Lecture Notes for CS-578 Cryptography (SS2006) Prof. Michael Backes

1. Historical Encryption Schemes and the One-time Pad

Lecture 1 Saarland University

In this chapter we examine some historic schemes and show how to break them quite easily. We furthermore introduce some basic notation to set up the ground for future studies of more evolved encryption schemes. More information can be found in the beautiful book *The Codebreakers* by David Kahn, which contains a very nice overview of the history of cryptography up to World War II).

1.1 Preliminaries

Let us first fix some important principles of secure encryption. There have been (and still are) encryption schemes where the algorithm is kept secret by the company/inventor. This is called *security by obscurity* and should be avoided under any circumstances: While the inventor usually claims that the system is secure, the user has no (easy) way to verify this claim. The algorithm might be flawed, and an attacker might invest enough time to break the scheme; in fact reverse-engineering crypto algorithms with nowadays tools is usually not that hard. Also, the inventor might have built in some sort of *trap-door*, which enables him to read all encrypted messages.

Thus a common design criterion for every modern cryptosystem is that the algorithms used for encryption and decryption are publicly known, and that the security of the scheme only relies on the secrecy of a short secret called the key. This is called *Kerkhoff's Principle*.

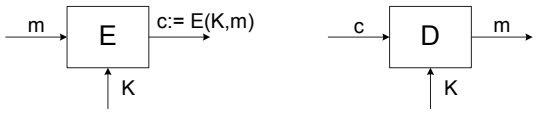


Figure 1.1: Basic Cipher

Let us conclude this brief overview by pointing out which parts an encryption system consists of; a depiction is given in Figure 1.1. For the next weeks we will deal with *symmetric encryption* only, i.e., both the *sender* (usually called Alice) and the receiver (usually called Bob) of a secret message share a common secret key K. There is an "efficient" algorithm E, called *encryption (algorithm)*, which takes the *key* and a *plaintext* and outputs a *ciphertext*, and there is an "efficient" algorithm D, called *decryption (algorithm)*, which takes the *key* and a *ciphertext* and outputs a plaintext. The notion of efficiency will be formally introduced later. We require that the decryption of a ciphertext yields the original message again. This property is called *correctness* of the encryption scheme.

Usually we denote symmetric keys by $K, K_1, K_2, \ldots$, messages by $m, m_1, m_2, \ldots$, and ciphertexts by $c, c_1, c_2, \ldots$. Sometimes we will denote plaintext messages with lower-case letters, i.e., `secret message`, and ciphertext messages with upper-case letters, i.e., `KAJFUG MSJFTOA`.

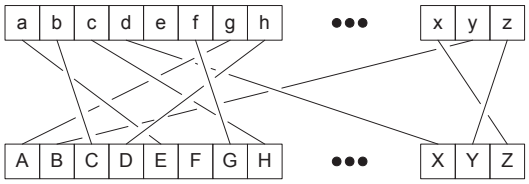


Figure 1.2: Substitution Cipher

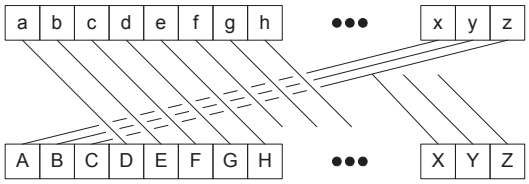


Figure 1.3: Shift-Cipher

1.2 Historic Ciphers

1.2.1 Substitution Cipher

The *substitution cipher* is the oldest and one of the most famous ciphers in history and literature, and it comes in several variants. For simplicity we assume a text to consist of letters only, removing any spaces and punctuation. We also assume that the substitution alphabet, i.e., the symbols the ciphertext consists of, are letters as well.

Then a key K of the substitution cipher is a permutation of the set $\{a, b, \ldots, z\}$, cf. Figure 1.2. A message $m = m_1||m_2||\ldots||m_q$ is encrypted by computing

$$c = K(m_1)||K(m_2)||\ldots||K(m_q),$$

where the m_i are single letters and where $||$ denotes concatenation of strings.

As K is a permutation there exists the (unique) inverse permutation K^{-1}. This is used to decrypt a given ciphertext by the following rule: For $c = c_1||\ldots||c_q$, compute

$$m' = K^{-1}(c_1)||\ldots||K^{-1}(c_q).$$

It is easy to see that this scheme is *correct*, i.e., that decryption of an encryption (with the same key) yields the message again. Formally,

$$\begin{aligned} D(K, E(K, m)) &= D(K, K(m_1)||\ldots||K(m_q)) \\ &= K^{-1}(K(m_1))||\ldots||K^{-1}(K(m_q)) \\ &= m_1||\ldots||m_p \\ &= m. \end{aligned}$$

A small example based on the key K shown in Figure 1.2: if the plaintext `abc` is encrypted under this key, the ciphertext becomes `ECH`.

Cryptanalysis The conceptionally simplest attack on any encryption scheme is the so-called *brute-force attack*, where each possible key is used to decrypt a ciphertext and the resulting message is investigated. Thus the practicability of this attack relies precisely on the size of the key-space. For the substitution cipher the size of the key-space is $26! \approx 2^{86}$, so a brute-force attack on a substitution cipher is clearly impractical.

However, using statistical attacks, the substitution cipher can be broken quite easily. The main observation is that letters in average text do not occur with the same probability. In English text, for example, the letter "e" appears with probability approx. 12%, while "z" has probability below

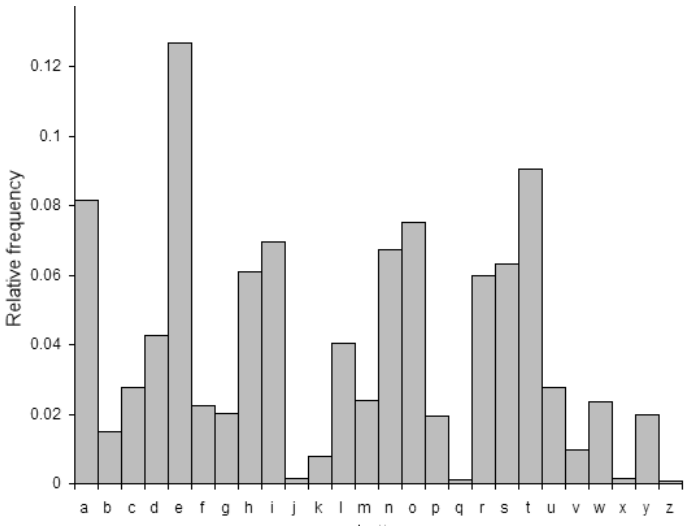


Figure 1.4: Letter frequencies in english Text

1%, cf. Figure 1.4. So a reasonable guess is that the most common letter in the ciphertext is the encryption of "e", or at least of one of the most common letters. Iterating this step will finally yield the plaintext. Of course letter frequencies depend on the writer of a text, on the topic, and so on, but in practice a text of a certain (not too small length) is easily breakable.

1.2.2 Shift-Ciphers and Caesar's Cipher

A special form of the substitution cipher is the shift-cipher. There the permutation K is chosen from a restricted set, namely by shifting the letters by a fixed number of positions in the alphabet. Given a number $\alpha_K \in \{0, \ldots, 25\}$, the permutation K maps each letter to its α_K'th successor, "wrapping" around after the Z and starting with A again if necessary (Figure 1.3). If one identifies $\{a, \ldots, z\}$ with $\{0, \ldots, 25\}$ then $K(\beta) = \beta + \alpha_K \bmod 26$. Julius Caesar used this cipher with a fixed value $\alpha_K = 3$, which is nowadays known as *Caesar's Cipher*. A variant is the so-called ROT-13 scheme, where $K_\alpha = 13$. This is sometimes used in Newsgroups and the Internet to prevent people from reading things by accident. For example, movie discussion boards "encrypt" details of the plot or the ending using ROT-13. This is not an encryption in a strict sense, however, the same mechanism is applied.

Cryptanalysis The size of the key-space is 26, rendering the Shift-cipher completely insecure. The plaintext can be recovered either by a brute-force attack testing each of the 26 keys and testing if the resulting plaintext makes any sense, or by frequency analysis as described above.

1.2.3 Vigenere Cipher

The Vigenere Cipher is a generalization of the Shift-Cipher. While the latter relies on a single permutation K to shift every letter by α_K positions, the Vigenere cipher uses n such permutations $K_0, \ldots, K_{n-1}$ to shift different letters by a different number $\alpha_{K_0}, \ldots, \alpha_{K_{n-1}}$ of positions. For $m =$

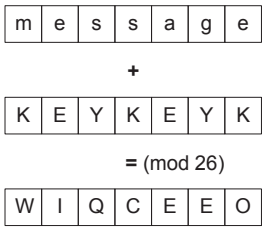


Figure 1.5: Vigenere Cipher

$m_0||\ldots||m_q$:

$$c_i := K_{i \bmod n}(m_i)$$

Decryption is done by shifting in the opposite direction:

$$m_i := K^{-1}_{i \bmod n}(c_i).$$

A small example is given in Figure 1.5.

Cryptanalysis The size of the keyspace is $26^n \approx 2^{4.7n}$, which becomes infeasible for brute-force attacks for moderate values of n. But again, statistical tests can be applied to break the scheme quite easily.

1.2.4 Rotor Machines and the Enigma

The Enigma was originally a commercial encryption machine that was later adopted by the military. A large number of variations of the Enigma machine exist; in the following we describe a generic model close to that used in the early days of World War II.

The machine comprises the following main elements. The *keyboard* contains letters from A to Z. In a first step, the letters from the keyboard are substituted using a *plug board*, which corresponds to the substitution cipher described above.

In the next step, the electric signal passes three *rotors*, each of which applies a fixed substitution to the letters, cf. Figure 1.6. If these rotors were static, this step would yield nothing than a substitution cipher again. However, after encrypting one letter, the rightmost rotor advances one step. After one full turn the middle rotor advances as well, and after one full turn of the middle rotor the leftmost rotor advances as well. Thus each letter is treated with a different substitution.

After passing the three rotors it enters the *reflector*. It permutes the letters again, but instead of passing it further it reflects the signal back to the same rotor, thus passing all three rotors in opposite direction, passing the plug-board one more time, and then reaching the *lamps*.

The reflector was invented to simplify decryption: The same setting can be used to decrypt the message again, as the signal passes the rotors in exactly the opposite direction. However, this is one of the main weaknesses of the Enigma, and one that helped to break it during World War II. Note that with the reflector, a letter is never encrypted to the same letter, as the three rotors perform a substitution and the reflector cannot "reflect" the signal back the same path through the rotors, as no electric circuit would be established then.

The key for the encryption consisted mainly of the permutation on the plug-board, the initial position of the rotors, and the order the rotors were placed into the machine.

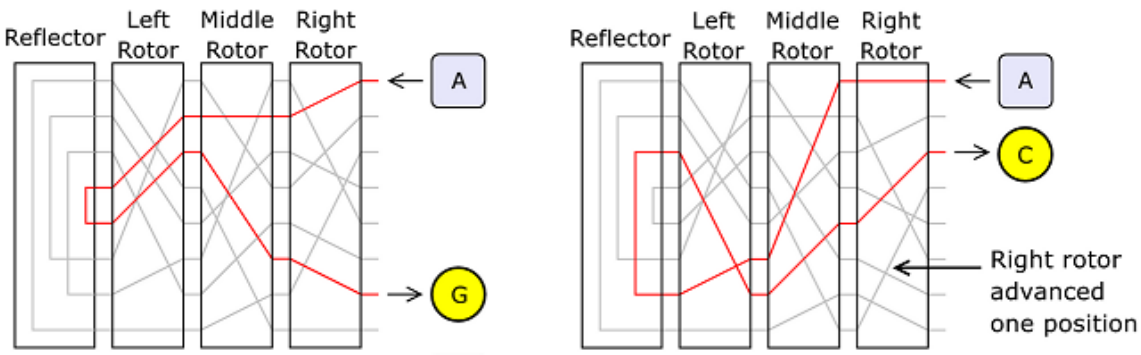


Figure 1.6: Functional description of the Enigma

Nowadays the Enigma can be easily broken by statistical analysis. The additional hardness was that cryptanalysts at that time neither knew the exact function of the Enigma or the wiring of the rotors. However, laziness of users facilitated this job, for example re-using the same key, using very weak keys (such as `AAA`), etc.

1.3 Classification of Attacks

In the previous section we implicitly assumed that the attacker intends to recover the plaintext given a ciphertext. While this is certainly a reasonable goal, there are other goals that are potentially easier to achieve, but should arguably be avoided as well. Apart from the goals an adversary pursues, there are different possibilities an attacker might exploit for attacking an encryption scheme, e.g., ciphertext-only attacks, or attacks where one assumes that certain plaintexts/ciphertext pairs are already known to the adversary. We will informally discuss the most common of these attack goals in the sequel.

1.3.1 Adversarial Goals

The following goals of an attacker are usually discussed:

- *Total break:* The attacker recovers the key that was used to encrypt ciphertexts. This will of course enable him to decrypt any ciphertext it gets.
- *Universal Break:* The attacker finds an alternative method to decrypt *any* ciphertext, without necessarily recovering the secret key.
- *Partial Break:* The attacker finds an alternative method to decrypt *some* distinguished ciphertexts.
- *Partial Information:* The attacker finds a method to compute *partial information* about the plaintexts given some ciphertext, e.g., individual bits, checksum, ...

Note that these attacks are listed in decreasing strength, in the sense that if an attacker can *universally break* an encryption scheme, then he can also *partially break* the scheme and compute

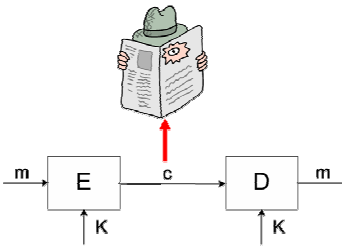


Figure 1.7: Known Cipher-text Attack

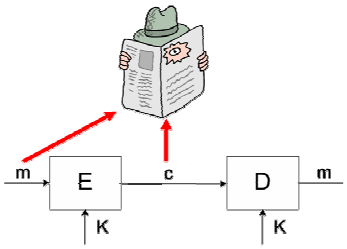


Figure 1.8: Known Plain-text Attack

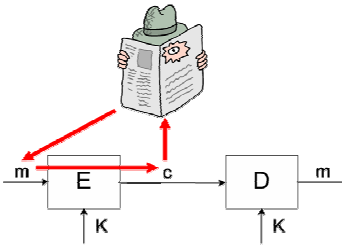


Figure 1.9: Chosen Plain-text Attack

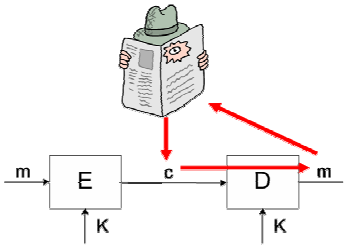


Figure 1.10: Chosen Cipher-text Attack

partial information, and so on. Note that in research, only the goal of preventing the adversary from obtaining any partial information is considered.

1.3.2 Adversarial Capabilities

The following adversarial capabilities are usually discussed:

- *Ciphertext-Only:* The adversary only sees one ciphertext (or a sequence of ciphertexts) encrypted with the (same) key, and he does not know the corresponding plaintext. This is illustrated in Figure 1.7.

- *Known-plaintext:* The adversary gets a sequence of paintext/ciphertext pairs encrypted using the same (unknown) key, and wants to attack an additional ciphertext whose corresponding plaintext is not known to him, cf. Figure 1.8.

- *Chosen Plaintext:* The adversary may choose a sequence of plaintext himself and get their encryptions using the same fixed key. After that, he wants to attack an additional ciphertext whose corresponding plaintext is not known to him, cf. Figure 1.9.

- *Chosen Ciphertext:* Similar to Chosen Plaintext, but the attacker may additionally choose ciphertexts and get their decryption under the considered key, cf. Figure 1.10. (This is the scenario typically considered in current research.)

1.4 Formal Definition of Ciphers

Before studying ciphers in more detail we start by giving a formal definition of symmetric encryption schemes.

Definition 1.1 (Symmetric Encryption Scheme) *A* symmetric encryption scheme *over* $(\mathcal{M}, \mathcal{C}, \mathcal{K})$ *is a tuple* (E, D) *where* E, D *are efficiently computable algorithms with* $\mathsf{E} : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$, $\mathsf{D} : \mathcal{K} \times \mathcal{C} \to \mathcal{M}$, *such that for all* $K \in \mathcal{K}$ *and for all* $m \in \mathcal{M}$*:*

$$\mathsf{D}(K, \mathsf{E}(K, m)) = m.$$

$\diamond$

Here

- E is called the *encryption* function,
- D is called the *decryption* function,
- $\mathcal{M}$ is a set called the *message space*. It contains all possible plaintexts, with typical examples being $\{a, \ldots, z\}^*$ (all words over the alphabet $a, \ldots, z$ of finite length), or $\{0,1\}^*$ (bitstrings of finite length).
- $\mathcal{C}$ is a finite set called the *ciphertext space*. It contains all encryptions that might occur in the scheme.
- $\mathcal{K}$ is a finite set called the *keyspace*. For symmetric encryption keys are drawn uniformly random from $\mathcal{K}$. We have seen for example $\mathcal{K} = \mathbb{N}_{26}$, $\mathcal{K} = (\mathbb{N}_{26})^*$, $\mathcal{K} = \mathsf{Perm}(\{a, \ldots, z\}) = \{p : \{a, \ldots, z\} \to \{a, \ldots, z\} \mid p \text{ is bijective}\}$

1.5 The One-time Pad

1.5.1 Definition of the One-time Pad

The One-time Pad, also called Vernam-cipher, was invented in 1917. It was the first scheme for which a security proof was given, namely by Claude Shannon in 1949. (Before that, not even definitions of security existed.) It may be defined over a variety of message spaces, for example $\{a, \ldots, z\}$ or $\{0,1\}$. We define it over bitstrings.

Cryptosystem 1 (One-time Pad) *Let* $\mathcal{M} = \mathcal{C} = \mathcal{K} = \{0,1\}^n$ *be bitstrings of a fixed length* n*. Both encryption and decryption are defined by the xor of the key with the plaintext or ciphertext, respectively. Formally:*

$$\begin{aligned} \mathsf{E}(K, m) &:= K \oplus m \\ \mathsf{D}(K, c) &:= K \oplus c \end{aligned}$$

It is important that only one message may be encrypted with each key, thus the name of the scheme.

It is easy to see that the One-time Pad is indeed a correct cipher: Let $K \in \mathcal{K}$ and $m \in \mathcal{M}$. Then

$$\begin{aligned} \mathsf{D}(K, \mathsf{E}(K, m)) &= \mathsf{D}(K, K \oplus m) \\ &= K \oplus (K \oplus m) \\ &= (K \oplus K) \oplus m \\ &= 0 \oplus m \\ &= m \end{aligned}$$

1.5.2 Some Basic Probability Theory Notation

Recall that (M, D) is a *discrete probability space* iff M is a finite or countable set and $D : M \to [0, 1]$ such that $\sum_{m \in M} D(m) = 1$ and $P_D(E) = \sum_{m \in E} D(m)$, writing $P(E)$ if D is clear from the context. For sets containing one element we abbreviate $P_D(\{m\})$ by $P_D(m)$.

A *probabilistic function* $A : M \to N$ assigns each input a distribution on the output, i.e., it constitutes a deterministic function $M \to \mathsf{Dist}(N)$ where Dist is the set of all distribution on N. Probabilistic assignment $n \leftarrow A(m)$ then means executing the probabilistic algorithm A with input m and assigning the value to n. For a finite set X, we write $x \leftarrow_{\mathcal{R}} X$ to denote uniform random drawing of an element from X and assigning it to x.

Let *pred* be a predicate on a certain domain M. Then one defines the event that *pred* is fulfilled as $E_{pred} := \{m \in M : pred(m)\}$, and the *probability that pred is fulfilled* given a probabilistic assignment D is defined as

$$P(pred(x); x \leftarrow D) := P_D(E_{pred}). \tag{1.1}$$

Given probabilistic algorithms $A : M \to N$ and $B : M \times N \to O$, we define their *sequential execution* $(A, B) : M \to N \times O$ by

$$P_{(A,B)(m)}(n, o) := P_{A(m)}(n) P_{B(m,n)}(o). \tag{1.2}$$

Using Equations 1.1 and 1.2 one obtains the following general formula. For all $m \in M$

$$P(pred(m, n, o);\ n \leftarrow A(m), o \leftarrow B(m, n)) = \sum_{n,o:\ pred(m,n,o)} P_{A(m)}(n) P_{B(m,n)}(o).$$

Note that one can define a more general version involving i algorithms A_i as well as version with several external messages $m_1, \ldots, m_l$.

1.5.3 Perfect Secrecy of the One-time Pad

The first definition of security for encryption schemes was the definition of *perfect secrecy* given by Claude Shannon in 1949. Intuitively it says, that any message is encrypted to a specific ciphertext with the same probability. This essentially means that, given a ciphertext, *no* adversary gains *any* information which plaintext was encrypted.

Definition 1.2 (Perfect Secrecy) *Let* (E, D) *be an encryption scheme on* $(\mathcal{M}, \mathcal{C}, \mathcal{K})$*. The encryption scheme provides* perfect secrecy *if and only if, for all* $m_0, m_1 \in \mathcal{M}$ *and for all* $c \in \mathcal{C}$*, the following holds:*

$$\mathsf{Pr}\left[c = c'; K \leftarrow_{\mathcal{R}} \mathcal{K}, c' \leftarrow \mathsf{E}(K, m_0)\right] = \mathsf{Pr}\left[c = c'; K \leftarrow_{\mathcal{R}} \mathcal{K}, c' \leftarrow \mathsf{E}(K, m_1)\right]$$

◇

Different equivalent variants of this definition exist based on statistical independence and entropy.

Shannon also proved that the One-time Pad satisfies this notion of perfect secrecy. Even if the proof is rather simple, it constituted a major step in crypto history since it was the first actual proof of security of a cryptographic scheme.

Proposition 1.1 *The One-time Pad provides perfect secrecy.*

Proof. Let $m_0, m_1 \in \mathcal{M}$ and $c \in \mathcal{C}$ be arbitrary and let $U_{\mathcal{K}}$ denote the uniform distribution on $\mathcal{K}$. For $i \in \{0,1\}$ it holds that

$$\begin{aligned}
\mathsf{Pr}\left[c = c'; K \leftarrow_{\mathcal{R}} \mathcal{K}, c' \leftarrow \mathsf{E}(K, m_i)\right] &= \sum_{K,c':\ c=c'} P_{U_{\mathcal{K}}}(K) \cdot P_{E(K,m_i)}(c') \\
&= \sum_{K} P_{U_{\mathcal{K}}}(K) \cdot P_{E(K,m_i)}(c) \\
&= \sum_{K} 1/|\mathcal{K}| \cdot \begin{cases} 1 & \text{if } c = K \oplus m_i \\ 0 & \text{else} \end{cases} \\
&= 1/|\mathcal{K}|
\end{aligned}$$

This holds for both i, so the claim follows. ∎

The One-time Pad has the disadvantage that the key is as long as the message and may not be reused. Thus deploying this scheme necessarily requires to securely transmit a large amount of keys. While this is done in very sensitive environments (the "Red Telephone" linking the White House with the Kremlin during the Cold War was encrypted with the One-time Pad) it is impractical for essentially any application. So the question naturally arises if one can securely encrypt with shorter keys. Unfortunately, Shannon also proved that this is not possible given the definition of perfect secrecy.

Proposition 1.2 (Optimality of the One-time Pad) *Let* (E, D) *be a cipher over* $(\mathcal{M}, \mathcal{C}, \mathcal{K})$*. If the cipher provides perfect secrecy, then* $|\mathcal{K}| \geq |\mathcal{M}|$.

Proof. Let $m_1 \in \mathcal{M}$, $K_1 \leftarrow \mathcal{K}$, and $c \leftarrow \mathsf{E}(K_1, m_1)$. Assume for contradiction that $|\mathcal{K}| < |\mathcal{M}|$. Let $\mathcal{M}_c := \{m \in \mathcal{M} \mid \exists K \in \mathcal{K} \colon \mathsf{D}(K, c) = m\}$ denote the set of plaintexts that the ciphertext c could be decrypted to using some key K. Since decryption is deterministic and since $|\mathcal{K}| < |\mathcal{M}|$, we have that $|\mathcal{M}_c| < |\mathcal{M}|$. Thus there exist some $m_2 \in \mathcal{M} \setminus \mathcal{M}_c$, and this m_2 satisfies that $\forall K \in \mathcal{K} : \mathsf{D}(K, c) \neq m_2$ by definition of $\mathcal{M}_c$.

For the plaintexts m_1, m_2 and the ciphertext c one obtains

$$\begin{aligned}
\mathsf{Pr}\left[c = c'; K \leftarrow_{\mathcal{R}} \mathcal{K}, c' \leftarrow \mathsf{E}(K, m_1)\right] &> 0 \\
&= \mathsf{Pr}\left[c = c'; K \leftarrow_{\mathcal{R}} \mathcal{K}, c' \leftarrow \mathsf{E}(K, m_2)\right].
\end{aligned}$$

This contradicts the assumption that the cipher provides perfect secrecy, and thus the assumption $|\mathcal{K}| < |\mathcal{M}|$ was not correct. ∎

2. Stream Ciphers

Stream ciphers constitute very practical encryption schemes that can be easily be implemented in hardware. They are widely used in prominent protocols such as SSL/TLS and 802.11b WEP.

2.1 Definition of Stream Ciphers

We have seen that the One-time Pad provides very strong security guarantees, but is not usable for most practical applications as keys need to be as long as the messages. One idea to overcome this problem is to not use keys that are fully random but keys that only look random. A relatively short string which is truly random is used to compute a larger string which, while of course not being truly random, is as good as being random in a sense formally defined below. This large string is called a *pseudorandom* string, and it can be used to replace the random key in the One-time Pad.

Such a cipher is called a synchronous stream cipher, where synchronous means that the pseudorandom string is created independently of the messages. Recall that we proved in the last chapter that such a cipher cannot provide perfect secrecy since its key space is much smaller than the message space; however, we will see that it still provides strong security guarantees.

