University of Tartu Prof. Dr. Dominique Unruh

Cryptology I

Short notes, spring 2012

Last Update: May 31, 2013

Important note: These notes are not supposed to be self-contained. Instead, they are intended as a reminder about which topics were discussed in the lecture. If you find mistakes in these notes, please send them to unruh@ut.ee.

Contents

Lecture on 2012-02-07

1 Historical ciphers

Shift cipher. The *shift cipher* uses a key $K \in \{1, \dots, 26\}$ (often represented by the letters $A = 1$, $B = 1$, ...). It operates on strings consisting of letters $A, \dots, Z$. (Of course, the shift cipher and the other ciphers described in this section can be based on other alphabets as well.) To encrypt a message $m = m_1 \dots m_n$, each letter m_i is replaced by the letter that occurs k places later in the alphabet. That is, write $x + k$ for the letter occurring k places after x in the alphabet (wrapping after Z; e.g., $D + 3 = G$, or $W + 5 = B$). Then the encryption of m under the key k is $(m_1 + k) \dots (m_n + k)$.

The shift cipher can easily be broken by trying out all keys and seeing for which key decrypting results in a meaningful plaintext (*brute-force attack*).

Vigenère cipher. The *Vigenère cipher* is a generalization of the shift cipher in different letter of the plaintext are shifted by different amounts. Its key is a string $k = k_1 \dots k_n$ with $k_i \in \{1, \dots, 26\}$ (again each k_i can be represented by a letter). The encryption of a message $m = m_1 \dots m_\ell$ under key k is $(m_1 + k_{1 \bmod n})(m_1 + k_{2 \bmod n}) \dots (m_1 + k_{(\ell-1) \bmod n})(m_1 + k_{\ell \bmod n})$.[1]

Since the key space of the Vigenère cipher has size 26^n, except for small n, a brute-force attack is not feasible.

[1] For integers a, b, $a \bmod b$ is the unique number i with $0 \le i < a$ such that $a = i + kb$ for some integer k. I.e., $a \bmod b$ is the remainder when dividing a by b.

If both the plaintext and the ciphertext is known, it is easy to deduce the key by computing the difference between each letter in the ciphertext and the corresponding letter in the plaintext. (Known-plaintext attack.)

If only a ciphertext is known, the Vigenère cipher can be broken using the following attack:

First, assume that we already know the length n of the key k. (Below, we see how to derive this length.) Then we can gather all the letters in the ciphertext $c = c_1c_2\dots c_\ell$ that were shifted by k_1: $c^{(1)} := c_1c_{1+n}c_{1+2n}\dots$. Then $c^{(1)}$ is the result of applying the shift cipher with key k_1 to the plaintext $p^{(1)} := p_1p_{1+n}p_{1+2n}\dots$). Thus we only have to break the shift cipher to find out k_1. But $p^{(1)}$ is not a meaningful plaintext (as it does not consist of consecutive letters). Thus testing each $k_1 \in \{1,\dots,26\}$ and checking whether the resulting plaintext is meaningful does not work. However, we can use the fact that the English language (and other languages) have typical *letter frequencies*. (I.e., some letters are considerably more common than others.) $p^{(1)}$ is expected to have a similar letter frequency as p. Thus, to find out k_1, we just find a k_1 such that $c^{(1)}$ decrypts to a plaintext with a letter frequency close to that of the English language (or whatever language the plaintext is written in).

To find out the length n of the key, we can either repeat the attack above for each candidate n, or we can use the following approach (other approaches also exist!): For a string x, let $I_C(x)$ be the probability that two randomly picked letters from x are equal. (I.e., $I_C(x) = \frac{1}{|x|^2}\sum_{i,j}\delta_{x_ix_j}$ where $\delta_{ab} := 1$ iff $a = b$ and $\delta_{ab} := 0$ otherwise.) We call $I_C(x)$ the *index of coincidence* of x. Different languages have different typical indices of coincidence. E.g., English text typically has $I_C(x) \approx 0.067$ while random text has $I_C(x) \approx 1/26 \approx 0.0385$. Since the index of coincidence does not change when applying a shift cipher, we expect that for the right key length n, we have $I_C(c^{(i)})$ to be close to the index of coincidence of the source language while for the wrong n it is expected to be closer to that of random text. Thus, we search for an n such that $I_C(c^{(i)})$ becomes as large as possible.

Lecture on 2012-02-09

Substitution cipher. The key of the *substitution cipher* is a permutation π on $\{A,\dots,Z\}$. The encryption of $m = m_1\dots m_\ell$ is $c = \pi(m_1)\dots\pi(m_\ell)$. I.e., each letter is replaced by its image under the permutation.

A brute-force attack is again infeasible. There are $26! \approx 4\cdot 10^{26}$ different keys.

However, frequency analysis helps to identify candidates for the image of the permutation of the most common letters. For example, in the encryption of an English text, if the letter x is the most common letter of the ciphertext, it is likely that $\pi(e) = x$ (since e is the most common letter in English text). In this way we get candidates for $\pi(m)$ for the most common letter m. However, due to statistical noise, this method will fail to give reliable results for rare letters. To break the substitution cipher one typically additionally makes use of the frequencies of *digrams* (two letter sequences) and *trigrams* (three letter sequences) in the plaintext language. (For example, the most common trigrams in English are *the* and *and*.) By identifying di/trigrams that appear often in the ciphertext, we get additional candidates for images of π. (E.g., if *kso* appears often, it

is a good guess that $\pi(t) = k, \pi(h) = s, \pi(e) = o$ or $\pi(a) = k, \pi(n) = s, \pi(d) = o$.) By combining the informations we get in this way, it is usually easy to break a substitution cipher.

Permutation cipher. The key of the permutation cipher is a permutation π on $\{1, \ldots, n\}$ for some n. To encrypt a plaintext m, we cut it into blocks of length n and reorder each block according to π. I.e., $c = m_{\pi^{-1}(1)} \ldots m_{\pi^{-1}(n)} m_{n+\pi^{-1}(1)} \ldots m_{n+\pi^{-1}(n)} m_{2n+\pi^{-1}(1)} \ldots m_{2n+\pi^{-1}(n)} \ldots$.[2]

The key-space has size $n!$, brute-force attacks are thus infeasible for reasonably large n.

To break the permutation cipher, one can try to reorder letters in the ciphertext in order to produce many common trigrams in the different blocks of the ciphertext. We do not give details here.

Enigma. The *Enigma* (a cipher machine used by the Germans in World War II) contains the following components:

- Keyboard: On the keyboard, a letter from A to Z can be typed. Depending on the letter that is typed, a corresponding wire that leads to the plugboard will carry electricity.
- Plugboard: The plugboard allows to select pairs of wires to be switched. That is, if we connect, e.g., A and Q on the plugboard, the letters A and Q will be swapped before being fed to the rotors.
- 3-4 rotors: The rotors are wheels that with 26 inputs on one side and 26 outputs on the other side. The inputs are connected to the outputs in a permuted fashion. Each rotors output is connected to the next rotors input. (Except for the last one, of course.) Upon each keypress, the rightmost (initial) rotors is rotated by one position. When the rightmost rotor finishes a turn, the next rotor is rotated by one position.
- Reflector: The reflector takes the output wires of the leftmost rotor, and connects them pairwise. This has the effect that the current coming from one output of the leftmost rotor flows back into another outputs and goes back through all the rotors, through the plugboard, to the lamps.
- Lamps: For each letter, we have a lamp that finally lights up and represents the current letter of the ciphertext.

An important property of the reflector is that the encryption of any letter in the plaintext can never be encrypted to itself. (I.e., if c is the encryption of m, then $c_i \neq m_i$ for all i.)

[2]Instead of using the inverse π^{-1} of the permutation, we could also use π itself. Choosing π^{-1} makes the description compatible with the example in the lecture.

The Enigma is highly non-trivial to break, but combined efforts of the British and the Polish managed to break it. The fact that the reflector ensures that no letter is mapped to itself was an important weakness (because it reveals something about the plaintext) that was exploited in breaking the Enigma.

2 Cipher and attack types

The ciphers presented in the preceding section fall into three categories:

- *Monoalphabetic substitution ciphers*: Each letter of the ciphertext depends only on the corresponding letter of the plaintext (i.e., the letter at the same position), the dependence is the same for each letter of the ciphertext. (Examples: shift cipher, substitution cipher.)
- *Polyalphabetic substitution ciphers*: Each letter of the ciphertext depends only on the corresponding letter of the plaintext (i.e., the letter at the same position), the dependence is different for different letters of the ciphertext. (Examples: Vigenère cipher, Enigma.)
- *Transposition ciphers*: The letters of the plaintext are not changed, only reordered. (Example: permutation cipher).

Of course, combinations of these types or encryption schemes that do not fall into any of these categories exist.

We also distinguish various attack types:

- *Ciphertext-only attack*: Here the attack is performed given only the knowledge of the ciphertext. This is the most difficult attack. We have described/sketched ciphertext-only attacks for the historical ciphers except the Enigma.
- *Known-plaintext attack (KPA)*: Here the attacker is additionally given the plaintext corresponding to the ciphertext. The goal is to learn the key (or to decrypt other ciphertexts for which we do not have the plaintext). For the ciphers in the preceding section (except the Enigma), known-plaintext attacks are trivial. Known-plaintext attacks are a relevant threat because in many cases the attacker may have some side information (such as typically used headers) that allow him to deduce (parts of) some plaintexts.
- *Chosen-plaintext attack (CPA)*: Here the attacker may even obtain ciphertexts corresponding to plaintexts of his choosing. This is motivated by the fact that often data we encrypt is not guaranteed to come from trusted sources. Thus some of our ciphertexts may contain plaintexts from the adversary.
- *Chosen-ciphertext attack (CCA)*: Here the attacker may in addition request to get the decryptions (plaintexts) of arbitrary ciphertexts of his choosing. This is motivated by the fact that we cannot always trusted the provenance of the ciphertexts we decrypt.

We will study security against CPA and CCA in more formal detail later in the lecture.

3 Perfect secrecy and the one-time pad

Perfect secrecy was introduced by Claude Shannon to be able to define and analyze the security of encryption schemes.

Definition 1 (Perfect secrecy) *Let K be the set of keys, let M be the set of messages, and let E be the encryption algorithm (possibly randomized) of an encryption scheme. We say the encryption scheme has* perfect secrecy *iff for all $m_0, m_1 \in M$ and for all c, we have that*

$$\begin{aligned}&\Pr[c = c' : k \xleftarrow{\$} K, c' \leftarrow E(k, m_0)]\\ =&\Pr[c = c' : k \xleftarrow{\$} K, c' \leftarrow E(k, m_1)].\end{aligned}$$

In the preceding definition, we used the following notation for probabilities: $\Pr[B : P]$ denotes the probability that B is true after executing the "program" P. In P, we use $x \xleftarrow{\$} Y$ to denote that x is uniformly randomly chosen from the set Y. And $x \leftarrow A(z)$ denotes that x is assigned the output of the algorithm or function A with arguments z.

Thus, the definition requires that if we encrypt m_0 or m_1 with a random key, the probability to get a given ciphertext c will be the same in both cases.

The following scheme has perfect secrecy:

Definition 2 (One-time-pad) *Let $K := \{0,1\}^n$ be the key space and $M := \{0,1\}^n$ the message space. (Notice that $K = M$.) Then the one-time-pad encryption $E(k, m)$ of message $m \in M$ under key $k \in K$ is defined as $E(k, m) := k \oplus m$ where $\oplus$ denotes the bitwise XOR.*

Lemma 1 *The one-time-pad has perfect secrecy.*

Lecture on 2012-02-14

Although the one-time-pad has perfect secrecy, it has three major disadvantages:

- The key has to be as long as the message.
- The one-time-pad is malleable. That is, by modifying the ciphertext, one can modify the contained plaintext. For example, given a ciphertext $c = E(k, m)$, by flipping the i-th bit of c, one gets a ciphertext of m' which is m with the i-th bit flipped. If m is, e.g., an order for a bank transaction of $1000\,e$, by flipping the right bit one gets a transaction of $9000\,e$. (This problem can be solved by applying "message authentication codes" to the message.)
- The one-time-pad does not allow to reuse keys. Given cipertexts $c_0 = E(k, m_0)$ and $c_1 = E(k, m_1)$ with the same key, we can compute $c_0 \oplus c_1 = m_0 \oplus m_1$ which leaks a lot of information about m_0 and m_1 (possibly all if m_0 and m_1 are, e.g., English words).

Can we improve on the one-time-pad in terms of the key length? I.e., is there an encryption scheme with perfect secrecy that has keys shorter than the message? The following theorem shows that this is impossible:

Theorem 1 *There is no encryption scheme that has the following three properties:*

- *Perfect secrecy.*
- $|K| < |M|$ *where* K *is the key space and* M *is the message space.*
- *When encrypting a message and then decrypting it, we always get the original message back.*

4 Stream ciphers and pseudo-randomness

A *(synchronous) stream cipher* is an encryption scheme that is constructed as follows: Let $G : K \to M$ be a pseudo-random generator (PRG), i.e., a function that takes a short random string $k \in K$ and outputs a long "random-looking" string $G(k) \in M$. Then, to encrypt a message $m \in M$, we compute $G(k)$, the so-called "key-stream" and then we XOR m with $G(k)$. I.e., the encryption of m is $G(k) \oplus m$. (This is reminiscent of the one-time-pad, except that we use the pseudo-random string $G(k)$ to pad the message instead of using a long random key.)

4.1 Linear feedback shift registers

When designing a stream cipher, the challenge is to construct a suitable pseudo-random generator. We first describe a classical approach for producing pseudo-random key-streams, the linear feedback shift register, LFSR. (Note: the LFSR on its own is not secure at all!)

A LFSR is described by its initial state $k \in \{0,1\}^n$, and by a set of indices $I \subseteq \{1,\ldots,n\}$. The LFSR contains an n-bit register $R = R_1 \ldots R_n$, initially loaded with k. In each step, the following happens: The value $b := \bigoplus_{i \in I} R_i$ is computed. Then the content of R is shifted to the right, and the rightmost bit of the LFSR is output (part of the keystream). The leftmost bit is initialized with b. I.e., formally, if R is the current state of the register, the next state R' will be: $R'_1 = b$, $R'_{i+1} = R_i$, and R_n will be output in this steps. The keystream consists of the output bits from all steps.

Unfortunately, LFSRs are not very secure. For example, one can, given I and a sufficiently long block in the keystream, compute the key by solving a system of linear equations (because the output of the LFSR is a linear function of its input).

However, LFSRs can be made stronger by combining several of them. For example:

- We use two LFSRs to get two keystreams, and then we combine the two keystreams using a nonlinear function F (where F is applied either bitwise or blockwise).
- We can use one LFSR to clock other LFSRs. For example, if the first LFSR outputs 0, then the second LFSR is clocked and its output added to the keystream. And if the first LFSR outputs 1, then the third LFSR is clocked and its output added to the keystream.

4.2 Digression – Best effort design and provable security

There are two main approaches for designing cryptographic schemes in modern cryptography.

Best-effort design. [3] While designing the scheme, one tries to make it as secure as possible, but without proving its security. One does, however, take into account the state-of-the-art in cryptanalysis (the science of attacking cryptosystems), and tries to see whether any know attack techniques allow to break the cipher. One publishes the scheme internationally, and lets other teams of researchers try to attack it. If any of the attacks works the scheme is discarded. Even if the scheme shows slight weaknesses under any of the attacks, the scheme is usually discarded (because such weaknesses might be an indication of more problems that could be found later). If the scheme survives all this, it can be used.

Provable security. In provable security, one designs the scheme based on some complexity-theoretic assumption. For example, one could assume that factoring large integers is hard. Based on this assumption, one mathematically proves that the scheme is secure. Although this does not necessarily imply that the scheme is indeed secure (because factoring might not be hard), it gives a great deal of trust into the scheme because the security of the scheme is based on a well-studied problem. Even if no cryptographer studies the new scheme, indirectly many people have studied its security by studying the underlying assumption. (Of course, one should use realistic and well-studied assumptions!)

Whether best-effort design or provable security is to be preferred is a matter of opinion and of the particular situation. In the case of stream ciphers, block ciphers, and hash functions, typically schemes developed using best-effort design are much faster. Also, for these kinds of functions, best-effort design seems to work well. On the other hand, public key encryption schemes and signatures do not lend themself very well to best-effort design: Such cryptosystems often have a rich underlying mathematical structure which may lead to many attack possibilities. A proof is needed to exclude all of them. Also, in the case of these cryptosystems, the gap in efficiency between provable security and best-effort design is much smaller.

