...
8. `00110011`	inverse of 7.

12.1.2 Unpredictable

Without knowing the coherence, which would be the case with randomness, no such problem would arise as with the former definition. Even if one were to know the first seven bits for the strings in table 1, the last bit could still be either zero or one. The unpredictability of every bit makes all 256 strings possible and the chance for any of the strings to be generated the same (1/256).

As clear and as simple as this definition might be, in general it will only suffice as just being a definition. For how can one label a string to be random if all strings are possible? There is no clear way to check whether a string was generated randomly or not. The chance of a bit to be zero or one should be the same. For when either of the two would have gotten a higher change of being generated, the unpredictability of the string would vanish.

Making a more usable definition, the following definition makes use of a feature of randomly generated information:

12.1.3 Uncompressible

Uncompressible means that there is no smaller way to represent the original in the same language as the original. The perfect compression program should be able to All the strings in table 1 are most probably uncompressible, as the binary description of the easiest string to compress ('eight times' zero or '$8 \cdot 0$') would probably exceed the eight bits. Taking a string long enough would erase this problem, see casus 12.2.

Casus 12.2: Compressing 64 zeros

1. Take the string $S = 0_0 0_1 0_2 \cdots 0_{63}$
2. The minimal binary description of 64 is `1000000`
3. Let '$\cdot$' be depicted by `1111111`
4. The compression would be: `1000000 1111111 0` (which is smaller than 64 bits)

(This is only a simple example: Real compression algorithms are far more complex)

Thus by this definition eight times zero is random where sixty-four times zero is not. This may be desired: the chance for eight zeros to be generated is far greater in comparison with a sixty-four zero string.

The set of unpredictable strings is formed by all strings possible, for every string has a probability of being made. This shows that both the set of incoherent and uncompressible strings are subsets of the set of unpredictable strings. The set of incoherent strings is a subset of the set of uncompressible strings as the definition of incoherence can be extremely tight, making it even possibly the empty set.

12.2 Random generators

Random generators come in two distinct different flavors. One the one hand there is the source of entropy to generate the randomness where on the other hand discrete algorithms try to do the same.

12.2.1 Source of entropy

Entropy is a concept in thermodynamics, but here we are more interested in how entropy is a concept in information theory. Information comes in streams, like in books: There is information already read and information soon to be read. Entropy is about how much the given information depicts the upcoming information.

As one is reading a normal English book, every character read gives information about the upcoming character. For instance: When one reads an 'y' the probability of the next letter being also an 'y' is quite minimal. An upcoming character is of course not affected by only the last read character, for instance: After reading a complete sentence a '.' has a high probability.

Now what makes a 'source of entropy'? A source of entropy is a source of information where the book example does not hold. Any given information does not change the probability of any upcoming information. The best example of a source of entropy is a radioactive source. Radioactive decay is totally unpredictable and sampling it gives pure random information. Not everybody has its own radioactive source sampler connected to his or her computer which makes it even better that there are online services which send streams of bits generated this way (encrypted if necessary). Another way to get a good source of entropy is to sample 'noise'. Noise as in when apparatus like your television or radio are not tuned properly. The 'background' information send through the unused frequencies is supposed to be totally random. There arises only one problem with this kind of sampling: These frequencies can easily be altered by anybody who knows the frequency and has any type of broadcaster. Still there are ways to check for this type of alterations and sampling noise can be done by anybody with a computer with a microphone, radio or web-cam connected.

One last source of entropy that should be explained is 'the user'. When working with computers, the user does many things that are supposedly totally unpredictable: Like the movement of his or her mouse. A user is very unpredictable and as there are a lot of users one should always try to use the resources at hand as best as possible. Nevertheless, carelessly sampling for instance this mouse movement does not make a good random generator for mouse movement is buffered and there are certain patters might arise. One strong pattern for Windows OS users is for a user to move its mouse to the bottom left corner of the screen, where the 'start' button is commonly situated. This does not mean that using the user as a source of entropy is a bad idea, for when the implementations is correct real random numbers can be generated this way.

Casus 12.3: Explanation of seed and offset

Offset	0	1	2	3	4	5	6	7	...
Seed 1	3	7	2	0	1	0	9	2	...
Seed 2	6	4	4	0	2	0	8	4	...
Seed ...				...					

There should not be a direct correlation between the different seeds. Here for instance the different seeds are generated by taking the numbers of Seed 1 and multiplying them with the current seed number, modulo 10.

When all obvious patterns are avoided, the sampling generates pure randomness. Or is this world more discrete than most would think and is every source of entropy just another pseudo random generator.

12.2.2 Pseudo random generator

As said in the beginning of this section, there are discrete algorithms, which try to generate random numbers. They try, for discrete algorithms can never create real randomness, as they are discreet and on some level, how deeply buried this may be, there is a pattern to be found. Numerous algorithms have been created to generate numbers as random as possible. The basis is this: The user (computer or person) gives a seed and a starting point (offset) and the algorithm generates as many numbers as needed from these two inputs (see 12.3).

Pseudo random generators can be strong or weak. What makes a pseudo random generator strong or weak is the degree of difficulty to recreate the starting input(seed, offset) from a given string of numbers generated by this generator. When it takes more than polynomial time to do this, the generator is truly strong else; it will be classified as being weak.

12.3 Mixing and skewness correction

In this section to random generator post-processing methods are briefly discussed. Both methods are widely used which make them necessary in any essay concerning randomness, but both methods also arent that fundamental for the question of this section that they will be discussed in real depth.

12.3.1 Mixing

By mixing different generators, it is possible to create a stronger one. Not all generators can be mixed together to make a stronger generator. If two or more generators are correlated is some way, the mixed string might even be weaker as the mixing can by accident enhance the correlation.

A simple way to mix different outputs is to XOR the different strings of data. When using this method at least 50% of the original strings is lost.

Casus 12.4: Output ent.exe for a jpeg file

```
Entropy = 7.980627 bits per character.
Optimum compression would reduce the size
of this 51768 character file by 0 percent.
-
Chi square distribution for 51768 samples is 1542.26, and randomly
would exceed this value 0.01 percent of the times.
-
Arithmetic mean value of data bytes is 125.93 (127.5 = random).
Monte Carlo value for Pi is 3.169834647 (error 0.90 percent).
Serial correlation coefficient is 0.004249
 (totally uncorrelated = 0.0).
```

12.3.2 Skewness correction

The output of real random generators should, if the string s long enough, have a normal distribution of zeros and ones in their binary representation. Any generator can have a small bias for either of the two. There are several skewness correction algorithms, which try to overcome this problem. One of the simplest algorithm break the 'to be corrected' string into parts of two bits. Every pair of bits that have two zeros or two ones in them are discarded. For the rest of the pairs, only the first bit is used.

12.4 Benchmark: ent.exe

To benchmark the randomness of a string, there are certain algorithms developed. The program 'ent.exe' performs several of these algorithms and outputs the different values. It is widely used as a randomness benchmark and the open source license made this transparent program interesting enough to discuss in this section as a general benchmark for randomness.

When a jpeg file is used as input, ent.exe outputs the values shown in casus 12.4.

12.4.1 Entropy

As described before, entropy tells you in what degree information depends on other information within a string.

Another way to look at this is to say that entropy is information density. In an information-dense string, every bit has a meaning and isnt dependable on other bits in the string. There would be no way to predict one bit, even if all other bits are known.

In Ent the entropy is given in how many bits are meaningful for the byte they together make up. For the jpeg file it is almost eight and this would indicate that it is a highly dense file, making it almost uncompressible.

12.4.2 Chi square distribution

The chi square distrubution is best explained on the Ent-website (http://www.fourmilab.ch/random/).

> The chi-square distribution is calculated for the stream of bytes in the file and expressed as an absolute number and a percentage which indicates how frequently a truly random sequence would exceed the value calculated. We interpret the percentage as the degree to which the sequence tested is suspected of being non-random. If the percentage is greater than 99% or less than 1%, the sequence is almost certainly not random. If the percentage is between 99% and 95% or between 1% and 5%, the sequence is suspect. Percentages between 90% and 95% and 5% and 10% indicate the sequence is "almost suspect".

Note that our JPEG file, while very dense in information, is far from random as revealed by the chi-square test.

12.4.3 Arithmetic mean

The arithmetic mean is calculated by summing all the decimal numbers expressed by every byte and then dividing that number by the total of bytes contained in the file. The decimals are from zero to 255, so the arithmetic mean should be about 127.5 for a quite long random string.

12.4.4 Monte Carlo Value for Pi

This algorithm takes three bytes as a X-coordinate end the next three as a Y-coordinate. The point (X, Y) is then being mapped on a two-dimensional square plane and when this point is within the circle enclosed by this square, it is called a 'hit'. This process repeats by taking the next six bytes as (X, Y) until no bytes are left. Using the correlation between all hits against all non-hits, an approximation of pi can be calculated. The closer the calculation leads to pi the less correlated the stream would be. This algorithm is adaptable for using bigger blocks as X and Y and why not include Z or even more dimensions. As for this two-dimension test, the jpeg file looks quite uncorrelated.

12.4.5 Serial correlation coefficient

This is the simplest of all tests and just shows how much a byte is dependent on the previous byte. On pure random bytes, this would be close to zero and the bytes from the jpeg also seem to be independent of their predecessors.

Summary and Conclusions

In this chapter, it became clear that different kinds of randomness are possible. Some are weak others are strong and then there are the real random numbers. Using the correct definition of the concept 'randomness', some kind of verification is possible to check whether a string was randomly generated or not. In the future real random numbers might be a necessary for more and more information is being encrypted. For every computer to have its own source of entropy might seem a little implausible, but hardware devices are currently being made which could easily be adopted to fit for instance your USB-port. Further investigation into parts of the computer itself as being a source of entropy is also being done. The future for compression, randomness and encryption seems bright which makes it even more interesting to keep watching.

Further Reading

For further information see the following websites:

- http://en.wikipedia.org/
- http://www.faqs.org/
- http://www.fourmilab.ch/
- http://www.7-zip.org/
- http://www.cs.berkeley.edu/~daw/rnd/
- http://ei.cs.vt.edu/~history/VonNeumann.html
- http://www.cse.ohio-state.edu/cgi-bin/rfc/rfc1750.html
- http://www.ciphergoth.org/crypto/unbiasing/
- http://www.random.org/
- http://www.softpanorama.org/Algorithms/random_generators.shtml
- http://www.merrymeet.com/jon/usingrandom.html
- http://www.lavarnd.org/
- http://random.hd.org/
- http://www.practicalotp.com/trng.htm

Chapter 13

Off-line Karma

Written by *Nelleke Steendam*

Do you upload as much as you download? Many people don't do that. To obligate the users of a peer-to-peer system to share as much as they use, we can use micropayment schemes. I will tell short what micropayments are and what they are used for. Therefore I used the sources, [Nie00] , [Shi00] and [JK00]. Then I will tell about the implementation of micropaymentschemes in Karma and Off-line Karma.

13.1 Micropayments

What are micropayments? The most common definition on the internet is: A micropayment is an electronic transaction consisting of the transfer of a very small sum of money. This is a very global definition because there are many system that say they are micropayment scheme, with different definitions of it. Micropayments are used for sale of tangible and non-tangible goods on the internet. The most important thing with micropayments is that the speed and the costs of processing the payment aren't to big. Because the goods that are transferred through the internet are usually very small and cost little.

Micropayments are used in different sectors. They can be used in closed networks, for example in peer-to-peer systems, where they can use there own currency. Micropayment is also used for systems based on dollars or euros, like telephone cards. This kind of micropayements are mainly used in monopoly-markets. Because monopoly-players can force the use of micropayments. In peer-to-peer systems (systems where user share resources with other users) it is in the benefit of the user to use such a system. Because the system can monitor and control the transactions of users to ensure that every user shares at least as much as it download. As well as there are a lot of definitions there also is a wide variety on implementations of micrpayment schemes. Each of these implementations focus on one of the properties of micropayment. There are many desirable properties in a micropayment scheme, I shall mention first the basic ones:
Divisibility; The user should be able to pay every amount of cash that wants, and it should be possible to receive exchange. Duration; We don't want that the valuta is able to expire, as happened for example with the gulden. Portability; The system should be usable at all networks and all operational systems. Offline-capability; The micropayment systems should not force the

user to contact a Trusted Third Party (TTP) during every transaction. This is a very important criterium considering the very hard latency requirements for the micropayment transactions.

This where the most basic properties.Second are extended features:

Transferability; Micropayment cash should be transferrable without any registration. Fault tolerance; Failure in software or hardware shouldn't influence a transaction between to parties.

The following properties are for security reasons:

Authenticity; For all parties it should be possible to confirm the origin of a message. Integrity; Integrity of payment means that it isn't possible to fraud the payment data. Interruption tolerance; The transaction should be completed for both parties or not at all. Non-repudiation; That it isn't possible to deny a transaction history for all parties. Third there are are two main desirable properties just for the system:
Scalability; When the market grows, the system should be able to handle the growth. Universality; The system should not be restricted in some way by a financial contract for example.

There are many different micropayment implementations, like CyberCoin, DigiCash, NetBill, PPay. They all have a big disadvantage, that the system needs a central bank or broker. Such a central point can become a bottleneck as the system grows. The only system originated before off-line Karma that doesn't need a central bank or broker is Karma.