Definition 2.1 (Synchronous Stream Cipher) $(\mathsf{E}, \mathsf{D}, \mathsf{G})$ *is a* synchronous stream cipher *over* $(\mathcal{M}, \mathcal{C}, \mathcal{K}, \mathcal{L})$ *iff* (E, D) *is a symmetric encryption scheme over* $(\mathcal{M}, \mathcal{C}, \mathcal{L})$*, and* $\mathsf{G} : \mathcal{K} \to \mathcal{L}^*$ *is an efficient function called the* pseudorandom generator *of the stream cipher.* $\diamond$

Elements of $\mathcal{K}$ are called *seeds*, $\mathcal{L}$ is called the *key-stream alphabet*, and output of G *pseudorandom strings*. Of course, the security of a stream cipher now crucially depends on the function G which creates long, almost random strings out of short, random strings. The central property this function should satisfy is called *unpredictability*.

2.2 Unpredictability of Pseudorandom Generators (PRGs)

Why do we need unpredictability? Assume an email is transmitted in an encrypted manner. Then one knows that large parts of the plaintext, e.g., the header, have a fixed format. Thus an attacker can obtain some bytes of the pseudorandom string at the beginning by computing the xor of the initial bits of the encryption and the known parts of the header. Perhaps he needs to guess where exactly the header is placed, but this is possible in reasonable time. If he was able to predict the behavior of the pseudorandom generator, then he would be able to decrypt the remaining parts of the message.

Slightly more formally, this means that if an attacker sees an initial prefix of the pseudorandom string generated by G, then he is not able to predict the next bit with probability better than $\frac{1}{2}$, i.e., the probability of purely guessing the bit, apart from a very small, so-called *negligible* advantage

ϵ. Before we give the rigorous definition of (next-bit) unpredictability, we first rigorously define the notions of efficient algorithms and negligible functions. Both constitute central concepts in modern cryptography.

Definition 2.2 (Efficient Algorithm) *Let $k \in \mathbb{N}$ be the security parameter. An algorithm A is* efficient *or* efficiently computable *iff it is computable in probabilistic polynomial-time (in k). We write $\mathsf{A} \in PPT$ if A is efficiently computable.* $\diamond$

A function is negligible if it decreases faster than the inverse of any polynomial. Formally we define it as follows:

Definition 2.3 (Negligible Function) *A function $f : \mathbb{N} \to \mathbb{R}_0^+$ is* negligible *iff for all $c \in \mathbb{N}$ there exists an $n_c \in \mathbb{N}$ such that for all $n \geq n_c$ we have*

$$f(n) \leq \frac{1}{n^c}.$$

$\diamond$

We are now ready to give the definition of next-bit unpredictability. For the sake of readability, we only address the case that the keystream alphabet is the set of all strings of a certain length k; similarly, we expand these strings to strings of length $p(k)$ where p is an arbitrary but fixed polynomial.

Definition 2.4 (Next-bit Unpredictability) *A deterministic efficient function $\mathsf{G} : \{0,1\}^k \to \{0,1\}^{p(k)}$ for a polynomial p is* (next-bit) unpredictable *iff for every $0 \leq i \leq p(k)$ and every attacker $\mathsf{A} \in PPT$, there exists a negligible function ϵ such that*

$$\begin{array}{lll} P[& b = b_i; & \# \textit{ The probability that } \mathsf{A}\textit{'s guess is correct} \\ & x \leftarrow_{\mathcal{R}} \{0,1\}^k, & \# \textit{ if the seed } x \textit{ is drawn uniformly at random} \\ & b_1 \ldots b_{p(k)} \leftarrow \mathsf{G}(x), & \# \textit{ if } \mathsf{G} \textit{ generated } p(k) \textit{ bits using the seed } x \\ & b \leftarrow \mathsf{A}(b_1 \ldots b_{i-1}) \,] & \# \textit{ and if } \mathsf{A} \textit{ guesses } b_i \textit{ knowing bits } b_1, \ldots, b_{i-1} \\ & \leq \frac{1}{2} + \epsilon(k). & \# \textit{ is only negligibly larger than pure guessing.} \end{array}$$

$\diamond$

2.3 Getting True Randomness in Practice

The seed for a pseudorandom generator, and keys for symmetric encryption schemes in general, should be as random as possible. One uses for example physical random number generators to get good randomness.

There are some physical sources that are supposed to produce good randomness, but the resulting bits may have a certain bias or some correlation. One usually circumvents this by taking the xor of bits obtained from different such sources, and by subsequently applying a randomness extractor further improving the quality of the output. An example for such a randomness extractor is the Von Neumann Correction.

Typical physical sources of randomness include:

- Thermal noise in various electric circuits, e.g., Zener-Diodes, optical sensors,

- atmospheric noise, captured by antennas,
- radioactive radiation, and
- more playfully, lava lamps. There is even a patent for creating random bits with lava lamps (US patent #5,732,138).

There is also a number of random sources that are more easily available, at the cost of producing weaker randomness. What is used also in practice is

- measurement of times between user key-strokes, and
- time needed to access different sectors on the hard-disk drive (the air turbulences caused by the spinning disk is supposed to be random).

2.4 Attacks Against the One-time Pad and Stream Ciphers, and How to Defend Against Them

Recall that the One-time Pad has perfect secrecy and thus is not vulnerable to a ciphertext-only attack. However, it turns out that the One-time Pad as well as stream ciphers are vulnerable to stronger attacks in which the attacker does more than passively easvesdropping on the wire; moreover, they are vulnerable against improper use of the scheme.

2.4.1 The Two-time Pad

One of the most important improper usages of the One-time Pad and stream ciphers is to re-use the random tape, i.e., to encrypt several messages with the same key. This may be called the "Two-time Pad", which is fully insecure. Assume that the adversary is given two encryptions $c_i = K \oplus m_i$ for $i \in \{1, 2\}$ for two messages m_1, m_2 with the same key K. Then the adversary simply computes the xor of c_1 and c_2 yielding:

$$c_1 \oplus c_2 = (m_1 \oplus K) \oplus (m_2 \oplus K) = m_1 \oplus m_2$$

Thus re-using the key leaks the XOR of the actual plaintexts. Assuming that both messages contain ordinary text or have some other form of exploitable redundancy, we can again use frequency analysis to recover the plaintexts m_1 and m_2 from $m_1 \oplus m_2$. Thus one has to be careful not to re-use a key when using the One-time Pad or stream ciphers. There are mainly two methods to realize this:

- One might use successive parts of the output stream to encrypt successive messages. This requires synchronization of the senders and the receivers streams by some means, usually by transmitting its position along with the encrypted message. This has disadvantages if the order of messages is changed in the transmission line or by the protocol.
- One might create a new seed for each message or file that needs to be encrypted. Then one additionally transmits the seed along with the message. Of course, the seed has to be transmitted secretly somehow. This can be done by combining the stream cipher with another cipher E^* to transmit the seed x as follows:

$$\mathsf{E}^*(x) || \mathsf{G}(x) \oplus m.$$

This has the advantage that a fast stream cipher can be used to encrypt the potentially very long message m, while a slower encryption scheme E^* is only needed to encrypt the shorter seed.

2.4.2 Active Attacks against the One-time Pad and Stream Ciphers

If an adversary not only reads messages but also is able to change messages, then attacks such as the following become possible. Consider a simple voting system, where each participant votes 0 or 1. This vote is encrypted using the OTP, i.e., each participant i shares a random bit $K_i \leftarrow_{\mathcal{R}} \{0,1\}$ with a central voting authority. Every participant sends the ciphertext $c_i \leftarrow K_i \oplus m_i$ to the central authority. The voting authority then computes $c_i \oplus K_i = m_i$ and counts all votes. Note that this is a very weak form of voting scheme since the voting authority can see how everybody voted, and it is provided for the sake of demonstration only.

Now assume that the attacker knows (e.g., by means of an opinion poll) that a majority of all participants are going to vote 0. Then by intercepting a vote c_i and replacing it with $c_i^* := c_i \oplus 1$ the attacker is able to invert the outcome of the vote, as these ciphertexts decrypt to

$$c_i^* \oplus K_i = (m_i \oplus K_i \oplus 1) \oplus K_i = m_i \oplus 1.$$

This attack applies to both the One-time Pad and stream ciphers. One says that the One-time Pad and stream ciphers are highly *malleable*, i.e., they offer no form of integrity of the plaintexts. Thus both ciphers should only be used with suitable integrity protection mechanisms. These are called MACs (message authentication codes) and will be discussed later in class.

2.5 Examples of Pseudorandom Generators

2.5.1 RC4

RC4 is a stream-cipher invented by Ron Rivest in 1987 for RSA Security, which also holds the trademark for it. The source code was originally not released to the public because it was a trade secret, but was posted to a newsgroup some time ago; thus people referred to this version as *alleged RC4*. Today it is known that alleged RC4 indeed equals RC4. While RC4 does not satisfy next-bit unpredictability, it is considered secure from a practical point of view if one takes certain precautions. It is used in many protocols such as SSL/TLS and 802.11b WEP.

Its main data structure is an array S of 256 bytes. The array is initializes to the identity before any output is generated, i.e., the first cell is initialized with 0, the second with 1 and so on. Then the cells are permuted using certain operations that depend on the current state and the chosen key K. In pseudocode, the RC4 initialization phase works as follows:

```
for i from 0 to 255
     S[i] := i
j := 0
for i from 0 to 255
     j := (j + S[i] + K[i mod keylength]) mod 256
     swap(S[i],S[j])
```

After initialization has been completed, the following procedure computes the pseudorandom sequence. For each output byte, new values of the variables i, j are calculated, the corresponding

cells are swapped, and the content of a third cell is output. This described again in pseudocode as follows:

```
i := 0
j := 0
while GeneratingOutput:
     i := (i + 1) mod 256
     j := (j + S[i]) mod 256
     swap(S[i],S[j])
     output S[(S[i] + S[j]) mod 256]
```

One easily sees that the first byte which is output depends on the content of 3 cells only. This property can be used to launch attacks against the cipher, so one usually discards the first 256 bytes of output generated by this algorithm to prevent these attacks.

2.5.2 Linear Feedback Shift Register (LFSR)

Linear Feedback Shift Registers constitute a very fast way of generating pseudorandom strings. An LFSR consists of a shift register that shifts its content for one bit at each clock, and a function that determines the feedback, i.e., the next value that enters the shift register, as a function of its current state. For LFSRs this is a linear function, i.e., the xor of certain specified bits in the state. The initial state of the shift register comprises the seed. An schematic sketch of an LFSR is given in Figure 2.1. The security of an LFSR is determined by its feedback function and how it is interlinked

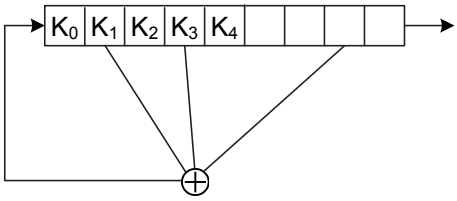


Figure 2.1: Linear Feedback Shift Register

with the surrounding operations. There also exist constructions with non-linear feedback. Note that LFSRs on their own cannot naively be used as stream ciphers because the first bits that are output are precisely the key bits which would make any stream cipher insecure. The solution taken here is that one throws away the first bits and subsequently combines LFSRs in some clever way either with other LFSRs or with other cryptographic primitives. For instance, one encrypts the output of an LFSR using a secure block cipher (see the next lecture), which yields a pseudorandom sequence provided as long as the bits output by an LFSR do not repeat themselves.

2.5.3 The Content Scrambling System (CSS)

We conclude with a brief description of the Content Scrambling System (CSS), which is a proprietary standard for encrypting the contents of DVDs. The key management is rather complex and we do not go into details here. The actual content is encrypted with a stream cipher whose corresponding

PRG, i.e., the generation of the pseudorandom sequence, will be described in the sequel. Figure 2.2 illustrate the PRG.

Each sector key consists of 5 bytes $K_0, \ldots, K_4$. The pseudo random generator comprises two LFSRs that operate as follows. The first LFSR has a length of 17 bits and is initialized with $1||K_0||K_1$, where the feedback is computed as the xor of bits 1 and 15. The second LFSR has a length of 25 bits initialized as $1||K_2||K_3||K_4$, where the feedback is calculated as the xor of bits 11, 19, 20, and 23. Both LFSRs are clocked eight times, each of them producing eight bits of output. This output is treated as bytes, added modulo 256 observing the carry bit from the previous addition. The resulting output is used to encrypt the actual data.

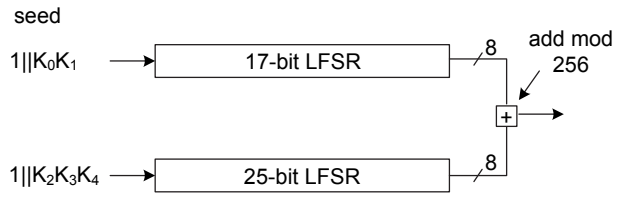


Figure 2.2: Content Scrambling System

The most apparent weakness of CSS is its short key size of 40 bits. This is a consequence of US export regulations that when when CSS was standardized. Brute-force attacks are reported to take 17 hours on a Celeron 366, i.e., on a very old hardware, and thus already much faster on more recent hardware. More sophisticated attacks exist that reduce the complexity of attacks to even only 2^{16}, thus taking only several seconds until CSS is successfully attacked.

3. Block Ciphers

Block ciphers are symmetric ciphers operating block-wise, i.e., on bitstrings of a fixed length. Common block sizes are 64, 128, or 256 bits.

3.1 Feistel Networks

Feistel networks are a particular structure for designing symmetric encryption schemes. They were described first by Horst Feistel in the context of the Lucifer cipher while working at IBM. Lucifer was the predecessor of the Data Encryption Standard (DES), which is built upon the same design. Other ciphers using Feistel networks include IDEA, RC5, Skipjack, and others.

A Feistel network parameterized by *round functions* $f_1, \ldots, f_d$ is a function $\{0,1\}^{2n} \to \{0,1\}^{2n}$ which adheres to the following program: It operates in d *rounds*, typically between 12 and 16, where the i-th round executes the following operations: (i) split the round input into two halves $L_{i-1}||R_{i-1}$, (ii) apply the round function f_i to the right half yielding $f_i(R_{i-1})$, (iii) compute the xor of this value with the left half $L_{i-1} \oplus f_i(R_{i-1})$, and (iv) swap the left and right side, thus yielding the round output $R_{i-1}||L_{i-1} \oplus f_i(R_{i-1}) =: L_i||R_i$. This process is depicted in Figure 3.1.

Feistel networks are appealing because of their simple design. The round functions f_i can be any functions; in particular they need not be invertible, but still the following proposition is obtained.

Proposition 3.1 (Feistel Networks are Permutations) *Every Feistel netweork $F : \{0,1\}^{2n} \to \{0,1\}^{2n}$ constitutes a permutation.*

Proof. First, we show how to invert a single round i. Let E_i denote the operations executed at the i-th round. We define the candidate inverse function D_i for input $L_i||R_i$ as $R_{i-1} := L_i$ and

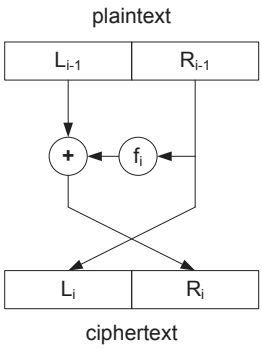


Figure 3.1: One Round in a Feistel Network

$L_{i-1} := R_i \oplus f_i(L_i)$. The following simple calculation shows that this is indeed the inverse function:

$$\begin{aligned}
\mathsf{D}_i(\mathsf{E}_i(L_{i-1}||R_{i-1})) &= \mathsf{D}_i(R_{i-1}||L_{i-1} \oplus f_i(R_{i-1})) \\
&= (L_{i-1} \oplus f_i(R_{i-1})) \oplus f_i(R_{i-1})||R_{i-1} \\
&= L_{i-1} \oplus (f_i(R_{i-1}) \oplus f_i(R_{i-1}))||R_{i-1} \\
&= L_{i-1}||R_{i-1}
\end{aligned}$$

The inverse of a d-round encryption may be straightforwardly defined by inverting the order in which the round functions f_i are applied:

$$\begin{aligned}
& \mathsf{D}_1(\mathsf{D}_2(\ldots \mathsf{D}_{d-1}((\mathsf{D}_d(\mathsf{E}_d(\mathsf{E}_{d-1}(\ldots \mathsf{E}_1(m))\ldots))\ldots)) \\
= \; & \mathsf{D}_1(\mathsf{D}_2(\ldots \mathsf{D}_{d-1}((\mathsf{E}_{d-1}(\ldots \mathsf{E}_1(m))\ldots))\ldots)) \\
= \; & \ldots \\
= \; & \mathsf{D}_1(\mathsf{E}_1(m)) \\
= \; & m
\end{aligned}$$

■

Feistel networks have the additional nice property that the same hardware circuit can be used for both encryption and decryption. This was particularly important in the 60ies and 70ies when the first block ciphers were designed, and when hardware was still much more expensive. There are two minor modification when a ciphertext should be decrypted in a Feistel network, but these do not affect the network itself but the handling of inputs of the circuit, i.e., messages and inputs of round functions. Namely (i) the round functions need to be applied in different order, and (ii) writing $c = L_d||R_d$ as the ciphertext, one inputs $R_d||L_d$ to the network for decryption, i.e., one exchanges the left and the right half of the ciphertext. Then one can easily verify that the round functions compute the decryption operation as defined in the above proof.

3.2 The Data Encryption Standard (DES)

For a long time DES was one of the most widely applied block ciphers. It was designed by IBM in collaboration with the NSA and published as a standard in FIPS PUB 46-3. There were rumors about weaknesses the NSA had built into it, but until now no evidence was found for this. The main point of criticism against DES was its limited keysize, which is nowadays the reason why pure DES is no longer used. In fact, the first DES break, in the sense of a total break given a certain number of plaintext/ciphertext pairs, was reported in 1997 by the DESCHALL project and took about 3 month. In 1998 The Electronic Frontier Foundation (EFF) built a specialized hardware device resulting in a break in roughly three days. Later they where cooperating with distributed.net breaking a challenge in 22 hours. The world record for breaking a DES encryption is currently 10 hours, and people are getting faster. A variant called Triple-DES (3DES) which we will discussed later in this chapter is still deployed and seems to provide good security.

3.2.1 Overall Construction of DES

DES is essentially a Feistel network of 16 rounds operating on blocks of 64 bits and using 56-bit keys. What makes DES different from a pure Feistel design is the initial permutation IP which is

applied before the first round starts, and whose inverse IP^{-1} is applied after the last round. The

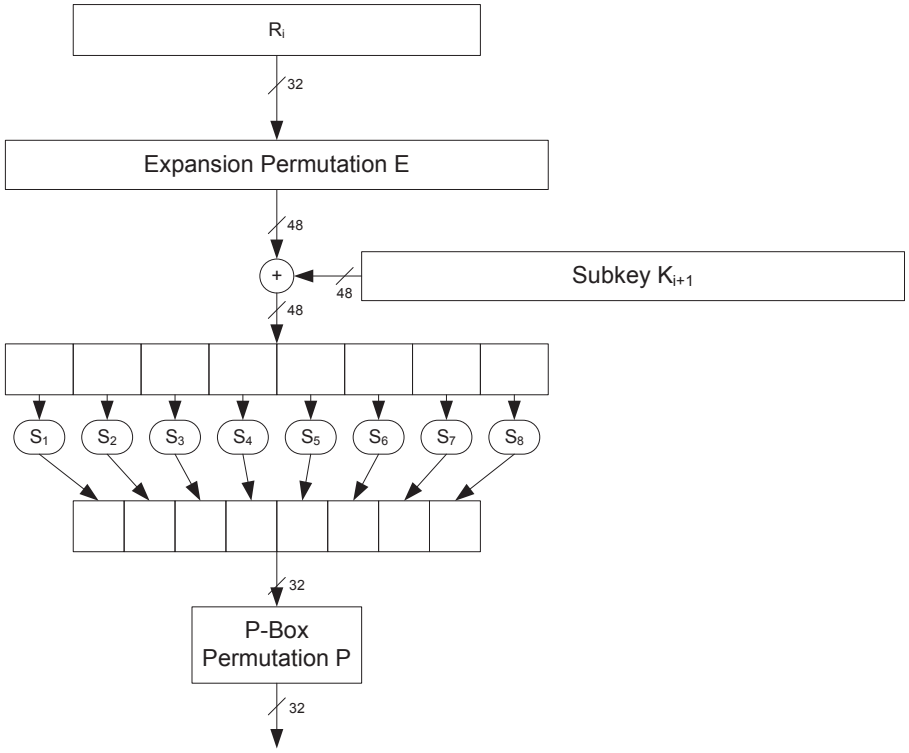


Figure 3.2: The DES round function.

DES round function f_i is depicted in Figure 3.2. Each round behaves as follows:

1. Expand the 32-bit message block to a 48-bit block by doubling 16 bits and permuting them as defined by the function E shown in Figure 3.4.

2. Compute the xor of that value with the round-key K_i which is derived from the key schedule as defined below;

3. Split the 48-bit into eight 6-bit blocks. Each of them is received as input by one of the eight *S-boxes* $S_1, \ldots, S_8$ as shown in Figure 3.5. Each S-box yields a 4-bit output, thus giving together yielding a 32-bit block.

4. The permutation P defined in Figure 3.6 is applied to the 32-bit block, obtaining the output of f_i.

3.2.2 Key schedule

The choice of the key used in each round is called *key-schedule*. Even if a DES key is composed of 64 bits, 8 bits are used for error detection, thus resulting in an actual key-size of 56 bits. In the first step, the parity bits are stripped off a 64-bit key K, and the bits are permuted by a fixed

IP:															
58	50	42	34	26	18	10	2	60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6	64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1	59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5	63	55	47	39	31	23	15	7

Figure 3.3: Initial permutation (IP) in DES (The first bit of the output is the 58-th bit of the input, the second bit of the output is the 50 bits of the input, and so on.)

E:					
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Figure 3.4: Bit-selection function E (The first bit of output is the 32-nd bit of the input, the second bit of output is the first of the input and so on.)

permutation PC-1, as shown in Figure 3.7. The output is divided into a 28-bit left half C_0 and a 28-bit right half D_0. Now for each round we compute

$$C_i = C_{i-1} \lll p_i$$
$$D_i = D_{i-1} \lll p_i$$

where $x \lll p_i$ means a cyclic shift on x to the left by p_i positions with $p_i = \begin{cases} 1 & \text{if } i = 1, 2, 9, 16 \\ 2 & \text{otherwise} \end{cases}$. Finally, C_i and D_i are joined together and permuted according to PC-2, obtaining the 48-bit round key. This permutation is reported in Figure 3.7.

3.2.3 The Security of DES

DES withstood attacks quite successfully apart from some attacks based on linear and differential cryptanalysis which we shall discuss later. However, the major weakness of DES is its limited keysize of 56 bits which is nowadays not enough to provide a reasonable level of security. For this reason, several improvements on the plain DES have been proposed: we will see some of them in the following lectures and in the homework sheets.

S_1:															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	O	6	13
S_2:															
15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S_3:															
10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S_4:															
7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S_5:															
2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	O	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S_6:															
12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S_7:															
4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S_8:															
13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Figure 3.5: DES S-Boxes (For a binary string $x_0x_1x_2x_3x_4x_5x_6x_7$, the output is computed by looking up the entry in row x_0x_7 and column $x_1x_2x_3x_4x_5x_6$. This representation has historic reasons.)

P:							
16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

Figure 3.6: DES permutation P (first bit of the output is 16-th bit of the input.)

PC-1:						
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

PC-2:					
14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Figure 3.7: DES Key schedule permutations PC-1 and PC-2

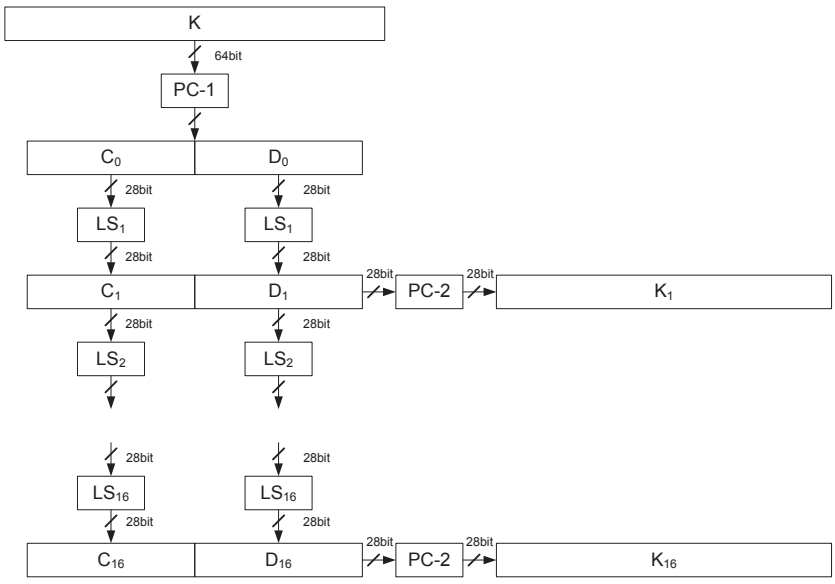


Figure 3.8: Key scheduling for DES

3.3 The Advanced Encryption Standard (AES)

The *advanced encryption standard* (AES) is the successor of the outdated DES standard. It was developed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen, and standardized as US FIPS PUB 197 in November 2001. It works on blocks of 128 bits, supports key sizes of 128, 192, and 256 bits and operates in 10, 12 and 14 rounds, respectively. AES is motivated by algebraic operations, and its implementation in hardware and software is compact and fast.

3.3.1 Construction

AES operates on an array of 4x4 bytes, which is initialized with a message $m = m_0||m_1||\ldots||m_{15}$ as follows:

m_0	m_4	m_8	m_{12}
m_1	m_5	m_9	m_{13}
m_2	m_6	m_{10}	m_{14}
m_3	m_7	m_{11}	m_{15}

$\rightarrow$

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

Each round applies the following manipulations to the state (i.e., the array):

1. a substitution operation on every single byte `SubBytes()`,
2. a byte permutation `ShiftRows()`,
3. a column manipulation `MixColumns()`
4. and an xor of the state with the round key `AddRoundKey()`.

An exception is the last round, where the `MixColumns()` operation is skipped. Now we discuss these operations in detail.

`SubBytes()`: This operation is similar to the S-Box substitution of DES. Each byte $a_{i,j}$ of the state is substituted by the output of a single S-Box. This S-Box has also a nice algebraic representation, however, we do not want to develop all the theory and simply give it in the form of a look-up Table 3.9. This is also used in implementations in which spending space to a fully specified table is not a major concern since table look-up is significantly times faster than on-the-fly calculations.

`ShiftRow()`: The lower three rows are shifted by one, two , and three positions, respectively.