Lecture on 2012-02-21

4.3 Stream ciphers with provable security

We now describe how to build streamciphers with provable security. For this, we first need to define the security of streamciphers that we wish to achieve.

Definition 3 (Negligible functions) *A positive function μ is called* negligible*, if for any positive polynomial p, it holds that for sufficiently large η we have $\mu(\eta) < 1/p(\eta)$.*

[3] The word "best-effort design" is not a generally known term, I chose the word to describe the methodology.

Definition 4 (IND-OT-CPA) *An encryption scheme (KG, E, D) consisting of a key-generation algorithm KG, an encryption algorithm E, and a decryption algorithm D is IND-OT-CPA (indistinguishable under one-time chosen plaintext attacks) if for any polynomial-time algorithm A there is a negligible function μ, such that for all $\eta \in \mathbb{N}$ we have that*

$$\left|\Pr[b' = b : k \leftarrow KG(1^\eta), b \xleftarrow{\$} \{0,1\}, (m_0, m_1) \leftarrow A(1^\eta), \\ c \leftarrow E(k, m_b), b' \leftarrow A(1^\eta, c)] - \tfrac{1}{2}\right| \leq \mu(\eta).$$

(Here we quantify only over algorithms A that output (m_0, m_1) with $|m_0| = |m_1|$.)

Intuitively, this definition means that no polynomial-time adversary, upon seeing an encryption of either m_0 or m_1 (both of his choosing), can guess which of the two messages has been encrypted (at least not better than by guessing).

Next, we define what assumption we use. For the moment, we use an assumption that is rather strong, namely the existence of pseudo-random generators.

Definition 5 (Pseudo-random generator) *A function $G : K \to M$ (where G, K, and M may depend on the security parameter η) is a pseudo-random generator (PRG) if for all polynomial-time algorithms A, there is a negligible function μ such that for all $\eta \in \mathbb{N}$ we have that*

$$\left|\Pr[b' = b : k \xleftarrow{\$} K, b \xleftarrow{\$} \{0,1\}, p_0 \leftarrow G(k), p_1 \xleftarrow{\$} M, b' \leftarrow A(1^\eta, p_b)] - \tfrac{1}{2}\right| \leq \mu(\eta).$$

Intuitively, this definition says that, without seeing the seed k, no polynomial-time adversary can distinguish between the pseudo-random $p_0 := G(k)$ and a truly random p_1.

We can then construct a streamcipher from a PRG:

Construction 1 *Fix a PRG $G : K \to M = \{0,1\}^n$. We construct a streamcipher (KG, E, D) with key space K and message space M as follows: The key generation $KG(1^\lambda)$ returns a uniformly random $k \in K$. Encryption: $E(k, m) := G(k) \oplus m$. Decryption: $D(k, c) := G(k) \oplus c$.*

Theorem 2 *If G is a PRG in the sense of Definition 5, then the streamcipher from Construction 1 is IND-OT-CPA secure.*

The main idea of the proof is to start with an adversary A that breaks the streamcipher, and to construct an adversary A' from A that breaks the PRG. (Such a proof is called a *reduction proof*.) **(not covered)**

5 Constructing pseudo-random generators

To get a streamcipher using Construction 1, we first need a PRG. We will now describe one possible way of constructing a PRG from more elementary building blocks.

Construction 2 *Let $\hat{G} : K \to K \times \{0,1\}$ be a PRG (with "1-bit-expansion"). We construct $G : K \to \{0,1\}^n$ as follows: For given $k \in K$, let $k_0 := k$ and compute for $i = 1, \ldots, n$: $(k_i, b_i) \leftarrow \hat{G}(k_{i-1})$. Let $G(k) := b_1 \ldots b_n$.*

Theorem 3 *If $\hat{G}$ is a PRG in the sense of Definition 5, then G is a PRG in the sense of Definition 5.*

A PRG with 1-bit-expansion can be constructed from elementary number-theoretic constructions. For example:

Construction 3 *Let N be a random Blum integer. (A Blum integer is the product of two primes p, q such that $p, q \equiv 3 \bmod 4$. When talking about random Blum integers, we mean that both primes are uniformly chosen from the set of primes of length η.) We define QR_N to be the set of all numbers $x^2 \bmod N$ where $x \in \{1, \ldots, N-1\}$ and x is invertible modulo N. (QR stands for* quadratic residue*.) We define $\hat{G} : \mathrm{QR}_N \to \mathrm{QR}_N \times \{0,1\}$ by $\hat{G}(k) := (k^2 \bmod N, lsb(k))$ where $lsb(k)$ denotes the least significant bit of k.*

Theorem 4 *If factoring random Blum integers is hard,*[4] *then $\hat{G}$ from Construction 3 is a PRG in the sense of Definition 5.*

Notice that by combining Theorems 2, 3, and 4, we get a provably secure streamcipher based on the assumption that factoring is hard. The resulting pseudo-random generator is called the Blum-Blum-Shub pseudo-random generator. Notice that this streamcipher is much too slow for practical purposes though.

Lecture on 2012-03-01

6 Block ciphers

A *block cipher* is an encryption scheme that deterministically encrypts messages of a fixed length. More precisely, a block cipher consists of an encryption function $E : \{0,1\}^n \times \{0,1\}^l \to \{0,1\}^l$, and a decryption function $D : \{0,1\}^n \times \{0,1\}^l \to \{0,1\}^l$ such that for all $k \in \{0,1\}^n$ and all $m \in \{0,1\}^l$, we have $D(k, E(k, m)) = m$. Here n is the key length and l the message/block length. (In practice, n and l are usually fixed. In theoretical analyses, n and l often depend on the security parameter η.)

The advantage of block ciphers is that block ciphers can typically be implemented in a very fast way (in software and hardware), and that they are much more versatile in their use than streamciphers (by using them as part of so-called "modes of operation", see Section 6.4).

Designing block ciphers. Most block ciphers are designed using best-effort design (see page 8). That is, when designing a block cipher, the block cipher is designed to withstand all known attack techniques. When designing a block cipher, any of the following would count as an attack:

[4] I.e., for any polynomial-time algorithm A, the probability that A outputs the factors of N given a random Blum integer N, is negligible.

- If there is any reasonable security definition for block ciphers such that the block cipher does not satisfy that security definition, then the block cipher is considered broken.
- If there is any attack on the block cipher that is any faster than a brute-force attack, then the block cipher is considered broken.

6.1 Data encryption standard (DES)

DES (data encryption standard) is a block cipher first published in 1977 that was in use for about 25 years. Today, DES is not considered secure any more. Its key length is only 56 bits, and it can be broken by a brute-force attack within only a day with modern special-purpose hardware. The block length of DES is 64 bits.

We will now describe the construction of DES.

Feistel networks. To do so, we first describe a concept that is used in DES and in many other block ciphers, the *Feistel network*.

A Feistel network is a particular way of encrypting messages. The core of a Feistel network for message blocks of length $2l$ is a function $F : \{0,1\}^n \times \{0,1\}^l \to \{0,1\}^l$ that takes a key and a message halfblock and outputs a halfblock. The encryption of a message m using an r-round Feistel network proceeds as follows:

- Split the message $m \in \{0,1\}^{2l}$ into two block $L, R \in \{0,1\}^l$.
- Run the following for $i = 1, \ldots, r-1$:
  - $L := L \oplus F(k_i, R)$. Here k_i is the so-called *round key* for the i-th round (see below).
  - $(L, R) := (R, L)$ (exchange L and R).
- $L := L \oplus F(k_r, R)$.
- Let c be the concatenation of L and R and return c.

Note that the operation of the Feistel network depends on so-called round keys $k_1, \ldots, k_r$. A block cipher that uses a Feistel network as part of its operation has to specify these round keys.

Notice that, when knowing the round keys of a Feistel network, one can easily decrypt. When c is the result of applying a Feistel network with round keys $k_1, \ldots, k_r$ to some m, and m' is the result of applying the same Feistel network (with the same function F) with round keys $k_r, \ldots, k_1$, then $m = m'$.

High-level structure of DES. The DES block cipher consists of the following components:

- A *key schedule*: The key schedule computes the round keys $k_1, \ldots, k_{16}$ from the 56-bit DES key k. Each k_i has length 48 bit and consists of a reordered subset of the bits of k. We do not describe the details of the key schedule.
- An *initial permutation*: This is a permutation (reordering) of the message block.
- A 16-round Feistel network with a round function F that will be described below.
- A *final permutation*: This is the inverse of the initial permutation.

To encrypt a message $m \in \{0,1\}^{64}$ with a key $k \in \{0,1\}^{56}$, we use the key schedule to derive the round keys $k_1, \ldots, k_{16}$. Then the message is reordered using the initial permutation, then fed through the Feistel network using the round keys $k_1, \ldots, k_{16}$, and then reordered using the final permutation. This gives the ciphertext.

To decrypt a DES ciphertext c, we perform exactly the same operations, except that we let the key schedule produce the round keys in opposite order $k_{16}, \ldots, k_1$. Since this will make the Feistel network decrypt, and since the initial and the final permutation are inverses of each other, this will decrypt c.

The Feistel round function F. The Feistel round function is the core of the DES block cipher. It performs the following steps (given a message halfblock $m \in \{0,1\}^{32}$ and a round key $k_i \in \{0,1\}^{48}$):

- *Expansion:* Computes $a \in \{0,1\}^{48}$ from m by duplicating some of the bits. We omit the details.
- *Use round key:* Compute $b := a \oplus k_i$.
- *Split:* Split b into 8 blocks $b_1, \ldots, b_8$ of 6 bits each.
- *Apply S-boxes:* Compute $d_1 := \mathrm{S1}(b_1)$, ..., $d_8 := \mathrm{S8}(b_8)$ where S1, ..., S8 are some fixed functions from 6 to 4 bits. The functions S1, ..., S8 are called *S-boxes*. They are designed to be as structureless as possible and are usually given as lookup tables.
- *Concatenate:* Let $d \in \{0,1\}^{32}$ be the concatenation of $d_1, \ldots, d_8$.
- *Permutation:* Apply a fixed permutation (reordering) to the bits in d, resulting in some reordered bitstring e. We omit the details.
- Return e.

The key components here are the S-boxes and the permutation. The S-boxes make sure that the dependencies between the input, the key, and the output of F are no simple linear functions but are instead as complicated as possible. Furthermore, to make sure that each input bit influences as many output bits as possible, the permutation shuffles the bits.

The concept of having complicated operations and of mixing the bits when designing ciphers was already identified by Shannon and called *confusion and diffusion*.

6.2 Meet-in-the-middle attack and 3DES

Since the main weakness of DES is its short key length, it seems natural to try to strengthen DES by applying DES twice with different keys, thus doubling the effective key length. More precisely, we can define the following block cipher:

Definition 6 (Double DES) *Let E' and D' denote the encryption and decryption algorithms of DES. Then we define the block cipher* double DES *with block length 64, key length 112, and encryption/decryption algorithms E and D as follows: $E(k, m) := E'(k_2, E'(k_1, m))$ and $D(k, c) := D'(k_1, D'(k_2, c))$ where k_1 and k_2 denotes the first and second half of $k \in \{0,1\}^{112}$, respectively.*

It seems that double DES is much stronger than DES: For a brute force attack, we need to try all 112 bit keys. However, there is another known plaintext attack attack on double DES, called the *meet-in-the-middle attack*. This attack is performed as follows:

- Let m and c be a plaintext/ciphertext pair. (I.e., $c = E(k, m)$ for some unknown key $k = k_1 k_2$ and E being the double DES encryption operation.)
- For each $k_1 \in \{0,1\}^{56}$, compute $a := E(k_1, m)$ and store (a, k_1) in a table.
- Sort the table containing the pairs (a, k_1) by the first component a. (Such that one can efficiently find (a, k_1) given a.)
- For each $k_2 \in \{0,1\}^{56}$, compute $b := D(k_2, c)$. Search for an (a, k_1) in the table with $a = b$. If found, call (k_1, k_2) a candidate key pair. (The candidate key pairs will be all $k_1 k_2$ with $E(k_1 k_2, m) = c$.)
- For each candidate key pair $k_1 k_2$, test whether $c' = E(k_1 k_2, m')$ for other message/ciphertext pairs (c', m'). If so, then $k_1 k_2$ is probably the right key.

This attack takes more time than breaking DES by bruteforce, because we additionally have to sort a table of 2^{56} entries. However, it takes considerably less time than doing a brute-force attack with key length 112 (that would need 2^{112} encryptions). Notice, however, that this attack needs a very large amount of memory.

Due to the meet-in-the-middle attack, double DES is not considered a good replacement for DES. Instead, a scheme called triple DES was used:

Definition 7 (Triple DES) *Let E' and D' denote the encryption and decryption algorithms of DES. Then we define the block cipher* triple DES *with block length 64, key length 168, and encryption/decryption algorithms E and D as follows: $E(k, m) := E'(k_3, D'(k_2, E'(k_1, m)))$ and $D(k, c) := D'(k_1, E'(k_2, D'(k_3, c)))$ where k_1, k_2 and k_3 denotes the first, second, and third 56 bit block of $k \in \{0,1\}^{168}$, respectively.*

Notice that even if we apply the meet-in-the-middle attack to 3DES, the time complexity will be in the order of 2^{112}. (We either have to iterate over all $k_1 k_2$ when building the table, or we have to iterate over all $k_2 k_3$ when using the table.)

A variant of 3DES uses keys with $k_1 = k_3$ (but k_1 and k_2 uniformly and independently chosen).

Also, when choosing $k_1 = k_2 = k_3$, then $E(k_1k_2k_3, m) = E'(k_1, m)$. Thus triple DES can emulate DES, saving costs in hardware when it is necessary to support both DES and 3DES for legacy reasons.

Lecture on 2012-03-06

6.3 Security definitions

An example of a security definition that block ciphers are usually expected to satisfy is the following. Intuitively, it means that, without knowing the key, the encryption/decryption looks like applying a totally random permutation and its inverse.

Definition 8 (Strong pseudo-random permutation) *A function $f : K \times M \to M$ (where f, K, and M may depend on the security parameter η) is a* strong pseudo-random permutation (strong PRP) *if the following two conditions are satisfied:*

- *There exists a deterministic polynomial-time algorithm for computing $f(k, m)$, and there exists a deterministic polynomial-time algorithm for computing $f^{-1}(k, m)$ where f^{-1} is a function such that $f^{-1}(k, f(k, m)) = m$ for all $k \in K$, $m \in M$.*
- *For all polynomial-time algorithms A, there is a negligible function μ such that for all $\eta \in \mathbb{N}$ we have that*

$$\big|\Pr[b = b' : b \xleftarrow{\$} \{0,1\}, k \xleftarrow{\$} K, \\ f_0 := f(k,\cdot), f_1 \xleftarrow{\$} \mathrm{Perm}_M, b' \xleftarrow{\$} A^{f_b, f_b^{-1}}(1^\eta)] - \frac{1}{2}\big| \leq \mu(\eta).$$

In the above definition, we use the notation A^X to denote that the algorithm A may make arbitrary many queries to the function X, that is, A gets $X(x)$ for any x that A sends to X. We write $f(k, \cdot)$ for the function mapping x to $f(k, x)$. And Perm_M denotes the set of all permutations (bijective functions) on M.

However, although strong PRP is quite a strong security guarantee, using a strong PRP for encrypting is not in general a good idea. Since a strong PRP (and any block cipher) is deterministic, when sending two messages encrypted with the same key, the adversary can tell whether the messages are equal or not. A better definition of security of encryption schemes is the following:

Definition 9 (IND-CPA) *An encryption scheme (KG, E, D) consisting of a key-generation algorithm KG, an encryption algorithm E, and a decryption algorithm D is IND-CPA (indistinguishable under chosen plaintext attacks) if for any polynomial-time algorithm A there is a negligible function μ, such that for all $\eta \in \mathbb{N}$ we have that*

$$\big|\Pr[b' = b : k \leftarrow KG(1^\eta), b \xleftarrow{\$} \{0,1\}, (m_0, m_1) \leftarrow A^{E(k,\cdot)}(1^\eta), \\ c \leftarrow E(k, m_b), b' \leftarrow A^{E(k,\cdot)}(1^\eta, c)] - \tfrac{1}{2}\big| \leq \mu(\eta).$$

(Here we quantify only over algorithms A that output (m_0, m_1) with $|m_0| = |m_1|$.)

In this definition, A has access to $E(k,\cdot)$. That is, A can produce arbitrary many additional encryptions beside the challenge encryption c. Thus the definition models the fact that the adversary knows nothing about the content of c even if the same key is used for other encryptions.

However, block ciphers (and any other deterministic encryption schemes) cannot satisfy this definition. In the next section, we describe how to use block ciphers to construct encryption schemes that are IND-CPA secure.