13.2 Karma

Karma is a system for a peer-to-peer network, desgined by Vivek Vishunmurthy, Sangeeth Chandrakumar and Emin Gün Sirer ([VVS01]). Peer-to-peer systems are designed to share resources through the internet. Most systems assume that all peers will contribute resources to system. In real life this isn't the case, we're dealing with freeloaders. Studies have shown that in the old Napster network 20 to 40% and in the Gnutella network 70% from the users are freeloaders. Freeloaders are people, who consume many more resources than they contribute. Karma is a framework for avoiding freeloaders in a peer-to-peer system. The main idea behind Karma is, that it keeps track of what the participants contribute and consume. They give each participant a scalar value, their karma, which tells the system, what the participant has contributed and consumed. So if the participant consumes something, then it will decrease, and when he contributes something to the system his karma value will increase. Usually a user receives a amount of karma when he joins a system. A group of nodes, called the bankset, keep track of the karma belonging to the users. A transaction can't not be completed when the karma of the consumer is to low. This forces the participant to upgrade their karma by sharing more resources. Karma is called a economic framework, because it works by keeping track of the resource purchasing capability of each peer. The system Karma possesses the properties of non-repudiation, certification and atomicity. That means that Karma protects a peer-to-peer system against malicious provider who promises to a resource but don't deliver it completely. Also against malicious consumers that receive a resource and later on claim that they didn't received it. And finally against transient states of the system where participants can observe intermediate states in process of transferring karma from one account to another. Karma uses a atomic transaction scheme that provides the consumer with the key to decrypt the resource at the same time that provider receives the certificate of the receipt. Beside that, karma also limits the effect of inflation and deflation, by making karma corrections periodically. With these corrections the total amount of karma available in the system is controlled. The design of the system Karma is based on achieving three fundamental properties. First, the system is designed for peer-to-peer systems and therefore karma needs to be completely distributed and requires no centralised functionality or trust. Second, because in

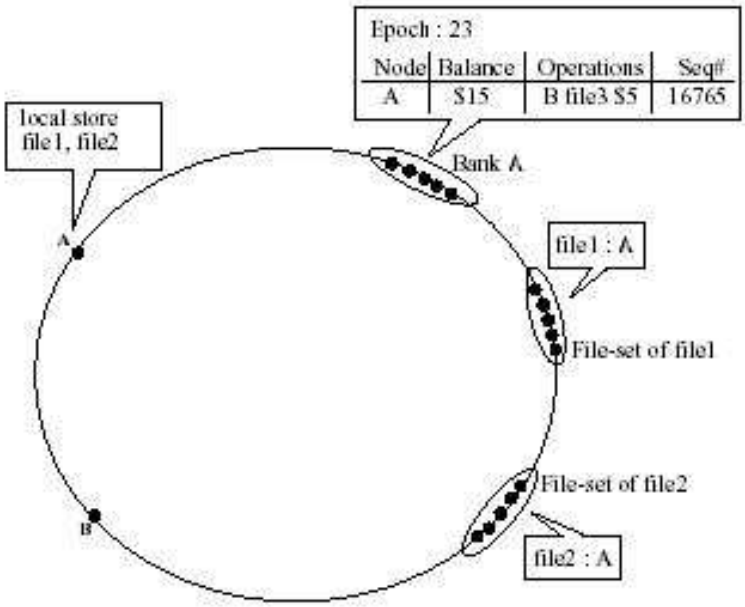


Fig. 1: Overview of system state. A has a balance of \$15 karma, and has recently paid B \$5 for file $file3$.

a loosely organised network of peers there are no failure-proof components, account data needs to be replicated, possibly extensively, to insure against loss and tampering. Third, because the system has to perform well, the coordination among the replicas has to be minimal.

Earlier, I said quickly that the bank-set is a group of nodes. I meant that bank_A keeps track of the karma of node A. Bank_A itself exist of several nodes, other participants, which are chosen with a hash function. Participant A has a nodeId(A), and de nodes closest to the Hash(nodeId(A)) form the bank-set of A. Members of bank_A are equally responsiblefor keeping track of the required information of A. The members of bank_A store the amount of karma that A possesses, the recent transaction with other nodes, (this can be used as proof), sequence number (against reply attacks) and the current epoch number. Epoch is a fixed period of time, usually several months.

The files in the system are sorted in fileId's, that exist out of nodes that possesses the similar file. DHT stores the nodes under fileId, which is the MD5 hash function of the file name. In the file-set is the actual file information replicated at a set of nodes. The file-set consist out of the k/2 nodes closest to the fileId in both direction. (This a possible definition) When node A joins the network, it sends a message with all the files it contributes. The file-set who receive a message add the node to the file-set. After a fixed time A will be dropped from the file-set. So every participant has to update his file communicating frequently. Arrival and departure of nodes can easily be handled, because of the frequently refreshment of the file-set. Epochs are important for the system to handle inflation and deflation of per-capita karma (the total karma divided by the number of active users). We speak of inflation when we nodes leave the system when they decrease their karma and of deflation when nodes leave when they have increase their karma. You don't want inflation and deflation, because you want that the framework keeps control of the total amount of Karma available in the framework. If there is no control of this total amount cheating is far more easily done by malicious people. To handle this the outstanding karma will

be re-valued after each epoch with a correction factor π.

$$\pi = \frac{karma_{old}}{karma_{new}} \times \frac{N_{new}}{N_{old}}$$

Where N stands for the total active users and karma stands for the total karma of the active users. Old stand for the value at the beginning of the epoch and new for the value at the end of the epoch. At the end of each epoch each node of the bank-set has to communicate the number of nodes that went inactive in this epoch and their karma value and the number of new nodes that joined the system. Each node receives this information from all other nodes, and uses it to compute the new value of karma and N and stores them. Then the node can compute the correction factor with stored old values, and revalue the karma of the node by his bank-set at the end of the epoch. The node also needs to add 1 to the epoch number. It's possible that nodes are in different epochs at the same time because each node defines his own epoch. If you go offline for a while (a few weeks) then other users go on into new epochs and you stay in your own epoch. This is no problem because the epoch is used for each peer individually and not for a group of peers together. When a node joins the system he has to get a bank-set of his own and possibly he has to become a member of a bank-set. The new node (N) has to select a random public key K_{public}, a private key $K_{private}$ and a value x such that MD5(K_{public}) equals MD5(x) in the lower n digits. The nodes of the potential bank-set of N, first check if N is really new to system. All nodes send a message with information of N's account if they found N in the system, if not they each send a message that N is a new member. These messages will be signed with the private keys, to prevent forgery. The majority decides than whether N is new or not. If N is new they all initialize the account of N, otherwise they use the balance of the majority.
The new node has to become a member of all bank-sets of other users M for which his identity is close to Hash(nodeId(M)). Therefore it has to communicate with k-1 nodes around him, in both direction. The nodes will give information on the relevant banks for node N. Node N will take for each node, of which he is a bank member, the majority of the balance information. If the information, that node N gets from the other nodes, don't gives a majority he will wait for a while. This can help because the balance information will change after the ending of an epoch.
The karma-file exchange consists of three steps. When node A wants to download from node B, it has to send a message to his bank, then $bank_A$ will send a message to $bank_B$, and $bank_B$ will send the message to node B. In the first step A request $bank_A$ to transfer a certain amount to $bank_B$. In the second step $bank_A$ will decrease the balance with that amount and ask $bank_B$ to increase with the same amount. In the last step $bank_B$ will increase the karma of B and inform ?? about it. Then can node B begin the transfer of the requested file. By file exchange is security also important and therefore is authentication important. In the first step did means that A has to sign his request and submit a sequence number, against replay attack. All bank nodes will store this transfer information, what can later be used as proof. In the second fase $bank_A$ request $bank_B$ to increase the karma of node B with a certain amount. After receiving the request $bank_B$ asks $bank_A$ to confirm the request. The bank nodes of B will challenge the bank nodes of A, to make sure that they are members of bank B. $Bank_B$ will accept the request if $k/2$ valid bank nodes of A confirm the request. In the last fase node B will challenge his bank in the same way as mentioned as above. If the valid majority confirm that he received the karma, then he will transfer the file to node A.

After the karma transfer is done the members of both banks will synchronise themselves. If this wouldn't be done, there would be more disagreements and after a while it wouldn't be possible to get a majority about the karma value.
We want to be able to place multiple requests. To make sure that malicious nodes can't make use of the complete synchronization, there are two steps added to the first fase. First bank $_A$

checks if node A possesses enough karma for the transfer. Second, is to make sure that every member of bank_A receives the request of A. This is solved by letting every member, who receive a request, forward it to a random other member of bank_A. If this is not done, then node A could just send exactly a majority the request. (Example, send one majority a request for file 1 and the other majority a request for file 2, with minimum overlay, then after these requests are done, the majority will give a higher karma then they should give.) This protocol depends on the fact that it expects nodes to participate in banks and to work on the behave of other nodes.

There are several attacks that can be used to the system.

Replay attack: These are ruled out as mentioned before by sequence number and signature's.

Malicious Provider: When the provider accept a payment but doesn't deliver the file, the money can be transferred back. Because of transfer information the bank keep track of.

Malicious consumer: When a consumer claims it didn't receive the file, while he did. The provider simply can prove that he delivered with the certificate that his bank-set gave when the transfer was completed.

Corrupt Bank-set: To prevent that a malicious node can control a bank we need secure entry algorithms.

Denail of Service Attack: When a malicious node tries to sabotage a transfer by sending non-confirming messages (NACKs) it will not work. Because beside the conforming message it also asks for an authencation.

13.3 Off-line Karma

As I said before, the big difference between Karma and the other is system is that Karma doesn't use a central bank or broker. The disadvantage of such central post is that is load grows linearly with the number of transactions, and therefore can be seen as a bottleneck. Beside that is that all the responsibility lies by one central point.

Jaap-Henk Hoekman and Flavio D. Garcia therefore believe that there shouldn't be a central broker or bank. And that all nodes should run the same client and there should be no difference between the relevance and responsibilities that each node possesses.

Karma possess the first property but not the second. If node is not very active and he is an a bank-set of a very active node, he has much higher load, than the nodes in his own bank-set. Jaap-Henk Hoekman and Flavio D developed a decentralised, off-line karma implementation ([GH04b] and [GH04a])for P2P and grid systems, where both properties are fullfilled. The system keeps track of the spending pattern of coins, and the responsibility of the central bank is distributed among nodes, that are random chosen. Transactions between nodes in this system don't need the interfering of a central bank anymore. To detect fraud the coins need to be reminted occasionally. The system allows people to go off- and on-line at all times as they pleases.

In the ideal case we would like to prevent double spending, only in the general case that isn't possible. Therefore want to detect it, and for a usable karma system for P2P and grid applications we demand the following properties:

Scalability; Transaction cost should be independent of the size of the network. No centralised control; The system should not depend on one or several central bank, broker or special nodes. There shouldn't be a predetermined hierarchy in the system. Load of Balance; The load of nodes should dependent on the number of transactions a node is involved with. Availability; Transactions between users will not be interrupted, when users are going off- and on-line. Even when a

group of users loose connection the transactions will not be interrupted. Double-spending detection; The system should detect double spended coins, and correct the fraudulent node backwards.

13.3.1 Description of the model

Off-line karma has three tasks minting the coins, transfer of karma coins and detection of double-spending.
There isn't a central bank, so the users have to mint the coins. They can do that by finding a collisions on the hash function. Finding of a collision will cost much time, and therefore it is easier to share resources. The minted coin gets the name of the minting user a sequence number, this makes each coin unique. The coin gets beside this a time stamp, which gives the time that coin was minted. Every user is allowed to mint 2^q karma coins.
Find a collision y such that $h_1(x) = h_2(y)$ and $x \neq y$, where:
$x = u||sn||ts$
sn =serial number: $|sn| \leq q$
ts =timestamp
New karma coin: $k_0 = (x, y)$
Transfer of a karma coin takes place when user pays with karma for resources of an other user. When an user sends a coin, he has to sign it with his signature. The receiver will verifies his signature and stores the coin. At each transfer an extra signature is add to the coin. The signature's of a coin, can be seen as the history of a coin. This is used by detecting double spended coins. When to coins have the same identity, they look at the history to find the corrupt user. He can be found by tracing where the history of users has been split. To prevent that the corrupt node spends the same coin twice by the same user, we need an extra step in the protocol. The receiver m of the coin has to send a random challenge z to the consumer s who wants to spend coin k_i. Then the consumer s has to send $k_{i+1} = \{k_i, z, m, C_s\}_s$ to the receiver m. Where C_s is the certificate of user s. When a corrupt node spends the same coin twice at the same user, the coin changes because of the challenge z.
A coin has to be reminted after a while for detecting double spending and clearing the history of the coin. If the history is not clear after some time the size of the coin becomes very large. The time stamp on a coin makes sure that it isn't possible to avoid the reminting of the coin, because de time of the timestamp plus a fixed period of time gives the expire date of a coin. That every coin will be reminted, means that every double spended coin will be detected. The reminting is done by set of users, we call reminters for the coin. The reminters are chosen, such that at least one is not a corrupted node and honest reminters possess the history of the previously reminted coins with the same identity. With saved identities the reminters can check if a coin is al ready be reminted and trace back the cheater. The reminters are chosen on the identity of the coin by using a hash function. The remint set of coin k is $R_k = \aleph_r(h(id(k)))$. This ensures us, that nobody can predict or control the reminters. All users in the reminter set have to verify the authenticity of the other user in the set. If the verification is succeeded, they can place their multisignature at the coin, $k_{new} = \{XY(k), ts, R_k, C_{R_k}, u\}_{R_k}$
We said that it was possible to leave and join the system at any time. It can be possible that a node leaves, that is in the remint set. Therefore the remint set will change in time, by the fact that users go off and online. The remint set will exist out the r users which indentity is closest to $h(id(k))$. This makes fraud possible, because you can construct your own remint set. And say later that all other users where off-line at that moment. To prevent this we need the density of the set R;

$$d(R_k) = max_{I \in R_k} |i - h(id(k))|$$

The density shows how much the remint set is spread, so what the maximum deviation of user

is to $h(id(k))$ of the coin k. It is not likely the amount of user will heavily fluctuate in a short period of time. When we limited the change of the density of the remint set, it is not possible anymore to choose completely your own remint set. Let α be the maximum rate for change for the density of the overlay network, d(t) the density at time t, and if T is the maximum time before reminting, then for t' between t and $t+T$ we demand that $\frac{1}{\alpha} \leq d(t') \leq \alpha d(t)$.
To prevent loss, nodes will update their database when ever a node leaves or joins the system, so there won't be loss of history. Another advantage of the timestamp is, that after a certain time, it isn't possible any more to have duplicates in the system of the coins. I mean that the timestamp makes sure that every coin will be reminted after a while. To keep the database in bounds, the history of the coins of which the expire date has past, will be deleted.