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

$\rightarrow$

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,0}$
$a_{2,2}$	$a_{2,3}$	$a_{2,0}$	$a_{2,1}$
$a_{3,3}$	$a_{3,0}$	$a_{3,1}$	$a_{3,2}$

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figure 3.9: S-box: substitution values for the byte xy (in hexadecimal format).

`MixColumns()`: The MixColumns operation replaces each column i as follows:

$$\begin{pmatrix} a'_{0,i} \\ a'_{1,i} \\ a'_{2,i} \\ a'_{3,i} \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} a_{0,i} \\ a_{1,i} \\ a_{2,i} \\ a_{3,i} \end{pmatrix},$$

where the multiplication is calculated in $GF(2^8)$.

`AddRoundKey()`: The round key is represented as a 4×4 matrix with bytes as elements and xored bit-wise with the current state.

3.3.2 Key Schedule

The round keys are derived from the main key by successively using functions `RotWord()` rotating a whole word and `SubWord()` applying the sbox of Figure 3.9 to four bytes in parallel. It is described in detail in FIPS PUB 197.

3.3.3 Security of AES

AES can, to a certain extend, be attacked with linear and differential cryptanalysis. However, these attacks can only reduce the effective keysize of AES with 128 bit key only by a few bits, i.e., AES currently yields a very comfortable degree of security.

It is worth to note that AES with 256-bit keys is even used for protecting "top secret" (highest NSA secrecy level) documents of the NSA. It is the first time in history that a publicly known algorithm is used for highly classified documents.

3.4 Attacks on Block Ciphers

3.4.1 Exhaustive Key Search

The conceptually simplest attack which is possible for any block cipher is exhaustive key search, also called brute force attack. Knowing a few plaintext/ciphertext pairs $(m_1, c_1), (m_2, c_2), \ldots$ (i.e. $c_i = \mathsf{E}(K, m_i)$), one tries to find the key K by testing every possible key.

To estimate the effectiveness of this attack we need to know how many keys would encrypt a fixed plaintext to a specific ciphertext. We will give an estimation for this in the sequel, where we replace the block cipher with a random function for every key. While this might not be accurate, it makes use of the intuition that the encryption function should essentially constitute a random. One says that the block cipher is considered an *ideal cipher*. This heuristic is used quite often, as an accurate analysis of, e.g., DES is far too complex. We will use this concept in a somehow intuitive manner and define it precisely later in the course.

We prove the following statement: For a randomly chosen $m \in \mathcal{M} = \{0,1\}^{64}$ and a randomly chosen key $K \in \{0,1\}^{56}$, the probability that there exists another key encrypting the message to the same ciphertext is very low. One also says that the *unicity distance*, i.e., the probability that a key is already uniquely determined after seeing one plaintext/ciphertext pair, is very high.

Lemma 3.1 *Let* (E, D) *be an ideal cipher, i.e., for each key* K*, the function* $\mathsf{E}(K, \cdot)$ *is "as god as" a randomly chosen function with the specific domains. Then*

$$\mathsf{Pr}\left[\exists K' \neq K : \mathsf{E}(K, m) = \mathsf{E}(K', m);\ m \leftarrow_{\mathcal{R}} \mathcal{M},\ K \leftarrow_{\mathcal{R}} \mathcal{K}\right] \leq \frac{1}{256}$$

□

Proof.

$$\begin{aligned}
& \mathsf{Pr}\left[\exists K' \neq K : \mathsf{E}(K, m) = \mathsf{E}(K', m);\ m \leftarrow_{\mathcal{R}} \mathcal{M},\ K \leftarrow_{\mathcal{R}} \mathcal{K}\right] \\
\overset{(1)}{\leq} & \sum_{K' \in \{0,1\}^{56}} \mathsf{Pr}\left[\mathsf{E}(K, m) = \mathsf{E}(K', m) \wedge K' \neq K;\ m \leftarrow_{\mathcal{R}} \mathcal{M},\ K \leftarrow_{\mathcal{R}} \mathcal{K}\right] \\
\overset{(2)}{\leq} & \sum_{K' \in \{0,1\}^{56}} \frac{1}{2^{64}} \\
\leq & \frac{2^{56}}{2^{64}} \\
= & 2^{-8}
\end{aligned}$$

For inequality (1) we exploited that the probability of a union is always less of equal than the sum of the probabilities of all elements of the union. Transformation (2) holds because E is an ideal cipher. This means that $\mathsf{E}(K, m)$ and $\mathsf{E}(K', m)$ are both uniformly random in $\{0,1\}^{64}$, thus the probability that they are equal is $\frac{1}{2^{64}}$. ∎

Thus the unicity distance of DES is at least $1 - 2^{-8}$.

Exhaustive Key Search in Practice: The DES Challenge was put forward by RSA Security to encourage research on the security of DES. A reward of 10000$ was offered for solving the challenge, i.e., for computing the key that was used to encrypt a specific plaintext/ciphertext pair of the form "`The unknown message is: ----`". In 1997 the DESCHALL project needed about 3 month to break the DES challenge with a distributed search. In 1998 The Electronic Frontier Foundation (EFF) built the specialized hardware device "Deep Crack" that was able to break DES keys in roughly three days, at rather moderate costs of about 250.000$. While this is certainly too expensive for individuals this is reasonable for large organizations or governments. Later the EFF and distributed.net together broke the challenge in 22 hours.

3.4.2 Linear Cryptanalysis

Suppose messages $m \in \mathcal{M}$ and keys $K \in \mathcal{K}$ are drawn uniformly random from their respective sets. Suppose you know $i_1, \ldots, i_r, j_1, \ldots, j_s, k_1, \ldots, k_t$ such that

$$\mathsf{Pr}\left[m^{(i_1)} \oplus \cdots \oplus m^{(i_r)} \oplus c^{(j_1)} \oplus \cdots \oplus c^{(j_s)} \oplus K^{(k_1)} \oplus \cdots \oplus K^{(k_t)} = 1\right] \geq \frac{1}{2} + \epsilon,$$

where $m^{(i)}, c^{(i)}, K^{(i)}$ denote the i-th bits of the bitstring m, the ciphertext c and the key K, respectively. Such relations can be found for DES with $\epsilon \approx 2^{-21}$. If one gets enough plaintext/ciphertext pairs, one can obtain partial information about the key. Namely for $1/\epsilon^2$ plaintext/ciphertext pairs (m_l, c_l) one has

$$\mathsf{Pr}\left[K^{(k_1)} \oplus \cdots \oplus K^{(k_t)} = \mathsf{MAJ}\left(m_l^{(i_1)} \oplus \cdots \oplus m_l^{(i_r)} \oplus c_l^{(j_1)} \oplus \cdots \oplus c_l^{(j_s)} \mid l = 1, \ldots, 1/\epsilon^2\right)\right] \geq 97.7\%.$$

where MAJ denotes the majority function, taken over all xors computed from any given plaintext/ciphertext pair. Obtaining 2^{42} plaintext/ciphertext pairs for DES thus allows one to get 14 bits of information of the key by these techniques. This enables an attacker to reduce the complexity of an exhaustive key search to 2^{42} DES operations, as one only has to test keys fulfilling the relation.

An even more involved method to break block ciphers is *differential cryptanalysis*, which we will not cover here.

3.4.3 Side-Channel Attacks

Side channel attacks do not attack an algorithm by its input/output behavior. Instead they use concrete implementations of an algorithm and seek to find side-channels that leak valuable information. Examples for side-channels that have been successfully exploited include time needed for encryption, electric power consumption of processors, and the electromagnetic emanation of a device. Some of these attacks were able to completely break a system in seconds, and several smartcards have been withdrawn after it was discovered that they were highly vulnerable to such attacks.

One of the most famous side channel attacks is due to Paul Kocher, who measured electric power consumption of microprocessors. He showed that by inspection of power consumption curves one can identify conditional branches the code took, and if these conditions depend on the secret key, that several popular algorithms can easily be broken.

3.5 Strengthening DES

Several ways were proposed for strengthening DES and, in particular, for solving its major problem, the short key-size. In the following, we shall present some of them.

3.5.1 Double-DES (2DES)

A first attempt on strengthening DES consisted in applying the encryption operation two times with different keys (K_1, K_2), thus computing $\mathsf{E}(K_1, \mathsf{E}(K_2, m))$. While the original intention was doubling the effective key-size, Double-DES does not improve much on the security of DES because of a "meet-in-the-middle" attack, which can be summarized as follows. Assume that we are given a plaintext/ciphertext pair (m, c), i.e., $c = \mathsf{E}(K_1, \mathsf{E}(K_2, m))$ for some K_1 and K_2:

1. Decrypt the given ciphertext c with every possible key $K_2^{(i)}$, where $1 \leq i \leq 2^{56}$. This yields a table with entries $(K_2^{(1)}, \mathsf{D}(K_2^{(1)}, c)), (K_2^{(2)}, \mathsf{D}(K_2^{(2)}, c)), \ldots$. This step requires 2^{56} executions of DES.

2. Sort this table with respect to the second entry. This takes steps in the order of $2^{56} \log(2^{56})$.

3. Encrypt the message m with any possible key K_1 and look up the resulting encryption in the second column of the table, until a matching ciphertext $\mathsf{E}(K_1, m)$ is found. Let i be the row containing such a ciphertext. This step takes at most 2^{56} executions of DES.

4. Let $K_2^{(i)}$ denote the key in the first column of the table at row i. Then the correct key is $(K_1, K_2^{(i)})$.

Since the sorting step does not produce a noticeable overhead over execution DES 2^{56} times, we ignore its cost in the overall complexity. Then one sees that the total number of DES executions is $2 \cdot 2^{56} = 2^{57}$, so the *effective key-length* of Double-DES is at most 57 bits. Since there is almost no improvement over DES, Double-DES is in fact never used.

3.5.2 Triple-DES (3DES)

A very common variant of DES is Triple-DES (3DES); in fact, this construction can be applied to any block cipher to increase its effective key-size. One calculates 3DES as follows: given three independent DES keys K_1, K_2, K_3, one computes

$$\mathsf{E}^{\mathsf{3DES}}((K_1, K_2, K_3), m) := \mathsf{E}(K_1, \mathsf{D}(K_2, \mathsf{E}(K_3, m))).$$

Decryption is given by

$$\mathsf{D}^{\mathsf{3DES}}((K_1, K_2, K_3), m) := \mathsf{D}(K_3, \mathsf{E}(K_2, \mathsf{D}(K_1, m))).$$

The keysize is $3 \cdot 56 = 168$ bits. However, a meet-in-the-middle attack is possible:

- As for Double-DES, compute a table with entries $(K_3^{(1)}, \mathsf{D}(K_3^{(1)}, c)), (K_3^{(2)}, \mathsf{D}(K_3^{(2)}, c)), \ldots$.
- Sort this table with respect to the second entry.
- For every possible pair of keys (K_1, K_2), encrypt the message m into $\mathsf{E}(K_1, \mathsf{D}(K_2, m))$ and look up this value in the table. This step needs 2^{112} DES executions.

As a consequence, the effective key-length of 3DES is at most 112 bits.

Figure 3.10: Encryption and Decryption in ECB Mode

3.5.3 DESX

Another variant of DES is DESX, which is not that widely used although it provides an even larger effective key size than 3DES and is even roughly three times faster.

A key is a triple (K_1, K_2, K_3) where $K_1, K_3 \in \{0,1\}^{64}$ and $K_2 \in \{0,1\}^{56}$. Thus K_2 is a DES key and K_1 and K_3 are random binary strings of the same length as the message blocks. Encryption and decryption are defined as

$$\mathsf{E}^{\mathsf{DESX}}((K_1,K_2,K_3),m) := K_3 \oplus \mathsf{E}(K_2, K_1 \oplus m), \text{ and}$$
$$\mathsf{D}^{\mathsf{DESX}}((K_1,K_2,K_3),c) := K_1 \oplus \mathsf{D}(K_2, K_3 \oplus c).$$

The following theorem was given by Kilian and Rogaway in 1998; it proves that DESX has an effective key size of at least 119 bit. The proof is omitted and can be found in the original paper.

Theorem 3.1 (Kilian, Rogaway 1998) *Let* (E,D) *be an ideal cipher with keysize* l *and blocksize* b. *Let* $(K_1,K_2,K_3) \in \{0,1\}^{2b+l}$, *and let*

$$\mathsf{EX}((K_1,K_2,K_3),m) := K_3 \oplus \mathsf{E}(K_2, K_1 \oplus m)$$

a cipher having keysize $k = 2b+l$. *Then the effective key size of* EX *is at least* $k-b-1 = b+l-1$ *bit. In particular, the effective key size of* DESX *is at least* $64+56-1 = 119$ *bits.* □

3.6 Modes of Operation

Given a block-cipher (E,D), there are several ways it can be used to actually encrypt long messages. FIPS PUB 81 defines four modes of operation (ECB, CBC, OFB, CFB), which can be applied to any block cipher. We will review them along with the counter mode (CTR) in the sequel.

3.6.1 Electronic Codebook mode (ECB)

The ECB mode corresponds to the "naive" use of a block cipher (E,D). The message m is split into a sequence $m_1, \ldots, m_t$ of blocks of the length that is handled by E, and each block m_i is encrypted with the same key K. Thus we obtain $c_i := \mathsf{E}(K, m_i)$ and $c_1, \ldots, c_t$ is the resulting encryption (cf. Figure 3.10).

The main weakness of ECB is that identical blocks m_i are encrypted to the same ciphertext c_i. This is especially a strong weakness if messages come from a source with low entropy, e.g., securely

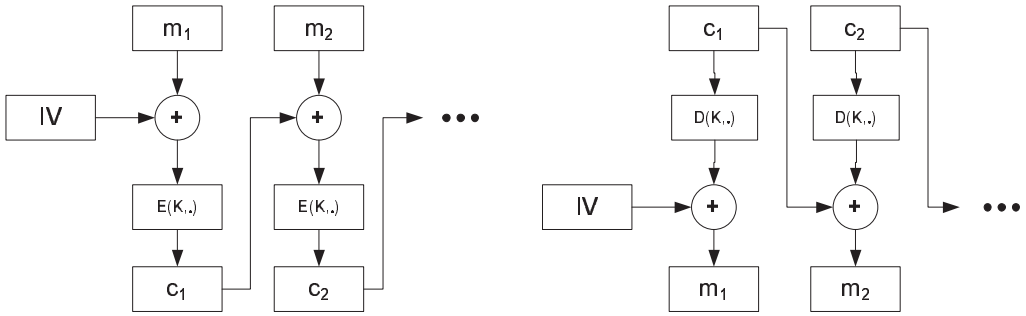


Figure 3.11: Encryption and Decryption in CBC Mode

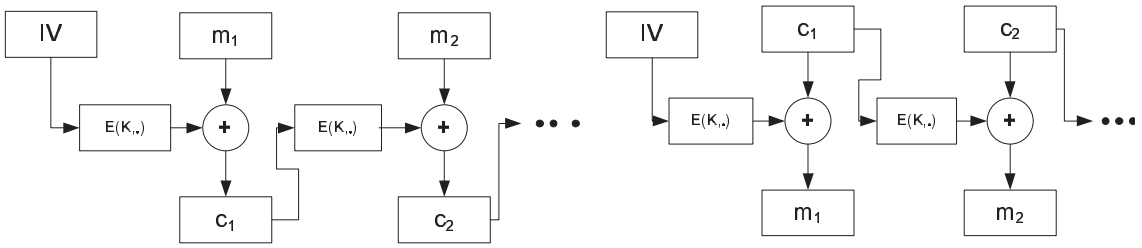


Figure 3.12: Encryption and Decryption in CFB Mode

encrypting (uncompressed) images is not possible using ECB. On the positive side, if one block c_i gets corrupted due to unreliable channels, then only one block is affected while all other blocks can still be decrypted.

3.6.2 Cipher Block Chaining Mode (CBC)

Again, the message m is split into a sequence $m_1, \ldots, m_t$. Encryption is then performed by the following operations:

$$\begin{aligned} c_1 &= \mathsf{E}(K, (m_1 \oplus IV)) \\ c_i &= \mathsf{E}(K, (m_i \oplus c_{i-1})), \text{ for } i > 1 \end{aligned}$$

The CBC mode is depicted in Figure 3.11. We do not give the formal definition in symbols again as they are straightforwardly derivable from the figure, similar for the following modes of operation. Each c_i is xored with the next block of the plaintext: A one-bit error in the ciphertext gives not only a one-block error in the corresponding message block but also a one-bit error in the next decrypted plaintext block. In contrast of the ECB mode, an initial value IV is passed to the encryption function for the first block. The initial value is used to ensure that two encryptions of the same plaintext yield different ciphertexts.

3.6.3 Cipher Feedback Mode (CFB)

This mode turns a block cipher into a stream cipher. A one-bit error in the ciphertext causes a one-bit error in the corresponding plaintext block and a block-error in the next decrypted plaintext

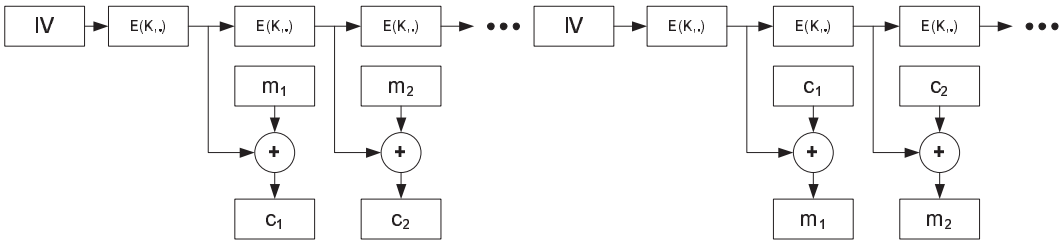


Figure 3.13: Encryption and Decryption in OFB Mode

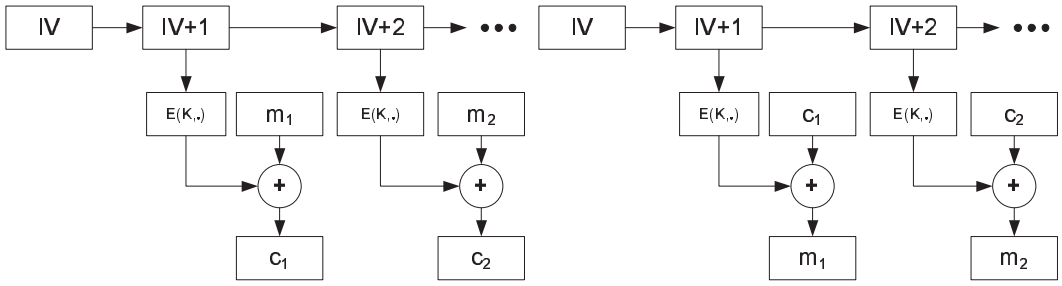


Figure 3.14: Encryption and Decryption in CTR Mode

block, i.e., the opposite of what happens in the CBC mode. The CFB is depicted in Figure 3.12.

3.6.4 Output Feedback Mode (OFB)

Similar to CFB, OFB mode enables a block cipher to be used as a stream cipher. It has the property that a one-bit error in the ciphertext gives only a one-bit error in the decrypted ciphertext.

3.6.5 Counter Mode (CTR)

Like CFB and OFB, counter mode turns a block cipher into a stream cipher. It generates each block of the stream cipher by encrypting successive values of a "counter". The counter can be any simple function which produces a sequence that is guaranteed not to repeat for a long time. However using an actual counter is the simplest and most popular choice. CTR is reported in Figure 3.14.

4. PRPs, PRFs, Semantic Security

This chapter is dedicated to central concepts of cryptography. We will introduce a new definition of security, so-called *semantic security* in different variants, which is the standard definition in modern cryptography.

4.1 Definitions and Basic Properties

Before investigating semantic security we will introduce the concepts of pseudo-random permutations (PRPs) and functions (PRFs). These are two important concepts and will enable us to prove security of certain modes of operations according to the definition of semantic security.

A (deterministic) function will be named pseudo-random, if no efficient adversary can distinguish it from what we call a *random function*. We will explain shortly the nature of such a random function. Consider the set of all functions from $\mathcal{X}$ to $\mathcal{Y}$ written $\mathsf{Func}(\mathcal{X},\mathcal{Y})$. For finite sets $\mathcal{X}$ and $\mathcal{Y}$, this set has $|\mathcal{Y}|^{|\mathcal{X}|}$ elements, thus the uniform distribution on this set is given by $P_{\mathcal{U}}[f] = \frac{1}{|\mathcal{Y}|^{|\mathcal{X}|}}$ for all $f \in \mathsf{Func}(\mathcal{X},\mathcal{Y})$. A random function is an element from $\mathsf{Func}(\mathcal{X},\mathcal{Y})$ that is drawn according to this uniform distribution. It has the following properties: (i) it is a deterministic function (!), (ii) for each $x \in \mathcal{X}$, $f(x)$ is distributed uniformly random in $\mathcal{Y}$, independently of any other $f(y)$, if $x \neq y$.

Random permutations are similarly defined, except that they are drawn from $\mathsf{Perm}(\mathcal{X})$, the set of all permutation of $\mathcal{X}$, which has $|\mathcal{X}|!$ elements.

4.1.1 Pseudo-random Permutations (PRPs)

Intuitively, a pseudo-random permutation is a keyed function, i.e., a deterministic function $\mathsf{E} : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{X}$ such that for a fixed but secret key K no machine can distinguish the resulting permutation from a random permutation. This is expressed, as most definitions in cryptography, in terms of a game, that an adversary plays against a *challenger*.

Definition 4.1 (PRP Challenger) *Let $\mathsf{E} : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{X}$ be a PRP candidate function. Then the PRP challenger for E is defined in three stages as follows:*

- *First, it randomly chooses a bit b.*
- *Secondly, depending on the value of b, it proceeds as follows:*
  - *If $b = 0$ let $K \leftarrow_{\mathcal{R}} \mathcal{K}$, $F := \mathsf{E}(K,\cdot)$.*
  - *If $b = 1$ let $F \leftarrow_{\mathcal{R}} \mathsf{Perm}(\mathcal{X})$.*
- *Finally, it receives a message $x \in \mathcal{X}$ and outputs $F(x)$. This stage is repeated arbitrarily often.*

$\diamond$

An adversary wins the game against the PRP-challenger if it is able to deduce the bit b significantly better than by pure guessing. The adversary can guess correctly with probability $\frac{1}{2}$ by simply outputting a random value b^*. As formalized by the following definition, its advantage captures

how much better an adversary can do than to purely guess the bit. In the following, $Exp_{\mathsf{A}}^{\mathsf{PRP}}(b)$ denotes the experiment where the adversary A interacts with the PRP challenger whose bit is chosen as b. Furthermore $Exp_{\mathsf{A}}^{\mathsf{PRP}}(b) = 0$ and $Exp_{\mathsf{A}}^{\mathsf{PRP}}(b) = 1$ denote the event that the adversary outputs 0 and 1 in the respective experiment.

Definition 4.2 (PRP advantage) *Let $\mathsf{E} : \mathcal{K} \times \mathcal{X} \to \mathcal{X}$ be a* PRP *candidate function. The advantage of the adversary A against the* PRP *challenger for E is defined as follows:*

$$Adv^{\mathsf{PRP}}[\mathsf{A}, \mathsf{E}] := \left| \mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{PRP}}(0) = 1\right] - \mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{PRP}}(1) = 1\right] \right|.$$

$\diamond$

The definition of security will be asymptotic, so we need a sequence of ciphers. Let $\mathsf{E} = (\mathsf{E}_n)_{n\in\mathbb{N}}$ be a sequence of functions where $\mathsf{E}_n : \mathcal{K}_n \times \mathcal{X}_n \to \mathcal{X}_n$. We say E is polynomial-time computable iff $\mathsf{E}_n(K, m)$ can be computed in time polynomial in n for all m, K. This parameter n is also called the *security parameter*. For a sequence of permutations $\mathsf{E} = (\mathsf{E}_n)_{n\in\mathbb{N}}$ and a sequence of adversaries $\mathsf{A} = (\mathsf{A}_n)_{n\in\mathbb{N}}$, we define

$$Adv^{\mathsf{PRP}}[\mathsf{A}, \mathsf{E}](n) := Adv^{\mathsf{PRP}}[\mathsf{A}_n, \mathsf{E}_n].$$

Definition 4.3 (Pseudo-random Permutation) *A* pseudorandom permutation *is a sequence $\mathsf{E} = (\mathsf{E}_n)_{n\in\mathbb{N}}$ of functions $\mathsf{E}_n : \mathcal{K}_n \times \mathcal{X}_n \to \mathcal{X}_n$, where for each $n \in \mathbb{N}$ and for each $K \in \mathcal{K}_n$:*

(1) *E_n is efficiently computable,*
(2) *$\mathsf{E}_n(K, \cdot)$ is bijective,*
(3) *$\mathsf{E}_n^{-1}(K, \cdot)$ is efficiently computable and*
(4) *$Adv^{\mathsf{PRP}}[\mathsf{A}, \mathsf{E}]$ is negligible in n, for all (sequences of) adversaries A.*

$\diamond$

4.1.2 Pseudo-random Functions (PRFs)

Pseudo-random functions are defined in a very similar manner, mainly by dropping the requirements (2) and (3) in Definition 4.3.

Definition 4.4 (PRF Challenger) *Let $\mathsf{E} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be a* PRF *candidate function. The* PRF challenger *for E is defined in three stages as follows:*

- *First, it randomly chooses a bit b.*
- *Secondly, depending on the value of b, it proceeds as follows:*
  - *If $b = 0$ let $K \leftarrow_{\mathcal{R}} \mathcal{K}$, $F := \mathsf{E}(K, \cdot)$.*
  - *If $b = 1$ let $F \leftarrow_{\mathcal{R}} \mathsf{Func}(\mathcal{X}, \mathcal{Y})$.*
- *Finally, it receives a message $x \in \mathcal{X}$ and outputs $F(x)$. This stage is repeated arbitrarily often.*

$\diamond$

Definition 4.5 (PRF Advantage) *Let $\mathsf{E} : \mathcal{K} \times \mathcal{X} \to \mathcal{X}$ be a* PRF *candidate function. The advantage of the adversary A against the* PRF *challenger for E is defined as follows:*

$$Adv^{\mathsf{PRF}}[\mathsf{A}, \mathsf{E}] := \left| \mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{PRF}}(0) = 1\right] - \mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{PRF}}(1) = 1\right] \right|.$$

$\diamond$

For a sequence of functions $\mathsf{E} = (\mathsf{E}_n)_{n\in\mathbb{N}}$ and a sequence of adversaries $\mathsf{A} = (\mathsf{A}_n)_{n\in\mathbb{N}}$ one defines

$$Adv^{\mathsf{PRF}}[\mathsf{A},\mathsf{E}](n) := Adv^{\mathsf{PRF}}[\mathsf{A}_n,\mathsf{E}_n].$$

Definition 4.6 (Pseudo-random Functions) *A* pseudo-random function *is a sequence* $\mathsf{E} = (\mathsf{E}_n)_{n\in\mathbb{N}}$ *of functions* $\mathsf{E}_n : \mathcal{K}_n \times \mathcal{X}_n \to \mathcal{Y}_n$*, where for each* $n \in \mathbb{N}$ *and for each* $K \in \mathcal{K}_n$*:*
(1) E_n *is efficiently computable and*
(4) $Adv^{\mathsf{PRF}}[\mathsf{A},\mathsf{E}](n)$ *is negligible in* n*.*

◇

We will drop the subscripts n if they are clear from the context, and we often simply speak of functions/adversaries instead of sequences of functions/adversaries in the following.