6.4 Modes of operation

Since a block cipher only allows to encrypt messages of fixed length, it is not, on its own, very useful. Instead, a block cipher is typically used as part of a *mode of operation* that describes how to apply it to larger messages. A mode of operation can also raise the security of the block cipher (e.g., by making the resulting encryption scheme IND-CPA secure).

In the following, we assume a block cipher with encryption E and decryption D and block length l and key length n. Furthermore, we assume that the messages we wish to encrypt have a length that is a multiple of l. Otherwise, *padding* needs to be used to enlarge the message to a multiple of l. Let m denote the message to be encrypted, and $m_1,\ldots,m_k$ be the message blocks of length l. And let k denote a key.

Electronic code book mode (ECB mode) In ECB mode, we compute $c_i := E(k,m_i)$, and the resulting ciphertext is $c := c_1 \ldots c_n$ (concatenation of the c_i).

ECB mode is not a very secure mode of operation. If m contains many repetitions, c will also contain repetitions. In general, c will leak information about the structure of m. E.g., when encrypting pictures using ECB mode, one can sometimes still recognize the picture.

Cipher block chaining mode (CBC mode) In CBC mode, to encrypt m, we first pick a random $IV \in \{0,1\}^l$. This value is called the *initialization vector*. Then we set $c_0 := IV$ and $c_{i+1} := E(k, m_{i+1} \oplus c_i)$. The resulting ciphertext is $c := c_0 c_1 \ldots c_k$.

Theorem 5 *Assume that E is a strong PRP. Let (KG', E', D') denote the encryption scheme that uses CBC mode with E as the block cipher. (That is KG' returns a uniformly chosen $k \in \{0,1\}^n$ and E' computes $c := c_0 c_1 \ldots c_k$ as described above.) Then (KG', E', D') is IND-CPA secure.*

Counter mode (CTR mode) In CTR mode, to encrypt m, we first pick a random $IV \in \{0,1\}^l$. This value is called the *initialization vector*. Then we set $c_0 := IV$ and $c_i := E(k, IV + i \bmod 2^l) \oplus m_i$. The resulting ciphertext is $c := c_0 c_1 \ldots c_k$.

Notice the strong similarity to stream ciphers.

Theorem 6 *Assume that E is a strong PRP. Let (KG', E', D') denote the encryption scheme that uses counter mode with E as the block cipher. (That is KG' returns a*

uniformly chosen $k \in \{0,1\}^n$ and E' computes $c := c_0 c_1 \dots c_k$ as described above.) Then (KG', E', D') is IND-CPA secure.

(upcoming)

7 Public key encryption

A *public key encryption scheme* (or *asymmetric encryption scheme*) is an encryption scheme in which there are two keys: A *public key* that is used for encrypting, and a *secret key* that is used for decrypting. It should not be possible to decrypt the message when knowing only the public key.

This allows a person A to distribute its public key to everyone, and to keep just a single secret key to decrypt all incoming messages. Distributing public keys is usually done using a *public key infrastructure (PKI)*: A PKI is a trusted authority that keeps a list of public keys of known individuals and provides these public keys to any interested party. (Using suitable mechanisms to ensure authenticity.)

In contrast, with *symmetric encryption schemes* (this means encryption schemes as in Sections 4 and 6 where encrypting and decrypting use the same key), each pair of potential communication partners needs to establish their own key pair, increasing the logistic challenge considerably.

Mathematics background. The following sections assume some knowledge about modular arithmetic. If you are not familiar with it, you might find a short overview in, e.g., Chapter 2 of [MvV96]. (You find it online here: `http://www.cacr.math.uwaterloo.ca/hac/about/chap2.pdf`.)

7.1 RSA

Definition 10 (Textbook RSA) *The RSA public key encryption scheme is the encryption scheme (KG, E, D) which is defined as follows:*

- Key generation: *$KG(1^\eta)$ does the following: Pick $p, q \xleftarrow{\$} Primes_\eta$ where $Primes_\eta$ is the set of all primes of length η. Let $N := pq$. Let $\varphi(N) := (p-1)(q-1)$. Pick $e \in \{0, \dots, \varphi(N)-1\}$ such that e is relatively prime to $\varphi(N)$. Compute d with $ed \equiv 1 \bmod \varphi(N)$. Let $pk := (N, e)$ and $sk := (N, d)$. Return (pk, sk).*
- Encryption: *$E(pk, m)$ with $pk = (N, e)$ and $m \in \{0, \dots, N-1)$ returns $m^e \bmod N$.*
- Decryption: *$D(sk, c)$ with $sk = (N, d)$ returns $c^d \bmod N$.*

Notice that textbook RSA is not well-suited as an encryption scheme on its own. For example, RSA is deterministic, and one can therefore recognize if the same message is encrypted twice. And, given a guess that the ciphertext c contains a message m, one can efficiently verify that guess. In particular, RSA is not IND-CPA secure. (See Definition 14 below.)

However, there are encryption schemes based on textbook RSA (which are sometimes misleadingly just called RSA) that improve on RSA and make it considerably more secure.

The security of textbook RSA cannot (according to the current state of knowledge) be reduced to some simpler assumption. It is, however, often used as an assumption on its own:

Definition 11 (RSA assumption) *For any polynomial-time algorithm A there exists a negligible function μ such that for all $\eta \in \mathbb{N}$, we have that*

$$\Pr\big[m = m' : p, q \xleftarrow{\$} Primes_\eta, N \leftarrow pq, m \xleftarrow{\$} \{0, \ldots, N-1\}, \\ e \xleftarrow{\$} \{0, \ldots, \varphi(N)-1\}, m' \leftarrow A(1^\eta, N, e, m^e \bmod N)\big] \leq \mu(\eta).$$

7.2 ElGamal encryption

Definition 12 (ElGamal encryption – generic group based description) *The ElGamal public key encryption scheme*[5] *is the encryption scheme (KG, E, D) which is defined as follows:*

- Key generation: *$KG(1^\eta)$ does the following: Pick the description of a group G (according to some distribution of groups which, when specified, makes this generic description concrete). Pick a generator g of G.*[6] *Pick $x \in \{0, \ldots, |G|-1\}$. Let $h := g^x$. Let $pk := (G, g, h)$ and $sk := (G, g, x)$.*
- Encryption: *$E(pk, m)$ with $pk = (G, g, h)$ and $m \in G$ picks $y \xleftarrow{\$} \{0, \ldots, |G|-1\}$ and returns $(g^y, m \cdot h^y)$.*
- Decryption: *$D(sk, c)$ with $sk = (G, g, x)$ and $c = (c_1, c_2)$ returns $c_2 \cdot (c_1^x)^{-1}$.*

The above can be made concrete by choosing G as follows: Let p be a prime of the form $p = 2q + 1$ where q is prime. Then G is the unique subgroup of $\mathbb{Z}_p^*$ of order (i.e., size) q. Or differently: $G := \{a : a = 1, \ldots, p-1, a^q \equiv 1 \bmod p\}$ where $(a, N) = 1$ means that the gcd of a and N is 1.

Definition 13 (ElGamal encryption – concrete instance) *The ElGamal public key encryption scheme*[7] *is the encryption scheme (KG, E, D) which is defined as follows:*

- Key generation: *$KG(1^\eta)$ does the following: Pick $p \xleftarrow{\$} Primes_\eta$ such that $p = 2q + 1$ for some prime q. Pick $g \in \{2, \ldots, p-1\}$ such that $g^q \equiv 1 \bmod p$. Pick $x \in \{0, \ldots, q-1\}$. Let $h := g^x \bmod p$. Let $pk := (p, q, g, h)$ and $sk := (p, q, g, x)$.*
- Encryption: *$E(pk, m)$ with $pk = (p, q, g, h)$ and $m \in \{1, \ldots, p-1\}$ and $m^q \equiv 1 \bmod p$: Pick $y \xleftarrow{\$} \{0, \ldots, q-1\}$ and return $(g^y \bmod p, m \cdot h^y \bmod p)$.*
- Decryption: *$D(sk, c)$ with $sk = (p, q, g, x)$ and $c = (c_1, c_2)$ returns $c_2 \cdot (c_1^x)^{-1} \bmod p$ where $(c_1^x)^{-1}$ denotes the integer satisfying $(c_1^x)^{-1} \cdot (c_1^x) \equiv 1 \bmod p$.*

We now proceed to state the security of ElGamal For this, we first need to state the definition of IND-CPA security for the case of public key encryption.

[5] Not to be confused with the unrelated ElGamal signature scheme.

[6] A generator of G is an element such that for every $a \in G$, there exists a $x \in \mathbb{Z}$ such that $a = g^x$.

[7] Not to be confused with the unrelated ElGamal signature scheme.

Definition 14 (IND-CPA (public key encryption)) *A public key encryption scheme (KG, E, D) consisting of a key-generation algorithm KG, an encryption algorithm E, and a decryption algorithm D is IND-CPA (indistinguishable under chosen plaintext attacks) if for any polynomial-time algorithm A there is a negligible function μ, such that for all $\eta \in \mathbb{N}$ we have that*

$$\big|\Pr[b' = b : (pk, sk) \leftarrow KG(1^\eta), b \xleftarrow{\$} \{0,1\}, (m_0, m_1) \leftarrow A(1^\eta, pk), \\ c \leftarrow E(pk, m_b), b' \leftarrow A(1^\eta, c)] - \tfrac{1}{2}\big| \leq \mu(\eta).$$

(Here we quantify only over algorithms A that output (m_0, m_1) with $|m_0| = |m_1|$.)

Notice that in contrast to Definition 9, the adversary gets the public key pk as input because pk is considered public information. Furthermore, we have not given the oracle $E(pk, \cdot)$ to A. Since A knows pk, he can compute anything himself that $E(pk, \cdot)$ would compute.

Next, we state the assumption under which the security of ElGamal is shown.

Again, there is a generic variant (parametrized over how we choose the group in which we work), and a concrete variant for integers modulo p.

Definition 15 (Decisional Diffie-Hellman (DDH) assumption – generic group based descriptio
Fix some algorithm GroupGen *that picks a cyclic group G.*

For any polynomial-time algorithm A there exists a negligible function μ such that for all $\eta \in \mathbb{N}$, we have that

$$\big|\Pr[b = b' : b \xleftarrow{\$} \{0,1\}, G \leftarrow \mathrm{GroupGen}(1^\eta), g \xleftarrow{\$} \{g : g \text{ is generator of } G\}, \\ x, y, z \xleftarrow{\$} \{0, \ldots, |G| - 1\}, b' \leftarrow A(1^\eta, p, g, g^x, g^y, (b = 0)?g^{xy} : g^z)] - \tfrac{1}{2}\big| \leq \mu(\eta).$$

Definition 16 (Decisional Diffie-Hellman (DDH) assumption – concrete instance) *For any polynomial-time algorithm A there exists a negligible function μ such that for all $\eta \in \mathbb{N}$, we have that*

$$\big|\Pr[b = b' : b \xleftarrow{\$} \{0,1\}, p \xleftarrow{\$} \mathit{SafePrimes}_\eta, q := (p-1)/2, g \xleftarrow{\$} \{g = 2, \ldots, p-1 : g^q \equiv 1 \bmod p\}, \\ x, y, z \xleftarrow{\$} \{0, \ldots, q-1\}, b' \leftarrow A(1^\eta, p, g, g^x \bmod p, g^y \bmod p, \\ (b = 0)?g^{xy} \bmod p : g^z \bmod p)] - \tfrac{1}{2}\big| \leq \mu(\eta).$$

Here $\mathit{SafePrimes}_\eta$ denotes the set of all primes p of length η such that $p = 2q + 1$ for some prime q.

Definition 17 (Decisional Diffie-Hellman (DDH) assumption) *For any polynomial-time algorithm A there exists a negligible function μ such that for all $\eta \in \mathbb{N}$, we have that*

$$\big|\Pr[b^* = 1 : p \xleftarrow{\$} \mathit{Primes}_\eta, g \xleftarrow{\$} \mathit{generators}_p, \\ x, y \xleftarrow{\$} \{0, \ldots, p-2\}, b^* \leftarrow A(1^\eta, p, g, g^x, g^y, g^{xy})] \\ - \Pr[b^* = 1 : p \xleftarrow{\$} \mathit{Primes}_\eta, g \xleftarrow{\$} \mathit{generators}_p, \\ x, y, z \xleftarrow{\$} \{0, \ldots, p-2\}, b^* \leftarrow A(1^\eta, p, g, g^x, g^y, g^z)]\big| \leq \mu(\eta).$$

(Here generators_p denotes the set of all generators modulo p.)

Theorem 7 *If the DDH assumption holds then the ElGamal encryption scheme is IND-CPA secure.*

7.3 Malleability and chosen ciphertext attacks

Malleability denotes the property of an encryption scheme that the adversary can change the content of a ciphertext without knowing the content. That is, upon reception of a ciphertext $c = E(pk, m)$, the adversary can produce a ciphertext $c' = E(pk, m')$ where $m' = f(m)$ and f is some function the adversary knows. (A precise definition of this is more complex because it also covers other dependencies between m and m', and possibly more than one input message.)

Note: Malleability denotes a weakness of an encryption scheme. To express that we have an encryption scheme where honest parties can transform $E(pk, m)$ into $E(pk, m')$, we speak of *homomorphic encryption.*

It is easy to see that both textbook RSA and ElGamal are malleable. Given an RSA ciphertext $c = E(pk, m)$ with $pk = (N, e)$, we can compute $c' = E(pk, am \bmod N)$ as $c' := a^e c \bmod N$. And given an ElGamal ciphertext $c = (c_1, c_2) = E(pk, m)$ with $pk = (p, g, h)$, we can compute $c' = E(pk, am \bmod p)$ as $c' = (c_1, ac_2 \bmod p)$.

But why is malleability a problem?

Auction example. Imaging an auction of the following kind: Two bidders send encrypted bids $c_1 := E(pk, bid_1)$ and $c_2 := E(pk, bid_2)$ to the auctioneer (pk is the public key of the auctioneer). The bids are encrypted in order to make sure that none of the bidders can make his bid depend on the bid of the other bidder. However, if E is malleable, this is not necessarily the case. Bidder 2 could, after seeing c_1, use malleability to produce a bid $c_2 = E(pk, 2 \cdot bid_1)$ and thus consistently overbid bidder 1. (This works both with textbook RSA and with ElGamal.)

Chosen ciphertext attack. Imagine the following protocol. A client wishes to send a command $cmd \in C := \{\texttt{cmd1}, \texttt{cmd2}\}$ to a server. The protocol we use is that the client sends $c := E(pk, cmd)$ to the server where pk is the public key of the server. The server decrypts c, and if the resulting plaintext is *not* in C, the server replies with `error`. We assume that `cmd1` and `cmd2` are encoded as integers so that we can use ElGamal to encrypt cmd. Our security requirement is that cmd stays secret from an adversary.[8]

To attack this protocol (when using ElGamal), the adversary can do the following: First, the adversary computes $a := \texttt{cmd1} \cdot \texttt{cmd2}^{-1}$ where $\texttt{cmd2}^{-1}$ is the integer with $\texttt{cmd2} \cdot \texttt{cmd2}^{-1} \equiv 1 \bmod p$. (Assuming $pk = (p, g, h)$.) Then, after intercepting $c = E(pk, cmd)$, using the malleability of ElGamal, he computes $c' = E(pk, a \cdot cmd \bmod p)$.

[8]With textbook RSA, this is obviously not the case, because to find out whether $c = E(pk, \texttt{cmd1})$, we just encrypt `cmd1` and compare the result with c. Since ElGamal is IND-CPA secure, this simple attack is not possible with ElGamal.

If $cmd = \texttt{cmd1}$, then $c' = E(pk, \texttt{cmd1}^2 \cdot \texttt{cmd2}^{-1})$. Since $\texttt{cmd1}^2 \cdot \texttt{cmd2}^{-1}$ is unlikely to be in C, the server responds with an error message. If $cmd = \texttt{cmd2}$, then $c' = E(pk, \texttt{cmd1} \cdot \texttt{cmd2}^{-1} \cdot \texttt{cmd2}) = E(pk, \texttt{cmd1})$. Hence the server does not reply with an error message. Thus, by observing whether the server sends an error message, the adversary learns the value of cmd.

Hence malleability can, in the case of a chosen ciphertext attack, lead to a loss of secrecy. Thus, at least in some situations, IND-CPA is too weak a security notion.

A stronger notion (that excludes malleability) is the following. It models that the scheme is secret even if the adversary can make a chosen ciphertext attack. More precisely, the adversary cannot learn anything about the content of a message c, even if he is allowed ask for decryptions of arbitrary messages (except a decryption of c itself, of course).