13.4 Protocol

Minting:
For a user u:
Initially: $K_u := \emptyset; sn_u := 0$
$sn_u := sn_u + 1$
$ts := now()$
$x := u||sn_u||ts$
Find y statisfying: $h_1(x) = h_2(y)$
$k := (x, y)$
$K_u := K_u \cup \{k\}$

Spending:
User u spends a coin at user m:
$m : picknoncez$
$m \rightarrow u : z$
$u : selectk \in K_u$
$u \rightarrow m : \{k, z, m, C_u\} \rightarrow k'$
$m : check(m, k', z)$
$u : K - u := K_u \ \{k\}$
$m : K_m := K_m \cup \{k\}$
Reminting:
User u remints a coin $k = \{\tilde{k}, z, u, C_s\}_s$:
$u : R = \aleph_r(h(id(k)))$
$u \rightarrow r_i : k, now(), R \rightarrow k', t', R' \quad \forall_i : r_i \in R$
$r_i : t' \approx now()$
. $R' = \aleph_r(h(id(k)))$
. $check(u, k', \perp)$
. $verifyHistory(k')$
. $k_{new} = \{XY(k'), t', R', C'_R, u\}$
$R \leftrightarrow u : \{k_{new}\}_R \rightarrow k_R$ (this is a three-round protocol)
u:checkBase(u, k_R)

With:
$Check(u, k, z)$:
If is Base(k) then checkBase(u, k)

else $\{k', z', u', C_s\}_s := k$
. return$(z' = z \vee z = \perp)$
. $\wedge\, u' = u$
. $\wedge\, validSig(k, s, C_s)$
. $\wedge\, check(s, k', \perp)$
checkbase(u, k):
if is reminted(k) then
. $\{k', newts, R', C_R, u'\}_R := k$
. return$u' = u$
. $\wedge\, R' = R = \aleph_r(h(id(k')))$
. $\wedge\, newts \in [now() - T, now()]$
. $\wedge$validSig(k, R, C_R)
else
. $(x, y) := k$
. $u'||sn||ts := x$
. return$h_1(x) = h_2(y)$
. $\wedge\, u' = u$
. $\wedge\, ts \in [now() - T, now()]$

audit(k, k'):
$\{\{k_o, z_1, u_1, C_0\}_{u_0}, z_m, u_m, C_{m-1}\}u_{m-1} := k$
$\{\{k'_o, z'_1, u'_1, C'_0\}_{u'_0}, z'_{m'}, u'_{m'}, C'_{m'-1}\}u_{m'-1} := k'$
for$(i = 1$ to $min(m, m'))$ do
. if$(z_i \neq z'_i \vee u_i \neq u'_i)$then return u_{i-1}

verifyHistory(k):
Hcoin $:= \{k' \in H | id(k) = id(k') \wedge ts(k) = ts(k')\}$
For each $k' \in$ Hcoin do
. $B := B \cup \{audit(k, k')\}$
$H : H \cup \{k\}$
return Hcoin $= \emptyset$

The only difference in the protocol for the off-line version is the checkbase: checkbase(u, k):
if is reminted(k) then

. return$u' = u$

. $\wedge\, R' = R$

. $\wedge\, newts \in [now() - T, now()]$
. $\wedge\, d(R) \leq \alpha d(now())$
. $\wedge\, validSig(k, R, C_R)$
else
. $(x, y) := k$
. $u'||sn||ts := x$
. return$h_1(x) = h_2(y)$
. $\wedge\, u' = u$
. $\wedge \in [now() - T, now()]$

Summary and Conclusions

Micropayments are scheme that are used for several applications. That the micropayments that are actually used, depends often on if it's forced by a monopoly player. But in the case of peer-to-peer system, the micropayment is of use for the user, and therefore has a great chance for succeeding. Karma was the first implementation for peer-to-peer systems without a central bank or broker. Only the load of the user of Karma is not equally divided. After Karma, Off-line Karma is developped, that doesn't need a central bank or broker and where for the load of the users depends on the amount of transcations they are involved in. Off-line Karma is a very new system, it is developed in september 2004. Therefore the are still some questions like, how efficient is the system, and when should we remint the karma coins.

Chapter 14

High-Bandwith Digital Content Protection

Written by *Jeffrey van Helden*

High-bandwidth Digital Content Protection (HDCP) is a cryptographic system developed by Intel that encrypts video on the DVI bus. The aim of HDCP is to prevent illegal copying of video contents by encrypting the signal. It works by adding circuitry within the DVI connection on both transmitter (DVD player, cable box, etc.) and the receiver (projector, LCD, etc.) that encrypts video content.

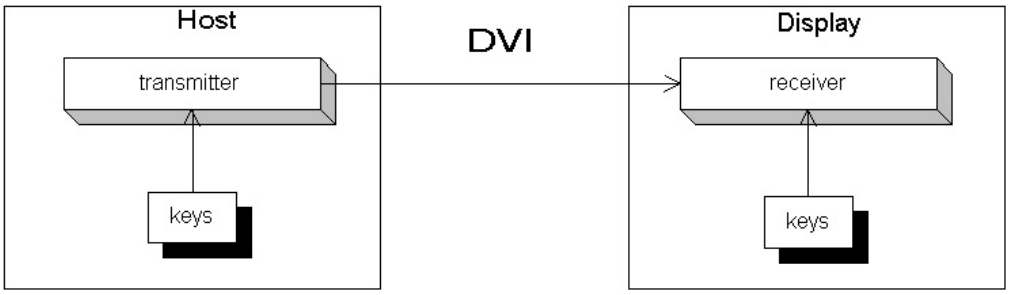


Figure 14.1: DVI bus

Digital Visual Interface (DVI) is a type of cable and connection created in 1999 by the Digital Display Working Group. It delivers exceptionally high quality video; a quality nearly as good as the original content. This previously unattainable quality has raised concern from Hollywood executives who fear video could be mass produced and illegally distributed, as what has happened within the music industry. In an effort to protect this video content, a method of security encryption was developed.

A summary of the protocol will be given in 14.2. A drawback of the HDCP authentication protocol is a weakness that can be used to recover the master key. Once you know the master key, you can decrypt any movie, and even create new HDCP devices that will work with the 'official' ones. If this master key is ever published, HDCP will provide no protection any longer. In 14.3 the main weakness will be described, together with a possible way to divert the master key from 40 public keys. The most famous person to break HCDP was the Dutch cryptographer Fergunson. The reason he did not publish his work is described in 14.1.

14.1 Freedom of speech: the Digital Millennium Copyright Act

From a point of professional interest Niels Fergunson analysed the specification of a technique that Intel had developed to prevent illegal copying of digital video. This specification was published on the internet by Intel itself [Int03]. When Fergunson read the paper, he immediately started to doubt the quality of the security offered. After analysing more thoroughly, he discovered a way to compute the secret master key; with which any private key can easily be computed given a public key.

According to Ferguson, HDCP is " *fatally flawed. My results show that an experienced IT person can recover the HDCP master key in about 2 weeks using four computers and 50 HDCP displays. Once you know the master key, you can decrypt any movie, impersonate any HDCP device, and even create new HDCP devices that will work with the 'official' ones* "

The normal procedure for a scientist would be to write an article and publish the result to colleague scientists. With the purpose to share his discovery such that discussion can improve the technique, he wrote the article but unfortunately, he was not allowed to publish it. A U.S. law called the Digital Millennium Copyright Act (DMCA) makes it illegal to distribute "circumvention technology", such as systems that break copyright protection schemes. As HDCP is used to protect copyrights, someone that publishes weaknesses of such a system would risk a penalty until 25 years in jail. According to the DMCA, Fergunson is not even allowed to discuss the article with any colleague, nor with Intel itself, because then he could be sued in the U.S. for spreading a method to bypass copy protection.

14.1.1 American logic

Another victim of DMCA is the Russian programmer Dimitri Skliarov. He was arrested in the US after illustrating a method on a hackers conference to bypass the copyright protection of electronic books (Adobe). He is charged with violating the DMCA (American law) while performing his work in Russia as an employee for a Russian firm. What he did was perfectly legal in Russia, and in most other countries in the world. The law works contra productive, as participation of critical cryptography researchers from all over the world is a necessity in the challenge to develop save digital protection in the future. In the long run, the DMCA will make it much easier to create illegal copies, because when research is forbidden, no one will develop good copyright protection schemes.

14.2 The protocol

Because DVI devices from many different manufacturers need to interoperate and perform key exchange with no user intervention, the HDCP authors choose to use an identity-based cryptosystem. In the HDCP scheme, device manufacturers purchase HDCP licenses from a trusted authority. The protocol is used in both Upstream and Downstream versions of HDCP. The Upstream version of HDCP is designed for the communication link between software running on the personal computer and the HDCP devices attached to that computer. The Downstream protocol is used between HDCP devices. First, an enumeration of all variables follows in 14.2.1, while a summary of the relevant details of the protocol will be further described below; the complete

specification can be found in [Int03]. The explanation of the protocol consists of three main parts, Authentication, Data Encryption and a Cipher.

14.2.1 Terminology

Each HDCP Device contains an array of 40, 56-bit secret device keys which make up its Device Private Keys (DPK), and a corresponding identifier called the Key Selection Vector (KSV). The KSV is a 40-bit binary value, and can be seen as the public key. A KSV contains exactly 20 ones and 20 zeros in a "random" sequence. The DPK's and KSV's of communication devices will hereafter be referred to as u and v, respectively. Furthermore, n_a is a pseudorandom nonce send by the transmitter A, which is used together with a one-way function h, to verify the result of the key agreement process.

14.2.2 Authentication

The communication exchange allows for the receiver to demonstrate knowledge of a secret device key. When devices A and B wish to communicate, in a practical situation this could for example be a DVD player with a flat-panel television, they exchange v_a and v_b. Device A computes the dot product $K = u_a * v_b$ and B computes $K' = u_b * v_b$ and they use this as their shared secret for the rest of their interactions. The way v_a is derived from u_a is unknown; but the trusted authority uses some secret information so that the above K is equal to K'. To verify that the key agreement process has been successful, A sends na, which is replied by B by sending $R' = h(K', n_a)$. When $h(K, n_a) = R'$ holds, it strongly suggest valid keys.

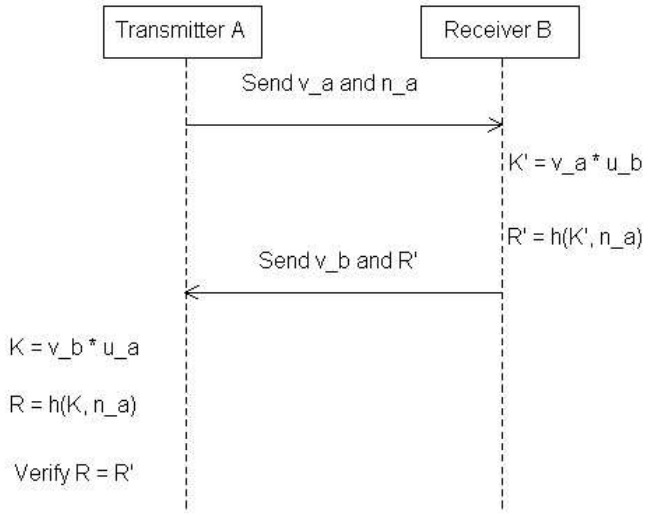


Figure 14.2: Authentication

14.2.3 Data Encryption

Now both HDCP Devices have generated a shared secret value that cannot be determined by eavesdroppers on this exchange. The shared secret can then also be used as a symmetric key to encrypt HDCP content. Without encryption, the data exchange would go as follows: The digital data stream is sent to the Transition Minimized Differential Signalling (TMDS) transmitter,

where it is encoded in a digital DVI-signal. After it is sent over the DVI-cable to the receiver, the TMDS receiver decodes the data stream to the proper digital data to eventually display it on the screen. By adding encryption to the protocol the signal sent over the DVI-cable becomes unrecognizable. HDCP Encryption is applied at the input to the TMDS Encoder and decryption is applied at the output of the TMDS Decoder. HDCP Encryption consists of a bit-wise exclusive-or (XOR) of the HDCP Content with a pseudo-random data stream produced by the HDCP Cipher. As XOR is a reversible function, the receiver can fulfil the same steps to decrypt.