4.1.3 Switching Lemma

We have already seen examples of pseudo-random permutations, namely blockciphers such as DES and AES are considered to be pseudo-random permutations. The switching lemma states that every pseudo-random permutation is also a pseudo-random function.

Lemma 4.1 (Switching Lemma) *Each PRP* E *on* $(\mathcal{K},\mathcal{X})$ *is also a PRF on* $(\mathcal{K},\mathcal{X},\mathcal{X})$*. More precisely, if* A *is an adversary making at most* q *queries, then we have*

$$\left|Adv^{\mathsf{PRP}}[\mathsf{A},\mathsf{E}] - Adv^{\mathsf{PRF}}[\mathsf{A},\mathsf{E}]\right| \leq q^2/(2|\mathcal{X}|).$$

□

Proof. First we bound the left side in one direction:

$$\begin{aligned}
& Adv^{\mathsf{PRP}}[\mathsf{A},\mathsf{E}] \\
= \;& \left|\mathsf{Pr}\left[Exp^{\mathsf{PRP}}_{\mathsf{A}}(0)=1\right] - \mathsf{Pr}\left[Exp^{\mathsf{PRP}}_{\mathsf{A}}(1)=1\right]\right| \\
= \;& \left|\mathsf{Pr}\left[Exp^{\mathsf{PRF}}_{\mathsf{A}}(0)=1\right] - \mathsf{Pr}\left[Exp^{\mathsf{PRP}}_{\mathsf{A}}(1)=1\right]\right| \\
= \;& \left|\mathsf{Pr}\left[Exp^{\mathsf{PRF}}_{\mathsf{A}}(0)=1\right] - \mathsf{Pr}\left[Exp^{\mathsf{PRF}}_{\mathsf{A}}(1)=1\right] + \mathsf{Pr}\left[Exp^{\mathsf{PRF}}_{\mathsf{A}}(1)=1\right] - \mathsf{Pr}\left[Exp^{\mathsf{PRP}}_{\mathsf{A}}(1)=1\right]\right| \\
\overset{(1)}{\leq} \;& Adv^{\mathsf{PRF}}[\mathsf{A},\mathsf{E}] + \left|\mathsf{Pr}\left[Exp^{\mathsf{PRF}}_{\mathsf{A}}(1)=1\right] - \mathsf{Pr}\left[Exp^{\mathsf{PRP}}_{\mathsf{A}}(1)=1\right]\right| \qquad (4.1)
\end{aligned}$$

The other direction is derived similarly:

$$\begin{aligned}
& Adv^{\mathsf{PRF}}[\mathsf{A},\mathsf{E}] \\
= \;& \left|\mathsf{Pr}\left[Exp^{\mathsf{PRF}}_{\mathsf{A}}(0)=1\right] - \mathsf{Pr}\left[Exp^{\mathsf{PRF}}_{\mathsf{A}}(1)=1\right]\right| \\
= \;& \left|\mathsf{Pr}\left[Exp^{\mathsf{PRP}}_{\mathsf{A}}(0)=1\right] - \mathsf{Pr}\left[Exp^{\mathsf{PRF}}_{\mathsf{A}}(1)=1\right]\right| \\
= \;& \left|\mathsf{Pr}\left[Exp^{\mathsf{PRP}}_{\mathsf{A}}(0)=1\right] - \mathsf{Pr}\left[Exp^{\mathsf{PRP}}_{\mathsf{A}}(1)=1\right] + \mathsf{Pr}\left[Exp^{\mathsf{PRP}}_{\mathsf{A}}(1)=1\right] - \mathsf{Pr}\left[Exp^{\mathsf{PRF}}_{\mathsf{A}}(1)=1\right]\right| \\
\overset{(1)}{\leq} \;& Adv^{\mathsf{PRP}}[\mathsf{A},\mathsf{E}] + \left|\mathsf{Pr}\left[Exp^{\mathsf{PRF}}_{\mathsf{A}}(1)=1\right] - \mathsf{Pr}\left[Exp^{\mathsf{PRP}}_{\mathsf{A}}(1)=1\right]\right| \qquad (4.2)
\end{aligned}$$

In both calculations (1) follows from the triangular inequality. Now let *Coll* denote the event that, for the adversary A making queries $x_1, \ldots, x_q$ in experiment $Exp^{\mathsf{PRF}}_{\mathsf{A}}(1)$, a collision occurs, i.e., for $x_i \neq x_j$ we have $F(x_i) = F(x_j)$, where F is a random function from $\mathcal{X}$ to $\mathcal{X}$. Thus we have

$$
\begin{aligned}
& \left| \mathsf{Pr}\left[Exp^{\mathsf{PRF}}_{\mathsf{A}}(1) = 1\right] - \mathsf{Pr}\left[Exp^{\mathsf{PRP}}(1) = 1\right] \right| \\
= \; & \Big| \mathsf{Pr}\left[Exp^{\mathsf{PRF}}_{\mathsf{A}}(1) = 1 \mid \mathit{Coll}\right] \cdot \mathsf{Pr}\left[\mathit{Coll}\right] + \mathsf{Pr}\left[Exp^{\mathsf{PRF}}_{\mathsf{A}}(1) = 1 \mid \neg \mathit{Coll}\right] \cdot \mathsf{Pr}\left[\neg \mathit{Coll}\right] \\
& \quad - \mathsf{Pr}\left[Exp^{\mathsf{PRP}}_{\mathsf{A}}(1) = 1\right] \Big| \\
= \; & \Big| \mathsf{Pr}\left[Exp^{\mathsf{PRF}}_{\mathsf{A}}(1) = 1 \mid \mathit{Coll}\right] \cdot \mathsf{Pr}\left[\mathit{Coll}\right] + \mathsf{Pr}\left[Exp^{\mathsf{PRF}}_{\mathsf{A}}(1) = 1 \mid \neg \mathit{Coll}\right] \cdot (1 - \mathsf{Pr}\left[\mathit{Coll}\right]) \\
& \quad - \mathsf{Pr}\left[Exp^{\mathsf{PRP}}_{\mathsf{A}}(1) = 1\right] \Big| \\
\overset{(2)}{=} \; & \left| \mathsf{Pr}\left[Exp^{\mathsf{PRF}}_{\mathsf{A}}(1) = 1 \mid \mathit{Coll}\right] \cdot \mathsf{Pr}\left[\mathit{Coll}\right] - \mathsf{Pr}\left[Exp^{\mathsf{PRF}}_{\mathsf{A}}(1) = 1 \mid \neg \mathit{Coll}\right] \cdot \mathsf{Pr}\left[\mathit{Coll}\right] \right| \\
= \; & \left| \mathsf{Pr}\left[\mathit{Coll}\right] \cdot \left(\mathsf{Pr}\left[Exp^{\mathsf{PRF}}_{\mathsf{A}}(1) = 1 \mid \mathit{Coll}\right] - \mathsf{Pr}\left[Exp^{\mathsf{PRF}}_{\mathsf{A}}(1) = 1 \mid \neg \mathit{Coll}\right] \right) \right| \\
= \; & \left| \mathsf{Pr}\left[\mathit{Coll}\right] \right| \cdot \left| \mathsf{Pr}\left[Exp^{\mathsf{PRF}}_{\mathsf{A}}(1) = 1 \mid \mathit{Coll}\right] - \mathsf{Pr}\left[Exp^{\mathsf{PRF}}_{\mathsf{A}}(1) = 1 \mid \neg \mathit{Coll}\right] \right| \\
\overset{(3)}{\leq} \; & \left| \mathsf{Pr}\left[\mathit{Coll}\right] \right|
\end{aligned}
$$

Here (2) follows from the fact that the experiment $Exp^{\mathsf{PRP}}_{\mathsf{A}}(1)$ is identical to the experiment $Exp^{\mathsf{PRF}}_{\mathsf{A}}(1)$ if no collision occurred, and (3) follows from the fact that the right term is between 0 and 1. In the last step of the proof we have to bound the expression $|\mathsf{Pr}\left[\mathit{Coll}\right]|$.

$$
\begin{aligned}
\left| \mathsf{Pr}\left[\mathit{Coll}\right] \right| &= \left| \frac{1}{|\mathcal{X}|} + \frac{2}{|\mathcal{X}|} + \ldots + \frac{q-1}{|\mathcal{X}|} \right| \\
&= \frac{q(q-1)}{2|\mathcal{X}|} \leq \frac{q^2}{2|\mathcal{X}|}
\end{aligned}
$$

■

4.1.4 Luby-Rackoff

The next theorem makes a statement about the opposite direction: A three-round Feistel network, where the round functions are PRFs, turns out to be a PRP. This can be seen as a justification for DES.

Theorem 4.1 (Luby-Rackoff) *A 3-round Feistel network whose three round functions are PRFs is itself a PRP.* □

4.2 On Definitions of Security

We have already seen that there are two parameters for defining security: The *capabilities* the adversary has, and the *goal* he pursues. Here we focus on the goal of *semantic security*, which intuitively captures that the adversary does not learn any partial information about the plaintexts. We distinguish the following adversary capabilities:

- Ciphertext-only attack (CT-only): Here the adversary sees one ciphertext only. Beforehand, he may choose two plaintexts, one of which is encrypted. If he cannot distinguish which of these (self-chosen) plaintexts is inside the given ciphertext, he learns no partial information. We already heave sees ciphers fulfilling this notion, e.g., the One-time Pad and stream ciphers.
- Chosen-plaintext Attack (CPA): Here the adversary sees (polynomially) many ciphertexts of self-chosen plaintexts, i.e., in addition to CT-only attacks, he may select arbitrary plaintexts that he sees the encryption of, before proceeding as above. Examples of ciphers fulfilling this stronger notion are randomized CBC based on PRPs as well as randomized counter mode based on PRFs.
- Chosen-ciphertext Attack (CCA): Additionally to the above, the adversary may choose ciphertexts that get decrypted for him. In order to avoid trivial distinguishability, these ciphertexts must of course be different from the ciphertext received as a response to the two challenge plaintext. We will treat CCA later in this lecture.

4.3 Semantic Security for Ciphertext-only Attack

Semantic security is one of the central definitions in cryptography. The intuition is that even if an adversary chooses two plaintexts and sees the encryption of one of them, he cannot tell which plaintext was encrypted. First we give the definition for encryption schemes where only one plaintext is encrypted and the resulting ciphertext is observed by the adversary.

Definition 4.7 (CT-only Challenger) *Let* (E, D) *be an encryption scheme on* $(\mathcal{K}, \mathcal{M}, \mathcal{C})$. *The* CT-only challenger *for* E *is defined in two stages as follows:*

- *First, it randomly chooses a bit* b *and a key* $K \leftarrow_{\mathcal{R}} \mathcal{K}$.
- *Secondly, it receives two plaintexts* $m_0, m_1 \in \mathcal{M}$, *with* $|m_0| = |m_1|$, *computes* $c \leftarrow \mathsf{E}(K, m_b)$ *and outputs* c.

$\diamond$

An adversary wins the game against the CT-only challenger if, intuitively, it is able to deduce the bit b better than by pure guessing after seeing c. In the following, $Exp_{\mathsf{A}}^{\mathsf{CT-only}}(b)$ denotes the experiment where the adversary A interacts with the CT-only challenger whose bit is chosen as b. Furthermore $Exp_{\mathsf{A}}^{\mathsf{CT-only}}(b) = 0$ and $Exp_{\mathsf{A}}^{\mathsf{CT-only}}(b) = 1$ denote the event that the adversary outputs 0 and 1 in the respective experiment. The advantage of an adversary is formalized by the following definition.

Definition 4.8 (CT-only Advantage) *Let* (E, D) *be an encryption scheme on* $(\mathcal{K}, \mathcal{M}, \mathcal{C})$. *The* advantage *of an adversary* A *against the* CT-only challenger *for* E *is defined as follows:*

$$Adv^{\mathsf{CT-only}}[\mathsf{A}, \mathsf{E}] := \left| \mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{CT-only}}(0) = 1 \right] - \mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{CT-only}}(1) = 1 \right] \right|.$$

$\diamond$

Again, the definition of semantic security is asymptotic, which means we are formally given a sequence of ciphers $(\mathsf{E}, \mathsf{D}) = (\mathsf{E}_n, \mathsf{D}_n)_{n \in \mathbb{N}}$ on $(\mathcal{K}_n, \mathcal{M}_n, \mathcal{C}_n)_{n \in \mathbb{N}}$ and a sequence of adversaries $\mathsf{A} = (\mathsf{A}_n)_{n \in \mathbb{N}}$, and we write

$$Adv^{\mathsf{CT-only}}[\mathsf{A}, \mathsf{E}](n) := Adv^{\mathsf{CT-only}}[\mathsf{A}_n, \mathsf{E}_n].$$

Again, we simply drop the subscript n in the following and simply speak of ciphers and adversaries, instead of sequences thereof.

Definition 4.9 (Semantic Security under Ciphertext-only Attack (CT-only)) *We say that a cipher* (E, D) *is* semantically secure under ciphertext-only attack (CT-only) *if for all efficient adversaries* A*, the advantage* $Adv^{\mathsf{CT-only}}[\mathsf{A}, \mathsf{E}]$ *is negligible in* n*.* ◇

4.3.1 Consequences of Semantic Security

Semantic security implies secrecy of every single bit of a plaintext. This is easy to see, so we do not give a formal proof. For contradiction suppose there exists an adversary A that can guess the i-th bit of a message with probability $\frac{1}{2} + p$ for some $0 < p \leq \frac{1}{2}$, after seeing its ciphertext. Then we construct an adversary B for the CT-only challenger as follows:

- Choose two messages, with $|m_0| = |m_1|$, such that the i-th bit in m_b is b,
- send these messages to the CT-only challenger and receive an encryption c,
- run A on input c, from which B obtains a bit b',
- and output b'.

The probability of success is given as follows:

$$\begin{aligned} Adv^{\mathsf{CT-only}}[\mathsf{B}, \mathsf{E}] &= \left| \mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{CT-only}}(0) = 1 \right] - \mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{CT-only}}(1) = 1 \right] \right| \\ &= |(1/2 - p) - (1/2 + p)| \\ &= 2p \end{aligned}$$

4.3.2 Semantic Security of the One-time Pad

The *one-time Pad* is semantically secure under ciphertext-only attack. This follows easily from perfect secrecy.

Lemma 4.2 *For* all *adversaries* A *(even for computationally unbounded ones) we have*

$$Adv^{\mathsf{CT-only}}[\mathsf{A}, \mathsf{E}^{\mathsf{OTP}}] = 0.$$

where $\mathsf{E}^{\mathsf{OTP}}$ *denotes the one-time pad encryption function.* □

4.3.3 Semantic Security of Deterministic Counter Mode (detCTR

Deterministic counter mode, i.e., counter mode with fixed initial value $IV = 0$, is semantically secure under ciphertext-only attack.

Theorem 4.2 (Semantic Security of detCTR mode) *For any* $L > 0$*: If* E *is a* PRF *over* $(\mathcal{K}, \mathcal{X})$ *then* $\mathsf{E}^{\mathsf{detCTR}}$ *over* $(\mathcal{K}, \mathcal{X}^L, \mathcal{X}^L)$ *is semantically secure. In particular, for any attacker* A *against* $\mathsf{E}^{\mathsf{detCTR}}$ *there exists an adversary* B *against* E *such that*

$$Adv^{\mathsf{CT-only}}[\mathsf{A}, \mathsf{E}^{\mathsf{detCTR}}] = 2 \cdot Adv^{\mathsf{PRF}}[\mathsf{B}, \mathsf{E}].$$

□

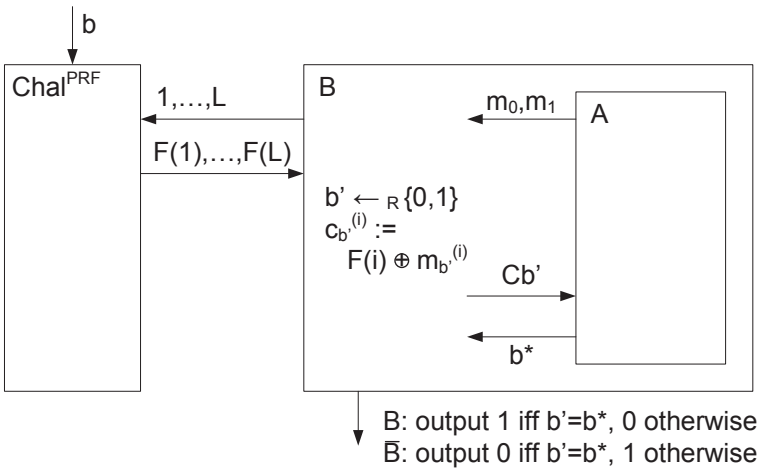


Figure 4.1: Construction of adversary B from A.

Proof. Given an adversary A against the CT-only challenger we construct an adversary B against the PRF challenger as in Figure 4.1. We need to estimate the following value

$$Adv^{\mathsf{PRF}}[\mathsf{B},\mathsf{E}] = \left|\mathsf{Pr}\left[Exp_{\mathsf{B}}^{\mathsf{PRF}}(0)=1\right] - \mathsf{Pr}\left[Exp_{\mathsf{B}}^{\mathsf{PRF}}(1)=1\right]\right| \tag{4.3}$$

First observe that for the second expression, we have

$$\mathsf{Pr}\left[Exp_{\mathsf{B}}^{\mathsf{PRF}}(1)=1\right] = \frac{1}{2}, \tag{4.4}$$

as the values $F(i)$ in this experiment are chosen independently at random. Thus the encryption $c_{b'}$ computed by B is as good as the one-time pad, so the adversary A gains absolutely no information on B's random bit b'.

Now let us calculate the first expression of the absolute value in Equation 4.3:

$$\begin{aligned}
& \mathsf{Pr}\left[Exp_{\mathsf{B}}^{\mathsf{PRF}}(0)=1\right] \\
= \;& \mathsf{Pr}\left[b'=b^* \mid b=0\right] \qquad (4.5)\\
= \;& \mathsf{Pr}\left[b'=b^*=0 \mid b=0\right] + \mathsf{Pr}\left[b'=b^*=1 \mid b=0\right] \\
= \;& \frac{1}{2}\cdot \mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{CT-only}}(0)=0\right] + \frac{1}{2}\cdot \mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{CT-only}}(1)=1\right] \\
= \;& \frac{1}{2}\cdot \left(1-\mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{CT-only}}(0)=1\right]\right) + \frac{1}{2}\cdot \mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{CT-only}}(1)=1\right] \\
= \;& \frac{1}{2} + \frac{1}{2}\cdot \left(\mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{CT-only}}(1)=1\right] - \mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{CT-only}}(0)=1\right]\right) \qquad (4.6)
\end{aligned}$$

If the right expression is positive, then Equation 4.6 equals $\frac{1}{2} + Adv^{\mathsf{CT-only}}[\mathsf{A},\mathsf{E}]$, which completes the proof.

Otherwise, if the right expression is negative, then we define the attacker $\bar{\mathsf{B}}$ to output 0 if B outputs 1 and vice versa. Then the same calculation as above yields

$$\mathsf{Pr}\left[Exp_{\bar{\mathsf{B}}}^{\mathsf{PRF}}(0)=1\right] = \frac{1}{2} + \frac{1}{2}\cdot Adv^{\mathsf{CT-only}}[\mathsf{A},\mathsf{E}] \tag{4.7}$$

Thus there exists some adversary B (either B itself or $\bar{\mathsf{B}}$ such that the following holds:

$$\begin{aligned} & Adv^{\mathsf{PRF}}[\mathsf{B},\mathsf{E}] = \left(\frac{1}{2} + \frac{1}{2} \cdot Adv^{\mathsf{CT-only}}[\mathsf{A},\mathsf{E}^{\mathsf{detCTR}}]\right) - \frac{1}{2} \\ \Leftrightarrow \quad & 2 \cdot Adv^{\mathsf{PRF}}[\mathsf{B},\mathsf{E}] = Adv^{\mathsf{CT-only}}[\mathsf{A},\mathsf{E}^{\mathsf{detCTR}}], \end{aligned}$$

which concludes the proof. ∎

4.4 Semantic Security under Chosen-plaintext Attack

The above definition of semantic security covered the case where the attacker sees a single encryption only. This is not sufficient for most uses of encryption schemes, but the notion can be extended to the more general case as well.

Definition 4.10 (CPA Challenger) *Let* (E,D) *be an encryption scheme on* $(\mathcal{K},\mathcal{M},\mathcal{C})$*. The* CPA challenger *for* E *is defined in three stages as follows:*

- *First, it chooses a bit* b *and a random key* $K \leftarrow_{\mathcal{R}} \mathcal{K}$*.*
- *Secondly, it receives a plaintext* x*. If* $x \notin \mathcal{M}$*, then it does nothing. Otherwise, it computes* $c \leftarrow \mathsf{E}(K,x)$ *and outputs* c*. It repeats this stage until the input satisfies the conditions of the third stage.*
- *It receives two plaintexts* $m_0, m_1 \in \mathcal{M}$ *with* $|m_0| = |m_1|$*, computes the encryption* $c \leftarrow \mathsf{E}(K,m_b)$ *and outputs* c*.*

◇

An adversary wins the game against the CPA challenger if, intuitively, it is able to deduce the bit b better than by pure guessing. In the following, $Exp^{\mathsf{CPA}}_{\mathsf{A}}(b)$ denotes the experiment where the adversary A interacts with the *CPA challenger* whose bit is chosen as b. Furthermore $Exp^{\mathsf{CPA}}_{\mathsf{A}}(b) = 0$ and $Exp^{\mathsf{CPA}}_{\mathsf{A}}(b) = 1$ denote the event that the adversary outputs 0 and 1 in the respective experiment.

Definition 4.11 (CPA Advantage) *Let* (E,D) *be an encryption scheme on* $(\mathcal{K},\mathcal{M},\mathcal{C})$*. The* advantage *of an adversary* A *against the* CPA challenger *for* E *is defined as follows:*

$$Adv^{\mathsf{CPA}}[\mathsf{A},\mathsf{E}] := \left|\mathsf{Pr}\left[Exp^{\mathsf{CPA}}_{\mathsf{A}}(b) = 0\right] - \mathsf{Pr}\left[Exp^{\mathsf{CPA}}_{\mathsf{A}}(b) = 1\right]\right|.$$

◇

Given a sequence of ciphers $(\mathsf{E},\mathsf{D}) = (\mathsf{E}_n,\mathsf{D}_n)_{n\in\mathbb{N}}$ on $(\mathcal{K}_n,\mathcal{M}_n,\mathcal{C}_n)_{n\in\mathbb{N}}$ and a sequence of adversaries $\mathsf{A} = (\mathsf{A}_n)_{n\in\mathbb{N}}$, and we write

$$Adv^{\mathsf{CPA}}[\mathsf{A},\mathsf{E}](n) := Adv^{\mathsf{CPA}}[\mathsf{A}_n,\mathsf{E}_n].$$

Definition 4.12 (Semantic Security under Chosen-plaintext Attack (CPA)) *A cipher* (E,D) *is* semantically secure under chosen-plaintext attack (CPA) *if for all efficient adversaries* A*, the advantage* $Adv^{\mathsf{CPA}}[\mathsf{A},\mathsf{E}]$ *is negligible in* n*.* ◇

4.4.1 Semantic Security of Modes of Operation

Theorem 4.3 (Semantic Security of CBC) *For any $L > 0$: If E is a PRP over $(\mathcal{K}, \mathcal{X})$ then $\mathsf{E}^{\mathsf{CBC}}$ over $(\mathcal{K}, \mathcal{X}^L, \mathcal{X}^L)$ is semantically secure. In particular, for any attacker A against $\mathsf{E}^{\mathsf{CBC}}$ making at most q queries there exists an adversary B against E such that*

$$Adv^{\mathsf{CPA}}[\mathsf{A}, \mathsf{E}^{\mathsf{CBC}}] = 2 \cdot Adv^{\mathsf{PRF}}[\mathsf{B}, \mathsf{E}] + \frac{2q^2L^2}{|\mathcal{X}|}.$$

□

This means that CBC is secure as long as $q \ll \frac{\sqrt{|\mathcal{X}|}}{L}$.

Theorem 4.4 (Semantic Security of Random Counter Mode (rndCTR)) *For any $L > 0$: If E is a PRP over $(\mathcal{K}, \mathcal{X})$ then $\mathsf{E}^{\mathsf{rndCTR}}$ over $(\mathcal{K}, \mathcal{X}^L, \mathcal{X}^{L+1})$ is semantically secure. In particular, for any attacker A against $\mathsf{E}^{\mathsf{rndCTR}}$ making at most q queries there exists an adversary B against E such that*

$$Adv^{\mathsf{CPA}}[\mathsf{A}, \mathsf{E}^{\mathsf{rndCTR}}] = 2 \cdot Adv^{\mathsf{PRF}}[\mathsf{B}, \mathsf{E}] + \frac{2q^2L}{|\mathcal{X}|}.$$

□

This means that random counter mode is secure as long as $q \ll \sqrt{\frac{|\mathcal{X}|}{L}}$.

5. MACs and Hash Functions

This chapter deals with message integrity, one of the core areas of cryptography.

5.1 Message Authentication Codes (MACs)

Message Authentication Codes (MACs) do not address privacy of data but provide data integrity, i.e., they prevent data from being changed by unauthorized users. MACs consist of two algorithms (S,V) ("sign" and "verify"). The sign algorithm takes a message and a key and computes a *tag*, also called *authenticator*. The verify algorithm takes a message, a key, and a tag and outputs true if it thinks the tag was correctly generated for the specified message, and false otherwise. One requires that the verify algorithm always outputs true if the tag was generated correctly using the sign algorithm with the respective message and key.