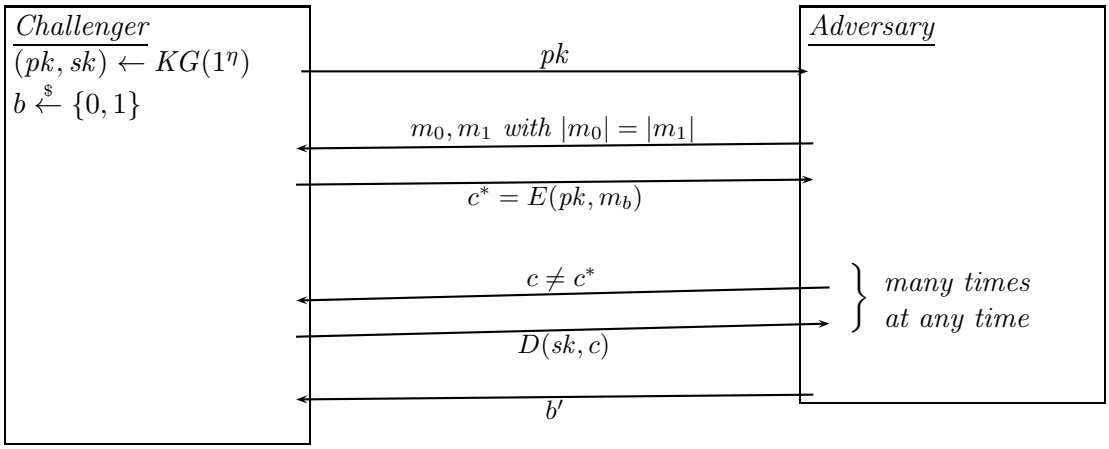
Definition 18 (IND-CCA (public key encryption)) *A public key encryption scheme (KG, E, D) consisting of a key-generation algorithm KG, an encryption algorithm E, and a decryption algorithm D is IND-CCA (indistinguishable under chosen ciphertext attacks) if for any polynomial-time algorithm A there is a negligible function μ, such that for all $\eta \in \mathbb{N}$ we have that*

$$\Big|\Pr[b' = b : (pk, sk) \leftarrow KG(1^\eta), b \xleftarrow{\$} \{0,1\}, (m_0, m_1) \leftarrow A^{D(sk,\cdot)}(1^\eta, pk), \\ c \leftarrow E(pk, m_b), b' \leftarrow A^{D(sk,\cdot)}(1^\eta, c)] - \tfrac{1}{2}\Big| \leq \mu(\eta).$$

Here we quantify only over algorithms A that have the following two properties:

- *A outputs (m_0, m_1) with $|m_0| = |m_1|$.*
- *In the second invocation of A, A only sends queries c' with $c' \neq c$ to $D(sk, \cdot)$.*

Visually, the game played by the attacker is the following:



Examples for IND-CCA secure encryption schemes are RSA-OAEP based on the RSA-assumption using the random oracle heuristic (the random oracle heuristic will be explained later in this lecture; see Section 12). And the Cramer-Shoup cryptosystem based on the DDH-assumption and the existence of collision-resistant hash functions (collision resistant hash functions will be explained later in this lecture, see Section 8).

7.4 Hybrid encryption

One disadvantage of the public key encryption schemes decrypted above is that they only allow to encrypt short messages (numbers modulo p and N, respectively). Of course, one could encrypt long messages by just picking a very large security parameter η (such that p and N are larger than the largest message we wish to encrypt), but this would be extremely inefficient. A somewhat better variant (which would, however, only give IND-CPA security even if the original scheme was IND-CCA secure) would be to split the message into many blocks and to encrypt each block individually.

There is, however, a better (more efficient and secure) solution, namely hybrid encryption. Hybrid encryption refers to the concept of using a symmetric encryption scheme (with a fresh key k) to encrypt the message, and a public key encryption scheme to encrypt the key k. The recipient can then get k by decrypting using his secret key, and using k he gets the message.

Definition 19 (Hybrid encryption) *Let (KG_1, E_1, D_1) be a symmetric encryption scheme. Let (KG_2, E_2, D_2) be a public key encryption scheme. Then we construct the* hybrid encryption scheme (KG, E, D) *as follows:*

- Key generation: $KG(1^\eta)$ *invokes* $(pk, sk) \leftarrow KG_2(1^\eta)$ *and returns* (pk, sk).
- Encryption: $E(pk, m)$ *does the following: Pick* $k \leftarrow KG_1(1^\eta)$. *Compute* $c_1 := E_1(k, m)$. *Compute* $c_2; = E_2(pk, k)$. *Return* $c := (c_1, c_2)$.
- Decryption: $D(sk, c)$ *with* $c = (c_1, c_2)$ *does the following: Compute* $k \leftarrow D_2(sk, c_2)$. *Compute* $m \leftarrow D_1(k, c_1)$. *Return* m.

Theorem 8 *If (KG_1, E_1, D_1) is IND-OT-CPA and (KG_2, E_2, D_2) is IND-CPA, then (KG, E, D) is IND-CPA.*

If (KG_1, E_1, D_1) is IND-CCA and (KG_2, E_2, D_2) is IND-CCA, then (KG, E, D) is IND-CCA.

The conditions on (KG_1, E_1, D_1) and (KG_2, E_2, D_2) can actually be weakened while still getting IND-CPA or IND-CCA security for (KG, E, D), respectively. In the context of hybrid encryption, one usually calls the public key scheme a *key encapsulation mechanism (KEM)* and the symmetric scheme a *data encapsulation mechanism (DEM)*.

8 Hash functions

A *hash function* $H : \{0,1\}^* \to \{0,1\}^n$ is a function that takes potentially long bitstrings as inputs and outputs a short (fixed length) bitstring. The intuition is that $H(x)$ is something like a "summary" of x that identifies x. There are many properties one expects from a good hash function. In a nutshell, a hash function should behave like a totally random function (and in particular operate in a totally "chaotic" way on its inputs). One particular property that a hash function should satisfy is the following:

Definition 20 (Collision resistance) *A function $H : M \to N$ (where H, M, and N may depend on the security parameter η) is* collision resistant *if for any polynomial-time algorithm A, there is a negligible function μ such that for all η, we have that*

$$\Pr[H(x) = H(x') \wedge x \neq x' : (x, x') \leftarrow A(1^\eta)] \leq \mu(\eta).$$

Typically, $M = \{0,1\}^*$ and $N = \{0,1\}^n$ in this definition.

This definition guarantees that $h := H(x)$ indeed identifies x: in any execution involving, the probability that more another x' with $h = H(x')$ occurs, is negligible.

8.1 Hash functions from compressions functions

Iterated hash. A basic construction for constructing hash functions is the following: Start with a so-called "compression function" $F : \{0,1\}^{n+t} \to \{0,1\}^n$. Then, to hash a message m, takes the start of m, compress it using F, add t more bits, compress again, etc. More formally:

Definition 21 (Iterated hash construction) *Let $F : \{0,1\}^{n+t} \to \{0,1\}^n$ be a function (the compression function). Let $iv \in \{0,1\}^n$ be a fixed initialization vector (that one is part of the definition of H_{IH}, is public, and usually described in the standard). Let $\{0,1\}^{*t}$ denote the set of all bitstrings whose length is a multiple of t. The* iterated hash *$H_{IH} : \{0,1\}^{*t} \to \{0,1\}^n$ is defined as follows: Let $m = m_1 \| \ldots \| m_k$ with each $m_i \in \{0,1\}^t$. Let $h_0 := iv$, and let $h_i := F(h_{i-1} \| m_i)$ for $i = 1, \ldots, k$. Then $H_{IH}(m) := h_k$.*

The iterated hash construction is, on its own, only useful for messages of fixed length. Namely, if F is collision resistant, we can show that H_{IH} does not have collisions in which both message have the same length:

Lemma 2 *Assume that F is collision-resistant. For any polynomial-time algorithm A, there is a negligible function μ such that for all η, we have that*

$$\Pr[H(x) = H(x') \wedge x \neq x' : (x, x') \leftarrow A(1^\eta)] \leq \mu(\eta).$$

when restricting A to output messages x, x' with $|x| = |x|'$.

However, H_{IH} is not necessarily collision resistant when allowing collisions between messages of different lengths: Assume we known some x^* with $F(iv \| x^*) = iv$ (this is, at least, not excluded by the assumption that F is collision resistant). Then $H_{IH}(x) = H_{IH}(x^* \| x)$, so x and $x^* \| x$ form a collision.

Merkle-Damgård. To avoid the collisions on messages of different lengths, one has to extend the iterated hash construction with some additional padding that makes sure that messages of different length are hashed differently. This can be done by putting the length of the message into the last block in a suitable way:

Definition 22 (Merkle-Damgård construction) *Fix a function* $F : \{0,1\}^{n+t} \to \{0,1\}^n$ *and an initialization vector* iv*. Fix an integer* $l \leq n$ *that depends on the security parameter* η *(such that* 2^l *is superpolynomial). We then define the* Merkle-Damgård construction $H_{MD} : \{0,1\}^* \to \{0,1\}^n$ *as follows: For a message* $m \in \{0,1\}^*$*, let* L *be the length of* m *encoded as an* l*-bit unsigned integer. Let* $m' := m\|1\|0^\alpha\|L$ *where* $\alpha \geq 0$ *is the smallest integer such that* $|m'|$ *is a multiple of* t*. Then* $H_{MD}(m) := H_{IH}(m')$*. (If* $|m| \geq 2^l$*,* $H_{MD}(x)$ *fails.)*

Notice that H_{MD} can hash messages of arbitrary length, not just of length a multiple of t.

Theorem 9 *If* F *is collision resistant, then* H_{MD} *is collision resistant.*

The Merkle-Damgård construction does not, however, "behave like a random function", see the discussion on page 24.

8.2 Constructing compression functions

It remains to answer how to construct a compression function in the first place. Two examples of such constructions are:

Davies-Meyer. Let E be a block cipher. Define the compression function F as $F(x\|y) := E(x,y) \oplus y$.

Miyaguchi-Preneel. Let E be a block cipher. Define the compression function F as $F(x\|y) := E(y,x) \oplus x \oplus y$. (Assuming the key length and message length of E are equal. Otherwise the construction needs to be slightly modified.)

Both constructions can be shown secure under the so-called ideal cipher heuristic which performs the proof under the (false) assumption that $E(k,\cdot)$ is an independently randomly chosen permutation for each k.

8.3 Birthday attacks

When attacking the collision resistance of a hash function (or the underlying compression function $F : \{0,1\}^{n+t} \to \{0,1\}^n$), a natural brute-force attack would be the following: Sample random pairs (x,y) and try whether $F(x) = F(y)$. On average, one needs 2^{n-1} tries to find a collision that way.

There is, however, a faster generic attack. Pick approximately $1.1774\sqrt{2^n}$ different values x_i. Compute $y_i := H(x_i)$. Sort the values y_i. When sorting, you will find out if there are two equal values $y_i = y_j$. If so, you have found a collision (x_i, x_j).

The probability of succeeding is about $\frac{1}{2}$ when picking $1.1774\sqrt{2^n}$ values. Thus this attack is considerably faster than a brute force attack, but it also uses much more memory.

9 Message authentication codes

A *message authentication code (MAC)* consists of three algorithms KG, MAC and *Verify*. A MAC is used to authenticate a message. If Alice and Bob share a key $k \leftarrow KG(1^\eta)$, then when Alice wants to send a message m, she computes a *tag* $t \leftarrow MAC(k, m)$ and sends both m and t to Bob. Bob verifies the tag by checking whether $\mathit{Verify}(k, m, t) = 1$. (If t was generated normally, this should always succeed.) Intuitively, security of a MAC requires that the adversary cannot produce a valid (i.e., passing verification) tag t for a message m that was not authenticated by Alice.

In most situations, *Verify* is defined by: $\mathit{Verify}(k, m, t) = 1$ iff $t = MAC(k, m) = t$. Thus, in the following, we will never specify *Verify* but just assume that it is thus defined. And unless mentioned otherwise KG always uniformly randomly picks a key k from a set K.

Definition 23 (Existential unforgeability) *We call a MAC $(KG, MAC, \mathit{Verify})$ existentially unforgeable (under chosen message attacks) (EF-CMA secure) iff for every polynomial-time algorithm A, there is a negligible function μ such that for all η we have that*

$$\Pr[\mathit{Verify}(k, m, t) = 1 \ \wedge \ m \textit{ fresh} : k \leftarrow KG(1^\eta), (m, t) \leftarrow A^{MAC(k,\cdot), \mathit{Verify}(k,\cdot)}(1^\eta)] \leq \mu(\eta).$$

Here "m fresh" means that m has never been sent by A to the $MAC(k, \cdot)$-oracle.

A naive construction. A naive construction of a MAC, suggested by the fact that a hash function should behave like a random function, is defined by $MAC(k, m) := H(k\|m)$ with keys k from a sufficiently large set K. Unfortunately, this construction is not secure, at least not if $H = H_{MD}$ is constructed using the Merkle-Damgård construction (Definition 22). Assume for simplicity that we use a compression function $F : \{0,1\}^{2n} \to n$ and keys $K = \{0,1\}^n$. Let $\mathit{pad}(m)$ denote the padding $1\|0^\alpha\|L$ that is added in the Merkle-Damgård construction. Notice that $\mathit{pad}(m)$ can be computed knowing only $|m|$.

Assume that the adversary intercepts a message m with corresponding tag $t = H_{MD}(k\|m)$. Then the adversary picks some x of his choice. Then let $m^* := m\|\mathit{pad}(k\|m)\|x$. By inspecting the Merkle-Damgård construction, one can see that $t^* := H_{MD}(k\|m^*)$ can be computed given only $t = H_{MD}(k\|m)$ and $|k|$ and $|m|$. Thus the adversary can forge a tag t^* for a message m^* that was not authenticated by an honest sender.

(Notice that this would not have happened if H would behave "like a random function". Thus this attack suggests a weakness of the Merkle-Damgård construction. However, the Merkle-Damgård construction is widely used, so it is preferable to construct MACs in a way that works with the Merkle-Damgård construction.)

HMAC. A construction of MACs from Hash functions that works with the Merkle-Damgård construction is the HMAC scheme.

Definition 24 (HMAC) *Let $F : \{0,1\}^{n+t} \to n$ be a compression function, and let $H := H_{MD}$ be the Merkle-Damgård construction based on F. Let $K := \{0,1\}^t$. Let $ipad \neq opad$ be two fixed n-bit strings.*

Then the HMAC MAC_{HMAC} is defined as follows: $MAC_{HMAC}(k,m) := H(k \oplus opad \| H(k \oplus ipad \| m))$.

Notice that although HMAC invoked H twice, the second (outer) invocation is performed on an input of length $t+n$ and thus is very fast even for long m.

To state the security of HMAC, we first need the following notion first:

Definition 25 (Pseudo-random function) *A function $f : K \times M \to N$ (where f, K, M, and N may depend on the security parameter η) is a* pseudo-random function (PRF) *if for all polynomial-time algorithms A, there is a negligible function μ such that for all $\eta \in \mathbb{N}$ we have that*

$$\begin{aligned} &\big| \Pr[b^* = 1 : k \xleftarrow{\$} K, b^* \leftarrow A^{f(k,\cdot)}(1^\eta)] \\ &- \Pr[b^* = 1 : \pi \xleftarrow{\$} \mathrm{Fun}_{M\to N}, b^* \leftarrow A^{\pi}(1^\eta)]\big| \leq \mu(\eta) \end{aligned}$$

Here $\mathrm{Fun}_{M\to N}$ denotes the set of all functions from M to N.

Compare this definition with that of a strong PRP (Definition 8). In fact, almost any strong PRP (and also any non-strong PRP) is a PRF:

Lemma 3 *Let $f : M \to M$ be a strong PRP. Assume that $|M|$ is superpolynomial. Then f is a PRF.*

The security of HMAC is given by the following theorem:

Theorem 10 *Assume that $F_1(x,y) := F(x\|y)$ and $F_2(y,x) := F(y\|x)$ are both PRFs. Then MAC_{HMAC} is EF-CMA secure.*

The intuitive reason why HMAC is secure (while the naive approach $H(k\|m)$ was not) is that in HMAC, we perform some "closing operation" after having hashed the message. This means that we when we extend the message, this introduces additional operations in the middle of the computation of $MAC_{HMAC}(k,m)$, thus the adversary cannot compute the new tag by just taking the old tag and applying some additional operations to it (as was the case with the naive approach).