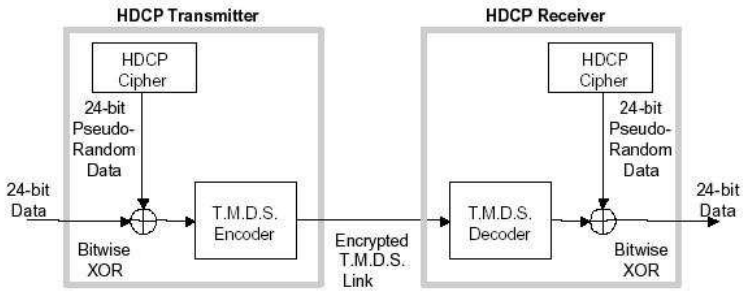


Figure 14.3: Data encryption

14.2.4 HDCP Cipher

The Cipher is a generator producing an independent data stream to be used for stream encryption. The structure of the HDCP Cipher consists of three layers. The block module operates as a block cipher during the authentication protocol. The block module and the output function are used together to produce the 24-bit pseudo random sequence that is used to encrypt the HDCP Content. Each layer is shortly described below.

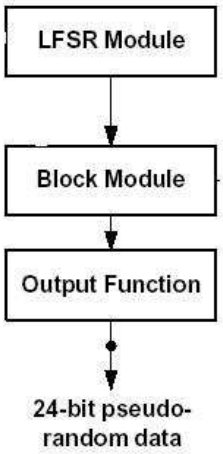


Figure 14.4: HDCP Cipher

1. A set of four Linear Feedback Shift Registers (LFSR's) are combined to one bit. This one bit feeds into the middle layer.

2. The middle layer consists of two separate "round function" components. One of these components, Round Function K, provides a key stream for the other component, Round Function B. Each of these two components operates on a corresponding set of three 28-bit registers. For Round Function K, a bit of the register takes its input from the LFSR module output. Both round functions produce three 28-bit registers.

3. The final layer takes four (out of six) 28-bit register outputs from the round functions through a compression function to produce a 24-bit block of pseudo-random bits for every clock pulse.

More about linear feedback shift registers and stream encryption can be found in [Tel02].

14.3 Weaknesses

The author travels regularly to the U.S.; therefore, he does not want to risk any sort of penalty regarding the DMCA. As a chapter about a relatively weak encryption algorithm will not be complete without touching the subject of a possible attack, only a brief summary is given in this chapter. It needs to be emphasized that the following method is completely based on other published information; comprehensive mathematical details of the attack can be found in [CGJ+01] and [Irw01].

The designers made a well-known cryptographic design mistake, that can easily be exploited by attackers. As the shared secret generation is entirely linear, the attack only needs 40 public/private key pairs such that the public key pairs span M; the module generated by all public keys. After recovering the private keys of 40 devices, one can attack every other interoperable HDCP device. The HDCP specification does not describe the key generation process but, based solely on the properties of generated keys, all possible key generation strategies can be characterized. This implicates that there is no secure implementation possible that follows from these properties.

14.3.1 A possible attack

The authority's secret information can be recovered by an attacker. As the mapping from public keys to private keys is linear, it can be represented by a matrix S. The secret can be captured in a 40*40 matrix, and linear algebra techniques are sufficient to recover it.

Recovering S can be done with a collection of 40 key pairs (v,u) such that the vi span the module spanned by all KSV's. Then the system of 40 equations with 40 unknowns: U = SV can be solved by applying for example the Gaussian elimination technique. One has to take into account that computations in HDCP are done mod 2^{56}, so not all basic facts from linear algebra hold.

Possessing the matrix S, the private key belonging to any KSV can be computed.

Summary and Conclusions

There are many ways to obtain the keys necessary to recover the authority's secret. An attacker can for example break open 40 devices of different HDCP video software utilities and extract the

keys via reverse engineering. Another, more legal approach, is to buy a set of key pairs. Device manufacturers can buy 10000 key pairs for $16000. If there is a set of 40 keys that span the total module spanned by the KSV's, and if anyone with bad intentions possesses such a set; the entire system collapses. Eavesdrop device communication, to device cloning, producing new accepted device keys, etc., everything is possible.

The U.S. (law system) does not do any effort to avoid future catastrophes. By making it impossible to publish articles about weaknesses of cryptographic systems that are meant to protect content, they break a set of main human rights, such as freedom of speech and publishing. Above all, they eventually dig their own grave by leaving holes in their own content protection.

Chapter 15

Advanced Access Content System

Written by *N.J.T. Munting*

This chapter will look into the development of the successor of the DVD format, focusing on the encryption techniques used for various purposes. At the moment two formats are being developed and promoted by different companies. Both systems hope to match or even exceed the success of the DVD format. The new standard also aims to provide better security features, which proved a big flaw in the DVD format.

First the security flaws of the DVD will be discussed in order to gain a better understanding of the demands that should be fullfilled by a successor. Then the specifications for the Advanced Access Content System, as far as known to date, will be discussed. This chapter will conclude with some thoughts on the success of this new content protection system.

15.1 Content Scrambling System

In the early nineties IBM decided it was time for single new standard for optical storage [Wik05b]. This standard should prevent another war between competing formats in vain of the battle between Betamax and VHS. IBM's initiative succeeded and the DVD Consortium was founded. The DVD was announced in 1995 and its first specification was released in September 1996. A number of security features were implemented in the DVD format in order to satisfy the film studios with their noble quest against piracy. These security features include region coding and CSS, short for Content Scrambling System.

CSS is an encryption system designed primarily to protect the DVD disk from being copied digitally [Kes00]. However, CSS has a series of serious design flaws, which enabled the system to be broken without much trouble. Surprisingly, initiatives to break CSS were not spearheaded by pirates trying to make illegal copies for distribution, but rather by users of open source software who lacked an implementation of a DVD player[1] for their operating system. To gain a better understanding of the flaws in CSS, the system will be explained first.

15.1.1 Key management

CSS uses a number of 40-bit keys for different purposes:

[1]The DVD player in this case would be computer software, but it can also be a home video player.

- **Authentication key (k_a).** A secret present in both the drive and the player used to negotiate the session key.
- **Session or bus key (k_b).** This key is used to encode the title and disk keys before sending them over the bus between the drive and the player. An eavesdropper might otherwise obtain these keys by reading from the bus.
- **Player key (k_p).** Player keys are issued by the DVD Copy Control Association (DVD CCA) to manufacturers of hardware or software DVD players. Each player is issued a few of the total of 409 available keys. A player key is required in the process of decrypting the disk key.
- **Disk key (k_d):** The disk key is used to encrypt the title keys.
- **Sector key (k_s):** This key is present in the plaintext header of a sector for optional additional encrypting of the sectors within each title.
- **Title key (k_t):** The title key is XORed with the session key and used to encrypt the data within a sector.

Every DVD player is equipped with a number of player keys. Each DVD disk has encrypted actual content (e.g. a movie) and a hidden area containing $e_d = E_{k_d}(k_d)$, which is an encryption of the disk key using the disk key itself, and $e_{p_1}, \ldots, e_{p_{409}}$, which is a table of encrypted disk keys, where each $e_{p_i} = E_{k_{p_i}}(k_d)$; i.e. an encryption of the disk key with the i-th player key.

The DVD player is able to decrypt the encrypted content on the disk after completion of the following process:

1. The player and drive verify each other's identity using the authentication key and a simple symmetric authentication protocol. In this authentication process a session key is also established.
2. After successful authentication the drive sends the hidden data on the disk to the player. The player uses one of the player keys (k_{p_i}) to decrypt the corresponding entry in the table (e_{p_i}) and obtains the disk key, i.e. $k_d = D_{k_{p_i}}(e_{p_i})$, and verifies this: $k_d = D_{k_d}(e_d)$. Remember that $e_d = E_{k_d}(k_d)$ and thus decrypting e_d using k_d should yield k_d again. In case the chosen player key does not produce the correct result, another one of the several possibilities can be used until the decryption of the disk key is successful. In case the operation does not succeed the player has been revoked by the DVD CCA.
3. Now that the disk key is successfully decrypted the title key can be obtained using the disk key.
4. Each sector that is sent from the drive to the player can be decoded by the player using the title key and the sector key present in the header of each sector.

15.1.2 Encryption and decryption

Obtaining the disk key is a big step towards the decryption of a DVD's content. However, some additional work needs to be done to get to the actual content. The content is encrypted on the disk using a stream cipher. The stream cipher is implemented by Linear Feedback Shift Registers (LFSRs), which produce a pseudo-random bit stream. This bit stream is than XORed with the plaintext content to produce the ciphertext content.

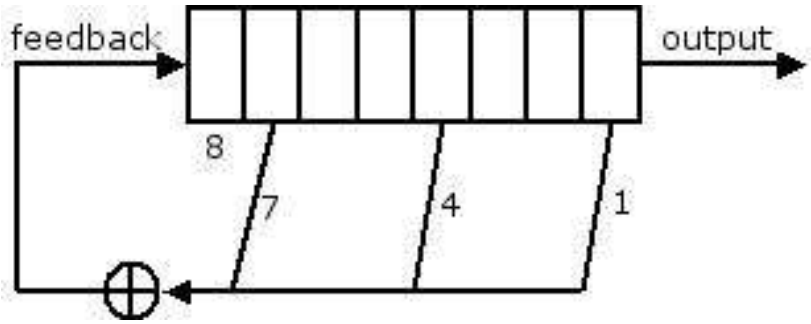


Figure 15.1: A typical Linear Feedback Shift Register

A basic LFSR works as shown in figure 15.1. The LFSR is seeded with a number of bits, in this case eight. At each clock the row of bits is shifted to the right and bit 1 is used as the output value. A new bit is shifted in on the left, which is a XOR of the tap bits. In figure 15.1 the taps are on bit 1, 4 and 7. When designing a LFSR of length n the taps should be chosen to get maximum cycling length $2^n - 1$. Furthermore, a LFSR should never be seeded with all bits set to zero. XORing the zeroes from the taps will always produce a zero and hence the output will only produce zeroes as well.

CSS uses two LFSRs to produce the pseudo-random bit stream; a 17-bit LFSR and a 25-bit LFSR (shown in figure 15.2). The first LFSR is seeded with the first two bytes of a key, the latter is seeded with the other three. Bit number 4 is extra and is always set to one to prevent null-cycling. This is all quite normal, but the LFSRs implemented in CSS have a somewhat peculiar design. The bits shifted out on the right, which are normally considered the output, are

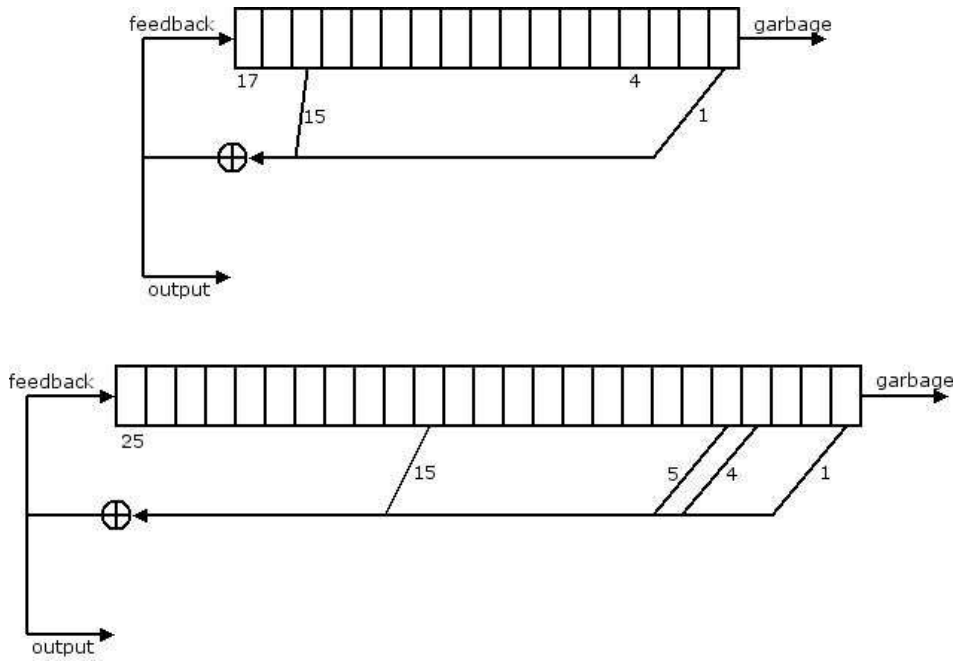
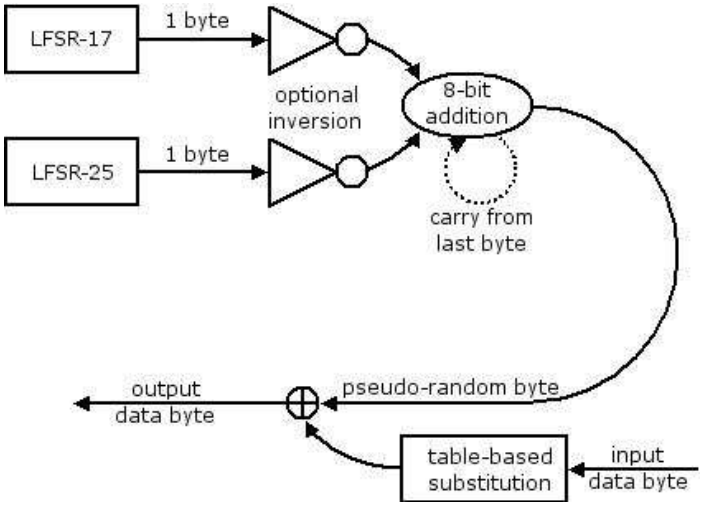


Figure 15.2: The Linear Feedback Shift Register used in CSS

Casus 15.3: Content Scrambling System encryption/decryption



The contents and the keys of a DVD disk are encrypted using CSS. Two LFSRs (see figure 15.2) are used to generate a pseudo-random bit stream. Both LFSRs are clocked at the same rate and the output is combined after eight clocks. The bytes from the LFSRs are optionally inverted, depending on the CSS mode [Kes00]. After this step both bytes are combined, together with a carry saved from the last byte, using simple linear addition. The carry of this addition is saved for the next round. The output of this addition operation is considered the pseudo-random bit stream. The input data byte[a] is first fed through a table-based substitution operation, after which it is XORed with the pseudo-random byte. This yields the output data byte[b]. Encryption/decryption for content would end here, but keys also go through an additional two-step mangling operation not described here. For additional information see [Kes00] and [Ste99].