Definition 5.1 (Message Authentication Code) *A* message authentication code *over* $(\mathcal{K},\mathcal{M},\mathcal{T})$ *is a tuple of functions* $\mathsf{I}=(\mathsf{S},\mathsf{V})$ *where* $\mathsf{S}:\mathcal{K}\times\mathcal{M}\to\mathcal{T}$ *and* $\mathsf{V}:\mathcal{K}\times\mathcal{M}\times\mathcal{T}\to\mathcal{BOOL}$ *such that:*

(1) S *and* V *are efficiently computable, and*

(2) *for all* $m\in\mathcal{M}$ *and* $K\in\mathcal{K}$*, and for* $t\leftarrow\mathsf{S}(K,m)$*, we have* $\mathsf{V}(K,m,t)=\mathsf{true}$.

◇

The definition of efficiency is again asymptotic, so strictly speaking we again consider sequences $\mathsf{S}=(\mathsf{S}_n)_{n\in\mathbb{N}}$ and $\mathsf{V}=(\mathsf{V}_n)_{n\in\mathbb{N}}$ with domains/ranges $\mathcal{K}=(\mathcal{K}_n)_{n\in\mathbb{N}}$, $\mathcal{M}=(\mathcal{M}_n)_{n\in\mathbb{N}}$, and $\mathcal{T}=(\mathcal{T}_n)_{n\in\mathbb{N}}$. This parameter n is again called *security parameter*. As usual, "efficient computable" is now defined as running in probabilistic polynomial time in n. In the following we will again omit to explicitly write these sequences in order to increase readability; the theorem statements also do not require explicit sequence notation.

Security of MACs is defined by requiring that no efficient adversary can *forge* a tag, i.e., intuitively no valid tag can be computed without knowing the key. This is, again, defined in terms of a cryptographic game with the following challenger.

Definition 5.2 (MAC Challenger) *Let* $\mathsf{I}=(\mathsf{S},\mathsf{V})$ *be a MAC over* $(\mathcal{K},\mathcal{M},\mathcal{T})$*. Then the* MAC challenger *for* I *is defined as follows:*

- *It first randomly chooses a key* $K\leftarrow_{\mathcal{R}}\mathcal{K}$.
- *It then receives messages* $m_i\in\mathcal{M}$ *and outputs* $t_i\leftarrow\mathsf{S}(K,m_i)$*. This stage may repeat arbitrarily but finitely often, thus yielding a sequence of pairs* $(m_1,t_1),\ldots,(m_q,t_q)$.
- *Finally, it receives a pair* (m^*,t^*)*. If* t^* *is a valid tag for* m^* *and this pair is not contained in the obtained sequence, i.e., if* $\mathsf{V}(K,m^*,t^*)=\mathsf{true}$ *and* $\forall i\in\{1,\ldots,q\}\colon(m_i,t_i)\neq(m^*,t^*)$*, then it outputs* true*, otherwise* false.

◇

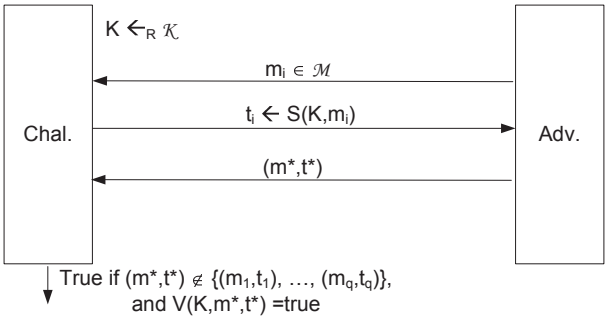


Figure 5.1: The Attack Game for MACs

The attack game between the MAC challenger and an adversary is depicted in Figure 5.1. The adversary wins the game if he is able to compute a tuple (m^*, t^*), such that the challenger outputs true. This tuple is called an *existential forgery*. Another type of forgery is *universal forgery*, where an adversary has been able to find a tag t' *for every* (given) message m'.

Definition 5.3 (MAC Advantage) *Let* $\mathsf{I} = (\mathsf{S}, \mathsf{V})$ *be a MAC over* $(\mathcal{K}, \mathcal{M}, \mathcal{T})$. *We let* $Exp_{\mathsf{A}}^{\mathsf{MAC}}$ *denote the experiment that an adversary* A *interacts with the MAC challenger for* I, *and we let* $Exp_{\mathsf{A}}^{\mathsf{MAC}} = \mathsf{true}$ *denote the event that the challenger finally outputs* true.

The advantage of an adversary A *against the* MAC *challenger for* I *is defined as*

$$Adv^{\mathsf{MAC}}[\mathsf{A}, \mathsf{I}] := \mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{MAC}} = \mathsf{true}\right].$$

◇

For a sequence of adversaries $\mathsf{A} = (\mathsf{A}_n)_{n \in \mathbb{N}}$ we as usual define

$$Adv^{\mathsf{MAC}}[\mathsf{A}, \mathsf{I}](n) := Adv^{\mathsf{MAC}}[\mathsf{A}_n, \mathsf{I}_n].$$

Definition 5.4 (Secure MAC) *Let* $\mathsf{I} = (\mathsf{S}, \mathsf{V})$ *be a MAC. Then* I *is* secure against existential forgery under chosen-message attack (CMA) *if* $Adv^{\mathsf{MAC}}[\mathsf{A}, \mathsf{I}](n)$ *is negligible for all (sequences of) efficient adversaries* A. *We often say* secure MAC *for brevity instead of a MAC that is secure against existential forgery under chosen-message attack.* ◇

5.1.1 Constructing MACs from PRFs

We will now see a first example of a MAC: it turns out that any PRF is a MAC. This construction is of course not entirely practical as the message space of PRFs we have seen so far is very small. However, we will later see how to increase the size of a PRF's message space, i.e., how to build PRFs with larger domains out of PRFs with smaller domains. For a PRF E over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$ we define the MAC $\mathsf{I}_{\mathsf{E}} = (\mathsf{S}, \mathsf{V})$ as follows:

- $\mathsf{S}(K, m) = \mathsf{E}(K, m)$,
- $\mathsf{V}(K, m, t) = \mathsf{true}$ if and only if $\mathsf{E}(K, m) = t$.

This construction yields a secure MAC, as long as E is a PRF and $\frac{1}{|\mathcal{Y}|}$ is negligible. This is captured in the following theorem.

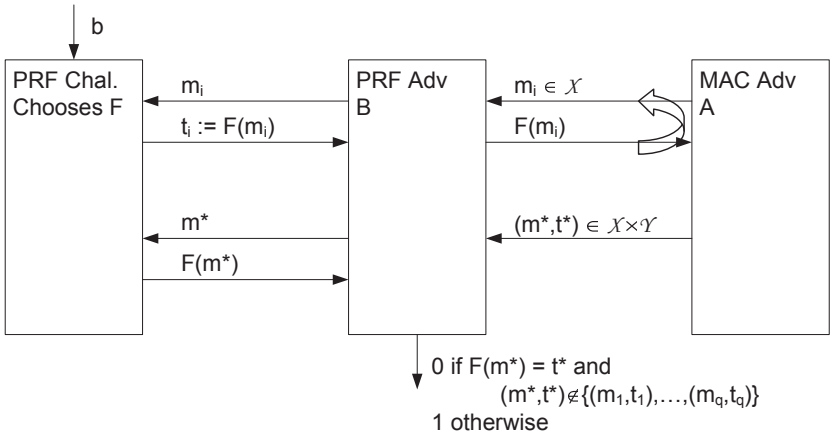


Figure 5.2: Any PRF is a MAC: Proof Sketch

Lemma 5.1 *Let* E *be a PRF over* $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$ *and* I_E *as constructed above. Then for every adversary* A *attacking* I_E *there exists an adversary* B *attacking* E *such that*

$$Adv^{\mathsf{MAC}}[\mathsf{A}, \mathsf{I}_\mathsf{E}] \leq Adv^{\mathsf{PRF}}[\mathsf{B}, \mathsf{E}] + \frac{1}{|\mathcal{Y}|}.$$

In particular, if $\frac{1}{|\mathcal{Y}|}$ *is negligible, then* I_E *is a secure MAC.* □

Proof. Let us consider the adversary B as defined in Figure 5.2. Consider the event $Exp^{\mathsf{PRF}}_{\mathsf{B}}(1) = 0$. We then have $\mathsf{F}(m^*) = t^*$ and $(m^*, t^*) \notin \{(m_1, t_1), \ldots, (m_q, t_q)\}$ for a random function F. First we assume $m^* = m_i$ for some i. Thus $t^* = \mathsf{F}(m^*) = \mathsf{F}(m_i) = t_i$, and hence $(m_i, t_i) = (m^*, t^*)$, which gives us a contradiction. Hence we have $m^* \notin \{m_1, \ldots, m_q\}$. However, this means that $\mathsf{F}(m^*)$ is a random element in $\mathcal{Y}$, thus the probability that $\mathsf{F}(m^*) = t^*$ for some fixed t^* is precisely $\frac{1}{|\mathcal{Y}|}$. Thus, we obtain $\mathsf{Pr}\left[Exp^{\mathsf{PRF}}_{\mathsf{B}}(1) = 0\right] = \frac{1}{|\mathcal{Y}|}$.

Next we consider the event $Exp^{\mathsf{PRF}}_{\mathsf{B}}(0) = 0$. We then have $\mathsf{F}(m_i) = \mathsf{E}(K, m_i) = \mathsf{S}(K, m_i)$, thus B always hands over the correct tag to A. Moreover B outputs true if $\mathsf{F}(m^*) = t^* \ \wedge \ (m^*, t^*) \notin \{(m_1, t_1), \ldots, (m_q, t_q)\}$. This is by definition equivalent to $\mathsf{V}(K, m^*, t^*) = \mathsf{true} \ \wedge \ (m^*, t^*) \notin \{(m_1, t_1), \ldots, (m_q, t_q)\}$. Thus, we obtain $\mathsf{Pr}\left[Exp^{\mathsf{PRF}}_{\mathsf{B}}(0) = 0\right] = \mathsf{Adv}^{\mathsf{MAC}}[\mathsf{A}, \mathsf{I}_\mathsf{E}]$.

Putting this together, we obtain

$$\begin{aligned} Adv^{\mathsf{PRF}}[\mathsf{B}, \mathsf{E}] &= \left|\mathsf{Pr}\left[Exp^{\mathsf{PRF}}_{\mathsf{B}}(0) = 1\right] - \mathsf{Pr}\left[Exp^{\mathsf{PRF}}_{\mathsf{B}}(1) = 1\right]\right| \\ &= \left|(1 - Adv^{\mathsf{MAC}}[\mathsf{A}, \mathsf{I}_\mathsf{E}]) - (1 - \frac{1}{|\mathcal{Y}|})\right| \\ &= \left|Adv^{\mathsf{MAC}}[\mathsf{A}, \mathsf{I}_\mathsf{E}] - \frac{1}{|\mathcal{Y}|}\right| \end{aligned}$$

and finally

$$Adv^{\mathsf{MAC}}[\mathsf{A}, \mathsf{I}_\mathsf{E}] \leq Adv^{\mathsf{PRF}}[\mathsf{B}, \mathsf{E}] + \frac{1}{|\mathcal{Y}|}.$$

■

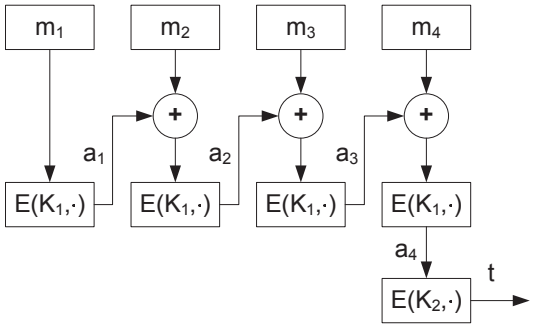


Figure 5.3: The (Encrypted) CBC-MAC Construction

5.2 Extending the Message Space of MACs

When using block ciphers such as DES or AES in the construction of Section 5.1.1, the message space consists of only 64 or 128 bits. This is of course not sufficient for most applications, so we explore several ways to increase the message space of PRFs. One of the most famous constructions for doing so is CBC-MAC, which is used in several standards. It is standardized as ANSI X9.9 and X9.19, as an ISO standard, and in FIPS 186-3, and it is also broadly used by the banking industry. Its main disadvantage is that it is fully sequential, i.e., its many block cipher invocations cannot be parallelized, e.g., by specialized hardware. This disadvantage is solved by PMAC (Parallel MAC), a recent system that also enjoys the property of being incremental, see below. A third approach is the HMAC construction, which relies on collision-resistant hash functions (CRHFs). CBC-MAC and PMAC will be introduced in this sections; the description of HMAC first requires reviewing some basic knowledge of hash functions and will be given thereafter.

Let us start with a brief observation: if a MAC $\mathsf{I}_\mathsf{E} = (\mathsf{S}, \mathsf{V})$ is a PRF over $(\mathcal{K}, \mathcal{M}, \{0,1\}^k)$, then we can shorten the tags by a certain amount without sacrificing security of the MAC. More specifically, if we delete all but the first w bits, then this "truncated PRF" is still a PRF. If additionally $(\frac{1}{2})^w$ is negligible, it also constitutes a secure MAC.

5.2.1 Encrypted CBC-MAC

The *Encrypted CBC-MAC*, which we will sometimes simply call CBC-MAC, is similar to the CBC-mode in symmetric encryption. For a given PRF E over $(\mathcal{K}, \mathcal{X}, \mathcal{X})$, one defines $\mathsf{E}^{\text{CBC-MAC}}$ over $(\mathcal{K}^2, \mathcal{X}^L, \mathcal{X})$ as follows; a graphical illustration is given in Figure 5.3.

- A CBC-MAC key consists of two PRF keys K_1, K_2.
- Given a message $m = m^{(1)}m^{(2)}\ldots m^{(s)}$ and a key (K_1, K_2), the signing algorithm computes $a_1 := \mathsf{E}(K_1, m_1)$ and $a_i := \mathsf{E}(K_1, m_i \oplus a_{i-1})$ for $i = 2, \ldots, s$.
- Finally, it computes the tag as $t := \mathsf{E}(K_2, a_s)$.
- Given a message m, a tag t, and a key (K_1, K_2), verification is done by recomputing the tag from m and by checking if the recomputed tag equals t.

This construction is secure, as stated in the following theorem.

Theorem 5.1 (Security of CBC-MAC) *Let $L > 0$, let E be a PRF over $(\mathcal{K}, \mathcal{X}, \mathcal{X})$, and let $\mathsf{E}^{\text{CBC-MAC}}$ be as constructed above. Then for each adversary A attacking $\mathsf{E}^{\text{CBC-MAC}}$ and making at*

most q queries, there exists an attacker B attacking E such that

$$Adv^{\mathsf{MAC}}[\mathsf{A}, \mathsf{E}^{\mathsf{CBC\text{-}MAC}}] < Adv^{\mathsf{PRF}}[\mathsf{B}, \mathsf{E}] + \frac{q^2}{|\mathcal{X}|} \cdot L^{o(1)}.$$

(Here $L^{o(1)}$ denotes a quantity that converges to 1, and which will not matter in the following.) In particular, if $\frac{1}{|\mathcal{X}|}$ is negligible, then $\mathsf{E}^{\mathsf{CBC\text{-}MAC}}$ is a secure MAC. □

Intuitively, this means that CBC-MAC is secure as long as $q \ll \sqrt{|\mathcal{X}|}$.

Raw CBC-MAC One might wonder if the additional encryption of the output a_s is really necessary, i.e., why not let $t = a_s$? We will show that dropping the last encryption does not result in a secure MAC. We call the resulting scheme *Raw CBC-MAC*, thus letting $t := a_s$. The following instructions describe how an adversary can win a game against a MAC challenger for Raw CBC-MAC.

- Choose a message $m \in \mathcal{X}$, i.e., a message that fits into one block, and request the tag t for it, receiving $t = \mathsf{E}(K, m)$.
- Output $(m \mid\mid m \oplus t, t)$ as a candidate forgery.

It is easy to see that t is indeed a valid tag for $(m \mid\mid m \oplus t)$. Calculating the tag t' of the message $m' = m \mid\mid m \oplus t$ yields

$$t' = \mathsf{E}(K, \mathsf{E}(K, m) \oplus (m \oplus t)) = \mathsf{E}(K, t \oplus (m \oplus t)) = \mathsf{E}(K, m) = t.$$

It turns out that this works as well for messages that span more than one block; however, the resulting forgeries and their correctness are lengthier to write down. We remark however that if the message length is fixed a-priori or if the receiver prepends the message length to the message, then Raw CBC-MAC can be proven secure.

5.2.2 Padding with CBC-MAC

If the message length is not a multiple of the block length then one needs to *pad* the message, i.e., one needs to append some bits to reach a multiple of the block size. This sounds trivial (and indeed is if one is only concerned about privacy), but has some hidden caveats if integrity is the goal.

- The easiest solution seems to be to append a string of 0's of the required length, or any other fixed string of appropriate length. This, however, turns out to be a bad idea, as one cannot distinguish between trailing zeros of the message and leading zeros of the pad. In particular, the messages 0 and 00 have the same tag, as they are both padded to 0^k where k is the block-size. This allows for easy creation of existential forgeries.

- A good method for padding, which is also described in the ISO standard, is the following: Append a 1 to the message and fill the remaining space with 0's. If the message size is a multiple of the block size, then append a new block $10 \ldots 0$. The inserted 1 unambiguously determines the begin of the padding.

Note again that padding was not an issue for encryption schemes, as padding any string does not violate privacy of the string.

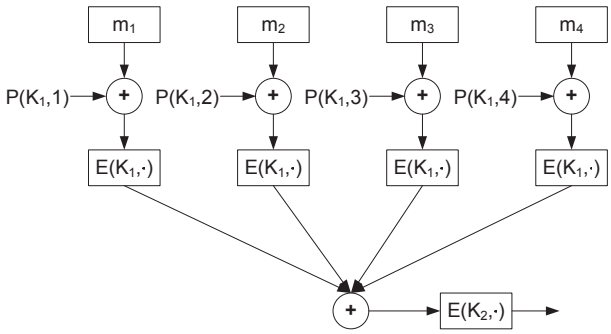


Figure 5.4: PMAC

5.2.3 PMAC

The CBC-MAC we just described is sequential, i.e., for computing the tag of a long message, all PRF invocations depend on the output of previous invocations; thus they cannot be parallelized even by specialized hardware. The PMAC (Parallel MAC) was designed to circumvent this problem. Let E a PRF over $(\mathcal{K}, \mathcal{X}, \mathcal{X})$, then we define a function $\mathsf{E}^{\mathsf{PMAC}} : \mathcal{K}^2 \times \mathcal{X}^L \to \mathcal{X}$ as follows:

- A key for PMAC consists of two PRF keys K_1, K_2.
- Furthermore it uses a special deterministic function $\mathsf{P} : \mathcal{K} \times \mathbb{N} \to \mathcal{X}$, which can be computed efficiently. It exploits properties of finite fields, which come later in the course, so that we do not describe the details here.
- Given a message $m = m_1 m_2 \dots m_r$ one computes

$$b_i := \mathsf{E}(K_1, m_i \oplus \mathsf{P}(K_1, i)) \text{ for } i = 1, \dots, r,$$

- Finally one computes $t := \mathsf{E}(K_2, \bigoplus_{i=1,\dots,r} b_i)$.

The resulting function is a secure MAC, which is formalized in the following theorem:

Theorem 5.2 (PMAC) *Let $L > 0$ and let E be a PRF over $(\mathcal{K}, \mathcal{X}, \mathcal{X})$. For $\mathsf{E}^{\mathsf{PMAC}}$ as defined above the following holds: For each adversary A attacking $\mathsf{E}^{\mathsf{PMAC}}$ and making at most q queries there exists an attacker B attacking E such that*

$$Adv^{\mathsf{MAC}}[\mathsf{A}, \mathsf{E}^{\mathsf{PMAC}}] \leq Adv^{\mathsf{PRF}}[\mathsf{B}, \mathsf{E}] + \frac{2q^2L^2}{|\mathcal{X}|}.$$

In particular, if $\frac{1}{|\mathcal{X}|}$ is negligible, then $\mathsf{E}^{\mathsf{PMAC}}$ is a secure MAC. □

We remark that PMAC is secure as long as $q \ll \frac{\sqrt{\mathcal{X}}}{L}$.

Another nice property is that PMAC is incremental, i.e., if a small part of a message is changed or added to the message, then the creation of the tag for the new message can easily be recomputed: Say we computed $t \leftarrow \mathsf{E}^{\mathsf{PMAC}}(K, m)$ for a long message m. Now if one block changed yielding another message m^*, one can recompute the new tag t^* for m^* very efficiently (either by assuming that E is a PRP, or by storing a little bit of intermediary information from the signing algorithm). This will be explored in detail in the homework exercises.

5.3 Hash Functions

Hash functions constitute a core cryptographic primitive. They map very long messages to a fixed-size value called *message digest* or *hash value*. One of the most important properties of hash functions is that one cannot find *collisions*, i.e., two different messages that are mapped to the same hash value. Hash functions often serve as basic building blocks for more complex primitives such as MACs, signatures, and so on.

Defining collision resistance of hash functions turns out to be a quite unsatisfactory task. The intuitively expected definition "no efficient adversary can find collisions" cannot be fulfilled by any hash function: As the message space is necessarily larger than the digest space, there always exist collisions $m, m^* \in \mathcal{M}$. Now take a look at the collection of adversaries A_{m_0,m_1} for $m_0, m_1 \in \mathcal{M}$, that simply output m_0 and m_1. Clearly, the adversary A_{m,m^*} exists (but nobody might know how to find it)! For this reason the following "definition" of collision-resistant hash functions is inherently imprecise, but it turns out that subsequent reduction proofs based on such hash functions can be turned into rigorous mathematical arguments again.

"Definition" 5.5 (Collision-Resistant Hash Function (CRHF)) *A (non-keyed)* hash function *is an "efficient" function* $\mathsf{H} : \mathcal{M} \to \mathcal{T}$. *A* collision *for* H *is a tuple* (m_0, m_1) *with* $\mathsf{H}(m_0) = \mathsf{H}(m_1)$ *and* $m_0 \neq m_1$. *A hash function* H *is* collision-resistant *if no "efficient"* A *adversary is known that finds a collision.*

5.3.1 Examples

Many proposals for collision-resistant hash functions exist; however, most of them have been broken already. Even the widely deployed hash functions MD5 and SHA-1 have recently been successfully attacked: While MD5 is definitely insecure (collisions can be found in roughly 1 hour on an ordinary PCs), the attacks on SHA-1 are rather theoretical in that they are still not efficient enough to find useful collisions in a short time. However, history has shown that it usually does not take long to further decrease the complexity of such attacks, once the first theoretical glitch has been found. Examples of well-known hash functions include

- MD5 (broken): 128-bits digest, hashes roughly 216 MB per second
- SHA-1 (broken) 160-bits digest, hashes roughly 68 MB per second
- SHA-256: 256-bits digest, hashes roughly 44.5 MB per second
- Whirlpool: 512-bits digest, hashes roughly 12.1 MB per second

5.3.2 Constructing Big-MACs from Small-MACs and CRHFs

We now present a generic construction how a MAC with a small message space can be turned into a MAC with a big message space by additionally exploiting a collision-resistant hash function. Let $\mathsf{I} = (\mathsf{S}, \mathsf{V})$ be a MAC over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$ and let H be a collision-resistance hash function $\mathsf{H} : \mathcal{M}^{\mathsf{big}} \to \mathcal{M}$. Then one can construct a MAC $\mathsf{I}^* = (\mathsf{S}^*, \mathsf{V}^*)$ by first hashing the message $m \in \mathcal{M}^{\mathsf{big}}$ and then applying the MAC:

- Let $\mathsf{S}^*(K, m) := \mathsf{S}(K, \mathsf{H}(m))$, and
- let $\mathsf{V}^*(K, m, t) := \mathsf{true}$ iff $\mathsf{V}(K, \mathsf{H}(m), t) = \mathsf{true}$.

One can prove the security of this construction assuming the security of the small MAC and the collision-resistance of the hash function:

Theorem 5.3 *Let* $\mathsf{H} : \mathcal{M}^{\mathsf{big}} \to \mathcal{M}$ *be a collision-resistant hash function and* $\mathsf{I} = (\mathsf{S}, \mathsf{V})$ *be a secure MAC over* $(\mathcal{K}, \mathcal{M}, \mathcal{T})$*. Then* $\mathsf{I}^* = (\mathsf{S}^*, \mathsf{V}^*)$ *as constructed above is a secure MAC over* $(\mathcal{K}, \mathcal{M}^{\mathsf{big}}, \mathcal{T})$*.* □

5.3.3 Generic Attack on CRHFs and the Birthday Paradox

The birthday paradox derives its name from the following question: if there are p people in a room, what is the probability that at least two of them have their birthday on the same day? Thus we are looking for the probability that there exist $i, j \in \{1, \ldots, n\}$, $i \neq j$, such that $t_i = t_j$, where t_i denotes the birthday of the i-th person. The somewhat surprising result is that for about 23 people the probability is already bigger than 50%. This result is stated and generalized in the following theorem.

Theorem 5.4 (Birthday Paradox) *Let* $t_1, \ldots, t_n$ *be independent randomly chosen integers in the set* $\{1, \ldots, B\}$*.*

$$\mathsf{Pr}\left[\exists i, j \in \{1, \ldots, n\}, i \neq j \colon t_i = t_j\right] \geq 1 - e^{\frac{-n(n-1)}{2B}}$$

□

In particular, for $n \geq 1.2 \cdot \sqrt{B}$ we have $\mathsf{Pr}\left[\exists i \neq j \in \{1, \ldots, n\} \colon t_i = t_j\right] \geq \frac{1}{2}$.