CBC-MAC. Instead of trying to use a hash function for constructing a MAC (and relying on the properties on the Merkle-Damgård construction), we can also try and use a block cipher in a suitable mode of operation to produce a MAC. One example of such a construction is the CBC-MAC scheme (which is similar to the CBC mode of operation, Section 6.4).

Definition 26 (CBC-MAC) *Let $E : \{0,1\}^n \times \{0,1\}^l \to \{0,1\}^l$ be a function. Then $MAC_{CBC\text{-}MAC}$ is defined as follows: For a key $k \in \{0,1\}^n$ and a message $m = m_1 \dots m_q \in \{0,1\}^{*l}$, let $c_1 := E(k, m_1)$ and $c_i := E(k, c_{i-1} \oplus m_i)$ for $i \geq 2$. Let $MAC_{CBC\text{-}MAC}(k, m) := t := c_q$.*

For fixed length messages, CBC-MAC is a secure MAC:

Lemma 4 *Assume that 2^l is superpolynomial. Assume that E is a PRF. Then for every polynomial-time algorithm A, there is a negligible function μ such that for all η we have that*

$$\Pr[t = MAC_{CBC\text{-}MAC}(k, m) = t \ \wedge \ m \text{ fresh} : k \leftarrow KG(1^\eta), (m, t) \leftarrow A^{MAC_{CBC\text{-}MAC}(k,\cdot)}(1^\eta)] \leq \mu(\eta).$$

Here we restrict A such that all messages m' sent to the oracles and the message m that is finally output have the same length.

If we allow messages of different lengths, CBC-MAC is insecure: Assume the adversary intercepts some one-block message $m \in \{0,1\}^l$ and a corresponding tag $t = MAC_{CBC\text{-}MAC}(k, m)$. It is easy to verify that $MAC_{CBC\text{-}MAC}(k, m') = t$ for $m' := m \| (m \oplus t)$. Thus, the adversary has found a valid tag (namely t) for a message $m' \neq m$. Hence CBC-MAC is not EF-CMA secure.

DMAC. However, there are simple modifications of the CBC-MAC scheme that make it EF-CMA secure. One is the DMAC scheme. Here, we just apply an additional encryption at the end of the MAC-computation to make sure that message extension attacks do not work. (Compare with HMAC!)

Definition 27 (DMAC) *Let $E : \{0,1\}^n \times \{0,1\}^l \to \{0,1\}^l$ be a function. Then MAC_{DMAC} is defined as follows: For a key $k = k_1 k_2 \in \{0,1\}^{2n}$ and a message $m = m_1 \dots m_q \in \{0,1\}^{*l}$, let $c_1 := E(k_1, m_1)$ and $c_i := E(k_1, c_{i-1} \oplus m_i)$ for $i \geq 2$. Let $MAC_{CBC\text{-}MAC}(k, m) := t := E(k_2, c_q)$.*

Notice that DMAC uses a key of twice the length because the final encryption uses another key.

Theorem 11 *If 2^l is superpolynomial and E is a PRF, then MAC_{DMAC} is EF-CMA secure.*

A direct construction. We can also directly combine a hash function and a PRF to get a MAC due to the following two lemmas:

Lemma 5 *If $E : K \times M \to M$ is a PRF, then E is an EF-CMA secure MAC.*

Notice that if E is, e.g., a block cipher, then E is not a very useful MAC because M consists only of one block messages.

Lemma 6 *If $MAC_1 : K \times M \to N$ is a PRF, and if $H : \{0,1\}^* \to M$ is collision resistant, then $MAC_2 : K \times \{0,1\}^* \to M$, $MAC_2(k,m) := MAC_1(k, H(m))$ is EF-CMA secure.*

Thus the construction $MAC(k,m) := E(k, H(m))$ gives us a EF-CMA secure MAC if E is a PRF and H collision resistant.

10 Cryptanalysis

Cryptanalysis is the science of how to break ciphers. More precisely, it is the science of recovering messages without access to the secret (e.g., the key).

Besides the attack scenarios described on Section 2 (ciphertext only, known plaintext, chosen ciphertext), there are the following attack scenarios:

- *Adaptive chosen-ciphertext:* We are allowed to chose the ciphertexts depending on decryptions we saw in earlier steps.
- *Related key:* Several messages are encrypted with different but not independent keys. (E.g., the first half of the key is the same for two encryptions.)
- *Side channel:* Use additional leakage of information to break a cryptosystem (e.g., the power consumption, the response time, etc. of the encryption/decryption operation).

Why study cryptanalysis? There are several reasons for studying cryptanalysis. First, cryptanalysis can help to design better cryptosystems: By studying weaknesses of ciphers, we find out what we have to do better. Second, cryptanalysis is a science that is relevant in war: For example, breaking the Enigma in World War II had a major impact on the war outcome. Third, breaking ciphers can be important for catching criminals or terrorists.

Lessons learned from breaking Enigma. The Enigma will never encrypt a letter to itself. This property means that we will learn something about the plaintext by looking at the ciphertext. Also, we cannot expect that the attacker does not know what algorithm we use: The attacker might (through stealth or force) have stolen our algorithm / implementation. Also, assuming that the adversary has to do a ciphertext only attack is not a good idea: There is usually at least some side information about the message, e.g., because it is a weather report, because we use fixed message format, etc.

Confusion and diffusion. See page 12.

Cache timing attacks. A cache timing attack is a side-channel attack in which the attacker's process run on the same machine as the honest user's process. (Think of a multi-user system.) We consider a very simplified view on AES: In a typical AES implementation, the following happens: In the first round, some part m of the message is XORed with some part k of the key. Then, some function F is applied to $m \oplus k$. For efficiency reasons, F is implemented as a lookup-table. I.e., the user will lookup $F[m \oplus k]$ in a table F.

Now consider the following attack: The attacker fills the cache with his own data (by running some loop accessing a lot of memory locations). Then the user encrypts some data. This will access $F[m \oplus k]$. Thus, the cache location corresponding to $F[m \oplus k]$ will have been accessed. (Actually, a lot of locations will have been accessed, since the operation F is applied to many parts of message and key, but we ignore this complication in this highly simplified exposition.) After the user encrypted, the adversary accesses his own data again. By measuring response times, he finds out which cache positions were overwritten by the user's algorithm, hence he knows $m \oplus k$. Assuming a known plaintext attack, the attack can compute k from this, hence he obtained a part k of the key.

Real-life cache timing attacks are much more involved because measurements about which cache positions are overwritten are very noisy, and not just a single cache position is overwritten, so we do not know which cache position corresponds to which value in the encryption algorithm.

10.1 Linear cryptanalysis

We will now illustrate the concept of *linear cryptanalysis* on a toy block cipher. The operation of the block cipher is depicted in Figure 1. We assume that all S-boxes implement the same function, described in Figure 2.

In the description of DES (Section 6.1) we have seen one common structure of block ciphers, the Feistel network. Here we see another structure, the *substitution-permutation network*. In a substitution-permutation network, we alternatingly permute the bits, apply substitutions to them, and mix in the key (i.e., XOR parts of the key to the bits). We do this for a number of rounds (usually, more rounds mean more security but less efficiency). A well-known example of a cipher using a substitution-permutation network is AES.

Looking at the structure of the toy cipher, the following question arises: Why do we have to mix in the key also after the last round? The answer is simple: Since each of the S-boxes is invertible, without the last mix-key-layer, we could just invert the S-box, and would effectively have lost a whole round.

Another question: In the toy cipher, we have that the key length is much bigger than the block length because we need some part of the key for each round. Is this necessary? The answer is no, we can use a smaller key by using a key schedule (as in the case of DES) that produces round keys out of a key that is shorter than the total length of the round keys.

In the following, we will often use the following lemma:

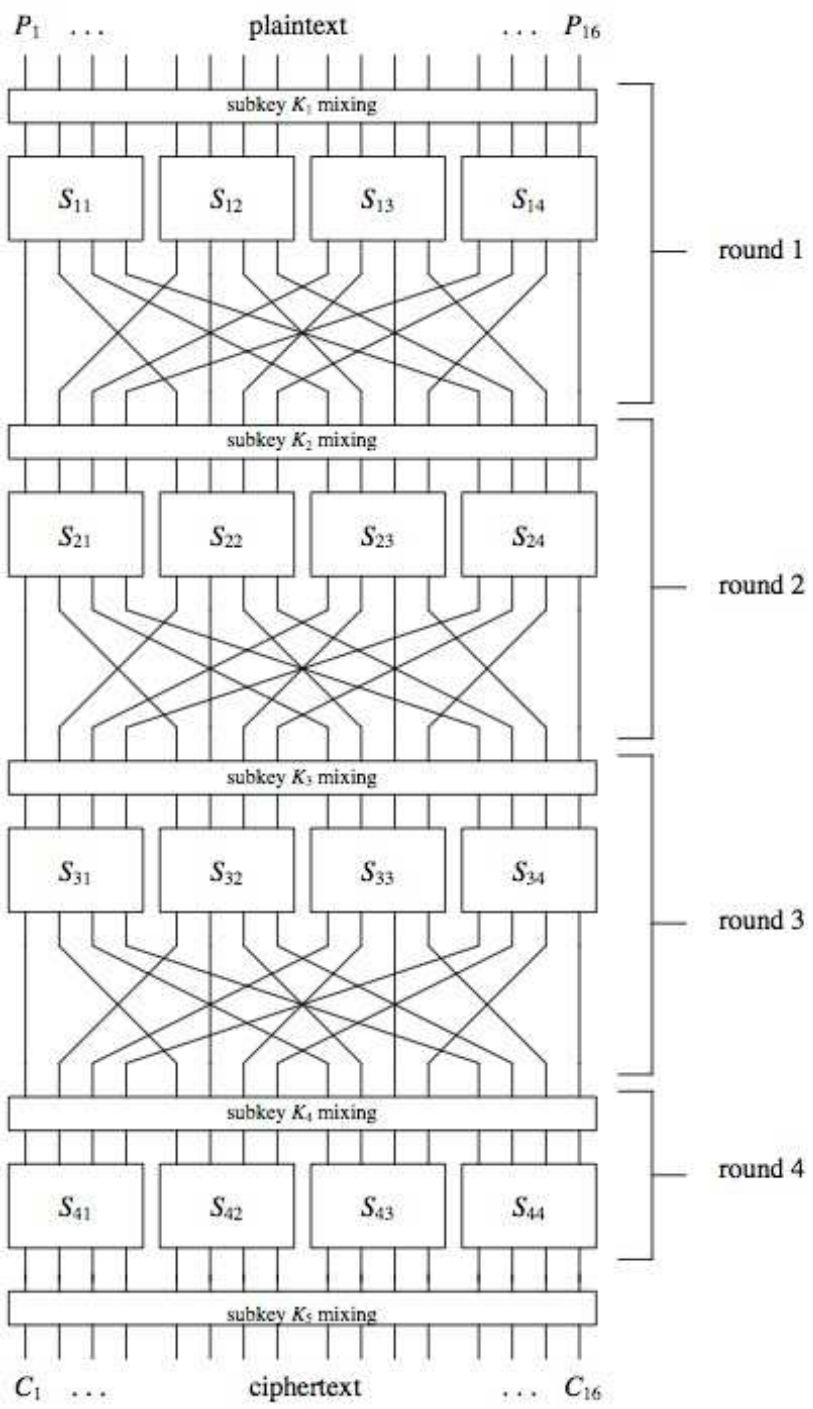


Figure 1: Toy cipher with block length 16 bit and key length 80 bits

input	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
output	E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7

Figure 2: The substitution implemented by the S-boxes

		Output Sum															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Input Sum	0	+8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	0	−2	−2	0	0	−2	+6	+2	+2	0	0	+2	+2	0	0
	2	0	0	−2	−2	0	0	−2	−2	0	0	+2	+2	0	0	−6	+2
	3	0	0	0	0	0	0	0	0	+2	−6	−2	−2	+2	+2	−2	−2
	4	0	+2	0	−2	−2	−4	−2	0	0	−2	0	+2	+2	−4	+2	0
	5	0	−2	−2	0	−2	0	+4	+2	−2	0	−4	+2	0	−2	−2	0
	6	0	+2	−2	+4	+2	0	0	+2	0	−2	+2	+4	−2	0	0	−2
	7	0	−2	0	+2	+2	−4	+2	0	−2	0	+2	0	+4	+2	0	+2
	8	0	0	0	0	0	0	0	0	−2	+2	+2	−2	+2	−2	−2	−6
	9	0	0	−2	−2	0	0	−2	−2	−4	0	−2	+2	0	+4	+2	−2
	A	0	+4	−2	+2	−4	0	+2	−2	+2	+2	0	0	+2	+2	0	0
	B	0	+4	0	−4	+4	0	+4	0	0	0	0	0	0	0	0	0
	C	0	−2	+4	−2	−2	0	+2	0	+2	0	+2	+4	0	+2	0	−2
	D	0	+2	+2	0	−2	+4	0	+2	−4	−2	+2	0	+2	0	0	+2
	E	0	+2	+2	0	−2	−4	0	+2	−2	0	0	−2	−4	+2	−2	0
	F	0	−2	−4	−2	−2	0	+2	0	0	−2	+4	−2	−2	0	+2	0

Table 4. Linear Approximation Table

Figure 3: Linear approximation table for the substitution in Figure 2

Lemma 7 (Piling-up lemma) *Let $X_1, \ldots, X_n$ be independent binary random variables with $\Pr[X_i = 0] = \frac{1}{2} + \varepsilon_i$ for all i. Then $\Pr[X_1 \oplus \ldots \oplus X_n = 0] = \frac{1}{2} + 2^{n-1} \prod_i \varepsilon_i$.*

Consider the substitution given in Figure 2. Assume that we feed it with input $X_1 \ldots X_4 \in \{0,1\}^4$. Then we get some output $Y_1 \ldots Y_4$. We can now, for example, compute the probability that $X_2 \oplus X_3 \oplus Y_1 \oplus Y_3 \oplus Y_4 = 0$ by counting for how many values of $X_1X_2X_3X_4$ we have $X_2 \oplus X_3 \oplus Y_1 \oplus Y_3 \oplus Y_4 = 0$. We get

$$\Pr[X_2 \oplus X_3 \oplus Y_1 \oplus Y_3 \oplus Y_4 = 0] = \frac{12}{16} = \frac{1}{2} + \frac{1}{4}. \tag{1}$$

In Figure 3, we have summarized the values of all these probabilities. This table uses the following compact encoding: If we are interested in the XOR of some input bits X_i and some output bits Y_i, we first write down a binary number which tells input bits we are interested in. E.g. for X_1, X_3, we write 1010. This number we convert to hexadecimal, giving A, this is called the *input sum*. We do the same for the Y_i, giving

the *output sum*. Then we look up in the table the corresponding value x. Then the probability that the XOR of the X_i and Y_i we are interested in is $\frac{1}{2} + \frac{x}{16}$.

For example, we can find (1) in this table: $X_2 \oplus X_3$ is encoded as 0110 which is 6 in hex. And $Y_1 \oplus Y_3 \oplus Y_4$ is encoded as 1011 which is B in hex. In the table, we find that for input sum 6 and output sum B, we have the entry $+4$, which means $\Pr[X_2 \oplus X_3 \oplus Y_1 \oplus Y_3 \oplus Y_4 = 0] = \frac{1}{2} + \frac{+4}{16}$ which is what (1) states.

Using the statistics from the linear approximation table, we can now find what is called a linear trail.

In the following, we name the bits that occur in an encryption using the toy cipher as follows: The i-th plaintext bit is called P_i. The i-th bit of the r-th round key is $k_{r,i}$. The i-th bit of the inputs to the S-boxes in round r is $U_{r,i}$, the output $V_{r,i}$.

We will now try to determine the bias of the XOR of some of the input bits, the key bits, and the inputs of the S-boxes in the last round. That is, for some suitable sets I_k, I_P, I_V, we wish to know ε such that

$$\Pr[\bigoplus_{(r,i)\in I_k} k_{r,i} \oplus \bigoplus_{i\in I_P} P_i \oplus \bigoplus_{i\in I_V} V_{4,i} = 0] = \frac{1}{2} + \varepsilon.$$

We are interested in sets I_k, I_P, I_V such that ε becomes as big as possible.

Assume we know such an ε. We can now perform a known plaintext attack on the toy cipher as follows:

- Guess the value for the last round key. I.e., guess values $k_{5,i}$ for some i.
- Using those values $k_{5,i}$ and a lot of ciphertexts $C^{(j)}$, we can compute the values $V_{4,i}^{(j)}$ for all $i \in I_V$.
- Given also the corresponding plaintexts $P^{(j)}$, we can compute $\bigoplus_{i\in I_P} P_i^{(j)} \oplus \bigoplus_{i\in I_V} V_{4,i}^{(j)}$ for many values j. Assuming we have guessed the values $k_{5,i}$ correctly, we will get a bias of either approximately ε (if $\bigoplus_{(r,i)\in I_k} k_{r,i} = 0$) or approximately $-\varepsilon$ (if $\bigoplus_{(r,i)\in I_k} k_{r,i} = 1$).