[a] Plain text in case of encryption and cipher text in case of decryption.
[b] Cipher text in case of encryption and plain text in case of decryption.

treated as garbage. Instead, the output uses the same value as the bit being shifted in on the left. After 8 clocks the 8 bits of both LFSRs are added using 8-bit addition and these form an output byte. This output byte is used for encryption by XORing it with a key or the data. For a more detailed description of this procedure please look at casus 15.3.

Decryption is of course very straighforward, since $(x \oplus a) \oplus a = x$. Thus repeating the encryption procedure on the ciphertext content or key will yield the plaintext content or key.

15.1.3 Weaknesses

- **Key length.** The key length of 40 bits was most likely chosen due to the export restrictions on cryptographic material out of the United States. The DVD Forum probably hoped that keeping the Content Scrambling System secret, would prevent the system from being broken. But even without the design flaws the 40-bit key was too short. DES, which uses a 56-bit key was broken by brute force the same year in which the first DVD players appeared on the market.

- **Security through obscurity.** The DVD Forum has chosen to keep the Content Scrambling System a secret for reasons mentioned above. This did not stop the system from being broken by open source users, trying to make their DVDs play on their operating systems. An anonymous posting of the source code for CSS sped up the process, but there is no doubt that the system would have been broken anyway, see [Fel04]. Even the Digital Millennium Copyright Act could not have prevented this. CSS can be considered another clear example of the fact that security through obscurity simply does not work.

- **LFSR implementation.** The chosen implementation of the LFSRs is not the only defect in the design of CSS, but certainly one of the most important ones. As described, the chosen implementation enabled much easier prediction of the LFSRs' output by guessing the initial state and clocking forward or backward. This feature was exploited for the best way to break CSS completely: obtaining the disk key. Remember that a disk contains a hidden area with $e_d = E_{k_d}(k_d)$. [Ste99] describes a way to calculate the disk key using e_d with a complexity of 2^{25}. This attack has a significant lower complexity than should be expected from a 40-bit key and takes about 20 seconds with a Pentium III, which was a typical system around that time.

In conclusion, CSS was far from being set up in the best way possible and breaking it was bound to happen.

15.2 Advanced Access Content System

Almost a decade after the introduction of the DVD, continued technological advances require a new standard with more storage capacity. The rise of High-Definition Television in the USA and Japan is one of the main reasons for the required extra storage. HDTV gives a sharper picture compared to the older systems, because of a much higher resolution. Of course this also impacts the size of the movies stored in this resolution on an optical storage disk. HD-DVD and Blu-ray promise to give the required storage capacity with maximum single-layer, one-sided disk contents of 15 GB and 25 GB respectively, which is a lot more compared to 5 GB for a single-layer, one-sided DVD disk.

The introduction of a new standard for video playback and recording gives the film studios a new opportunity to require a better content protection system. Advanced Access Content System, which is being developed by IBM, Intel, Microsoft, Panasonic, Sony, Toshiba, The Walt Disney Company and Warner Bros. is the most likely candidate for content protection on both HD-DVD and Blu-ray. The biggest difference compared to CSS is the technical specification, see [Adm04], which can be downloaded by anyone from `http://www.aacsla.com/`.

The specification reveals some important differences with CSS. Instead of designing a potentially faulty encryption system themselves, the AACS Licensing Administrator decided to choose Advanced Encryption Standard to provide most of their encryption needs. AES is a block cipher approved by the United States government. It is being and has been extensively studied by cryptography experts worldwide. No successful attacks have been confirmed and AES is considered secure.

15.2.1 Key management

However, adopting AES as their encryption system certainly does not solve all problems. The key management system used in AACS had to be designed by themselves. Key management is of course always the most important security issue in an encryption system and this will proof not to be an easy task. The reason this is difficult is because the people who need the keys to decrypt the contents will potentially try to compromise the system.

The key management system in AACS will rely primarily on the ability to revoke compromised device keys when necessary as stated in [Adm04]:

> [...] Each compliant device or application is given a set of secret Device Keys when manufactured. The actual number of keys may be different in different media types.

Casus 15.4: Advanced Encryption Standard

AES or Rijndael is a block cipher, which was chosen by the National Institute of Standards and Technology out of several other candidates to be the successor to DES [ST01]. AES is a substitution-permutation network and the algorithm has a neat mathematical foundation. The algorithm works on 128-bit data blocks and it can use key sizes of 128, 192 or 256 bits. The number of rounds in AES depends on the key length and is respectively 10, 12 and 14 rounds. AES uses a 4×4 byte representation of the data block for each step performed in a round.
There are four steps in one round, namely:

1. **Byte substitution.** Each byte in the array is substituted using a lookup table.
2. **Row shifting.** Each row in the array is shifted a defined number of steps.
3. **Column mixing.** The four bytes in one column in the array are combined using a linear transformation. This step is omitted in the final round.
4. **Round key adding.** Each byte in the array is XORed with the corresponding byte of the round key.

For further information see [Wik05a] and [Tel02].

> These Device Keys, referred to as $K_{di}(i = 0, 1, \ldots, n-1)$, are provided by AACS LA, and are used by the device or application to process the MKB to calculate K_m. The set of Device Keys may either be unique per device, or used commonly by multiple devices. The license agreement describes details and requirements associated with these two alternatives. A device shall treat its Device Keys as highly confidential, as defined in the license agreement. [...]
>
> [...] The Media Key Block (MKB) enables sytem renewability. The MKB is generated by AACS LA, and allows all compliant devices, each using their set of secret Device Keys, to calculate the same K_m. If a set of Device Keys is compromised in a way that threatens the integrity of the system, an updated MKB can be released that causes a device with the compromised set of Device Keys to be unable to calculate the correct K_m. In this way, the compromised Device Keys are "revoked" by the new MKB. [...]

CSS was also designed with the ability to revoke the similar player keys, but the DVD Copy Control Association never got the chance to put it to good use. The disk key could be calculated from the information stored in the hidden area, without using a player key, in effect making every player compromised. Utilising AES for key encryption will probably not allow this to happen as easily.

15.2.2 Media Key Block

AACS uses the following 128-bit keys:

- **Device key (k_d).** Each licensed player contains several of these device keys. A device key allows the disk key to be extracted from the Media Key Block by the player.

- **Subsidiary device key (k_s) and processing key (k_p).** Keys used in the process of calculating the media key using the device key.
- **Media key (k_m).** Each disk or medium contains a Media Key Block, which allows players to access the disk key, given the correct player key. A hash of the disk key, combined with the volume identifier and usage rules, is used to decrypt the title key.
- **Title key (k_t).** Each title on the disk is encrypted using the title key, which can be either different or the same for each title on the disk.

The arrangement of a disk protected by AACS is very similar to the layout of a DVD disk. The disk contains the MKB, the encrypted title keys and the encrypted content. The Media Key block contains the following elements:

- **Verify Media Key record.** Contains $c_v = E_{k_d}(v)$, with the first 64 bits of v equal to $0123456789ABCDEF$ (in hexadecimals) padded with a 64-bit random number to a total of 128 bits. The device can verify whether it has obtained the right k_d by calculating $v' = D_{k_d}(c_v)$ and verifying that the first 64 bits of v' are the correct value.
- **Explicit Subset-Difference record.** This part of the MKB contains information on how to perform the bit expansion used to obtain the processing key. This information is stored in a number called a subset-difference.
- **Media Key Data record.** The Media Key Data record contains encrypted media keys for each corresponding subset-difference.

The exact manner in which the media key is calculated is not discussed here, but can be read in the [Adm04]. However, the following describes briefly how the key is obtained. The device key is used in a bit expansion function, which expands the number of bits from 128 to 384 bits. The subset-difference information from the MKB lists whether the bit expansion should repeat with the first or the last 128 bits, or whether the middle 128 bits should be used as the processing key. In the latter case the bit expansion is finished and the media key can be decrypted using the processing key and the media key data corresponding to the subset-difference used in the bit expansion.

15.2.3 Encryption and decryption

The encryption and decryption of the keys and the content is done with a number of AES based functions. Some of these are implementations of NIST approved AES uses described in [Dwo01], others appear to be designed by the AACS.

For encryption and decryption of keys the Electronic Codebook mode of AES is used, which is the "normal" mode of AES described in casus 15.4 and outlined in [Dwo01]. Every block of plaintext is encrypted to ciphertext independent of the other plaintext blocks. Likewise, decryption of a ciphertext block can occur independently of the other ciphertext blocks, if the key is known.

Content is encrypted and decrypted using the Cipher Block Chaining mode of AES, which is also covered in [Dwo01]. Encryption is done as follows: $c_1 = E_k(p_1 \oplus iv)$ and $c_j = E_k(p_j \oplus c_{j-1})$ for $j = 2, \ldots, n$, where c_i is the i-th ciphertext block, p_i is the i-th plaintext block and iv is the initialisation vector used for the first encryption. Decryption is straightforward: $p_1 = D_k(c_1) \oplus iv$ and $p_j = D_k(c_j) \oplus c_{j-1}$.

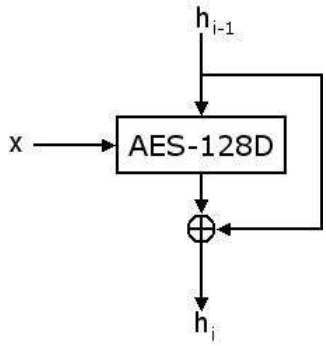


Figure 15.5: AACS AES-BASED HASHING FUNCTION

AACS also uses a hashing function, shown in figure 15.5, which is based on the AES decryption mode. The plaintext x is divided into i pieces of 128 bits. x_1 and h_0 are used as the first input, where h_0 is a given initial value and $h_1 = D_{x_1}(h_0) \oplus h_0$ is the hash of these two values. The subsequent hashes, if necessary, of $x_2, \ldots, x_i$ are performed with $h_1, \ldots, h_{i-1}$ respectively.

The hashing function described above is important in calculating the title key from the media key. The media key, as h_0, is used in a hash function together with the volume identifier, as x_1. The volume identifier is a unique random number for every instance (i.e. disk) of a specific title or just for a certain title, depending on the implementation the content provider wishes to use. The result of this hash h_1 is used in a hash with the usage rules x_2 resulting in h_2. The result of this last hash is the key with which the encrypted title keys can be decrypted. The obtained title keys can then be used to decrypt each corresponding title as needed. Note that hashing the volume identifier and the usage rules can take more than one round of the hashing function, but for simplicity it is assumed that one round is enough.

15.2.4 Possible weaknesses

Despite the improvements of AACS over CSS, there are still factors which form potential weaknesses and can make AACS a failure for the new optical storage disks as CSS was for the DVD. See also [Per05]

- **Key management.** As stated before, key management is the biggest issue in content protection. The devices have to be able to decrypt the encrypted content, which means that they should be able to obtain the key for doing that in one way or another. In AACS the media key is obtained using the device key. This device key is stored in the hardware or software, depending on the type of implementation. However, anything that is stored in a "hidden" area can be reverse engineered.

  AACS counts on this by being able to revoke certain device keys for newer media after a device key is compromised, which implies that certain devices will stop playing newer content after its device key is compromised, until after an upgrade takes place. If device keys are compromised at a high enough rate, the consumer will be stuck with a piece of software or hardware that needs constant upgrading, which is annoying in both cases and could be impossible in the latter case.

- **Decoded content.** Generally speaking, a consumer would buy a movie to view it. This implies that the contents will have to be decoded at some point. No matter how late the

final decoding takes place, it is inevitable that the content will be decoded at some point [Per05]. Even possible implementations of trusted computing can not prevent this[2].

AACS might count on the Digital Millennium Copyright Act to protect them, but this will probably not prevent unauthorised copying, it simply makes it illegal. In the United States CSS was protected under the same law, but the copying of DVDs is probably as common in the United States as it is in Europe, where such a law has not yet been enforced everywhere.

- **Professional piracy.** Consumer recorders implementing AACS are not allowed to write into the Media Key Block area of the DVD, except for its extension, which should prevent bit-by-bit copying. Professional pirates usually have access to the same machinery with which the official disks are made. The Blu-ray Disk Association and HD-DVD forum will try to anticipate this by giving a very limited number of factories the right to make the disks. This sounds like security through obscurity, which has not proven to be a very successful concept in the past.

Summary and Conclusions

The Advanced Access Content System is designed to prevent another debacle comparable to the failure of the Content Scrambling System for the protection of the copyrighted content on DVDs. There have been some improvements in the design of the system, which makes the chances for success better, but not more than that. There are a few difficult problems associated with content protection, illustrated in section 15.2.4, which also apply to AACS.

Trying to keep parts of the production process under control probably will not prevent professional pirates from copying disks. Neither will protection by some law, which does not have undisputed success as it is.

Personally I think Advanced Access Content System will not be broken as easily as the Content Scrambling System, but it most likely will not provide the protection the film studios are hoping for. However, it can be stated without any doubt that this new system will provide the consumer with extra problems, which presumably no one is waiting for.

[2]Whether AACS will require a trusted computing platform is not yet clear, but see [Zag04]

Chapter 16

Bluetooth Security

Written by *Gerben D. Oostra*

Bluetooth is a wireless communication technology for short ranges, to eliminate wires between nearby devices. Bluetooth allows for wireless data communications within 10 to 100 meters, without requiring a line of sight. Because it is easier to attack wireless communication, Bluetooth defines link level authentication and encryption. It is claimed that the used frequency hopping spread makes it more difficult to 'tap the wire'.