Proof. First we calculate the probability that all n values are different: For the first value this is 1, for the second $1 - \frac{1}{B}$, and so on. We thus obtain

$$\begin{aligned}
\mathsf{Pr}\left[\exists i, j \in \{1, \ldots, n\}, i \neq j \colon t_i = t_j\right] &= 1 - 1 \cdot \left(1 - \frac{1}{B}\right) \cdot \ldots \cdot \left(1 - \frac{n-1}{B}\right) \\
&= 1 - \prod_{i=1}^{n-1} (1 - \frac{i}{B}) \\
&\overset{(1)}{\geq} 1 - \prod_{i=1}^{n-1} e^{-\frac{i}{B}} \\
&= 1 - e^{-\sum_{i=1}^{n-1} \frac{i}{B}} \\
&= 1 - e^{-\frac{n(n-1)}{2B}}.
\end{aligned}$$

where in step (1) we exploited that $1 - x \leq e^{-x}$ for all $x \geq 0$. ∎

Note that the birthday paradox gives rise to a generic attack against all hash functions $\mathsf{H} : \mathcal{M} \to \mathcal{T}$:

- Let $p = 1.2 \cdot \sqrt{|\mathcal{T}|}$.
- Randomly choose $m_1, \ldots, m_n \in \mathcal{M}$,
- Compute hash values $t_1 := \mathsf{H}(m_1), \ldots, t_n := \mathsf{H}(m_n) \in \mathcal{T}$.
- With probability at least $\frac{1}{2}$, there exist $i \neq j \in \{1, \ldots, n\}$ such that $t_i = t_j$. Thus the adversary can output (m_i, m_j).

Consequently, if the length of hash values was 64 bits, then $|\mathcal{T}| = 2^{64}$ and the attack would require 2^{32} applications of the hash function. Typically, the length of hash values is 160 bits (SHA-1) or 256 bits (SHA-256): thus the attack would require 2^{80} or 2^{128} applications of the hash function, respectively, which is infeasible. We only mention here that there is already a specific attack on SHA-1 requiring only 2^{63} applications.

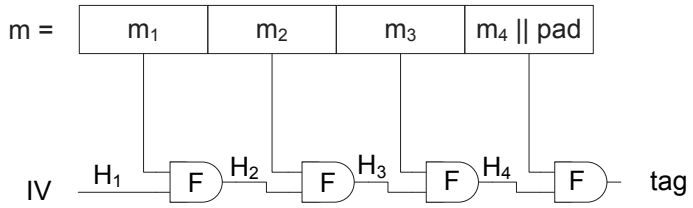


Figure 5.5: Merkle-Damgard Construction for Hash Functions

5.3.4 Merkle-Damgard Construction

Given a collision-resistant hash function F (called compression function in this subsection) that maps a fixed-length bitstring to a shorter bitstring, one can construct a collision-resistant hash function with a huge domain using the Merkle-Damgard construction.

Note that there are variations of the described construction. The one given here can be applied to bitstrings up to a length of 2^{64} bits. This bound, however, is big enough for any practical purpose, since $2^{64} \approx 10^{19}$ bits or 2305843 Terrabytes can be hashed.

Construction Let $\mathsf{F}\colon \{0,1\}^b \times \{0,1\}^n \to \{0,1\}^n$ be a collision-resistant compression (hash) function, and $IV = H_1 \in \{0,1\}^n$ be an initial value. This IV is a fixed string whose precise value does not matter here.

Given $m = m_1 \dots m_r$, where each m_i is a b-bit block, the hash-value $\mathsf{H}(m)$ is computed as follows, see the graphical illustration in Figure 5.5:

- $H_{i+1} := \mathsf{F}(m_i, H_i)$ for $i = 2, \dots, r$,
- $\mathsf{H}(m) := H_{r+1} := \mathsf{F}(m_r \mid\mid pad, H_r)$.

The padding pad is computed as follows: If $|m|$ is not a multiple of b then a block $pad = 10\dots0||msg\_len$ is appended, where msg_len is the bitstring representation of the message length (message length uses exactly the last 64 bits of the padded message), and the number of 0's is chosen so that $|m| + |pad|$ becomes a multiple of the block size b. If it already is a multiple then a new block $pad = 10\dots0||msg\_len$ with sufficiently many 0's is appended.

Lemma 5.2 *If the compression function* $\mathsf{F} : \{0,1\}^b \times \{0,1\}^n \to \{0,1\}^n$ *is collision-resistant, then the Merkle-Damgard Hash Function (MD Hash)* $\mathsf{H} : \{0,1\}^{2^{64}} \to \{0,1\}^n$ *as defined above is also collision-resistant.* □

Proof. We proof this lemma by reduction, i.e., we prove that if one finds a collision for the hash function H, then one also has a collision for the compression function F.

Let $m, m' \in \{0,1\}^{2^{64}}$ be a collision, i.e., $m \neq m'$ and $\mathsf{H}(M) = \mathsf{H}(m')$. Let us write $m = m_1 \dots m_t$ and $m' = m'_1 \dots m'_r$ as in the construction, and let us denote the intermediate values appearing in the computation of $\mathsf{H}(m)$ and $\mathsf{H}(m')$ with $H_1, \dots, H_t, H_{t+1}$ and $H_1, \dots, H'_r, H'_{r+1}$, respectively. Then we have $\mathsf{H}(m) = \mathsf{H}(m') \Rightarrow H_{t+1} = H'_{r+1} \Rightarrow \mathsf{F}(m_t \mid\mid pad, H_t) = \mathsf{F}(m'_r \mid\mid pad', H'_r)$. This means that either the arguments of the compression function are equal, or we found a collision, in which case we are done. So let us assume $(m_t \mid\mid pad, H_t) = (m'_r \mid\mid pad', H'_r)$. This in particular implies $pad = pad'$; consequently the length of m, m' are equal, thus $t = r$. Now we can proceed iteratively backwards as follows: From $H_i = H'_i$ it follows that $\mathsf{F}(m_{i-1}, H_{i-1}) = \mathsf{F}(m'_{i-1}, H'_{i-1})$, so either we found a collision for F, thus we are done, or the arguments are equal $(m_{i-1}, H_{i-1}) = (m'_{i-1}, H'_{i-1})$.

Thus, finally, we have $t = r$ and $m_i = m'_i$ for all $i = 1, \ldots, t$, thus $m = m'$, contradicting the assumption. So we had to find a collision for F in one of the above steps. ∎

5.3.5 Davies-Meyer

For using the Merkle-Damgard construction, it remains to come up with a suitable compression function. One possibility would be to use the so-called *Davies-Meyer construction*: Starting with a PRP E on $(\mathcal{K}, \mathcal{X}, \mathcal{X})$, one defines the compression function $\mathsf{F} : \mathcal{K} \times \mathcal{X} \to \mathcal{X}$ as $\mathsf{F}(M, H) := \mathsf{E}(M, H) \oplus H$. Note, however, that security of this construction can only be proved when E is an "ideal cipher", i.e., a collection of random permutations.

Theorem 5.5 (Davies-Meyer yields Collision-Resistant Compression Functions) *If E is a collection of random permutations on $(\{0,1\}^k, \{0,1\}^n, \{0,1\}^n)$, then finding collision for F takes time $2^{n/2}$.* □

An example for a hash function using this construction is SHA-256, which is an MD hash function using a Davies-Meyer compression function based on a cipher called SHACAL-2.

5.3.6 Miyaguichi-Preneel

An alternative construction is the Miyaguichi-Preneel construction, which is similar to Davies-Meyer: Let a PRP E on $(\mathcal{X}, \mathcal{X}, \mathcal{X})$ be given, and let $g : \mathcal{X} \to \mathcal{X}$ be a function mapping chaining variables from $\mathcal{X}$ to other elements of $\mathcal{X}$ (details do not matter here). Then one defines $\mathsf{F} : \mathcal{X} \times \mathcal{X} \to \mathcal{X}$ as $\mathsf{F}(M, H) := \mathsf{E}(g(H), M) \oplus H \oplus M$.

An examples for a hash function using this construction is Whirlpool, an MD hash function using a Miyaguichi-Preneel compression function based on a cipher called W (derived from AES).

5.4 HMAC

We have seen how to construct MACs with a large message space from collision-resistant hash functions and secure MACs that themselves have only a small message space. We will finally investigate how one can omit the small-MAC function and construct MACs with large message spaces directly from hash functions. The idea is to combine the message and the key in a special manner and input the result to the hash function. However, it is important to combine them in a suitable manner, as naive attempts are insecure. Let $\mathsf{H} : \mathsf{M} \to \mathsf{T}$ a Merkle-Damgard hash function. One may try the following constructions:

- $\mathsf{S}(K, M) := \mathsf{H}(K||m)$: This is completely insecure, as you will prove in the homework exercises.
- $\mathsf{S}(K, m) := \mathsf{H}(m||K)$: This can be proven secure if H is a collision-resistant hash function *and* F is a PRF. So you need two assumptions, which is not the best you can achieve.
- $\mathsf{S}((K_1, K_2), m) := \mathsf{H}(K_1||m||K_2)$: This so-called *envelope method* is secure if F is a PRF. However, it is rarely used in practice.

The method usually used in practice is HMAC. Let H be a MD-hash function with block-size n, K the key padded with 0's to the block-size n, $ipad = 3636\ldots36$, and $opad = 5c5c\ldots5c$, both in hexadecimal notation. Then signing in HMAC is defined as

- $\mathsf{HMAC} : \mathsf{S}(K, m) := \mathsf{H}(K \oplus opad \;||\; \mathsf{H}(K \oplus ipad \;||\; M))$

The verify algorithm again works by recomputing the tag.

Theorem 5.6 *If the compression function F is a PRF (when either input is used as the key), then the HMAC construction yields a secure MAC.* □

5*. Secure Channels and Key Management

5.4 Combining Privacy and Integrity

After introducing ciphers and MACs the goal now is to combine privacy and integrity. This is often called a *secure channel*, as it provides security even against active attacks. An adversary should neither be able to read any data transmitted, nor to change data without being detected.

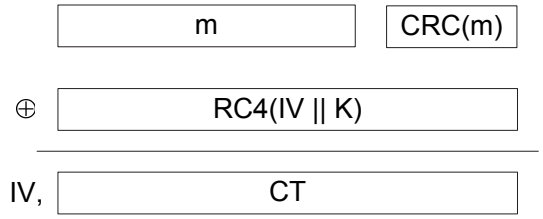


Figure 5.1: Schematic Encryption in 802.11b WEP

Let us again have a look at our favorite example, 802.11b WEP. In Figure 5.1 we sketch the relevant parts of its encryption routine. There are several weaknesses in this construction:

- Using Cyclic Redundancy Checks (CRCs) for message integrity is a poor choice, especially in the presence of active adversaries. CRC has been designed to detect errors caused by bad transmission conditions, e.g., accidentally flipped bits, not against active adversaries. First of all, it is unkeyed and, consequently, an adversary can recompute its value after changing the message. But CRC is even linear, in the sense that $\mathsf{CRC}(m \oplus B) = \mathsf{CRC}(m) \oplus \mathsf{CRC}(B)$. This can be used for the following attack against 802.11b WEP. Assume the attacker can guess which message is contained in an encryption, or at least which structure the message has. Assume he knows the message is `"My Bid is 0100$"`. Then he can easily compute an m' such that $m \oplus m' =$ `"My Bid is 0900$"`. Then he constructs $m' \,||\, \mathsf{CRC}(m')$ and xors this to c yielding

  $$c' = m \oplus m' \,||\, \mathsf{CRC}(m) \oplus \mathsf{CRC}(m') = m \oplus m' \,||\, \mathsf{CRC}(m \oplus m').$$

  This is obviously a valid ciphertext for $m \oplus m'$, thus violating message integrity.

- Another weakness is the generation of the keys that are used for the RC4 stream-cipher. The IV is used as counter, and $IV \,||\, K$ is used as key for RC4. However, RC4 is not designed to be used with related keys such as in this construction. In fact, Fluhrer, Mantin, and Shamir proved that if RC4 is used with such weak keys, then an adversary knowing only 10^6 (prefixes of) outputs of RC4 can compute the key K.

  Instead, a good method for deriving the keys would be computing $K_i := \mathsf{F}(K, IV + i)$ for a PRF F.

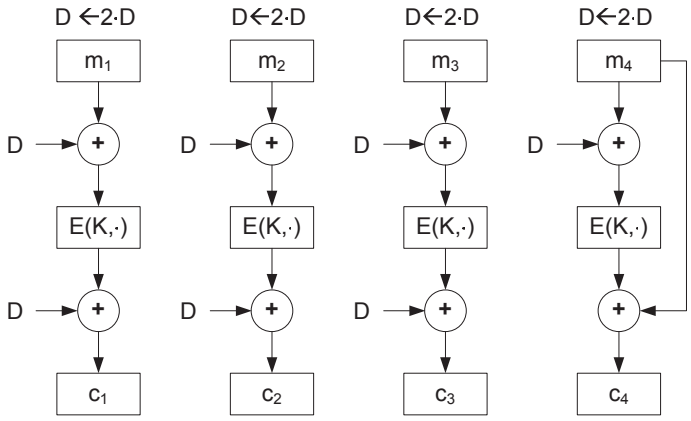


Figure 5.2: Offset Codebook Mode (OCB)

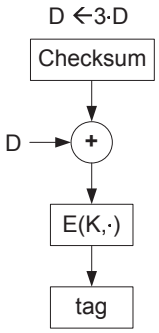


Figure 5.3: Tagging

Now let us examine more generally which possibilities exist to combine a cipher E and a MAC S, and if the resulting construction yields privacy and integrity:

- "Encrypt-then-MAC" (used in IPSec): Let $c \leftarrow \mathsf{E}(K_1, m), t \leftarrow \mathsf{S}(K_2, c)$. Then send (c, t).

  This is the recommended construction. One can prove that, for *any* CPA-secure cipher E and for *any* secure MAC S, this construction provides secrecy and integrity.

- "MAC-and-encrypt" (used in SSH v2): Let $t \leftarrow \mathsf{S}(K_1, m), c \leftarrow \mathsf{E}(K_2, m)$. Then send (c, t).

  This construction is, in general, insecure, as the definition of a MAC does not exclude, e.g., that the message is appended to the tag. However, for specific MACs such as HMAC in SSH v2, this construction is secure.

- "MAC-then-encrypt" (used in SSL): Let $t \leftarrow \mathsf{S}(K_1, m), c \leftarrow \mathsf{E}(K_2, m \mid\mid t)$. Then send c.

  This method is, in general, insecure, as, informally speaking, the tag and the encryption might interact in some bad manner. However, for specific MACs such as HMAC as in SSL, this construction is secure.

5.4.1 Offset Codebook Mode (OCB)

Constructions such as "Encrypt-then-MAC" have the disadvantage that one needs roughly $2 \cdot n$ invocations of the blockcipher for a message of length n blocks. This can be decreased to roughly $n + 1$ invocations by using OCB mode, which we will briefly describe in the sequel.

The OCB mode provides privacy and integrity at the same time. Since it is also easily parallelizable, it is easy to speed up its computation on specialized hardware. It makes use of computations in $GF(2^{128})$ to obtain different values D for the different xors in Figure 5.2. The exact description is omitted as we have not yet introduced finite fields.

Let a PRF E be given. The initialization vector IV is picked at random, and the initial D is computed as $D \leftarrow \mathsf{E}(K, IV)$. The encryption is computed according to Figure 5.2. Then a checksum is computed from these values as $checksum = m_1 \oplus m_2 \oplus \ldots \oplus m_{last-1} \oplus c_{last}$ and the tag is computed as in Figure 5.3. The ciphertext is $c = (IV, c_0, \ldots, c_{last}, tag)$.

The OCB mode is an optional part in 802.11i, using AES as the underlying cipher.

5.5 Key Exchange

When more than two parties want to communicate secretly with each other, they need to share multiple secret keys. There are several ways to handle these keys.

The naive approach is sketched in Figure 5.4: Each pair of participants shares a secret key which is used to encrypt every messages sent between these two parties. This has the obvious disadvantage that the number of keys each party has to store is $n-1$, so the total number of keys is quadratic in the number of participants. Furthermore, it is not clear how these keys should have been established in large networks such as the Internet.

A more effective way is using a central authority, often called *Key Distribution Center (KDC)*, as sketched in Figure 5.4. Each participant shares a secret key, and he may invoke the KDC to generate session keys if he wants to communicate with another party. This requires storage of 1 key per party: however, the KDC needs to be trusted and always be online, so it is a single point of failure. There exist several variants of this scheme, which we will not investigate any further.

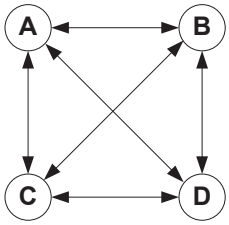


Figure 5.4: Naive Key Exchange

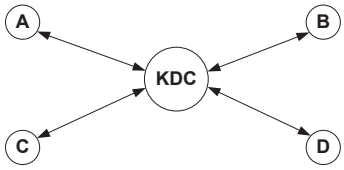


Figure 5.5: Key Exchange with a central KDC

5.5.1 Merkle Puzzles

Merkle puzzles provide a method for establishing a secret key between two parties Alice and Bob, without relying on a central authority. It was invented by Merkle in 1974 but never used in practice. Suppose Alice and Bob want to establish a shared secret key without sharing any secret before, assuming that an adversary cannot change transmitted messages, i.e., it is an passive eavesdropper. Then they can use the following method:

(1) Alice creates 2^{20} puzzles $P_i := \mathsf{E}(K_i, \text{“Puzzle } X_i\text{”} \mid\mid K_i^*)$, where K_i, X_i, K_i^* are chosen at random. The cipher E has a very short keysize, i.e., $|K_i| = 20$ bits, and $|K_i^*| = 128$ bits. She sends $(P_1, \ldots, P_{2^{20}})$ to Bob.

(2) Bob picks up a random puzzle P_i, and breaks the encryption by brute-force, thus finding K_i^* and X_i. He sends X_i to Alice and keeps K_i^*.

(3) Alice can identify the correct K_i^* when receiving X_i and looking it up.

The time needed for these computations is evaluated as follows: Alice needs 2^{20} operations to encrypt the puzzles, whereas Bob needs 2^{20} operations to brute-force the puzzle he has picked up. The attacker, however, cannot relate the (randomly chosen) X_i with any puzzle P_i, so he intuitively has to brute-force all the puzzles, resulting in a runtime of $2^{20} \cdot 2^{20} = 2^{40}$.

So one can say the Merkle puzzle has a *quadratic gap*: Whereas the honest parties perform $O(n)$ steps, an adversary needs to perform $O(n^2)$ steps to recover the secret. However, such a quadratic gap is considered not to be enough for many applications. Imagine a small handheld device and an attacker owning a cluster of several fast computers. In the second half of this course we will see that public key cryptography allows for an exponential gap.

6. Basic Number Theory Facts

In this chapter we introduce some basic facts about $\mathbb{Z}_p$ and $\mathbb{Z}_p^*$ for primes p. In particular, we study the existence and computational complexity of finding inverses, of exponentiation, of solving linear and quadratic equations and of extracting square roots. Later in the course we will move from groups $\mathbb{Z}_p$ for primes p to groups $\mathbb{Z}_n$ where n is a composite number (usually consisting of the product of two primes). We do not give any proofs here since they can be found in essentially every textbook on the topic; moreover, none of these proofs will matter in the subsequent lectures.

6.1 Basic Number Theory Facts – Arithmetic Modulo Primes

In the following, we are dealing with large primes p, e.g., primes that are 1024 bits long.

6.1.1 $\mathbb{Z}_p$ and Some Basic Facts

For a prime p let $\mathbb{Z}_p := \{0, 1, 2, \ldots, p-1\}$. Elements of $\mathbb{Z}_p$ can be added modulo p and multiplied modulo p as usual. For adding two elements $a, b \in \mathbb{Z}_p$, we often write $a + b \bmod p$ instead of $a + b$ to make clear in which group the addition is performed, similarly for multiplication.

It turns out that all elements $g \in \mathbb{Z}_p$ except for $g = 0$ are *invertible (with respect to multiplication)*, i.e., for every such g there exists some element h such that $g \cdot h = 1 \bmod p$. We write g^{-1} to denote the inverse of g. We denote by $\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$ the set of invertible elements in $\mathbb{Z}_p$.

Let us now mention *Fermat's little theorem.*

Theorem 6.1 (Fermat) *For any $g \in \mathbb{Z}_p^*$, we have $g^{p-1} = 1 \bmod p$.* □

Here and in the following, we complement definitions and algorithms with simple examples to foster basic understanding.

Example 6.1 *As an example for illustrating Fermat's little theorem, we have* $3^4 \bmod 5 = 81 \bmod 5 = 1 \bmod 5$.

Fermat's little theorem reasons about exponentiation in $\mathbb{Z}_p^*$. Getting to the computational side, it is easy to see that addition and multiplication modulo p can be done in polynomial time (with respect to the bitlength of p). In contrast to $\mathbb{Z}$ however, it turns out that even exponentiation in $\mathbb{Z}_p^*$ can be achieved in polynomial time in the bitlength of p, using the so-called *repeated squaring technique.*

Definition 6.1 (Repeated Squaring Technique) *Let $g \in \mathbb{Z}_p^*$ and $a \in \{0, \ldots, p-2\}$. Then $g^a \bmod p$ can be computed as follows. Let the bit representation of a be given by $a = \sum_{i=0}^{k-1} a_i 2^i$. Then we have*

$$\begin{aligned} g^{\sum_{i=0}^{k-1} a_i 2^i} &= (g)^{a_0} \cdot (g^2)^{\sum_{i=1}^{k-1} a_i 2^i} \\ &= (g)^{a_0} \cdot (g^2)^{a_1} \cdot (g^4)^{\sum_{i=2}^{k-1} a_i 2^i} = \ldots \\ &= (g)^{a_0} \cdot (g^2)^{a_1} \cdot (g^4)^{a_2} \cdots (g^{2^{k-1}})^{a_{k-1}} \end{aligned}$$

It is easy to see that the computing time is bounded by $O(k^3)$ since it takes $O(k^2)$ time to square in $\mathbb{Z}_p$, and we need to do k repeated squarings to successively compute $(g^{2^i} \bmod p)$ for all $1 \leq i \leq k$. $\diamond$

Example 6.2 *Let $g = 3$, $a = 5$, and $p = 7$. Thus $a_0 = a_2 = 1$ and $a_1 = 0$. We first compute $g = 3$, $g^2 = 3^2 = 2 \bmod 7$, $g^4 = 2^2 = 4 \bmod 7$. Thus we get $g^a = g^{a_0} \cdot (g^2)^{a_1} \cdot (g^4)^{a_2} = g \cdot g^4 = 3 \cdot 4 = 5 \bmod 7$.*

Fermat's little theorem furthermore entails a very simple procedure for computing the inverse of an element $g \in \mathbb{Z}_p^*$: one simply sets $g^{-1} := g^{p-2} \bmod p$. One easily verifies that $g \cdot g^{p-2} = g^{p-1} = 1 \bmod p$, according to Fermat's little theorem.

Example 6.3 *The inverse of 3 modulo 5 is given by $3^{5-2} = 27 = 2 \bmod 5$, and indeed $2 \cdot 3 = 1 \bmod 5$.*

Note further that we have $2^{-1} \bmod p = \frac{p+1}{2}$ for all p, since $2 \cdot \frac{p+1}{2} = p + 1 = 1 \bmod p$.

Moreover, joining the repeated squaring technique with this way of computing inverses entails an efficient algorithm for solving linear equations of the form $a \cdot x = b \mod p$: one simply computes $x = b \cdot a^{-1} = b \cdot a^{p-2} \bmod p$. One again easily verifies that $a \cdot (b \cdot a^{p-2}) = b \cdot a^{p-1} = b \cdot 1 = b \bmod p$, where we again exploited Fermat's little theorem.

What however remains unclear at this point is how to solve quadratic equations or equations of even higher order. We will discuss this in the next subsection in detail for quadratic equations while referring to the literature for equations of higher order.

6.1.2 $\mathbb{Z}_p^*$ and Some Basic Facts

We start with the following important theorem.

Theorem 6.2 *For every prime p, $\mathbb{Z}_p^*$ constitutes a* cyclic group*, i.e., there exists some $g \in \mathbb{Z}_p^*$ such that $\mathbb{Z}_p^* = \{1 = g^0, g, g^2, g^3, \ldots, g^{p-2}\}$. An element g with this property is called a* generator *of $\mathbb{Z}_p^*$.* $\square$

In other words, if g is a generator of $\mathbb{Z}_p^*$, then for every $h \in \mathbb{Z}_p^*$ there exists some $i \in \{0, \ldots, p-2\}$ such that $h = g^i \bmod p$.

Example 6.4 *The element 3 constitutes a generator of $\mathbb{Z}_7^*$ since $\{1, 3, 3^2, 3^3, 3^4, 3^5, 3^6\} = \{1, 3, 2, 6, 4, 5\} (\text{mod} 7) = \mathbb{Z}_7^*$.*

However, not every element of $\mathbb{Z}_p^*$ is a generator. The number 2, for example, is not a generator of $\mathbb{Z}_7^*$, since we we have $\{1, 2, 2^2, 2^3, 2^4, 2^5, 2^6\} = \{1, 2, 4\} (\text{mod} 7) \neq \mathbb{Z}_7^*$.

Although not every element is a generator, every element generates at least a *subgroup* $\langle g \rangle$ of $\mathbb{Z}_p^*$, defined as $\langle g \rangle := \{h \in \mathbb{Z}_p^* \mid \exists i \in \{0, \ldots, p-2\} \colon h = g^i\}$. For generators g of $\mathbb{Z}_p^*$, we thus have $\langle g \rangle = \mathbb{Z}_p^*$.

The *order* of $g \in \mathbb{Z}_p^*$ is defined to be the size of the group it generates. Equivalently, it is the smallest positive integer i such that $g^i = 1 \bmod p$. We denote the order of $g \in \mathbb{Z}_p^*$ by $\mathsf{ord}_p(g)$. If we use the aforementioned examples, we obtain $\mathsf{ord}_7(3) = 6$ and $\mathsf{ord}_7(2) = 3$. If the factorization of $p-1$ is known then there is a simple and efficient algorithm to determine $\mathsf{ord}_p(g)$ for any $g \in \mathbb{Z}_p^*$ (We will see this later).

Let us conclude this subsection with the famous theorem of Lagrange.

Theorem 6.3 (Lagrange) *For all $g \in \mathbb{Z}_p^*$ we have that $\mathsf{ord}_p(g)$ divides $p-1$.* □

6.1.3 Quadratic Residues and Quadratic Non-Residues

The *square root* of an element $g \in \mathbb{Z}_p^*$ is an element $h \in \mathbb{Z}_p^*$ such that $h^2 = g \bmod p$. We as usual write $\sqrt{g}$ instead of h. For instance, we have that $\sqrt{2} = 3 \bmod 7$ since $3^2 = 2 \bmod 7$. Square root however do not necessarily exist, e.g., 3 does not have a square root modulo 7.