Thus, if we guessed $k_{5,i}$ correctly, we expect to have a bias $\pm\varepsilon$. For wrong $k_{5,i}$, we expect that the bias will be much smaller (because we have chosen I_k, I_P, I_V such that ε is big). Thus, we can identify a correct guess of $k_{5,i}$ by making a suitable statistics on plaintext/ciphertext pairs $(P^{(j)}, C^{(j)})$. This will leak some of the key bits which is already an attack. By performing this attack for more than one choice of I_k, I_P, I_V, we can learn even more bits.

Note that for computing the statistics to a sufficient precision, we need about $1/\varepsilon^2$ plaintext/ciphertext pairs. And the time-complexity of the attack is in the order of $2^m/\varepsilon^2$ where m is the number of keybits $k_{5,i}$ we have to guess.

But how to we compute the bias ε for given sets I_k, I_P, I_V?

First, we have to choose some bits in each round of the cipher. These are the bits upon which we wish to base our statistics. In Figure 4, we have chosen the input bits P_5, P_7, P_8.

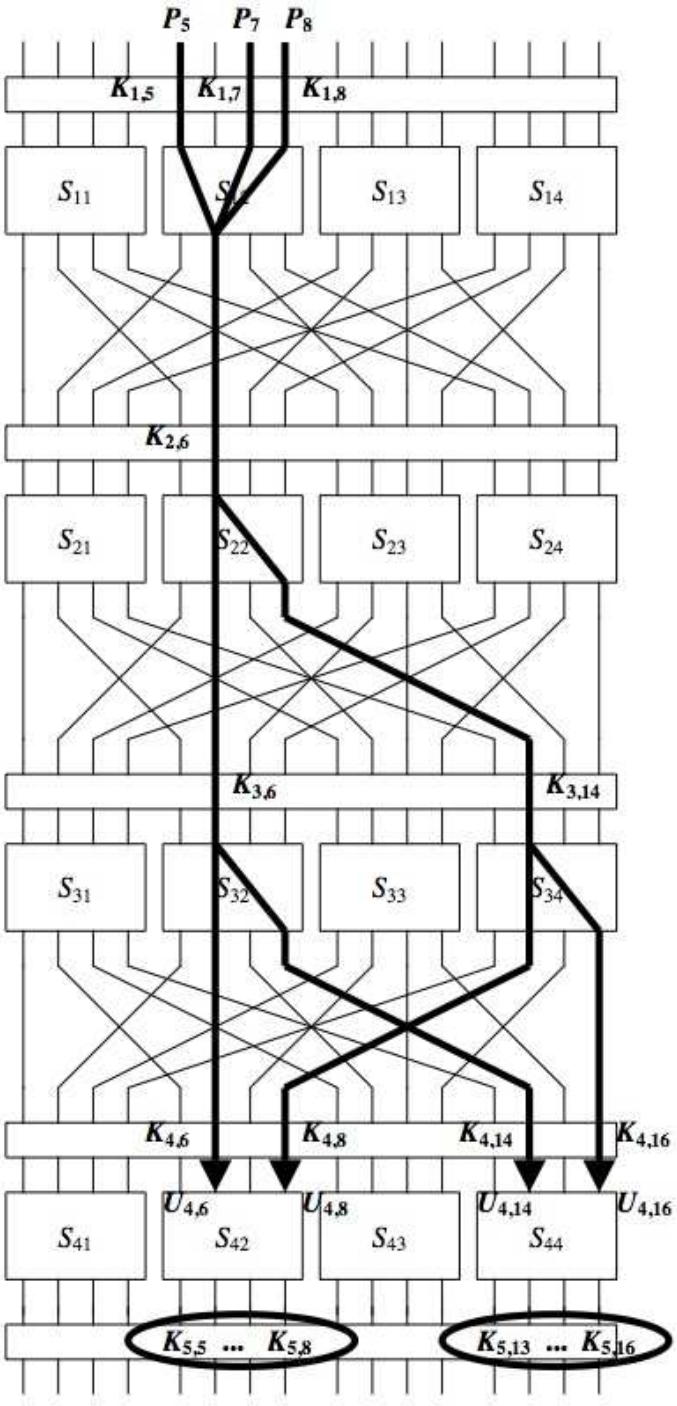


Figure 4: Example linear trail of the toy cipher

From these, the bits $U_{1,5}, U_{1,7}, U_{1,8}$ depend directly. Then we choose the output bit $V_{1,6}$. Upon $V_{1,6}$, the bit $U_{2,6}$ depends. The bit $U_{2,6}$ goes into the S-box S_{22}. Of that S-box, we choose the output bits $V_{2,6}$ and $V_{2,8}$. Upon these, the input bits $U_{3,6}$ and $U_{3,14}$ will depend. These go into S-boxes S_{32} and S_{34}. We pick outputs $V_{3,6}, V_{3,8}, V_{3,14}, V_{3,16}$. And the inputs $U_{4,6}, U_{4,14}, U_{4,8}, U_{4,16}$ depend directly on these.

From the linear approximation table, we then expect the following dependencies between the inputs and outputs of the different S-boxes:

$$\begin{aligned}
S_{12}: &\quad U_{1,5} \oplus U_{1,7} \oplus U_{1,8} \oplus V_{1,6} = 0 && \text{with prob. } \frac{1}{2} + \frac{1}{4} \\
S_{22}: &\quad U_{2,6} \oplus V_{2,6} \oplus V_{2,8} = 0 && \text{with prob. } \frac{1}{2} - \frac{1}{4} \\
S_{32}: &\quad U_{3,6} \oplus V_{3,6} \oplus V_{3,8} = 0 && \text{with prob. } \frac{1}{2} - \frac{1}{4} \\
S_{34}: &\quad U_{3,14} \oplus V_{3,16} \oplus V_{3,16} = 0 && \text{with prob. } \frac{1}{2} - \frac{1}{4}
\end{aligned}$$

Notice that each $U_{r,i}$ is of the form $k_{r,i} \oplus V_{r-1,j}$ for some (known) j. (Or $k_{1,i} \oplus P_i$ for $r = 1$.) Using these equalities, we get:

$$\begin{aligned}
S_{12}: &\quad B_1 := P_5 \oplus k_{1,5} \oplus P_7 \oplus k_{1,7} \oplus P_8 \oplus k_{1,8} \oplus V_{1,6} = 0 && \text{with prob. } \frac{1}{2} + \frac{1}{4} \\
S_{22}: &\quad B_2 := V_{1,6} \oplus k_{2,6} \oplus V_{2,6} \oplus V_{2,8} = 0 && \text{with prob. } \frac{1}{2} - \frac{1}{4} \\
S_{32}: &\quad B_3 := V_{2,6} \oplus k_{3,6} \oplus V_{3,6} \oplus V_{3,8} = 0 && \text{with prob. } \frac{1}{2} - \frac{1}{4} \\
S_{34}: &\quad B_4 := V_{2,14} \oplus k_{3,14} \oplus V_{3,16} \oplus V_{3,16} = 0 && \text{with prob. } \frac{1}{2} - \frac{1}{4}
\end{aligned}$$

Using the piling up lemma, and the (heuristic and not really true) assumption that everything is independent, we get $\Pr[B_1 \oplus B_2 \oplus B_3 \oplus B_4 = 0] = \frac{1}{2} - \frac{1}{32}$.

And since

$$B_1 \oplus B_2 \oplus B_3 \oplus B_4 = V_{3,6} \oplus V_{3,8} \oplus V_{3,14} \oplus V_{3,16} \oplus P_5 \oplus P_7 \oplus P_8 \oplus k_{1,5} \oplus k_{1,7} \oplus k_{1,8} \oplus k_{2,6} \oplus k_{3,6} \oplus k_{3,14}$$

we have found an equation of the form of (1). This allows to perform the attack.

Further reading. A tutorial on cryptanalysis (covering the above and differential cryptanalysis) [Hey].

11 One-way functions

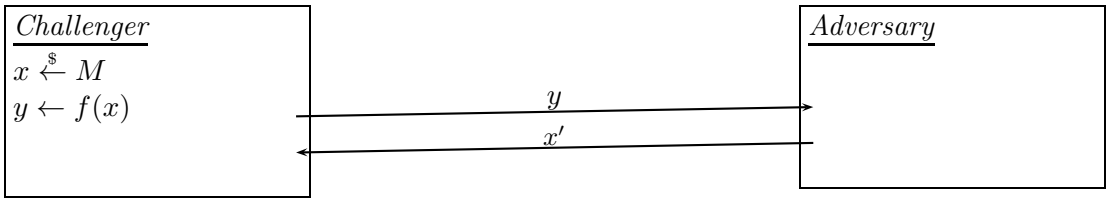
One of the weakest assumptions in cryptography is the existence of so-called one-way functions. Roughly, a one-way function is a function that is easy to compute, but hard to invert. (I.e., given an image, it is hard to find any preimage of that image.)

Formally, one-way functions are defined as follows:

Definition 28 (One-way function) *A function $f : M \to N$ (where f, M, N may depend on the security paramter η) is* one-way *iff it can be computed in deterministic polynomial-time and for all polynomial-time algorithms A, there exists a negligible function μ such that for all η, we have:*

$$\Pr[f(x) = f(x') : x \xleftarrow{\$} M, y \leftarrow f(x), x' \leftarrow A(1^\eta, y)] \leq \mu(\eta)$$

Visually, the game played by the attacker is the following:



Although a one-way function seems to be very weak and rather useless, using only one-way functions, quite a lot of cryptographic primitives can be constructed:

Theorem 12 *Assume that one-way functions exist. Then there are:*

(a) Pseudo-random generators

(b) Pseudo-random functions

(c) Strong pseudo-random permutations

(d) Symmetric encryption schemes (IND-CCA secure)

(e) MACs (EF-CMA secure)

(a) is shown using a generalization of the Blum-Blum-Shub construction (Construction 2). (b) by building pseudo-random functions from pseudo-random generators using a tree-based construction. (c) by using a 4-round Feistel network with a PRF as round function. (d) by using a strong PRP as a block cipher in a suitable mode of operation. (e) by using a PRF as a MAC.

Also, we can construct signature schemes from one-way functions, see Section 13.1.

What is not known is whether it is possible to construct collision-resistant hash functions or public-key encryption schemes from one-way functions.

12 Random Oracle Model

In many cases, cryptosystems that use hash functions are difficult or impossible to prove based only on simple assumptions about the hash function (such as collision-resistance). Instead, one would like to use the fact that a typical hash function behaves very much like a totally random function.

This leads to the following analysis technique: When analyzing a cryptosystem that uses a hash function $H : M \to N$, one models the hash function as a uniformly randomly chosen function out of the space of all functions from M to N. Such a uniformly randomly chosen function is called a *random oracle*.

More precisely, when analyzing whether a scheme X (that uses a hash function $H : M \to N$) satisfies a given security definition S, one does the following: First, one writes X as a collection of algorithms that use an oracle H (instead of directly using the hash function), and one changes the security definition S such that at the beginning of the execution, H is chosen randomly ($H \xleftarrow{\$} \mathrm{Fun}_{M\to N}$), and then one changes S further by giving oracle access to H to all algorithms (in particular the adversary).

To illustrate this, consider the definition of one-way functions. If we build a one-way function f using a hash function (e.g., $f(x) := H(x) \oplus x$), we would represent f as an algorithm that expects an oracle H and computes $f^H(x) := H(x) \oplus x$. The security of f would then be expressed by the following definition.

Definition 29 (One-way function (in the random-oracle model)) *A function $f^H : M \to N$ using an oracle $H : M' \to N'$ (where f, M, N, M', N' may depend on the security paramter η) is* one-way *iff it can be computed in deterministic polynomial-time (given oracle access to H) and for all polynomial-time oracle algorithms A, there exists a negligible function μ such that for all η, we have:*

$$\Pr[f^H(x) = f^H(x') : H \xleftarrow{\$} \mathrm{Fun}_{M'\to N'}, x \xleftarrow{\$} M, y \leftarrow f(x), x' \leftarrow A^H(1^\eta, y)] \leq \mu(\eta)$$

See also Definition 32 in the next section for a more complex example.

Schemes in the random oracle model are usually much simpler to analyze than schemes using real hash functions. However, random oracles do not exist in real life. So, an analysis of some cryptosystem in the random-oracle model has, strictly speaking, no impact on any real-life scheme.

One does, however, commonly use the following heuristic when analyzing schemes in the random-oracle model:

> **Random oracle heuristic.** In order to "prove" the security of a scheme X that uses a hash function H, first analyze X in the random oracle model (i.e., replacing H by a random oracle). Then, if X is secure in the random oracle model, conclude that X is secure using a suitably good real-life hash function (such as, e.g., SHA-256).[9]

Note that the random oracle model itself is a well-defined, mathematically rigorous model (it just describes something that does not exist, but that does not make it informal), the random oracle heuristic is not fully well-defined, and it is more of a guideline for heuristically analyzing the security of protocols.

[9] A better variant of the random oracle heuristic would actually be the following: If H is implemented as, e.g., Merkle-Damgård with a compression function F, model F as a random oracle (and treat H as a Merkle-Damgård construction based on this oracle F). However, this is often not done.

Unsoundness of the random oracle heuristic. Moreover, the random oracle heuristic can, in general, lead to wrong results. Consider the following protocol: Let $H : \{0,1\}^* \to \{0,1\}^\eta$ be a hash function. Bob picks a secret $s \xleftarrow{\$} \{0,1\}^\eta$. Then Bob expects the description of a Boolean circuit C from the adversary. Then Bob picks $x \xleftarrow{\$} \{0,1\}^\eta$ and checks whether $C(x) = H(x)$. If so, Bob sends s to the adversary.

In the random oracle model, this protocol is secure (i.e., the adversary cannot guess s. This is – roughly – due to the fact that a random function $H \xleftarrow{\$} \mathrm{Fun}_{\{0,1\}^* \to \{0,1\}^\eta}$ will not have a short description, and thus C will not describe H (or even be close to it). Thus $C(x) = H(x)$ only with negligible probability.

If H is a hash function, however, there is a circuit C_H that describes this hash function. The adversary just sends $C := C_H$ and receives s from Bob. Thus, the scheme is insecure for *any* hash function, even though it is secure in the random oracle model!

Relevance of the random oracle heuristic. Even though the random oracle heuristic is, in general, unsound, it is of practical importance nonetheless.

- So far, only very contrived examples of cryptosystems have been found where the random oracle heuristic fails.
- Schemes in the random oracle model are typically simpler, more efficient, and easier to prove that schemes in the standard model (i.e., the model without random-oracle).
- It is better to use a well-known and well-tested heuristic in the security proof than not to prove security at all. Since a too inefficient scheme will typically not be adopted on a wide basis, it may be preferable to propose a scheme in the random oracle model than to propose a considerably more inefficient scheme (in the latter case, a totally unproven scheme might be adopted instead).

Nevertheless, the use of the random oracle heuristic is a matter of taste and opinion.

Lazy sampling. An important technique when analyzing protocols in the random oracle model uses the following fact:

Lemma 8 (Lazy sampling) *Fix M, N. Let H_{lazy} denote the following (stateful) algorithm: H_{lazy} maintains a table each $m \in M$ to a value $h_m \in N \cup \{\bot\}$; initially $h_m = \bot$ for all m. When invoked as $H_{lazy}(m)$ for $m \in M$, H_{lazy} first checks whether $h_m \neq \bot$. In this case, it returns h_m. Otherwise, it chooses $h_m \xleftarrow{\$} N$ and returns h_m.*

Then H_{lazy} is indistinguishable from a random $H \xleftarrow{\$} \mathrm{Fun}_{M \to N}$. More formally: For any oracle algorithm A (not necessarily polynomial-time) and all η, we have

$$\Pr[b^* = 1 : H \xleftarrow{\$} \mathrm{Fun}_{M \to N}, b^* \leftarrow A^H(1^\eta)] = \Pr[b^* = 1 : b^* \leftarrow A^{H_{lazy}}(1^\eta)].$$

This fact is usually used in a security proof by first replacing the random oracle by H_{lazy}, and then continuing the proof in that setting. The advantage of this approach is that when using H_{lazy}, it is typically easier to track the dependencies of the values $H(m)$, and to make slight changes to the $H(m)$ in subsequent proof steps.