In this chapter it will be shown that the frequency hopping spread spectrum can be exploited, that the authentication can be deceived to impersonate devices, and that the optional encryption cannot prevent these attacks.

16.1 Bluetooth specification

In this section the different aspects of Bluetooth will be explained. First the radio frequency will be handled, followed by the connection establishment, authentication, and finally encrypted communication. These are required to explain possible attacks on Bluetooth in the following section.

16.1.1 Radio frequency

Bluetooth works in a broadband frequency range of 2.45 GHz with the ISM (International Scientific, and Medical Band) spectrum [LFE+01]. ISM allows multiple devices to communicate on the same frequency, and avoids interference with a frequency hopping technology. The broadband is divided into 23 or 79 (depending on the country code) narrow-band frequencies, through which the devices switch in a pseudo-random manner. In the U.S. and most other countries 79 bands have been assigned for use by Bluetooth devices, in Spain and France only 23 [JW01].

Bluetooth implements the frequency hopping with Gaussian Frequency Shift Keying (GFSK) [LFE+01]. To successfully use the hopping scheme devices must transmit and receive at synchronized time moments and time intervals. They also must transmit and receive on the right frequency. This is achieved with the frequency selection module (FSM), which uses the country code, the device address, and the clock as input. Which device's address and clock are used as input depends on the phase of the connection establishment.

Each device has a unique 48 bit address (BD_ADDR), consisting of a 24 bit lower address part (LAP), a 4 bit upper address part (UAP) and a 20 bit non-significant address part (NAP) [LFE+01]. Besides this unique address, each device has a self determined unit key. This is calculated with the E_{21} algorithm, using the BD_ADDR and a random number as input. This key is generated when the device is started for the first time. Each device also has a 28 bit clock (CLK) running at 3.2 kHz and thus cycles once in each 23.3 hours. In each communication, the master transmits in even time slots, and the slave may response in the following odd slots [Kg03].

Next to the avoided interference of signals, it is stated that the pseudo random frequency hopping sequence improves security because it should be harder to listen to a conversation. In section 16.2.2 it will be explained how the frequency hopping sequence can be exploited using a relay attack.

16.1.2 Connection establishment

To establish a connection to another device, the device first has to be discovered. This is done using an inquiry [Kg03]. To create a communication link between two devices, the paging procedure is executed.

Inquiry

The inquiry is used to discover unknown devices in the area [Kg03]. Each device has a general inquiry access code (GIAC), which is used to inquire that device. A device can be stated to only be temporarily (Limited discoverable), or not at all (Non-discoverable) discoverable. Depending on the state, a device listens for inquiry packets containing its GIAC on different frequencies. If the client receives such packet, it responds with its BD_ADDRESS and clock setting. Because the devices are not synchronized at this moment, the inquirer (master) does not know the exact clock setting. The FSM of the inquirer uses the GIAC as address and the clock of the inquiring device as inputs. The sequence consists of 32 frequencies and has a period of length 32.

Paging [Kg03]

When a device is discovered, paging is used to establish a connection. A device can be non-connectable; it then will never scan for page requests. The FSM uses the device address and clock of the slave as input, and its sequence consists of 32 frequencies, and has a period of length 32.

The paging procedure starts with a page request. The master guesses the slaves clock, and determines the corresponding frequency and the frequencies belonging to the previous eight and next seven clock ticks. This is called the A-Train. The master quickly sends ID packets containing the device access code (DAC) of the paged slave on these A-train frequencies. If no response is received, the master sends the packet again on the remaining frequencies (the eight before and after the A-Train). This is repeated for either 128 or 256 times, until the slave's response is received. The slave scans for ID packets containing its DAC on different frequencies. After reception of such ID packet, the slave responds with the same ID packet. When the master receives the slave response, both devices are synchronized because they both know on which frequency to listen (and the clock from which it is derived). The master responds with an FHS packet containing its own device address and its own actual clock setting. The slave responds to this with another ID packet. When the master receives the second slave response, it is sure that both know the master's address and clock setting.

After paging, the master's address and clock setting are used by both the master and slave as input to the FHS, such that both use the same pseudo-random frequency sequence.

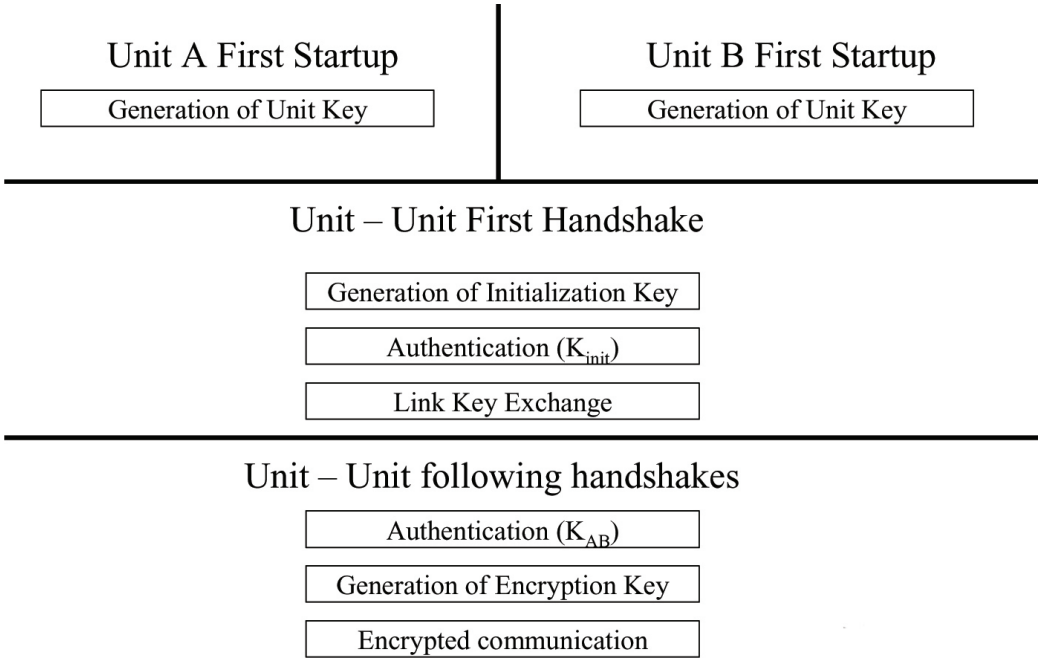


Figure 16.1: Key management [LFE+01]

Pairing [LFE+01]

When two devices have never communicated with each other before, they are first paired. Pairing is used to exchange link keys, such that they can authenticate each other in the future without requiring any PIN input. An overview of the generated key is shown in figure 16.1. To authenticate each other for the first time, without any prior known link key, an initialization key is generated. The E_{22} algorithm takes as input the user entered PIN, the PIN length (L), and a random number (IN_RAND), and returns the key. The device requesting the communication is the claimant, the other the verifier. The verifier chooses the random number, and sends it un-encrypted to the claimant. The E_{22} results in the same initialization key if and only if they have entered the same PIN. This initialization key is used as a temporary link key.

$$K_{AB} = E_{22}(IN\_RAND, PIN, L) \tag{16.1}$$

Then, just as if they have met before, the two devices authenticate each other with their (temporary) link key. Then they force to change link keys. During communication, all packets are encrypted by exclusive or'ing it with the (temporary) link key (K_{AB}).

$$X^{cipher} = X \oplus K_{AB} \tag{16.2}$$

Authentication [LFE+01]

To verify that two devices are communicating with the device they think they are communicating with, an authentication scheme is implemented in Bluetooth. It is a challenge/response system. One device (A) sends a random number (A_CHAL) to the other (B). Device B then calculates a value SRES', using the E_1 algorithm with as input values the received random value, its own device address and the link key (K_{AB}). Device B sends the value SRES' back to device A, which has calculated SRES in the same way. SRES and SRES' are equal if their link keys were equal, if they were, authentication is successful. Next to a SRES value, the E_1 algorithm also returns

an authenticated ciphering offset (ACO) which can be used in generating an encryption key.

$$challenge = A\_CHAL \tag{16.3}$$

$$(SRES, ACO) = E_1(BD\_ADDR_B, K_{AB}, A\_CHAL) \tag{16.4}$$

$$response = SRES \tag{16.5}$$

Link key exchange [LFE+01]

It is essential to agree upon a key with which the two devices will authenticate each other in the future. Depending on the memory constraints and required level of security, the link key (K_{AB}) can be the unit key of one of the devices (K_A or K_B), or a combination derived of both unit keys.

The unit key is generated virtually only once in each devices lifetime, the first time it is turned on [LFE+01]. The device chooses a random number, and calculates the unit key using the E_{21} algorithm.

$$K_A = E_{21}(BD\_ADDR_A, RAND_A) \tag{16.6}$$

To use one unit key, it is transmitted after simply exclusive or'ing it with the current (temporary) link key. The receiving device XOR's it with its own (equal) link key, having the unit key of a certain device.

$$K_A^{cipher} = K_A \oplus K_{AB} \tag{16.7}$$

$$K_{AB}^{new} = K_A = K_A^{cipher} \oplus K_{AB} \tag{16.8}$$

To combine the two unit keys, each device sends a random number (K_RAND) to the other (encrypted by XOR'ing it with the current link key). Both devices then calculate two device related keys, and XOR's them both to obtain the resulting link key (K_{AB}). The device related keys are calculated by using the E_{21} algorithm, with the devices BD_ADDR and a transmitted random number as input.

$$K_{AB}^{new} = E_{21}(K\_RAND_A, BD\_ADDR_A) \oplus E_{21}(K\_RAND_B, BD\_ADDR_B) \tag{16.9}$$

If the devices have exchanged link keys, the connection is dropped such that a new authentication phase is started.

Encryption key generation [LFE+01]

The encryption key is generated with the E_3 algorithm using the link key, a random number (EN_RAND_A), and a ciphering offset number (COF). The random number is generated by one device and transmitted to the other (encrypted by XOR'ing it with the current link key). The COF can either be the authenticated ciphering offset (ACO) generated during authentication, or the concatenation of the device address of the sender. The ACO is used for one-to-one encrypted communication; the second is used for an encrypted broadcast communication.

$$COF^{broadcast} = BD\_ADDR + BD\_ADDR \tag{16.10}$$

$$COF^{private} = ACO \tag{16.11}$$

$$K_c = E_3(EN\_RAND_A, K_{AB}, COF) \tag{16.12}$$

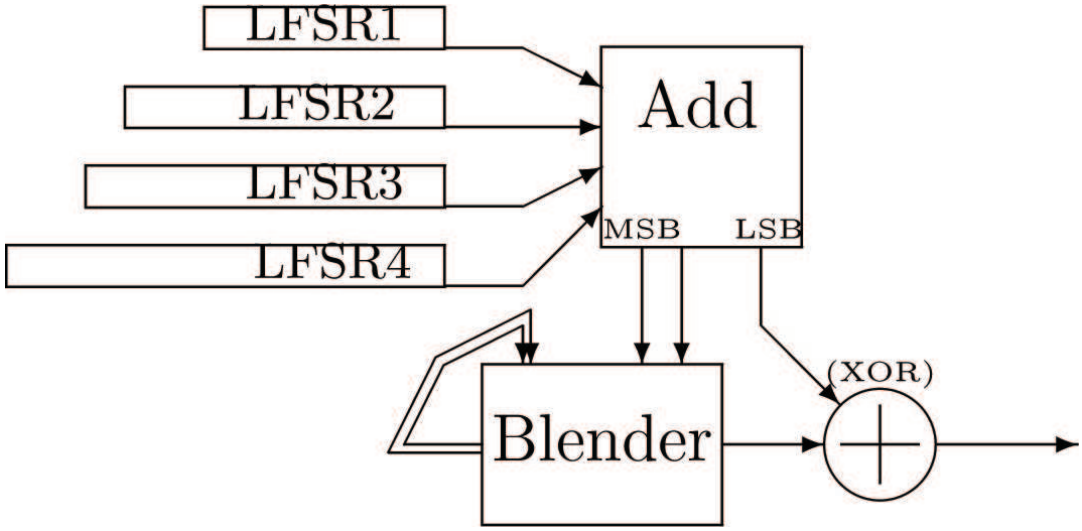


Figure 16.2: E_0 keystream generator [FL01]

Encrypted communication [LFE+01]

Encryption and decryption is done by XOR'ing the data with an encryption stream. This cipher stream is generated by the E_0 algorithm, with as input the BD_ADDR of the master, the clock of the master and their encryption key. Decryption is symmetric.

$$K_{cipher} = E_0(BD\_ADDR_A, CLK_A, K_c) \tag{16.13}$$

$$data_{A-B}^{cipher} = data_{A-B} \oplus K_{cipher}, data_{B-A}^{cipher} = data_{B-A} \oplus K_{cipher} \tag{16.14}$$

$$data_{A-B} = data_{A-B}^{cipher} \oplus K_{cipher}, data_{B-A} = data_{B-A}^{cipher} \oplus K_{cipher} \tag{16.15}$$

The E_0 key stream generator consists of four linear feedback shift registers (LFSR), with a total length of 128 bits, and a four bit finite state machine, the blender FSM [FL01]. For each outputted bit, the LFSRs are clocked once. The two most significant bits (MSB) of their sum are used as input to the blender FSM. The least significant bit of the sum is XOR'ed with the blenders output. The BD_ADDR, the clock and the encryption key are used to generate the initial state. This is shown in figure 16.2.