An element $g \in \mathbb{Z}_p^*$ is called a *Quadratic Residue* (or *QR* for short) if it has a square root in $\mathbb{Z}_p^*$. Otherwise it is called a *Quadratic Non-Residue* (or *QNR* for short).

How many square roots does an element $g \in \mathbb{Z}_p^*$ have? Consider the equation $x^2 = y^2 \bmod p$. This is equivalent to $0 = x^2 - y^2 = (x-y)(x+y) \bmod p$. Since $\mathbb{Z}_p$ is an "integral domain" we know that $x = y \bmod p$ or $x = -y \bmod p$. Hence, elements in $\mathbb{Z}_p^*$ have either zero square roots or two square roots. If h is the square root of g modulo p then $-h$ is also a square root of g modulo p.

The following theorem of Euler tells us how to decide if an element is a quadratic residue or not.

Theorem 6.4 (Euler) *An element $g \in \mathbb{Z}_p^*$ is a QR if and only if $g^{(p-1)/2} = 1 \bmod p$.* □

Example 6.5 *We have that 2 is a QR in $\mathbb{Z}_7^*$ since $2^{(7-1)/2} = 1 \bmod 7$; on the other hand 3 is a QNR in $\mathbb{Z}_7^*$ since $3^{(7-1)/2} = -1 \bmod 7$.*

A special case we are often interested in are the roots of the multiplicative unit: For every $g \in \mathbb{Z}_p^*$, we have that $h = g^{(p-1)/2}$ is a square root of 1 since $h^2 = g^{p-1} = 1 \bmod p$ according to Fermat's little theorem. One the other hand, the element 1 can have at most two square roots as we have seen, and 1 and -1 are of course square roots of 1 modulo every prime p. Consequently, we know that $g^{(p-1)/2} \bmod p \in \{1, -1\}$ for every $g \in \mathbb{Z}_p^*$.

Next we define the Legendre symbol.

Definition 6.2 (Legendre Symbol) *For $g \in \mathbb{Z}_p$, we define*

$$\left(\frac{g}{p}\right) = \begin{cases} 1 & \text{if } g \text{ is a QR in } \mathbb{Z}_p \\ -1 & \text{if } g \text{ is not a QR in } \mathbb{Z}_p \\ 0 & \text{if } g = 0 \bmod p \end{cases}$$

◇

Euler's theorem implies that $\left(\frac{g}{p}\right) = g^{(p-1)/2} \bmod p$. Thus the Legendre symbol can be efficiently computed, e.g., using the repeated squaring technique.

We conclude with some easy additional facts:

1. Let g be a generator of $\mathbb{Z}_p^*$ and let $h = g^r$ for some integer r. Then h is a QR in $\mathbb{Z}_p^*$ if and only if r is even. This implies that the Legendre symbol reveals the parity of r. More precisely, $\left(\frac{g^r}{p}\right) = \left(\frac{g}{p}\right)^r$.

2. Since $x = g^r$ is a QR if and only if r is even, it follows that exactly half the elements of $\mathbb{Z}_p$ are QR's.

3. When $p = 3 \bmod 4$ computing square roots of a QR $g \in \mathbb{Z}_p^*$ is easy. One simply computes $h_1 = g^{p+1}/4 \bmod p$ and $h_2 = -g^{p+1}/4 \bmod p$. This gives the correct result since $h_1^2 = h_2^2 = g^{p+1/2} = g \cdot g^{p-1/2} = g \cdot \left(\frac{g}{p}\right) = g \cdot 1 = g \bmod p$.

4. When $p = 1 \bmod 4$ computing square roots of a QR $g \in \mathbb{Z}_p^*$ is possible but somewhat more complicated (one needs a randomized algorithm).

5. There exists a simple algorithm for solving quadratic equations in $\mathbb{Z}_p$: We know that if a solution to $ax^2 + bx + c = 0 \bmod p$ exists then it is given by

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} (\mathrm{mod} p).$$

   Hence, the equation has a solution in $\mathbb{Z}_p$ if and only if $A := b^2 - 4ac$ is a QR in $\mathbb{Z}_p^*$ (or if $A = 0$). Using our algorithm for taking square roots in $\mathbb{Z}_p^*$ we can find $\sqrt{A} \bmod p$ and recover x_1 and x_2.

6. What about higher-order equations in $\mathbb{Z}_p$? We only need to know that there exists an efficient randomized algorithm that solves any equation of degree d in time polynomial in d and the length of p.

6.1.4 Efficient Computing in $\mathbb{Z}_p$ — Upper Time Bounds

We briefly summarize here in which time the respective operations can be computed.

1. Adding two elements $x, y \in \mathbb{Z}_p$ can be done in linear time in the length of p.

2. Multiplying two elements $x, y \in \mathbb{Z}_p$ can be done in quadratic time in the length of p. If p is n bits long, more clever (and practical) algorithms work in time $O(n^{1.7})$ (rather than $O(n^2)$).

3. Inverting an element $x \in \mathbb{Z}_p^*$ can be done in quadratic time in the length of p.

4. Using the repeated squaring technique, $x^r \bmod p$ can be computed in time $(\log_2 r)O(n^2)$ where p is n bits long. Note that the algorithm takes linear time in the length of r.

6.1.5 Summary of Easy and Hard Problems

Let p be a 1024 bit prime. The following problems constitute easy problems in $\mathbb{Z}_p$:

1. Adding and multiplying elements.

2. Generating a random element in $\mathbb{Z}_p$ or a random generator of $\mathbb{Z}_p^*$.

3. Computing $g^r \bmod p$ even if r is very large.

4. Inverting an element and solving linear systems.

5. Testing if an element is a QR and computing its square root if it is a QR.

6. Solving polynomial equations of degree d can be done in polynomial time in d.

The following problems are believed to be hard (intractable) in $\mathbb{Z}_p$:

1. Let g be a generator of $\mathbb{Z}_p^*$. Given $x \in \mathbb{Z}_p^*$, find r such that $x = g^r \bmod p$. This is known as the *discrete logarithm problem.*

2. Let g be a generator of $\mathbb{Z}_p^*$. Given $i, j \in \mathbb{Z}_p^*$ where $i = g^a$ and $j = g^b$. Find $h = g^{ab}$. This is known as the *Diffie-Hellman problem.*

3. Finding roots of sparse polynomials of high degree. For example finding a root of: $x^{2^{1000}} + 23 \cdot x^{2^{167}} + 9 \cdot x^{2^{62}} + x^3 + 7 = 0 \bmod p$.

7. Public-key Encryption, Diffie-Hellman, ElGamal

This is our first chapter on *public-key encryption*, also called *asymmetric encryption*. Each party has a pair of keys: The *public key* is known to everybody and is used for encryption, while the *secret/private key* is kept secret and is used for decryption. Thus everybody can encrypt messages with the public key while only the owner of the corresponding secret key can decrypt them. Since only public keys have to be transmitted, key distribution is considerably simplified as only the authenticity of exchanged keys is relevant. This is an important difference with respect to distribution of symmetric keys where also the privacy of exchanged keys has to be guaranteed.

7.1 Definition of Public-key Encryption Schemes

The next definition formalizes the notion of public-key encryption schemes:

Definition 7.1 (Public-key Encryption Scheme) *A public-key encryption scheme consists of three efficient algorithms* $(\mathsf{Gen}, \mathsf{E}, \mathsf{D})$*, which are defined as follows:*

- *The randomized* key generation *algorithm* Gen *takes the security parameter in unary representation (for defining complexity in the length of the security parameter) as input and returns a pair* (pk, sk) *of keys, the* public key *pk and the matching* secret key *sk.*
- *The* encryption algorithm E *takes the public key pk and a plaintext m and returns a ciphertext c. This algorithm may be randomized.*
- *The deterministic* decryption algorithm D *takes the secret key sk and a ciphertext c and returns a plaintext m.*
- *The* message space $\mathcal{M}_{pk}$ *associated to a public key pk is the set of plaintexts m for which* $\mathsf{E}(pk, m)$ *never returns a distinguished error symbol* $\downarrow$*. We require that, for any key-pair* (pk, sk) *that might be output by* Gen *and for any message* $m \in \mathcal{M}_{pk}$*, if* $c \leftarrow \mathsf{E}(pk, m)$ *then* $m = \mathsf{D}(sk, c)$*.*

$\diamond$

We assume that all algorithms (including challengers and adversaries in the following) get the security parameter in unary representation as input so that the efficiency of such algorithms can be meaningfully measured in the security parameter. We don't write this explicitly in the following to increase readability.

7.2 Semantic Security of Public-key Encryption Schemes against CPA

Security of public-key encryption schemes against chosen-plaintext attack (passive adversaries) is defined similar to the symmetric case.

Definition 7.2 (CPA Challenger) *Let* $\mathsf{PubEnc} = (\mathsf{Gen}, \mathsf{E}, \mathsf{D})$ *be a public-key encryption scheme, and let* n *be the security parameter. The* CPA challenger *for* PubEnc *and* n *is defined as follows:*

- *First, it chooses a bit* b*, creates keys* $(pk, sk) \leftarrow \mathsf{Gen}(n)$*, and outputs* pk*.*
- *It then receives two plaintexts* $m_0, m_1 \in \mathcal{M}_{pk}$ *with* $|m_0| = |m_1|$*, computes an encryption* $c \leftarrow \mathsf{E}(pk, m_b)$*, and outputs* c*.*

◇

An adversary wins the game against the CPA challenger if, intuitively, it is able to deduce the bit b better than by pure guessing. In the following, $Exp_{\mathsf{A}}^{\mathsf{CPA}}(b)$ denotes the experiment where the adversary A interacts with the CPA challenger whose bit is chosen as b. Furthermore $Exp_{\mathsf{A}}^{\mathsf{CPA}}(b) = 0$ and $Exp_{\mathsf{A}}^{\mathsf{CPA}}(b) = 1$ denote the event that the adversary outputs 0 and 1 in the respective experiment.

The advantage can now be defined as usual. Note that the security parameter n induces a uniform sequence of public-key encryption schemes, each of which is defined for a fixed value of n, as well as a uniform sequence of CPA challengers and adversaries. Thus, the public-key encryption algorithms Gen, E, D, the challenger CPA, and the adversary A constitute individual algorithms that are parameterized by the security parameter. Hence the advantage naturally also constitutes a function of n. (This might be clearer for public-key encryption than it was for symmetric encryption, since key generation as well as encryption and decrypting algorithms are defined for arbitrary values of n already. This stands in contrast to, e.g., AES, which was only defined for fixed key sizes, and where the definition of sequences consequently might have looked artificial.)

Definition 7.3 (CPA Advantage) *Let* $\mathsf{PubEnc} = (\mathsf{Gen}, \mathsf{E}, \mathsf{D})$ *be a public-key encryption scheme. The* advantage *of an adversary* A *against the CPA challenger for* E *is defined as follows (as a function of the security parameter* n *since* PubEnc*, the* CPA *challenger, and* A *are parameterized in* n*):*

$$Adv^{\mathsf{CPA}}[\mathsf{A}, \mathsf{PubEnc}](n) := \left| \Pr\left[Exp_{\mathsf{A}}^{\mathsf{CPA}}(0) = 1 \right] - \Pr\left[Exp_{\mathsf{A}}^{\mathsf{CPA}}(1) = 1 \right] \right|.$$

◇

Definition 7.4 (Semantic Security of Public-key Encryption against CPA) *A public-key encryption scheme* $\mathsf{PubEnc} = (\mathsf{Gen}, \mathsf{E}, \mathsf{D})$ *is* semantically secure against chosen-plaintext attack (CPA) *if for all efficient adversaries* A*, the advantage* $Adv^{\mathsf{CPA}}[\mathsf{A}, \mathsf{PubEnc}](n)$ *is negligible in* n*.* ◇

7.3 Semantic Security of Public-key Encryption Schemes against Active Adversaries (CCA2)

Security against active adversaries is defined similarly, but the adversary is given the possibility to additionally decrypt any ciphertexts it wants. Note that there are two variants of this. For *non-adaptive chosen-ciphertext attack* or *lunchtime attack*, the adversary is allowed to decrypt messages only before selecting the messages m_0, m_1. For *adaptive chosen-ciphertext attack* the adversary may ask for decryptions even after receiving the challenge-ciphertext, except that he is forbidden to ask for the decryption of the challenge-ciphertext itself. The latter notion is more powerful and constitutes the standard security definition of public-key encryption in modern cryptography. It is abbreviated CCA2.

Definition 7.5 (CCA2 Challenger) *Let* $\mathsf{PubEnc} = (\mathsf{Gen}, \mathsf{E}, \mathsf{D})$ *be a public-key encryption scheme,* n *be the security parameter, and* $\downarrow$ *denote an arbitrary error symbol that is not in the range of* E *for*

any pk and m. The CCA2 challenger *for* PubEnc *and n maintains a variable c^* and is defined as follows:*

- *First, it chooses a bit b, sets $c^* := \downarrow$, creates keys $(pk, sk) \leftarrow \mathsf{Gen}(n)$, and outputs pk.*
- *If it receives a decryption query c it does the following: If $c \neq c^*$, it computes $m := \mathsf{D}(sk, c)$ and outputs m.*
- *If it receives two plaintexts $m_0, m_1 \in \mathcal{M}_{pk}$ with $|m_0| = |m_1|$, it computes an encryption $c \leftarrow \mathsf{E}(pk, m_b)$, sets $c^* := c$, and outputs c.*

$\diamond$

An adversary wins the game against the CCA2 challenger if it is able to deduce the bit b better than by pure guessing. In the following, $Exp_{\mathsf{A}}^{\mathsf{CCA2}}(b)$ denotes the experiment where the adversary A interacts with the CCA2 challenger whose bit is chosen as b. Furthermore $Exp_{\mathsf{A}}^{\mathsf{CCA2}}(b) = 0$ and $Exp_{\mathsf{A}}^{\mathsf{CCA2}}(b) = 1$ denote the event that the adversary outputs 0 and 1 in the respective experiment.

The advantage is then defined as a function of n as usual.

Definition 7.6 (CCA2 Advantage) *Let* PubEnc $= (\mathsf{Gen}, \mathsf{E}, \mathsf{D})$ *be a public-key encryption scheme. The* advantage *of an adversary* A *against the CCA2 challenger for* PubEnc *is defined as follows (as a function of the security parameter n again since* PubEnc*, the* CCA2 *challenger, and* A *are parameterized in n):*

$$Adv^{\mathsf{CCA2}}[\mathsf{A}, \mathsf{PubEnc}](n) := \left| \mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{CCA2}}(0) = 1 \right] - \mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{CCA2}}(1) = 1 \right] \right|.$$

$\diamond$

Definition 7.7 (Semantic Security of Public-key Encryption against CCA2) *A public-key encryption scheme* PubEnc $= (\mathsf{Gen}, \mathsf{E}, \mathsf{D})$ *is* semantically secure against adaptive chosen-ciphertext attack (CCA2) *if for all efficient adversaries* A*, the advantage $Adv^{\mathsf{CCA2}}[\mathsf{A}, \mathsf{PubEnc}](n)$ is negligible in n.*

$\diamond$

7.4 The Discrete Logarithm and Diffie-Hellman Problems

Next we will define three problems that are conjectured to be hard in certain groups. They can be defined in arbitrary (cyclic) groups. In practice, however, they are most often used in $\mathbb{Z}_p^*$ for a prime p or in subgroups of prime order q of $\mathbb{Z}_p^*$ where p is a prime as well and $q | p-1$. We will denote such subgroups of $\mathbb{Z}_p^*$ by G_q. The following definitions will be specifically given for $\mathbb{Z}_p^*$ and for subgroups G_q of $\mathbb{Z}_p^*$, i.e., consider a group G which is either G_q or $\mathbb{Z}_p^*$ for some p, q. For $g \in G$ one defines the *discrete exponentiation* as

$$\mathsf{DExp}_g : \mathbb{N} \to G, \quad x \mapsto g^x.$$

The *discrete logarithm* is defined as

$$\mathsf{DLog}_g : G \to \mathbb{N}, \quad h \mapsto x.$$

where $x \in \mathbb{N}$ is the smallest natural number such that $g^x = h$, with $\mathsf{DLog}_g(h) := \infty$ if no such x exists. The discrete logarithm DLog is conjectured to be hard to compute in G, and in many more groups as well. For a generator g of G, the *Diffie-Hellman function* is defined as

$$\mathsf{DH}_g : G \times G \to G, \quad (g^x, g^y) \mapsto g^{xy}.$$

The *computational Diffie-Hellman (CDH) problem* is defined as follows: Let $x, y \leftarrow_{\mathcal{R}} \{1, \ldots, |G|\}$ and g a generator of G, compute the Diffie-Hellman function $\mathsf{DH}(g^x, g^y)$, i.e., given g^x, g^y, compute g^{xy}. This is conjectured to be hard in G. This in particular led to the Diffie-Hellman key exchange protocol and to the ElGamal encryption scheme, which we will describe later.

It is easy to see that if the discrete logarithm is easy to compute, then the computational Diffie-Hellman problem is easy to solve. In other words, solving the CDH problem is at most as difficult as computing the discrete logarithm. The converse direction is, in general, unknown.

The *Decisional Diffie-Hellman (DDH) problem* is closely related to the CDH problem. The difference is that the adversary is not required to compute the value g^{xy}, but he needs to distinguish it from a value g^z for a random z. This assumption will play an important role in the sequel, thus we define it formally, again in terms of a game. We however only define it for subgroups G_q of $\mathbb{Z}_p^*$, where the DDH problem is indeed considered hard; in $\mathbb{Z}_p^*$ the DDH problem can be efficiently solved, see below.

For a rigorous definition of the game, the following technical subtlety arises: The group G_q is a subgroup of $\mathbb{Z}_p^*$, so strictly speaking we needed two "security parameters" n and n^* measuring the size of p and q, respectively. We avoid this by saying that the bit-length of q is the primary security parameter n, and that n^* can be derived from n via a public polynomially bounded function $n_p(\cdot)$, i.e., $n^* := n_p(n)$.

Definition 7.8 (DDH Challenger) *The* DDH challenger *(for subgroups G_q of $\mathbb{Z}_p^*$) for a given security parameter n is defined as follows:*

- *Firstly, it randomly chooses an n-bit prime q and an $n_p(n)$-bit prime p such that $q|p-1$. After that, it randomly chooses an element $g \in \mathbb{Z}_p^*$ of order q.*
- *Secondly, it randomly chooses a bit b and values $x, y, z \leftarrow_{\mathcal{R}} \{1, \ldots, q\}$. Depending on the value of b, it outputs*
  - *$(q, p, g, g^x, g^y, g^{xy})$ if $b = 0$.*
  - *(q, p, g, g^x, g^y, g^z) if $b = 1$.*

$\diamond$

An adversary wins the game against the DDH challenger if it is able to deduce the bit b significantly better than by pure guessing. Again, the adversary advantage captures how much better an adversary can do than to purely guess the bit. In the following, $Exp_{\mathsf{A}}^{\mathsf{DDH}}(b)$ denotes the experiment where the adversary A interacts with the DDH challenger whose bit is chosen as b. Furthermore $Exp_{\mathsf{A}}^{\mathsf{DDH}}(b) = 0$ and $Exp_{\mathsf{A}}^{\mathsf{DDH}}(b) = 1$ denote the event that the adversary outputs 0 and 1 in the respective experiment. The advantage is defined as usual.

Definition 7.9 (DDH Advantage) *The advantage of an adversary A against the DDH challenger is defined as follows (as a function of the security parameter n again):*

$$Adv^{\mathsf{DDH}}[\mathsf{A}](n) := \left| \mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{DDH}}(0) = 1\right] - \mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{DDH}}(1) = 1\right] \right|.$$

$\diamond$

Assumption 1 (Hardness of the DDH Problem in G_q) *The DDH problem in groups G_q as constructed above is conjectured to be hard, i.e., it is conjectured that $Adv^{\mathsf{DDH}}[\mathsf{A}](n)$ is negligible in the security parameter n.*

It is again easy to see that, if the CDH problem is easy, i.e., if one can compute g^{xy} given g^x and g^y, then the DDH problem can be decided efficiently by comparing the computed value with the given value. The converse is not known in general, but we will see now that in $\mathbb{Z}_p^*$ the DDH problem is easy to solve, while the CDH problem is conjectured to be hard. To see that the DDH problem is easy to solve in $\mathbb{Z}_p^*$, we make the following observation: We have seen in Remark 1 in Section 6.1.3 that if g is a generator of $\mathbb{Z}_p^*$, then g^r is a quadratic residue (QR) iff r is even; hence for r randomly chosen from $\{1,\ldots,p-1\}$, g^r is a QR with probability $\frac{1}{2}$. However, if $x, y \leftarrow_{\mathcal{R}} \{1,\ldots,p-1\}$ then the probability that g^{xy} is a QR is precisely $\frac{3}{4}$.

Thus an adversary can do the following: receiving (q,p,g,h,i,k) from the DDH challenger, the adversary outputs 0 if k is a QR, and 1 if k is not a QR. (Note that this can be easily computed using the Legendre symbol.) Then one easily calculates

$$\mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{DDH},\mathbb{Z}_p^*}(0)=0\right]=\frac{3}{4} \quad \text{and} \quad \mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{DDH},\mathbb{Z}_p^*}(1)=0\right]=\frac{1}{2},$$

and thus

$$Adv^{\mathsf{DDH},\mathbb{Z}_p^*}[\mathsf{A}]=\left|\frac{3}{4}-\frac{1}{2}\right|=\frac{1}{4}.$$

Before continuing with applications of the above concepts, let us briefly take a look at a property of the DLog called *random self-reducibility*. It turns out that, if one can compute the DLog for a small fraction of inputs, then one can compute it for all inputs.

Lemma 7.1 (Random Self-reducibility of DLog) *Let g be a generator of $\mathbb{Z}_p^*$. Suppose there exists an algorithm A for computing $\mathsf{DLog}_g(h)$ in time T whenever $h \in S \subseteq \mathbb{Z}_p^*$, where $|S| = \epsilon \cdot |\mathbb{Z}_p^*|$. Then there exists an algorithm B for computing $\mathsf{DLog}_g(h)$ for all h in expected time T/ϵ.* □

Proof. Since g is a generator, the discrete log of h to the base g is an integer x where $0 < x \leq p-1$ and $h = g^x$. The algorithm B picks a random $y \leftarrow_{\mathcal{R}} \{1,\ldots,p-1\}$, uses A to compute the discrete logarithm z of $h \cdot g^y$, and tests if the result was correct. It repeats the choice of y and the subsequent testing until a correct discrete logarithm is output. The element g^y is distributed uniformly in $\mathbb{Z}_p^*$ as y is uniformly distributed in $\{1,\ldots,p-1\}$, thus the probability that $h \cdot g^y \in S$ is ϵ. In this case A computes the logarithm z of $h \cdot g^y$ correctly. B then simply outputs $z - y \bmod p-1$ as the discrete logarithm of h. This is correct since

$$g^{z-y} = g^z \cdot g^{-y} = h \cdot g^y \cdot g^{-y} = h.$$

The expected running time of algorithm B is T/ϵ: it needs to do an expected number of $\frac{1}{\epsilon}$ loops, each of which takes time T (plus a small polynomial overhead in each loop for choosing the values y and testing if the logarithm was correctly computed, which we neglected). ■

7.5 Diffie-Hellman Key Exchange

The Diffie-Hellman key exchange was invented by Wittfield Diffie and Martin Hellman in 1976. Using this protocol it is possible for two parties Alice and Bob to agree on a secret value in the presence of a passive adversary observing their entire communication. The algorithm is depicted in Figure 7.1. It was traditionally proposed for $\mathbb{Z}_p^*$ for a large prime p but works in principle in any cyclic group (and is today usually used in subgroups G_q of $\mathbb{Z}_p^*$).

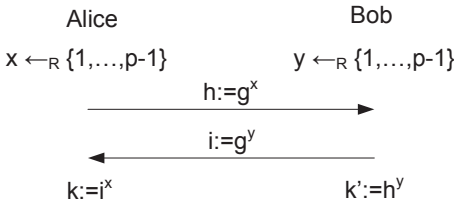


Figure 7.1: Diffie-Hellman Key Exchange

Let g be a generator of $\mathbb{Z}_p^*$ which is publicly known. Alice chooses $x \leftarrow_{\mathcal{R}} \{1, \ldots, p-1\}$ and sends $h := g^x \in \mathbb{Z}_p^*$ to Bob, whereas Bob chooses $y \leftarrow_{\mathcal{R}} \{1, \ldots, p-1\}$ and sends $i := g^y \in \mathbb{Z}_p^*$ to Alice. Then Alice computes $k := i^x$ and Bob computes $k' := h^y$. Then an easy calculation shows that

$$k = i^x = (g^y)^x = g^{xy} = (g^x)^y = h^y = k'.$$

So both Alice and Bob know a common secret $k \in \mathbb{Z}_p^*$ that they can use do derive keys for other encryption schemes such as AES.

7.6 The ElGamal Encryption System

The ElGamal encryption scheme is the first public-key encryption scheme that we will explore. Its security is based on the hardness of the Decisional Diffie-Hellman problem. The ElGamal encryption scheme $\mathsf{E}^{\mathsf{ElGamal}} = (\mathsf{Gen}, \mathsf{E}, \mathsf{D})$ is defined as follows for security parameter n:

- Key generation Gen chooses an n-bit prime q and an $n_p(n)$-bit prime p such that $q|p-1$. After that, it randomly chooses an element $g \in \mathbb{Z}_p^*$ of order q. Furthermore, it chooses a random value $x \leftarrow_{\mathcal{R}} \{1, \ldots, q\}$ and computes $h := g^x$. It sets $pk = (q, p, g, h)$ and $sk = (q, p, g, x)$ and outputs the key-pair (pk, sk).
- Encryption $\mathsf{E}(pk, m)$ for messages $m \in G_q := \langle g \rangle$ is computed as follows: Choose $y \leftarrow_{\mathcal{R}} \{1, \ldots, q\}$, compute $i := g^y$, $c' := m \cdot h^y$ and output $c := (i, c')$.
- Decryption $\mathsf{D}(sk, (i, c'))$ works by computing $m := c' \cdot (i^x)^{-1}$.