13 Signature schemes

A *signature scheme* consists of three algorithms KG, $Sign$ and $Verify$. A signature is used to authenticate messages. To be able to authenticated messages, Alice generates a key pair $(pk, sk) \leftarrow KG(1^\eta)$ and publicizes the public key pk (but keeps the secret key sk secret).[10] Then, when Alice wants to send a message m, she computes a *signature* $\sigma \leftarrow Sign(sk, m)$ and sends both m and σ to Bob. Bob verifies the signature by checking whether $Verify(pk, m, \sigma) = 1$. (If σ was generated normally, this should always succeed.) Intuitively, security of a signature scheme requires that the adversary cannot produce a valid (i.e., passing verification) signature σ for a message m that was not signed by Alice.

That is, signatures schemes are the public-key variant of MACs. Accordingly, the security definition for signature schemes closely resembles the one for MACs (Definition 23):

Definition 30 (Existential unforgeability) *We call a signature scheme* $(KG, Sign, Verify)$ existentially unforgeable (under chosen message attacks) *(EF-CMA secure) iff for every polynomial-time algorithm A, there is a negligible function μ such that for all η we have that*

$$\Pr[Verify(pk, m, \sigma) = 1 \ \wedge \ m \text{ fresh} : (pk, sk) \leftarrow KG(1^\eta), (m, \sigma) \leftarrow A^{Sign(sk,\cdot)}(1^\eta, pk)] \leq \mu(\eta).$$

Here "m fresh" means that m has never been sent by A to the $Sign(sk, \cdot)$-oracle.

Notice that A now gets access to the public key pk (since that one is not kept secret). Furthermore, in contrast to Definition 23, the adversary A does not get access to a $Verify(pk, \cdot)$-oracle: since A knows pk, he can simulate that oracle himself.

Naive construction of signature schemes – a warning. One sometimes sees descriptions of the concept of signatures that suggest the following: Take a public-key encryption scheme (KG, Enc, Dec) and use it as a signature scheme as follows: Produce keys by $(pk, sk) \leftarrow KG(1^\eta)$. To sign a message m, run $\sigma \leftarrow Dec(sk, m)$. To verify a signature σ on m, check whether $m = Enc(sk, m)$. This approach is a bad idea due to the following reasons:

- In general, it does not even work: Since ciphertexts in public-key encryption schemes usually have some structure (e.g., an ElGamal ciphertext is a pair of integers), many messages (e.g., those that are not pairs) will simply be rejected by Dec. Even if we encode messages to have the right structure, there may be some internal checksums or similar inside a ciphertext that make Dec fail whenever the ciphertext is not produced by encrypting a message. (Most IND-CCA secure public-key encryption schemes have such "checksums".)
- If the encryption scheme is randomized, Enc will produce a different ciphertext each time it is invoked. Thus $Enc(pk, Dec(sk, m)) \neq m$ most of the time.

[10] Publicizing pk could, analogously to the case with encryption, be done using a PKI.

- And even if the above problems are resolved (e.g., textbook RSA does not have the two problems above), the security of the resulting scheme is still doubtful. An encryption scheme is not designed to be a signature scheme when reversed, and the security of the encryption scheme therefore implies in no way security of the result signature scheme.

I presume that the reason for the persistence of this approach is that it can be done (with very weak security!) using RSA which was the first known public-key cryptosystem.

13.1 Signatures from one-way functions.

We now describe how to construct a signature scheme based on very weak assumptions. We will only need one-way functions and collision-resistant hash functions. (And below we mention how to do it only with one-way functions.)

One-time signatures. The first step is to construct so-called one-time signatures. These are signature schemes that are secure only when the honest signer produces at most one signature. More precisely, a one-time signature scheme should satisfy the following security definition:

Definition 31 (Existential one-time unforgeability) *We call a signature scheme* $(KG, Sign, Verify)$ existentially one-time unforgeable (under chosen message attacks) *(EF-OT-CMA secure) iff for every polynomial-time algorithm* A that calls its oracle at most once*, there is a negligible function* μ *such that for all* η *we have that*

$$\Pr[Verify(pk, m, \sigma) = 1 \ \wedge \ m \text{ fresh} : (pk, sk) \leftarrow KG(1^\eta), (m, \sigma) \leftarrow A^{Sign(sk,\cdot)}(1^\eta, pk)] \leq \mu(\eta).$$

Here "m fresh" means that m *has never been sent by* A *to the* $Sign(sk, \cdot)$*-oracle.*

We construct the following scheme:

Construction 4 (Lamport's one-time signatures) *Let* f *be a one-way function. We define a signature scheme* $(KG_{Lamport}, Sign_{Lamport}, Verify_{Lamport})$ *with message space* $\{0,1\}^\eta$ *as follows:*

- ***Key generation:*** $KG_{Lamport}(1^\eta)$ *picks values* $x_i^j \xleftarrow{\$} \{0,1\}^\eta$ *for* $i = 1, \ldots, \eta$*,* $j = 0, 1$*. Let* $y_i^j := f(x_i^j)$*. Let* $sk := (x_1^0, \ldots, x_\eta^0, x_1^1, \ldots, x_\eta^1)$ *and* $pk := (y_1^0, \ldots, y_\eta^0, y_1^1, \ldots, y_\eta^1)$*. Return* (pk, sk)*.*
- ***Signing:*** $Sign_{Lamport}(sk, m)$ *with* $sk = (x_1^0, \ldots, x_\eta^0, x_1^1, \ldots, x_\eta^1)$ *and* $m \in \{0,1\}^\eta$ *returns* $\sigma := (x_1^{m_1}, \ldots, x_\eta^{m_\eta})$*.*
- ***Verification:*** $Verify_{Lamport}(pk, m, \sigma)$ *with* $pk = (y_1^0, \ldots, y_\eta^0, y_1^1, \ldots, y_\eta^1)$ *and* $m \in \{0,1\}^\eta$ *and* $\sigma = (\sigma_1, \ldots, \sigma_\eta)$ *checks whether* $y_i^{m_i} = f(\sigma_i)$ *for* $i = 1, \ldots, \eta$*. If so, it returns* 1*.*

Theorem 13 *Let f be a one-way function. Then $(KG_{Lamport}, Sign_{Lamport}, Verify_{Lamport})$ is EF-OT-CMA secure.*

Notice that the security of the scheme breaks down completely if we produce more than one signature with a given signing key.

Lamport's scheme only allows us to sign messages of length η. If we wish to sign longer messages, we can use the following construction:

Construction 5 *Let $(KG, Sign, Verify)$ be a signature scheme with message space M and let $H : \{0,1\}^* \to M$ be a collision-resistant hash function. Define $(KG_H, Sign_H, Verify_H)$ as follows: $KG_H(1^\eta) := KG(1^\eta)$. $Sign_H(sk, m) := Sign(sk, H(m))$. $Verify_H(pk, m) := Verify(pk, H(m))$.*

Lemma 9 *If $(KG, Sign, Verify)$ is EF-CMA secure (or EF-OT-CMA secure), then $(KG_H, Sign_H, Verify_H)$ is also EF-CMA secure (or EF-OT-CMA secure, respectively).*

Chain-based construction. We first show a simple construction for constructing a signature scheme from a one-time signature scheme. Later, we will show a more advanced construction that resolves a big drawback of the simple construction.

Construction 6 (Chain-based signatures) *Let $(KG_1, Sign_1, Verify_1)$ be a one-time signature scheme. We construct $(KG_{chain}, Sign_{chain}, Verify_{chain})$ as follows:*

- ***Key generation:*** *$KG_{chain}(1^\eta)$ returns $(pk_1, sk_1) \leftarrow KG_1(1^\eta)$.*
- ***Signing:*** *The signing algorithm is stateful. It maintains a list L, initially empty, a counter i, initially $i := 1$, and a secret key, initially $sk := sk_1$. When invoked as $Sign_{chain}(m_i)$, it does the following: Compute $(pk_{i+1}, sk_{i+1}) \leftarrow KG_1(1^\eta)$. Compute $\sigma_i := Sign_1(sk, (pk_{i+1}, m_i))$. Append m, pk_{i+1}, σ_i to L. Set $sk := sk_{i+1}$. Increment i. Return $\sigma := L$.*
- ***Verification:*** *$Verify_{chain}(pk_1, m, \sigma)$ with $\sigma = (m_1, pk_2, \sigma_1, \ldots, m_n, pk_{n+1}, \sigma_n)$: Check whether $Verify_1(pk_i, (pk_{i+1}, m_i), \sigma_i) = 1$ for all $i = 1, \ldots, n$. Check if $m_n = m$. If so, return 1.*

Notice that you could reduce the length of the signatures by assuming a stateful verifier. In this case, you only need to send the new parts of L when signing.

Theorem 14 *If $(KG_1, Sign_1, Verify_1)$ is EF-OT-CMA secure, then $(KG_{chain}, Sign_{chain}, Verify_{chain})$ is EF-CMA secure (but stateful).*

The proof bases on the fact that each secret key sk_i is used at most once.

Tree-based construction. The main disadvantage of the chain-based construction is that it uses a stateful signer. The following construction remedies this problem:

Construction 7 (Tree-based signatures) *Let $(KG_1, Sign_1, Verify_1)$ be a one-time signature scheme. Assume without loss of generality that KG_1 and $Sign_1$ use at most η random bits per invocation (if necessary, this can be ensured by using a PRG). Let $F : K \times \{0,1\}^* \to \{0,1\}^\eta$ be a PRF.*

We construct $(KG_{tree}, Sign_{tree}, Verify_{tree})$ with message space $\{0,1\}^\eta$ as follows:

- ***Key generation:*** *$KG_{tree}(1^\eta)$ picks $k_1, k_2 \xleftarrow{\$} K$ and $(pk_\varepsilon, sk_\varepsilon) \leftarrow KG_1(1^\eta; F(k_1, \varepsilon))$.)Here ε stands for the empty word. The notation $X(a;b)$ denotes an invocation of the algorithm X with input a and using b as its randomness.) KG_{tree} returns (pk, sk) with $pk := pk_\varepsilon$ and $sk := (k_1, k_2, sk_\varepsilon)$.*
- ***Signing:*** *$Sign_{tree}(sk, m)$ with $sk = (k_1, k_2, sk_\varepsilon)$ does the following: For $i = 0, \ldots, \eta - 1$, let*

$$(pk_{m_{|i}0}, sk_{m_{|i}0}) \leftarrow KG_1(1^\eta; F(k_1, m_{|i}0))$$
$$(pk_{m_{|i}1}, sk_{m_{|i}1}) \leftarrow KG_1(1^\eta; F(k_1, m_{|i}1))$$
$$\sigma_i \leftarrow Sign_1(sk_{m_{|i}}, (pk_{m_{|i}0}, pk_{m_{|i}1}); F(k_2, m_{|i})).$$

  Here $m_{|i}$ denotes the first i-bits of m.

  Let

$$\sigma_{end} \leftarrow Sign_1(sk_m, m; F(k_2, m)).$$

  Return $\sigma := (pk_{m_{|0}0}, pk_{m_{|0}1}, \ldots, pk_{m_{|\eta-1}0}, pk_{m_{|\eta-1}1}, \sigma_0, \ldots, \sigma_{\eta-1}, \sigma_{end})$.

- ***Verification:*** *$Verify_{tree}(pk, m, \sigma)$ with $pk = pk_\varepsilon$ and $\sigma = (pk_{m_{|0}0}, pk_{m_{|0}1}, \ldots, pk_{m_{|\eta-1}0}, pk_{m_{|\eta-1}1}, \sigma_0, \ldots, \sigma_{\eta-1}, \sigma_{end})$ does the following: For $i = 0, \ldots, \eta - 1$, check whether $Verify_1(pk_{m_{|i}}, (pk_{m_{|i}0}, pk_{m_{|i}1}), \sigma_i) = 1$. Check whether $Verify_1(pk_m, m, \sigma_{end}) = 1$. If all checks succeed, return 1.*

Notice that in this construction, any sk_x is used several times, but always to sign the same message. And in each invocation, it uses the same randomness, so effectively sk_x is used only once.

Theorem 15 *If $(KG_1, Sign_1, Verify_1)$ is EF-OT-CMA secure, then $(KG_{tree}, Sign_{tree}, Verify_{tree})$ is EF-CMA secure.*

Plugging things together. By Theorem 13 we get a one-time signature scheme (EF-CMA secure) which can be extended to allow signing of arbitrary messages (not just length η) by Lemma 9. Based on that scheme, by Theorem 15, we get a signature scheme which is EF-CMA secure (and whose message space can be extended using Lemma 9). These constructions need one-way functions, PRFs, and collision-resistant hash functions. By Theorem 12(b), PRFs can in turn be constructed from one-way functions. Thus we get:

Corollary 1 *If one-way functions and collision-resistant hash functions exist, then there is an EF-CMA secure signature scheme (with message space* $\{0,1\}^*$*).*

Note: Instead of the collision-resistant hash function, one can also use a so-called *universal one-way hash function (UOWHF).* (This requires a slight change of the construction.) A UOWHF can be constructed from one-way functions. We omit the details and only mention the final result:

Theorem 16 *If one-way functions exist, then there is an EF-CMA secure signature scheme (with message space* $\{0,1\}^*$*).*

Note that this construction is extremely inefficient (especially if we construct UOWHFs from OWFs) and thus unusable in practice.

13.2 Full domain hash

We now describe another construction of a signature scheme, the full domain hash (FDH) construction. (We describe RSA-FDH; in general, FDH can based on other assumption than RSA, too.)

Construction 8 *Let* $H : \{0,1\}^* \to \{0,1\}^{3\eta}$ *be a hash function (or the random oracle, respectively).*[11]

The RSA-full domain hash scheme (RSA-FDH) $(KG_{FDH}, Sign_{FDH}, Verify_{FDH})$ *is defined as follows:*

- ***Key generation:*** $KG_{FDH}(1^\eta)$ *picks two* η*-bit primes* p, q*, computes* $N := p, q$*, picks uniform* e, d *with* $ed \equiv 1 \bmod \varphi(N)$*, and sets* $sk := (N, d)$ *and* $pk := (N, e)$*. (This is the same key generation as for textbook RSA.)*
- ***Signing:*** $Sign_{FDH}(sk, m)$ *with* $sk = (N, d)$ *returns* $H(m)^d \bmod N$*.*
- ***Verification:*** $Verify_{FDH}(pk, m, \sigma)$ *with* $pk = (N, e)$ *returns* 1 *iff* $H(m) \equiv \sigma^e \bmod N$*.*

Since only have a proof of RSA-FDH in the random oracle model, we restate the definition of EF-CMA security in the random oracle model:

Definition 32 (Existential unforgeability (in the random oracle model)) *Given sets* M, N *(domain and range of the random oracle), we call a signature scheme* $(KG, Sign, Verify)$ existentially unforgeable (under chosen message attacks) in the random oracle model *(EF-CMA secure in the ROM) iff for every polynomial-time algorithm* A*, there is a negligible function* μ *such that for all* η *we have that*

$$\Pr[\mathit{Verify}^H(pk, m, \sigma) = 1 \ \wedge \ m \textit{ fresh} : H \xleftarrow{\$} \mathrm{Fun}_{M \to N}, \\ (pk, sk) \leftarrow KG^H(1^\eta), (m, \sigma) \leftarrow A^{H, Sign^H(sk, \cdot)}(1^\eta, pk)] \leq \mu(\eta).$$

Here "m fresh" means that m has never been sent by A to the $Sign^H(sk, \cdot)$*-oracle.*

[11] We use the range $\{0,1\}^{3\eta}$ such that $H(m) \bmod N$ will be almost uniform for an N that is the produce of two η-bit primes.

We then have:

Theorem 17 *If the RSA assumption holds,* $(Sign_{FDH}, Verify_{FDH})$ *is EF-CMA secure in the random oracle model.*

14 Symbolic cryptography

To ensure trust in a cryptographic protocol, the security of the protocol is typically proven. However, proving the security of a protocol by hand has two drawbacks:

- Humans tend to make accidental mistakes. Cryptographic proofs of complex systems are usually quite long and contain a lot of case distinctions etc. Thus, one cannot expect that a hand-written security proof of a complex system will be correct.
- Real-life protocols may be so complex as to defy manual security analysis altogether. Since the complexity of a proof increases with the complexity of the system, some systems may need proofs in which humans simply lose track.

In light of these problems, we would like *automated verification* in which a computer finds the security proof, since a computer will not make accidental mistakes. (To deal with the first problem only, *computer-aided verification* is also an option: here, the proof is (partially) constructed by a human, but rigorously checked by the computer.)