16.2 Bluetooth Security

Possible security threats concerning mobile ad hoc networks are impersonation and eavesdropping [LFE+01]. Two communicating Bluetooth devices have two shared secrets. The first is the exact clock setting. The second is the link key. The clock setting (with the device address) determines the frequency hopping sequence, while the link key is the base of all encryption and authentication protocols.

This gives the attacker two required objectives for eavesdropping and impersonation. The frequency hopping sequence must be known to eavesdrop on the communication. The link key is needed to either decrypt communication or authenticate the attacker. These objectives can be achieved in two manners, the first is just listening, the second is relaying.

16.2.1 Passive attack on Bluetooth

When an attacker wishes to minimize the chance to be detected, he can use a passive attack. The attacker determines the frequency hopping sequence used for communicating. This sequence depends on the clock and Bluetooth device address of the master. Jakobsson and Wetzel [JW01] show how the master's clock and address can be found.

In the inquiry response, the slave device reveals its clock and device address. These are the seed for the paging hopping sequence. During paging the master reveals its clock and device address.

Because transmitted messages do not necessarily are transmitted in one time-slot, it is not sufficient to listen to all paging frequencies. However, by scanning through the 32 inquiry frequencies, and eavesdropping on the response messages, the paging hopping sequence seed can be determined. This then can be used to determine the master's clock and address.

The attacker then can eavesdrop on any further transmitted message. This attack needs 32 parallel frequency listeners. If the two devices have not previously met, then the attacker is able to calculate the PIN, and all derived keys. These keys then can be used for further impersonation attacks, or to eavesdrop on encrypted communication.

PIN determination

If the initialization key is the first unknown key for the attacker, then the security relies on the PIN (which has no minimum length).If the PIN is of a regular size of 4 digits, then the key space is limited to 10,000 different values. This is an easy brute force attack.

To verify off-line whether the PIN is correct, the attacker needs one challenge-response authentication transcript and the random number generated for the initialization key [JW01]. These can be simply retrieved by the passive attack as described above.

Then an exhaustive search can be done on all PIN values up to a certain length. For each PIN the possible initialization key can be generated, for each initialization key the SRES' can be generated using the BD_ADDRESS and the random value, and this can be verified with communicated SRES.

16.2.2 Active attack on Bluetooth

An attacker can participate in the communication between two devices, giving him the power to alter messages and calculate the PIN. To calculate the PIN code, the initialization key seed and a challenge-response tuple is required. In the previous section 16.2.1 it is shown how to perform a brute force attack on the PIN. It is also possible to initiate such attack.

PIN determination

An attacker can start communication with the device to impersonate, and transmit a random number as seed for the initialization key. The attacker sends another random number as challenge, and retrieves the SRES' response. The attacker then can perform a brute force attack on the PIN, verifying that the resulting SRES value is equal to the received SRES'.

Unit key stealing

After two devices have authenticated each other to each other, the link key is determined. If the master device is low on resources, the master will always use its unit key as link key. However, the slave then knows the unique unit key of the master, and then is able to impersonate the master

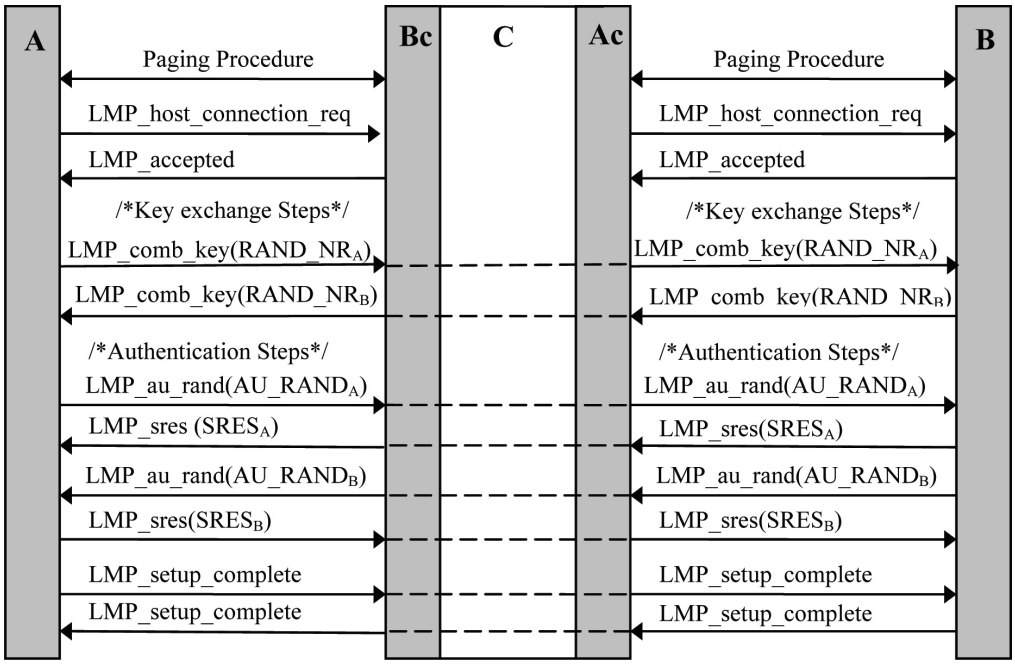


Figure 16.3: Two sided initiated relay attack [LcA+04]

and eavesdrop on all communication from the master [LFE+01], [JW01]. Lamm et al. [LFE+01] refer to this as unit key stealing.

Relay attack

Another active attack is the relay attack. The attacker relays every message to the other device, impersonating both devices. This attack can be started by initiating or intercepting communication between the two devices to impersonate [JW01]. The general two-sided relay attack is shown in figure 16.3.

Initiating relay attack The attacker initiates a relay attack by contacting each one of two devices to impersonate or eavesdrop. The attacker either makes both of them slaves or both masters. This is needed to let the two devices follow different frequency hop sequences. Therefore they will not see the messages they transmit to each other. The attacker then can relay every message to the other. Because the attacker initiates the communication, it will first become the master. If one attacked device is already the master of a piconet, it prefers to have the attacker as a slave in the piconet to prevent jamming. The attack requires that then the other attacked device also is a master relative to the attacker. However, only slaves can require becoming masters. Kügler [Kg03] therefore explains that both attacked devices must request for a role switch while establishing a connection, which is an unlikely case. Relaying messages also fails if an encrypted link is used. The master's address is used for the encryption stream, and both devices believe that the other is the master.

Kügler [Kg03] gives two improvements. Jakobsson and Wetzel [JW01] initiate the communication. Kügler shows that both intercepting and initiating are possible attacks, even when an encrypted link is used. The attacker establishes a connection to both devices, posing to be the other device. The attacker is the master, and therefore can determine the clock setting to be used. The encryption key uses the master's address and clock setting. It however, does not depend on the most significant bit of the clock setting. The attacker therefore chooses the clock

of device A (CLK_A) XOR'ed with 2^{27} (the clock is 27 bits). Next, the attacker switches roles with device A (which A may already have requested). A then uses its own clock setting CLK_A, as it is the master, while device B still uses $CLK_A \oplus 2^{27}$. They use the same channel hopping sequence, but at different offsets, as they both think that A is the master. The attacker now impersonates both devices to each other, and has the power to change relayed packets.

An attacker can initiate the relay attack on two devices, but this does not make the two devices transmit interesting files to each other. Kügler [Kg03] shows that it is also possible to intercept a communication startup using a relay attack.

Intercepting relay attack The interception attack of Kügler [Kg03] makes use of the paging procedure. The master sequentially sends a number of page requests until the slave responds. The attacker responds faster than the slave to the page request. This can be achieved with two methods.

The first method is 'jamming'. The attacker scans all 32 page hopping frequencies in parallel, and responds as fast as possible. It therefore will not respond slower than the paged slave, but possibly at the same time. However, the channel then will be jammed, and the attacker gets another chance.

The second method is 'busy waiting', and only requires one listening device. The attacker starts establishing a connection with the slave before the master does, but does not send the FHS packet. This leaves the slave waiting for a FHS packet. The attacker can now respond to the master's page request (the ID packet), without having the slave as competitor. The slave waits for FHS packets until a timeout occurs, after this timeout it restarts listening for ID packets. If no ID packet is received, the slave returns in normal operation. If the FHS timeout occurs, the attacker can again initiate a connection leaving the slave again in a "half open" connection.

At this point, the attacker has a connection with the master. The attacker restarts the paging procedure with the slave, and finally responds with a relayed FHS packet containing a different clock setting. The master and slave will therefore use the same channel hopping sequence, but at a different offset. If the slave requests a role switch, the attacker relays the packets. When the new master sends a FHS packet, the attacker can again change the clock setting, again leaving both devices at different offsets in the same new channel hopping sequence.

Relaying (encrypted) communication The next problem is to attack an encrypted link. Bluetooth uses a stream cipher for optional encryption, which depends on the master's device address and clock setting. It therefore changes for every received or transmitted packet. While encryption doesn't allow the attacker to eavesdrop on the communication, the attacker may manipulate every intercepted packet.

The first possible situation is unidirectional communication. Then only one device sends data, while the other device acknowledges reception. The attacker should let the slave have the clock run at least one tick behind the master. The attacker buffers the data from the master, and transmits them when the slave has the same clock setting. If the communication is not encrypted, then the packets can be relayed at any moment. As Bluetooth offers reliable communication, with ACK and NAK packets not encrypted, the attacker can respond immediately with an ACK to the master and discard the ACK message from the slave. However, if the slave responds with a NAK packet, the attacker is unable to resend the packet. Therefore, the attacker should respond with a NAK packet to the master until the slave has responded with an ACK. A drawback is the major bandwidth reduction.

The second possible situation is bidirectional communication. The encryption cipher stream depends on the clock setting of the master. However, the attack requires that the two devices

have different clock settings, to not let them hear each other. It is unfortunately possible to relay encrypted communication with a certain clock offset. The encryption sequence depends on only the first 26 clock bits, discarding the most significant bit. By flipping the most significant bit for one device at the connection establishment, both use the same encryption key but different frequency hopping spectra. The most significant bits however do need to be equal. Therefore, the attacker has to relay a packed within the same transmission frame. Bluetooth has an uncertainty window of $10\mu s$ because clocks do not drift with whole clock ticks. This gives the attacker $10\mu s$ to relay the packet.

Changing communication When the attacker relays all (encrypted) communication, he has the power to flip certain bits. In the previous section it is shown that the attacker needs to relay packets within $10\mu s$ if the communication is encrypted. It may take more time to determine which bits to flip. The solution is to not relay the messages, and reply with NAK packets to the transmitter. The attacker replies with NAK packets until he knows which bits to flip. In the next resent message, the attacker can flip the bits and immediately bitwise relay the message. The attacker can continue to reply with NAK packets until the other device gives the desired response.

16.2.3 Encryption key determination

If only a communication is recorded, without any authentication messages, the cipher can be attacked. This can be done using a 128 bits known plain-text attack with 2^{100} bit operations, or with a known birthday attack in time and memory complexity 2^{66} [JW01].

Fluhrer and Lucks [FL01] derive the cipher key from a set of known key stream bits in time varying from $O(2^{84})$ if 132 bits are available to $O(2^{73})$ given 2^{43} known key stream bits. They use a linear consistency attack, described in 1989 by Zeng, Yang, and Rao [ZYT89]. The used E_0 key stream generator in Bluetooth uses 4 linear feedback shift registers (LFSR), and a 4-bit finite state machine (Blender FSM). The LFSR's are added, the two most significant bits (MSB) are input to the blender. The blenders output is XOR'ed with the least significant bit. This is shown in figure 16.2.

The LFSR's are of bit length 25 (LFSR1), 31(LFSR2), 33(LFSR3), and 39(LFSR4). Fluhrers and Lucks' [FL01] base attack technique is guessing the initial states of part of the cipher, and checking for consistency. For each possible starting state a set λ of linear equations for LFSR3 and LFSR4 is maintained. They repeatedly examine a possible state n of LFSR1, LFSR2 and the blender FSM. The exclusive or of these with the known key stream bit Z_n, should be equal to the exclusive or of LFSR3 and LFSR4. If the exclusive or is zero, a branch is created for the λ rules $(LFSR3_n = 0 \wedge LFSR4_n = 0)$ and $(LFSR3_n = 1 \wedge LFSR3_n = 1)$. If the exclusive or is one, then to λ the rule $(LFSR3_n \neq LFSR4_n)$ is added. For $n \geq 33$ the tap equations of LFSR3 are added to λ. For $n \geq 39$ the tap equations of LFSR4 are added to λ. If λ becomes inconsistent, backtrack to the next state. If it is consistent, then the next blender FSM state can be computed using the current (known) state, and the (known) number of LFSRs that output a one.

Fluhrer and Lucks [FL01] show that the base attack can be further improved. They note that the base attack is more efficient if the outputs of LFSR3 and LFSR4 exclusive or'ed together happens to have a high hamming weight (the number of '1'-bits in sequence). Therefore, they assume that, at a specific point in the key stream, the next n+1 bits of $(LFSR3 \oplus LFSR4)$ are n ones followed by a zero. Here n will be less than the length of the LFSRs. The algorithm becomes the following. Select a position in the known key stream that is the start of more than 132 consecutive known bits. Examine each possible combination of the 4 bits of the blender FSM

state, 25 bits of LFSR1 state and the last 30-n bits of LFSR2 state. Compute the initial $n+1$ bits of LFSR2 state that is consistent with the exclusive or of LFSR3 and LFSR4, consisting of n ones and then a zero. Run the base attack on that setting. Stop if it finds a consistent initial setting.

If multiple non-consecutive packets are received, the attack can be done independent one each one of them. However, the exclusive or of the clock values are known, and therefore the exclusive or of the first level LFSRs are known [FL01].