Unfortunately, the reduction proofs that are typically used in cryptography are too difficult for the computer: they usually require clever insights that the computer does not have. To be able to automate the analysis of protocols, a heuristic is therefore used: Instead of modeling operations like encryption and decryption as the application of probabilistic algorithms on bitstrings, we simplify the world and model them as a operations on abstract terms. E.g., instead of representing an encryption as a bitstring (e.g., 010010110), we directly represent it as a term (e.g., $enc(pk, m)$). Although this does not represent reality, it is a useful idealization. (We discuss pros and cons of this approach below.)

The modeling of cryptography in which messages are represented as bitstrings (and not terms) is usually called the *computational model*.

Advantages and disadvantages The big advantage of modeling cryptography symbolically is that in the symbolic model automated verification is much simpler. At the current state of the art, many protocols can only be analyzed in the symbolic model – in these cases, it is better to perform verification in the symbolic model than no verification at all. (This motivation is similar to that of the random oracle model, see Section 12.)

One should, however, use the symbolic model with care. To be safe, one needs to identify everything the adversary can do and add it to the deduction rules. If one overlooks something, an attack is overlooked. (Note: There is even no suitable formal definition of "everything the adversary can do", so it is very difficult to tell when the list of

deduction rules is sufficiently complete.) While for encryption, the list of deduction rules seems straightforward, for other cryptographic schemes (e.g., schemes with homomorphic properties), it is far from clear which rules to use.

In some specific cases, however, it can be proven that the symbolic model is sound. Such results are called *computational soundness results*.[12]

14.1 Analysis of the Needham-Schröder-Lowe protocol

To illustrate how a symbolic model is used, we will investigate the following two protocols for mutual authentication.

Needham-Schröder protocol. In this protocol, there are two roles, initiator and responder. We assume that each party X has a secret key sk_X and that everyone knows the corresponding public key pk_X. When a party A is running as the initiator (with argument B – indicating the intended partner), and B is running as the responder (with argument A), the following steps are executed:

- A picks a random value R_1 and sends $enc(pk_B, (A, R_1))$ to B.
- B decrypts, picks a random R_2, and then sends $enc(pk_A, (R_1, R_2))$ to A.
- A decrypts, and sends $enc(pk_B, R_2)$ to B.

The intended property of this protocol is mutual authentication, that is, A will not believe to be talking to B unless B also intended to talk to A and vice verse. Somewhat more formally:

Definition 33 (Mutual authentication – informal) *Whenever some A successfully finishes*[13] *an execution as initiator with argument B, then the party B has started an execution as responder with argument A. And vice versa (with initiator and responder switched).*

The Needham-Schröder protocol does not satisfy this property: Assume that A runs as initiator with argument C, that B runs as responder with argument A, and that C is corrupted (i.e., the adversary) and communicates with both A and B. That is, A believes correctly that she is talking to C, but B believes incorrectly that he is talking to A. C performs the following steps:

- When getting $enc(pk_C, (A, R_1))$ from A, it decrypts and sends $enc(pk_B, (A, R_1))$ to B.
- When B responds with $enc(pk_A, (R_1, R_2))$, C forwards this message to A.
- When A responds with $enc(pk_C, R_2)$, C decrypts and sends $enc(pk_B, R_2)$ to B.

[12] Note: this notion of computational soundness has nothing to do with the notion of computational soundness introduced in Section 15. They are just homonymes.

[13] That is, all the messages it receives are as it expects them from the protocol description.

- Then B successfully finishes the execution.

Since B successfully finished an execution with argument A, but A never started an execution *with argument* B, this shows that the Needham-Schröder protocol does not satisfy mutual authentication.

The following protocol is supposed to fix this problem:

Needham-Schröder-Lowe protocol (NSL). The situation is the same as in the Needham-Schröder protocol, but the following steps are executed:

- A picks a random value R_1 and sends $enc(pk_B, (A, R_1))$ to B.
- B decrypts, picks a random R_2, and then sends $enc(pk_A, (R_1, R_2, B))$ to A.
- A decrypts, and sends $enc(pk_B, R_2)$ to B.

Notice that the only difference to the Needham-Schröder protocol is that the message $enc(pk_A, (R_1, R_2, B))$ additionally contains the name B.

Symbolic modeling of NSL. We would like to make sure that NSL is not also susceptible to attack similar to the above. For this, we will analyze the security in the symbolic model. The first step is to write down how the protocol and the capabilities of the adversary are modeled.

For illustration purposes, we consider a special case only, corresponding to the attack on the Needham-Schröder protocol described above. Namely, we consider an execution in which A runs as initiator with argument C, and B runs as responder with argument A. In this case, to check security, it is sufficient to check whether B successfully finishes. (A full-fledged analysis would have to consider arbitrary many executions of arbitrary parties in arbitrary rules with arbitrary arguments.)

We start by modeling the capabilities of the adversary:

$$\frac{\vdash enc(pk_C, x)}{\vdash x}\ \textsc{Dec} \qquad \frac{\vdash x \qquad P \in \{A, B, C\}}{\vdash enc(pk_P, x)}\ \textsc{Enc}$$

$$\frac{}{\vdash A \qquad \vdash B \qquad \vdash C}\ \textsc{Name} \qquad \frac{\vdash x \qquad \vdash y}{\vdash (x, y)}\ \textsc{Pair}$$

$$\frac{\vdash (x, y)}{\vdash x \qquad \vdash y}\ \textsc{Split} \qquad \frac{i \in \mathbb{N}}{\vdash \hat{R}_i}\ \textsc{Rand}$$

These rules are read as follows: The expression $\vdash t$ stands for "the adversary knows (or: can know) the term t". If, for some assignment for the variables x and y, the preconditions in the upper half of the rule are satisfied, then the conclusions in the lower half are also satisfied. E.g., if $\vdash enc(pk_C, (A, R_1))$, then the preconditions of the rule Dec are satisfied (with $x := (A, R_1)$), and hence $\vdash (A, R_1)$ holds.

Notice that there is no rule for $\vdash R_1$ or $\vdash R_2$. This models the fact that the adversary cannot guess these random values unless he first gets some term that contains them. (Keep in mind, though, that R_1, R_2 are not modeled as random values, but as unique symbols.)

Furthermore, we need to model what the adversary C can learn from interacting with the protocol. For example, A sends $enc(pk_C, (A, R_1))$, so we have the rule MSG1 below. Furthermore, if C knows a term of the form $enc(pk_A, (R_1, x, C))$ for any x,[14] he can send this term to A, who will answer with $enc(pk_C, x)$. This leads to the rule MSG3. And finally, if C knows $enc(pk_B, (A, y))$, then he can send this to B and gets $enc(pk_A, (y, R_2, B))$. Hence rule MSG2.

$$\frac{}{\vdash enc(pk_C, (A, R_1))}\ \text{MSG1} \qquad \frac{\vdash enc(pk_B, (A, y))}{\vdash enc(pk_A, (y, R_2, B))}\ \text{MSG2}$$

$$\frac{\vdash enc(pk_A, (R_1, x, C))}{\vdash enc(pk_C, x)}\ \text{MSG3}$$

These rules describe everything the adversary can learn.

Formally, we define $\vdash$ as follows:

Definition 34 (Deduction relation) *$\vdash$ is the smallest relation satisfying the deduction rules above.*

This means that $\vdash t$ holds if and only if one can deduce $\vdash t$ by applying the rules above.

Security proof of NSL. We will now show the security of NSL. In the special case we are analyzing, it is sufficient to show that B never receives the message $enc(pk_B, R_2)$. That is, we have to show that $\nvdash enc(pk_B, R_2)$. ($\nvdash$ is the negation of $\vdash$.)

Consider the following grammar:

$$M ::= enc(pk_A | pk_B | pk_C, M) \mid A \mid B \mid C \mid R_1 \mid \hat{R}_i \mid (M, M) \mid enc(pk_A, (R_1, R_2, B))$$

Security is based on the following claim:

Lemma 10 *For any term t satisfying $\vdash t$, we have that t matches the grammar of M.*

This claim is easily proven by checking that for each deduction rule, we have that if all terms in the preconditions match the grammar, then the terms in the postcondition also match the grammar. For example, for rule DEC, we need to check that if $enc(pk_C, t)$ matches the grammar for some term t, then t matches the grammar.

Since $enc(pk_B, R_2)$ does not match the grammar, it immediately follows that:

Theorem 18 $\nvdash enc(pk_B, R_2)$.

Thus the NSL protocol is secure (in the symbolic model and our specific case).

[14] In our analysis, we consider (x, y, z) as a shortcut for $(x, (y, z))$, so that we do not need to formally introduce triples; pairs are sufficient.

15 Zero-knowledge

In many situations in cryptography, it is necessary for one party to convince another party that certain data has a certain property without revealing too much information. We illustrate this by an example:

Peggy wishes to purchase some goods from Vendor. These goods are only sold to people aged at least 18. Peggy has a certificate issued from the government that certifies her birthday (i.e., the government signed a message of the form "Peggy was born on 17. March 1979"). To prove that Peggy is at least 18 years old, she might simply send the certificate to the Vendor. But, for privacy reasons, she does not wish to reveal her birthday, only the fact that she is at least 18 years old.

This can be done by a so-called zero-knowledge proof: Peggy proves that she knows[15] a valid signature on a message of the form "Peggy was born on x" where x is a date at least 18 years before the present date. Such a zero-knowledge proof will then not reveal anything beyond that fact. In particular, the value of x will not be revealed.

15.1 Definition

A *proof system* for a relation R is a protocol consisting of two machines (P, V). The *prover* P expects arguments $(1^\eta, x, w)$ with $(x, w) \in R$. The value x is usually called the *statement* and w the *witness*. The relation R models which witnesses are valid for which statements. Intuitively, one can see x as describing some mathematical fact (e.g, "there exists a value that has property P") and w as the proof of that fact (e.g., a particular value with property P). Then $(x, w) \in R$ just means that w is a valid proof for x. The *verifier* V expects arguments $(1^\eta, x)$. Notice: the verifier does not get w, otherwise there would not be any sense in proving the existence of such a w to the verifier.

The first (and most obvious) property of a proof system is that it should work. That is, when prover and verifier are honest, the verifier should accept the proof. This property is called completeness:

Definition 35 (Completeness) *We call a proof system* (P, V) complete *if there exists a negligible function* μ *such that for all* η *and all* $(x, w) \in R$*, we have*

$$\Pr[\langle P(1^\eta, x, w), V(1^\eta, x)\rangle = 1] \geq 1 - \mu(\eta).$$

Here $\langle A(a), B(b)\rangle$ *stands for the output of* B *after an execution in which* A *runs with arguments* a *and* B *runs with arguments* b *and* A *and* B *interact.*

The second thing one expects from a proof is that one cannot prove wrong things. This is captured by the following definition:

[15] The fact that Peggy *knows* a particular value (as opposed to the fact that this value exists) can be modeled formally. In this exposition, however, we will only give definitions for proofs that show the existence of certain values (Definition 36).

Definition 36 (Computational soundness) *A proof system (P, V) is* computationally sound[16] *with soundness error s iff for every polynomial-time machine P^* there exists a negligible function μ such that for all η, all $x \notin L_R$, and all z, we have*

$$\Pr[\langle P^*(1^\eta, x, z), V(1^\eta, x)\rangle = 1] \leq s(\eta) + \mu(\eta)$$

Here $L_R := \{x : \exists w.(x, w) \in R\}$.

Of couse, we want proof systems with soundness error $s = 0$ (or equivalently: negligible s). Nevertheless, it is useful to have a definition for proof systems with soundness error $s > 0$ since the soundness error can be reduced by repetition of the proof.

Finally, we wish to formulate that when the prover runs with statement x and witness w, the verifier learns nothing except the fact that x is a true statement:

Definition 37 (Zero-knowledge) *A proof system (P, V) is* computational zero-knowledge *if for any polynomial-time V^* there exists a polynomial-time S such that for every polynomial-time D there exists a negligible function μ such that for all η, z, $(x, w) \in R$, we have that*

$$\begin{aligned}\big|\Pr[b^* = 1 : out \leftarrow \langle P(1^\eta, x, w), V^*(1^\eta, x, z)\rangle, b^* \leftarrow D(1^\eta, z, out)] \\ - \Pr[b^* = 1 : out \leftarrow S(1^\eta, x, z), b^* \leftarrow D(1^\eta, z, out)]\big| \leq \mu(\eta).\end{aligned}$$

Intuitively, this definition means the following: Whenever the verifier can learn something (which he outputs as *out*), there is a corresponding *simulator* that computes a value *out* that cannot be distinguished from the real *out* (by the *distinguisher* D). That is, whatever the verifier learns, the simulator can compute himself. But the simulator has no information about the witness w, hence his output does not depend on w. Thus the verifier learns nothing about w.

The additional input z (called the *auxiliary input*) models the fact that there might be some additional information that the adversary has (e.g., from prior protocol executions).

15.2 Graph isomorphism

We now describe a particular zero-knowledge proof system, namely the proof system for graph isomorphism. Give two graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$, an *isomorphism* from G_1 to G_2 is a permutation π on V such that $E_2 = \{(\pi(a), \pi(b)) : (a, b) \in E_1\}$. (Intuitively, an isomorphism is a shuffling of the graph.) We write $\pi(G_1) = G_2$. We call two graphs G_1 and G_2 isomorphic if there is an isomorphism from G_1 to G_2.

For the rest of this section, let $R := \{((G_1, G_2), \pi) : \pi(G_1) = G_2\}$. That is, a statement for this relation is a pair (G_1, G_2) of graphs, and a witness for these graphs is an isomorphism between them. Hence a proof system for this relation will prove that G_1 and G_2 are isomorphic.

[16] Note: this notion of computational soundness has nothing to do with the notion of computational soundness introduced in Section 14. They are just homonymes.

Construction 9 (Graph isomorphism proof system)
- *The prover P runs with input $(1^\eta, (G_1, G_2), \pi)$ with $\pi(G_1) = G_2$. The verifier V runs with input $(1^\eta, (G_1, G_2))$.*
- *The prover picks a uniformly random permutation $\rho \xleftarrow{\$} \mathrm{Perm}_V$ and computes the graph $H := \rho(G_1)$. (Notice that H is isomorphic to G_1 and G_2.) The prover sends H to the verifier.*
- *The verifier picks $i \xleftarrow{\$} \{1, 2\}$ and sends i to the prover.*
- *The prover computes σ as: $\sigma := \rho$ if $i = 1$ and $\sigma := \rho \circ \pi^{-1}$ if $i = 2$. Then the prover sends σ to the verifier.*
- *The verifier checks whether $\sigma(G_i) = H$. If so, he accepts (outputs 1).*

Theorem 19 *The proof system (P, V) from Construction 9 is complete, computational zero-knowledge, and computationally sound with soundness error $\frac{1}{2}$.*

The soundness error can be decreased to 2^{-n} by repeating the proof system n times sequentially (and letting the verifier accept only if he accepts each exeuction). E.g., $n := \eta$.

References

[Hey] Howard M. Heys. A tutorial on linear and differential cryptanalysis. `http://www.engr.mun.ca/~howard/PAPERS/ldc_tutorial.ps`.

[MvV96] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, Florida, 1996. Online available at `http://www.cacr.math.uwaterloo.ca/hac/`.

[Unr11] Dominique Unruh. Lecture "Cryptology I", fall 2011. Webpage is `http://www.cs.ut.ee/~unruh/crypto1-11/`.

Index

Symbol index

$\|x\|$	Absolute value / length / cardinality of x	
$I_C(x)$	Index of coincidence of x	3
δ_{ab}	Kronecker's delta (= 1 iff $a = b$)	3
$\Pr[B : P]$	Probability of B after executing P	6
$x \xleftarrow{\$} Y$	x is uniformly randomly chosen from set Y	6
$x \leftarrow A$	x is assigned output of algorithm A	6
$\oplus$	Bitwise XOR	6
QR_n	Quadratic residues modulo n	10
A^X	Algorithm A with oracle access to X	14
Perm_M	Set of all permutations on M	14
$Primes_\eta$	Primes of length η	16
pk	Usually denotes a public key	
sk	Usually denotes a secret key	
$SafePrimes_\eta$	Safe primes of length η	18
$\{0,1\}^{*n}$	Bistrings with length multiple of n	22
H_{IH}	Iterated hash	22
H_{MD}	Merkle-Damgård construction	23
MAC_{HMAC}	HMAC message authentication code	25
$\mathrm{Fun}_{M \to N}$	Set of functions from M to N	25
$MAC_{CBC\text{-}MAC}$	CBC-MAC message authentication code	26
MAC_{DMAC}	DMAC message authentication code	26
$\vdash t$	Adversary can deduce (know) the term t	44
$\langle A, B \rangle$	Interaction of machines A and B	46