Because of this total attack, the real security level of E_0 is no more than 73-84 bits. The security depends on the amount of key stream available to the attacker, but not on the key length if above 64 bits. The Bluetooth specification requires that the length of keys may be changed, to easily increase the security. This attack shows that all keys of length more than 64 bits are equally secure.

Jakobsson and Wetzel [JW01] explain that a more practical solution is to determine the stream cipher output using a known sent/received plain-text, with which the corresponding received/sent cipher-text can be decrypted. This is possible because the stream cipher output is reused for decryption [JW01]. Using this method, the number of known bits can be increased, lowering the complexity of determining the full encryption key using the method of Fluhrer and Lucks [FL01]. I think the knowledge of absolute CLK (27 used bits) and BD_ADDR (48 bits) values can lower the complexity of the encryption key attack. The attacker needs to know these to be able to eavesdrop on the communication, so he could easily use them to attack the encryption key.

16.3 Bluetooth improvements

In the previous sections, different attacks on bluetooth have been explained. Now some solutions to prevent the attacks will be given.

The man in the middle attacks can be prevented [Kg03] by using all clock bits as input to the encryption stream, and encrypting packet headers too. The first prevents that an attacker can bring both devices out of synchronization. The second prevents an attacker to acknowledge packets and forward them later, because the acknowledge flag is included in the packet header. The man in the middle attack, a relay attack, is based on a deception: both victims think they are in the same piconet [LcA+04]. However, they are not. If the victims can include some information about their actual piconets in SRES, then relay attacks can be detected. If the victims' piconets are close together, then including the LAP (lower address part) of the master BD_ADDR and master clock in SRES is sufficient. This could easily be done by using an exclusive or of the random challenge value, the LAP and clock of the master instead of just the random value. If the piconets are not close to each other, the attacker can use the same address and clock for both victims. Because they are not close, they will not hear each other. Another limited use, but efficient precaution could be to exchange the challenge messages (AU_RAND) before sending out the responses [LcA+04]. In this way, the SRES value cannot be relayed. This solution works only if the attacker is the verifier in both piconets, which corresponds to a one-sided attack.

The brute force attack on the PIN could easily be prevented by using sufficient long and sufficient random PINs [JW01]. If the users chose the PINs uniformly at random, then 64bit PINs appear to be secure.

The unit key can be protected by either using a set of unit keys, one for each device it communicates with, or by using the unit key as input to a pseudo-random generator with the address of the other Bluetooth device as seed [JW01]. This limits the required amount of storage.

Summary and Conclusions

Bluetooth provides link level security, consisting of authentication and encryption. These are however cryptographically not secure. One needs additional application level encryption and authentication for real secure communication. Bluetooth itself could be improved in next editions, as explained in the previous section 16.3.

Bibliography

[Ada97] Adams, C. Constructing symmetric ciphers using the cast design procedure. In proc. *Designs, Codes and Cryptography 12* (1997), Kluwer Academic Publishers, pp. 283–316.

[Adm04] Administrator, A. L. *Advanced Access Content System, Technical Overview (informative).* AACS Licensing Administrator, 2004.

[AEP94] A. Ekert, B. Huttner, G. M. P. and Peres, A. Eavesdropping on quantum-cryptographical systems. *Phys. Rev. A* **50**, 2 (1994), 1047–1055.

[AMT01] A. Miyaji, M. N. and Takano, S. New explicit conditions of elliptic curve traces for fr-reduction. *IEICE Trans. Fundamentals* (2001).

[AMV93] A. Menezes, T. O. and Vanstone, S. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Trans. Inform. Theory* **39** (1993), 1639–1646.

[BDJR00] Bellare, M., Desai, A., Jokipii, E., and Rogaway, P. A concrete security treatment of symmetric encryption.
`http://www-cse.ucsd.edu/users/mihir/papers/sym-enc.ps.gz`.

[Ben92] Bennett, C. H. Quantum cryptography using any two nonorthogonal states. *Phys. Rev. Letters* **68**, 21 (1992), 3121–3124.

[Beu03] Beukers, F. *Elementaire Getaltheorie.* Mathematisch Instituut Universiteit Utrecht, 2003, pp. 26–32.

[Bih98] Biham, E. Cryptanalysis of multiple modes of operation. *Journal Of Cryptology* **11** (1998), 45–58.

[BK98] Balasubramanian, R. and Koblitz, N. The improbability that an elliptic curve has subexponential discrete log problem under the menezes-okamoto-vanstone algorithm. *Journal of Cryptology* **11** (1998), 141–146.

[Bra93] Brands, S. An efficient off-line cash system based on the representation problem. Tech. rep., 1993.
`http://ftp.cwi.nl/CWIreports/AA/CS-R9323.pdf`.

[BS91] Biham, E. and Shamir, A. Differential cryptanaysis of des-like cryptosystems. In proc. *Advances in Cryptology, CRYPTO '90* (Berlin, 1991), vol. 537 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 2–21.

[CB] Curran, K. and Bailey, K. An evaluation of image based steganography methods.

[CGJ+01] Crosby, S., Goldberg, I., Johnson, R., Song, D., and Wagner, D. A cryptanalysis of the high-bandwidth digital content protection system. *LNCS 2320 Security and Privacy in Digital Rights Management* (2001), 192–200.

[Cro] Crowell, W. Introduction to the venona project. `http://www.nsa.gov:8080/docs/venona/index.html`.

[DBS04] Dan Boneh, B. L. and Shacham, H. Short signatures from the weil pairing. *Journal of Cryptology* **17** (2004), 297–319.

[Dwo01] Dworkin, M. Recommendations for block cipher modes of operation: Methods and techniques. Special Publication SP-800-38A, National Institute of Standards and Technology, 2001.

[Eke91] Ekert, A. K. Quantum cryptography based on bell's theorem. *Phys. Rev. Letters* **67**, 6 (1991), 661–663.

[Fel04] Felten, E. W. Dvd replacement still insecure. http://www.freedom-to-tinker.com/archives/000730.html, 2004.

[FL01] Fluhrer, A. and Lucks, S. Analysis of the e_0 encryption system. In proc. *Lecture notes in Computer Science 2259* (2001), Springer-Verlag, pp. 38–48.

[FR94] Frey, G. and Ruck, H. A remark concerting m-divisibility and the discrete logarithm in the divisor class group of curves. *Math. Comput.* **62** (1994), 865–874.

[GH04a] Garcia, F. D. and Hoepman, J.-H. Off-line karma: A decentralized currency for peer-to-peer and grid applications.

[GH04b] Garcia, F. D. and Hoepman, J.-H. Off-line karma: Towards a decentralized currency for peer-to-peer and grid applications (brief abstract).

[Hes] Hess, F. Exponent group signature schemes and efficient identity based signature schemes based on pairings.

[Int03] Intel. High-bandwidth digital content protection system. `http://www.digital-cp.com/data/HDCPSpecificationRev1_1.pdf` (2003).

[Irw01] Irwin, K. Four simple cryptographic attacks on hdcp. *http://www.angelfire.com/realm/keithirwin/HDCPAttacks.html* (2001).

[JK00] Jari Kytöjoki, V. K. Micropayments-requiremens and solutions.

[Jou04] Joux, A. A one round protocol for tripartite diffie-hellman. *Journal of Cryptology* **17** (2004), 263–276.

[JQYY02] Joye, M., Quisquater, J., Yen, S., and Yung, M. Observability analysis - detecting when improved cryptosystems fail. *Lecture Notes in Computer Science* **2271** (2002).

[JW01] Jakobsson, M. and Wetzel, S. Security weaknesses in bluetooth. In proc. *Lecture notes in Computer Science 2020* (2001), Springer-Verlag, pp. 176–191.

[Kat05] Katos, V. A randomness test for block ciphers. *Applied Mathematics and Computation* **162** (2005), 29–65.

[Kes00] Kesden, G. Content scrambling system. http://www-2.cs.cmu.edu/ dst/DeCSS/Kesden/, 2000.

[Kg03] Kgler, D. "man in the middle" attacks on bluetooth. In proc. *Lecture notes in Computer Science 2742* (2003), Springer-Verlag, pp. 149–161.

[KR96] Kilian, J. and Rogaway, P. How to protect des against exhaustive key search. In proc. *Advances in cryptology-CRYPTO '96* (1996), vol. 1109 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 252–267.

[LcA+04] Levi, A., Çetintas, E., Aydos, M., Kaya Koç Çetin, and Çaglayan, M. U. Relay attacks on bluetooth authentication and solutions. In proc. *Lecture notes in Computer Science 3280* (2004), Springer-Verlag, pp. 278–288.

[LFE+01] Lamm, G., Falauto, G., Estrada, J., Gadiyaram, J., and Cockerham, D. Bluetooth wireless networks security features. In proc. *Proceedings of the 2001 IEEE* (2001), Society for Industrial and Applied Mathematics, pp. 265–272.

[LHL93] Lenstra, A. and H.W. Lenstra, J. (eds.). *The development of the number field sieve*, vol. 1554 of *Lecture Notes in Mathematics*. Springer-Verlag, 1993.

[LHT97] Lee, J., Heys, H., and Tavares, S. Resistance of a CAST-like encryption algorithm to linear and differential crytanalysis. In proc. *Designs, Codes and Cryptography 12* (1997), Kluwer Academic Publishers, pp. 267–282.

[LN02] Lefranc, S. and Naccache, D. Cut-&-paste attacks with java. *Lecture Notes in Comnputer Science* **2587** (2002).

[Mat94] Matsui, M. Linear cryptanalysis methods for des cipher. In proc. *Advances in Cryptology, EUROCRYPT '93* (Berlin, 1994), vol. 765 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 386–397.

[Mil04] Miller, V. S. The weil pairing, and its efficient calculation. *Journal of Cryptology* **17** (2004), 235–261.

[Nao03] Naor, M. On cryptographic assumptions and challenges. *Lecture Notes in Comnputer Science* **2729** (2003).

[Nie00] Nielsen, J. The case for micropayments. *Journal of Cryptology* **13** (2000), 361–396.

[Per05] Perry, T. S. Dvd copy protection: Take 2. http://www.spectrum.ieee.org/WEBONLY/publicfeature/jan05/0105ldvd.html, 2005.

[PH] Provos, N. and Honeyman, P. Detecting steganographic content on the internet.

[Poi02] Pointcheval, D. Practical security in public-key cryptography. *Lecture Notes in Computer Science* **2288** (2002), 1–17.

[Pom05] Pomerance, C. Smooth numbers and the quadratic sieve. In *Algorithmic Number Theory: Lattices, Number Fields, Curves and Cryptography*, J. Buhler and P. Stevenhagen (eds.), Cambridge University Press, 2005.

[PS00] Pointcheval, D. and Stern, J. Security arguments for digital signatures and blind signatures. *Journal of Cryptology* **13** (2000), 361–396.

[PSB04] Paulo S.L.M. Barreto, Ben Lynn, M. S. Efficient implementation if pairing-based cryptosystems. *Journal of Cryptology* **17** (2004), 321–334.

[Sha49] Shannon, C. Communication theory of secrecy systems. *Bell System Technical Journal* **28** (1949), 656–715.

[Shi00] Shirky, C. The case against micropayments. *Journal of Cryptology* **13** (2000), 361–396.

[Sho97] Shor, P. W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Scientific Computing* **26** (1997). `http://www.arxiv.org/abs/quant-ph/9508027`.

[SLL+] Sung, J., Lee, S., Lim, J., Lee, W., and Yi, O. Concrete security analysis of ctr-ofb and ctr-cfb modes of operation. `http://cist.korea.ac.kr/Tr/TR01_18.ps`.

[SSSW98] Salter, C., Saydjari, O., Schneier, B., and Wallner, J. Towards a secure system engineering methodology. Tech. rep., NSA, 1998.

[ST01] Standards, N. I. of and Technology. Announcing the advanced encryption standard. Federal Information Processing Standards Publication FIPS-197, National Institute of Standards and Technology, 2001.

[Ste99] Stevenson, F. A. Cryptanalysis of contents scrambling system. http://www-2.cs.cmu.edu/ dst/DeCSS/FrankStevenson/analysis.html, 1999.

[Ste05] Stevenhagen, P. The number field sieve. In *Algorithmic Number Theory: Lattices, Number Fields, Curves and Cryptography*, J. Buhler and P. Stevenhagen (eds.), Cambridge University Press, 2005.

[Tel02] Tel, G. *Cryptografie: Bescherming van de Digitale Maatschappij*. Addison-Wesley, 2002 (363 pp.).

[VVS01] Vivek Vishnumurthy, S. C. and Sirer, E. G. Karma: A secure economic framework for peer-to-peer resource sharing.

[Wik05a] Wikipedia. Advanced encryption standard, 2005. http://en.wikipedia.org/wiki/AES.

[Wik05b] Wikipedia. Dvd, 2005. http://en.wikipedia.org/wiki/DVD.

[XDT03] Xu, S., Doumen, J., and Tilborg, v. H. On the security of digital signatures based on error - correcting codes. *Designs, Codes and Cryptography* **28** (2003).

[Zag04] Zagar. Son of css: Hd-dvd v. skylink..., 2004. http://zagarsbrain.blogspot.com/2004/12/son-of-css-hd-dvd-v-skylink.html.

[Zei99] Zeidler, E. *Applied Functional Analysis*, corrected third printing edition. 1999.

[ZYT89] Zeng, K., Yang, C.-H., and T.Rao. On the linear consistenc test (lct) in cryptanalysis with applications. In proc. *Lecture notes in Computer Science 435* (1989), Springer-Verlag, pp. 164–176.

Index