The correctness of this scheme can be verified as follows: Let $pk = (q, p, g, h = g^x)$, $sk = (q, p, g, x)$ and $m \in G_q$. Then

$$\begin{aligned} \mathsf{D}(sk, \mathsf{E}(pk, m)) &= \mathsf{D}(sk, (g^y, m \cdot h^y)) \\ &= m \cdot h^y \cdot (g^y)^{-x} \\ &= m \cdot g^{xy} \cdot g^{-xy} \\ &= m. \end{aligned}$$

One can easily see that a total break of this scheme, i.e., the ability to compute the secret value x given an encryption c, requires solving the discrete logarithm problem, and recovering the plaintext from an encryption requires solving the Computational Diffie-Hellman problem. However, we will see that for semantic security against CPA, it's the hardness of the Decisional Diffie-Hellman problem that we have to be assuming.

Theorem 7.1 (CPA-Security of ElGamal) *If the DDH problem for subgroups G_q of $\mathbb{Z}_p^*$ as constructed above is hard, then $\mathsf{E}^{\mathsf{ElGamal}} = (\mathsf{Gen}, \mathsf{E}, \mathsf{D})$ is semantically secure against chosen-plaintext*

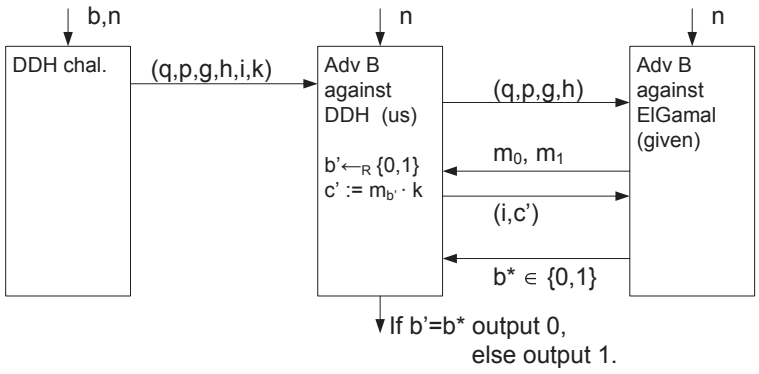


Figure 7.2: The Security of ElGamal against Passive Adversaries

attack (CPA). □

Proof. Given an adversary A against the CPA challenger for ElGamal we construct an adversary B against the DDH challenger as depicted in Figure 7.2. We need to calculate

$$Adv^{\mathsf{DDH}}[\mathsf{B}] = \left|\mathsf{Pr}\left[Exp_{\mathsf{B}}^{\mathsf{DDH}}(0)=1\right] - \mathsf{Pr}\left[Exp_{\mathsf{B}}^{\mathsf{DDH}}(1)=1\right]\right|. \tag{7.1}$$

First, observe that for the second expression we have

$$\mathsf{Pr}\left[Exp_{\mathsf{B}}^{\mathsf{DDH}}(1)=1\right] = \frac{1}{2}, \tag{7.2}$$

as the value k output by the DDH challenger in this experiment is g^z for z randomly chosen from $\{1,\ldots,q\}$. Thus g^z is random in G_q and consequently the encryption $c' = m_{b'} \cdot k$ computed by B constitutes a One-time Pad in G_q. Thus in the experiment $Exp_{\mathsf{B}}^{\mathsf{DDH}}(1)$ the adversary A gains no information on B's random bit b'. Now let us calculate the first expression of the absolute value in Equation 7.1:

$$\begin{aligned}
& \mathsf{Pr}\left[Exp_{\mathsf{B}}^{\mathsf{DDH}}(0)=1\right] \\
= \;& \mathsf{Pr}\left[b' \neq b^* \mid b=0\right] \qquad (7.3)\\
= \;& \mathsf{Pr}\left[b'=1 \wedge b^*=0 \mid b=0\right] + \mathsf{Pr}\left[b'=0 \wedge b^*=1 \mid b=0\right] \\
= \;& \frac{1}{2}\cdot \mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{CPA}}(1)=0\right] + \frac{1}{2}\cdot \mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{CPA}}(0)=1\right] \\
= \;& \frac{1}{2}\cdot \left(1 - \mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{CPA}}(1)=1\right]\right) + \frac{1}{2}\cdot \mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{CPA}}(0)=1\right] \\
= \;& \frac{1}{2} + \frac{1}{2}\cdot \left(\mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{CPA}}(0)=1\right] - \mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{CPA}}(1)=1\right]\right) \qquad (7.4)
\end{aligned}$$

Thus we obtain

$$Adv^{\mathsf{DDH}}[\mathsf{B}] = \frac{1}{2} Adv^{\mathsf{CPA}}[\mathsf{A}]$$

which concludes the proof.

■

Thus, the ElGamal encryption scheme is CPA-secure in groups where the DDH problem is conjectured to be hard. Remember that this is however not the case in $\mathbb{Z}_p^*$, and we leave it as a simple exercise to show that CPA-security of ElGamal does not hold in $\mathbb{Z}_p^*$ (this can be shown similar to how we solved the DDH problem in $\mathbb{Z}_p^*$).

However, it turns out that in any group, ElGamal encryption as described above is completely insecure against chosen-ciphertext attack (CCA2), i.e., against active attackers.

Proposition 7.1 (ElGamal is not CCA2-secure) *Given the ElGamal encryption scheme* $\mathsf{E}^{\mathsf{ElGamal}} = (\mathsf{Gen}, \mathsf{E}, \mathsf{D})$ *(over an arbitrary group), there exists an adversary* A *such that*

$$Adv^{\mathsf{CCA2}}[\mathsf{A}, \mathsf{E}^{\mathsf{ElGamal}}] = 1.$$

Proof. The adversary A picks two arbitrary challenge messages $m_0 \neq m_1 \in G$ and asks for an encryption of one of them, yielding a ciphertext (i, c) of m_b. The adversary chooses an arbitrary c' and lets the $\mathsf{CCA2}$ challenger decrypt (i, c') yielding a message $m' = c'/i^x$. The adversary then simply computes $i^x = c'/m'$ and $m^* = c/i^x$. By construction, we have $m^* = c/i^x = m_b$ so that the adversary outputs 0 if $m^* = m_0$ and 1 if $m^* = m_1$. ∎

Note that this attack could be countered by testing if a value i was reused. While this might not be practical in applications, there are additionally more sophisticated attacks that circumvent this additional test: Instead of asking the $\mathsf{CCA2}$ challenger to decrypt the ciphertext (i, c'), i.e., for the same i that also occurred in the challenge-ciphertext, the adversary can *blind* this value i as follows. The adversary chooses an arbitrary $z \in \{1, \ldots, |G|\}$ as well as an arbitrary $c' \in G$ and lets the $\mathsf{CCA2}$ challenger decrypt (i^z, c'). The element i^z is distributed uniformly in G, thus the $\mathsf{CCA2}$ challenger cannot tell it apart from a correctly generated g^r for some other r. Thus the $\mathsf{CCA2}$ challenger decrypts the ciphertext and returns $m' := c'/i^{xz}$. Then the adversary can compute $k' := i^{xz} = c'/m'$ and $i^x := k'^{(z^{-1} \in G)}$. Finally, he can decrypt c' by computing $m := c'/i^x$ and then proceed as in the previous proof.

Thus, in order to make ElGamal-style encryption schemes secure against chosen-ciphertext attack, one needs a more sophisticated construction, which will be described in the next chapter.

8. The Cramer-Shoup Public-key Encryption System

This chapter is dedicated to the Cramer-Shoup public-key encryption scheme, which constitutes the first (and essentially still only) efficient public-key encryption scheme that is provably secure under chosen-ciphertext attack under standard cryptographic assumptions.

8.1 Keyed Hash Functions

In order to present the Cramer-Shoup encryption scheme, we first have to briefly introduce keyed hash functions. In contrast to their unkeyed counterparts, keyed hash functions have a rigorous security definition that can in fact be fulfilled by candidate functions. Let Gen denote a probabilistic algorithm called the key generation algorithm. Then a family $H = (\mathsf{hash}(pk, \cdot))_{pk \in [\mathsf{Gen}(n)]}$ (here $[\mathsf{Gen}(n)]$ denotes the carrier set of the probabilistic algorithm $\mathsf{Gen}(n)$, i.e., the set of all values that can be output by $\mathsf{Gen}(n)$ with non-zero probability) is called a keyed collision-resistant family of hash functions iff for all efficient algorithms A, the probability

$$\mathsf{Pr}(m \neq m^* \wedge \mathsf{hash}(pk, m) = \mathsf{hash}(pk, m^*); pk \leftarrow \mathsf{Gen}(n), (m, m^*) \leftarrow \mathsf{A}(n, pk))$$

is negligible in n.

8.2 Definition of the Cramer-Shoup Public-key Encryption Scheme

The Cramer-Shoup encryption scheme is quite similar to ElGamal encryption in subgroups G_q of $\mathbb{Z}_p^*$ of prime order, but the ciphertext contains additional fields. Intuitively, the purpose of these fields is that only someone who knows the plaintext can construct them correctly. However, there is no notion of "proof of knowledge" in the security proof, nor an intuitive explanation yet why exactly this choice of fields is good; they were simply chosen so that the proof worked. The key is also extended corresponding to the additional fields in the ciphertext.

8.2.1 Key Generation

Choose a subgroup G_q of prime order q of $\mathbb{Z}_p^*$ as for ElGamal encryption in subgroups of $\mathbb{Z}_p^*$, i.e., randomly choose an n-bit prime q and an $n_p(n)$-bit prime p such that $q|p-1$ (recall that $n_p(\cdot)$ was the polynomial function that relates the lengths of p and q). Choose a random generator g_1 of G_q. (Alternatively, these parameters can be fixed in advance and publicly known, cf. the description of Elgamal.) Choose a second generator g_2 of G_q. Choose five elements x_1, x_2, y_1, y_2, z randomly from $\mathbb{Z}_q$. Here z will play the role of the secret ElGamal key. Then compute

$$s := g_1^{x_1} \cdot g_2^{x_2}, \quad t := g_1^{y_1} \cdot g_2^{y_2}, \quad h := g_1^z.$$

Here h will play the role of the public key of ElGamal, and thus g_1 plays the role of g in ElGamal. Finally generate a key pk_{hash} for a keyed collision-resistant family of hash functions. The resulting actual hash function is abbreviated as

$$\mathsf{H}(\cdot) := \mathsf{hash}(pk_{\mathsf{hash}}, \cdot).$$

The complete public key is then

$$pk := (q, p, g_1, g_2, s, t, h, pk_{\mathsf{hash}})$$

and the secret key

$$sk := (pk, x_1, x_2, y_1, y_2, z).$$

8.2.2 Encryption

Choose r randomly from $\mathbb{Z}_q$. This corresponds to y in ElGamal encryption. Still as in ElGamal encryption, the ciphertext c contains components

$$i_1 := g_1^r, \quad c^* := h^r \cdot m.$$

In addition, compute

$$i_2 := g_2^r, \quad \alpha := \mathsf{H}(i_1, i_2, c^*), \quad v := s^r \cdot t^{r\alpha}.$$

The ciphertext is

$$c := (i_1, i_2, c^*, v).$$

Let us start with some intuition on this construction: In some way, you would expect that an active attacker who tries to reuse or blind i_1 as with ElGamal must reuse or blind i_2 in the same way so that they remain consistent; then α will be more or less fixed. (He may be able to choose c^* freely, but it seems he cannot do much better than choose some values c^* and get a list of values α to choose from.) None of these values α will be that from the original ciphertext. Hence it seems he cannot reuse or blind the given v, but must compute it by knowing r. Then, however, he would also know the ciphertext. However, all this was not a proof of security (nor, e.g., does it really explain the need for i_2).

8.2.3 Decryption

Given pk and $c = (i_1, i_2, c^*, v)$, the recipient recomputes $\alpha := \mathsf{H}(i_1, i_2, c^*)$. Now he must verify v. However, even with the secret key, he cannot retrieve r. Instead, he tests whether

$$i_1^{x_1+y_1\alpha} \cdot i_2^{x_2+y_2\alpha} = v.$$

If this is true, he decrypts m as with ElGamal, i.e., by computing

$$k := i_1^z, \quad m := c^*/k.$$

8.2.4 Correctness of Decryption

If a value m is output at all, its correctness follows exactly as with ElGamal: $k = i_1^z = g_1^{rz} = h^r$ and therefore c^*/k is the original m.

It remains to be shown that any correctly constructed v passes the test. Here we have

$$v = s^r \cdot t^{r\alpha} = (g_1^{x_1} \cdot g_2^{x_2})^r \cdot (g_1^{y_1} \cdot g_2^{y_2})^{r\alpha} = g_1^{rx_1+ry_1\alpha} \cdot g_2^{rx_2+ry_2\alpha} = i_1^{x_1+y_1\alpha} \cdot i_2^{x_2+y_2\alpha}. \tag{8.1}$$

The central question we have to ask ourselves here is the following: Does this test reject all incorrect quadruples? In other words, for any quadruple (i_1, i_2, c^*, v) that passes the test, is there an r and m such that $i_1 = g_1^r$, $i_2 = g_2^r$, $c^* = h^r \cdot m$, and $v = s^r \cdot t^{r\alpha}$ for $\alpha := \mathsf{H}(i_1, i_2, c^*)$? Equivalently, is there an r such that $i_1 = g_1^r$, $i_2 = g_2^r$, and $v = s^r \cdot t^{r\alpha}$ (then m is defined uniquely as c^*/h^r)?

The only possible value for r (modulo q) is already fixed by i_1. (This is so since $g_1^r = g_1^{r'}$ if and only if $r = r' \bmod q$.) We now distinguish two cases:

- If $i_2 = g_2^r$, then it is clear that v must also be of the required form (to see this, just write "$= v$" at the end instead of the beginning of Equation 8.1 and read it backwards).

- For any other i_2 (i.e., $i_2 \neq g_2^r$), a suitable value

  $$v := i_1^{x_1+y_1\alpha} \cdot i_2^{x_2+y_2\alpha}$$

  also exists. However, we will see in a lemma below that an attacker who does not know the secret key will not be able to find such a v except with negligible probability.

8.3 Security Proof

Theorem 8.1 *The Cramer-Shoup encryption scheme is secure under chosen-ciphertext attack (CCA2) under the Decisional Diffie-Hellman assumption.* □

Proof. The proof constitutes a complex reduction showing that if a successful attacker $\mathsf{A_{CS}}$ against the Cramer-Shoup system exist, i.e., $\mathsf{A_{CS}}$ succeeds in an game against the CCA2 challenger, we can construct a successful attacker $\mathsf{A_{DDH}}$ against the Decisional Diffie-Hellman assumption. For readability, we assume that $\mathsf{A_{CS}}$ consists of four algorithms $\mathsf{A}_1, \ldots, \mathsf{A}_4$ for the respective phases of the CCA2 experiment. (A_1 outputs ciphertexts and expects the corresponding plaintexts, A_2 outputs two challenge messages m_0^*, m_1^* and expects the ciphertext c of one of the messages, A_3 outputs ciphertexts different from c and expects the corresponding plaintexts, while A_4 finally outputs the guess b^*. The structure of $\mathsf{A_{CS}}$, of $\mathsf{A_{DDH}}$, and of the overall reduction is depicted in Figure 8.1. More precisely, the figure shows the desired input/output behavior of $\mathsf{A_{CS}}$ and the given input/output behavior of $\mathsf{A_{CS}}$. What remains to be done is to define the actual adversary $\mathsf{A_{DDH}}$ and prove that it wins against the DDH challenger with not-negligible probability, provided that $\mathsf{A_{DDH}}$ would win its game against the CCA2 challenger with not-negligible probability. The actual proof has to fill in how $\mathsf{A_{DDH}}$ treats incoming messages and computes outgoing messages.

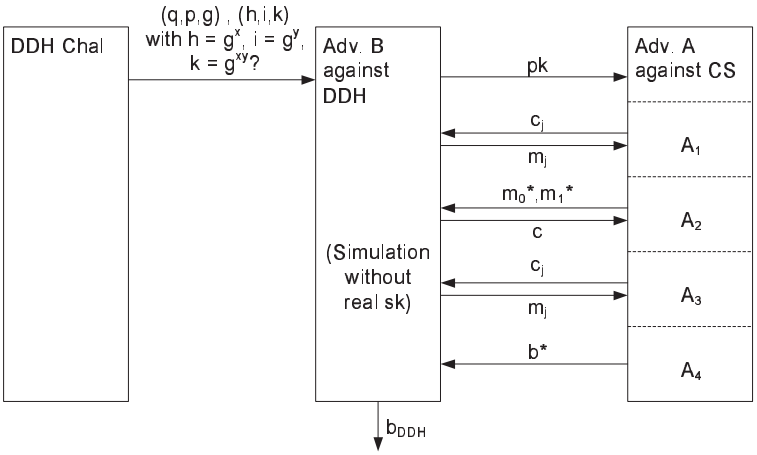


Figure 8.1: Structure of the Security Proof of the Cramer-Shoup Encryption Scheme

8.3.1 Intuition on the Reduction

Basic idea: The basic idea of the actual proof is the following: The given Diffie-Hellman tuple (h, i, k) (which is either (g^x, g^y, g^{xy}) or (g^x, g^y, g^a) for some random a) is used in the public key and ciphertext in a way so that everything is essentially correct if $k = g^{xy}$ in fact holds. In the following we use the notation $h = g^x$ and $i = g^y$ because such values x and y have to exist. (Note, however, that $\mathsf{A}_{\mathsf{DDH}}$ does not know them.)

- Hence, if $k = g^{xy}$, we give a correct simulation to A_{CS} and thus the success probability of A_{CS} is significantly better than $\frac{1}{2}$. This however also means that the probability that $b^* = b$ is significantly better than $\frac{1}{2}$.
- In the other case, where k is random, it will be shown that everything A_{CS} sees is essentially independent of b. Hence $b^* = b$ will hold with probability almost exactly $\frac{1}{2}$.

Thus the final step of $\mathsf{A}_{\mathsf{DDH}}$ is to set $b_{\mathsf{DDH}} = 0$ if and only if $b^* = b$. Thus $b_{\mathsf{DDH}} = 0$ can be interpreted as a guess that $k = g^{xy}$ is given, and in fact $b_{\mathsf{DDH}} = 0$ will be output with significantly larger probability in this case than in the other case.

Where to use the Diffie-Hellman Triple: The main decision is now where $\mathsf{A}_{\mathsf{DDH}}$ can reasonably use the given input, i.e., in what places A_{CS} expects a Diffie-Hellman triple among its inputs. The choice made here is to use

$$g_1 := g \text{ and } (g_2, i_1, i_2) := (h, i, k),$$

where g_1, g_2 are the two generators in the public key, and i_1 and i_2 the first two components of the ciphertext c that has to be produced in the challenge phase (against adversary A_2).

Using g and h as g_1 and g_2 is no problem: g and h are two random generators of G_q just as g_1 and g_2 should be.[1]

[1] Recall that $\mathsf{A}_{\mathsf{DDH}}$ has to work in the context of the Decisional Diffie-Hellman Assumption, i.e., its parameters are chosen according to the definition of the DDH assumption. Thus g is a random generator by construction, and $h = g^x$ for a random exponent x and thus also a random generator.

If $k = g^{xy}$ (i.e., the case that we would like to give a good simulation for), then i_1, i_2 are also chosen with the correct probability distribution because then $i_1 = i = g^y = g_1^y$ and $i_2 = k = g^{xy} = h^y = g_2^y$. In other words, y plays the role of r in a correct ciphertext. However, $\mathsf{A_{DDH}}$ does not know r and therefore has to compute the remaining components of the ciphertext c in another way than usual.

We now show the entire construction of $\mathsf{A_{DDH}}$. For the moment, we could think that $\mathsf{A_{DDH}}$ could choose sk in a perfectly correct way starting from (p, q, g_1, g_2), because none of the remaining components was used for the outside inputs i and k. Nevertheless, for reasons that will be seen in the case "k random", $\mathsf{A_{DDH}}$ chooses sk differently so that it has a bit more flexibility.

8.3.2 Construction of $\mathsf{A_{DDH}}$

In addition to the construction, we immediately show that everything is correct in the case $k = g^{xy}$.

1. Actions before $\mathsf{A_{CS}}$ is called:

   - The group and the first generator in pk are those given from the outside, i.e., p and q are the same for $\mathsf{A_{CS}}$ as for $\mathsf{A_{DDH}}$, and $g_1 := g$.
   - The second generator, g_2, is h from the input.
   - Now, instead of five, six elements x_1, x_2, y_1, y_2, z_1, z_2 are chosen randomly from $\mathbb{Z}_q$. The components s and t of the public key are computed in the correct way, but the last component is now
   $$h_{CS} := g_1^{z_1} \cdot g_2^{z_2}.$$
   - The hash key pk_{hash} is generated as usual.

   The change in the choice of h_{CS} does not alter the probability distribution of pk: Both the correct g_1^z and the simulated $g_1^{z_1} \cdot g_2^{z_2}$ are uniformly random elements of G_q. In fact, we can define the appropriate z as $z := z_1 + x \cdot z_2$ because $g_2 = g_1^x$. However, $\mathsf{A_{DDH}}$ does not know x and thus z.

2. Interaction with $\mathsf{A_1}$: Now $\mathsf{A_{DDH}}$ gives pk to $\mathsf{A_{CS}}$, so that the first active attack phase, $\mathsf{A_1}$, starts. We have to show how $\mathsf{A_{DDH}}$ decrypts the ciphertexts $c_j = (i_{j,1}, i_{j,2}, c_j^*, v_j)$ that $\mathsf{A_1}$ sends. As $\mathsf{A_{DDH}}$ knows the entire secret key, this is quite easy; we only have to adapt the decryption to the changed value h_{CS}: The verification of v_j is exactly as before (it uses neither z nor h). Now, in a normal decryption, k_j would be computed as $i_{j,1}^z$. If we define r_j such that $i_{j,1} = g_1^{r_j}$ (although $\mathsf{A_{DDH}}$ does not know this r_j), we can rewrite this as
   $$k_j = i_{j,1}^z = g_1^{r_j z} = (g_1^{z_1} \cdot g_2^{z_2})^{r_j} = g_1^{r_j z_1} \cdot g_2^{r_j z_2}.$$
   If the ciphertext is correct, this is $i_{j,1}^{z_1} \cdot i_{j,2}^{z_2}$. Thus $\mathsf{A_{DDH}}$ sets
   $$k_j := i_{j,1}^{z_1} \cdot i_{j,2}^{z_2}$$
   and then decrypts $m_j := c_j^*/k_j$ in the normal way.

   However, if the ciphertext is not correct (and it was sent by the attacker algorithm $\mathsf{A_1}$, so nobody guarantees that it is correct), then $i_{j,2}$ need not be $g_2^{r_j}$, and thus $\mathsf{A_{DDH}}$ gives $\mathsf{A_1}$ another answer than an actual CCA2 challenger would. This might alter the success probability of $\mathsf{A_1}$ and thus destroy our proof. Fortunately, we can show in Lemma 8.1 below that this case only occurs with negligible probability.

3. Interaction with A_2: When A_1 stops and A_2 sends m_0^* and m_1^*, then $\mathsf{A}_{\mathsf{DDH}}$ first chooses $b \in \{0,1\}$ as usual. Then it puts the given i and k into the ciphertext as described above, i.e., it uses them as i_1 and i_2. Recall that this means that r is the unknown y if $k = g^{xy}$.

   Now $\mathsf{A}_{\mathsf{DDH}}$ has to compute the remaining components c^* and v of this ciphertext, so that A_{CS} gets an input with the correct distribution in the case $k = g^{xy}$. Here c^* should be $h_{CS}^y \cdot m_b^*$. As $\mathsf{A}_{\mathsf{DDH}}$ does not know r, but it does know the secret key (in contrast to a normal CCA2 challenger who encrypts), it encrypts more or less as it would usually decrypt, i.e., it computes h_{CS}^y as $(g_1^{z_1} \cdot g_2^{z_2})^y = g_1^{yz_1} \cdot g_2^{yz_2} = i^{z_1} \cdot k^{z_2}$. This means it actually sets

$$c^* := i^{z_1} \cdot k^{z_2} \cdot m_b^*.$$

   Finally, the correct v certainly fulfills the verification. Hence the only possible choice is $i^{x_1+y_1\alpha} \cdot k^{x_2+y_2\alpha}$, where $\alpha = \mathsf{H}(i, k, c^*)$. This computation is easy for $\mathsf{A}_{\mathsf{DDH}}$ because it knows the secret key.

4. Interaction with A_3: Finally, $\mathsf{A}_{\mathsf{DDH}}$ has to interact with A_3, i.e., to perform decryptions again. This is exactly as in the interaction with A_1, except that $\mathsf{A}_{\mathsf{DDH}}$ compares each ciphertext c_j with the computed challenge ciphertext c first, so that it does not decrypt the secret.

5. Final computation: After A_3 stopped, A_4 starts and outputs a bit b^*. We already described above that $\mathsf{A}_{\mathsf{DDH}}$ outputs $b_{\mathsf{DDH}} = 0$ iff $b^* = b$, i.e., if A_4 guesses correctly.

This finishes the construction of $\mathsf{A}_{\mathsf{DDH}}$. We summarize it in Figure 8.2.

8.3.3 Proof of Correct Simulation

We now have to show how $\mathsf{A}_{\mathsf{DDH}}$ behaves in the two cases.

Behavior if $k = g^{xy}$. For this case, we have, with one exception, already shown during the construction that $\mathsf{A}_{\mathsf{DDH}}$ gives A_{CS} all its inputs with the correct respective probabilities. Once we have filled in the exception by Lemma 8.1, this means that A_{CS} will output a correct guess $b^* = b$ with probability significantly larger than $\frac{1}{2}$, and thus $\mathsf{A}_{\mathsf{DDH}}$ will output $b_{\mathsf{DDH}} = 0$ with that same probability.

Lemma 8.1 *If $k = g^{xy}$, the probability that A_{CS} succeeds in sending any ciphertext c_j whose first two components are not of the form $i_{j,1} = g_1^{r_j}$, $i_{j,2} = g_2^{r_j}$ for some r_j, but which nevertheless passes the verification, is exponentially small. This holds both with interacting with $\mathsf{A}_{\mathsf{DDH}}$ and in a normal game against a CCA2 challenger.* □

Proof. Actually, we can even show this property information-theoretically, i.e., even if A_{CS} were computationally unrestricted.[2] We show that for *every* wrong ciphertext $(i_{j,1}, i_{j,2}, c_j^*, v_j)$, the probability that it passes the verification is small. This probability is over the possible secret keys, more precisely the conditional probability over those secret keys that are still possible given everything that the attacker has seen. (In other words, not even exhaustive search helps because there is no

[2] This may sound like an unnecessary complication, but in fact information-theoretic proofs are usually simpler than computational ones.

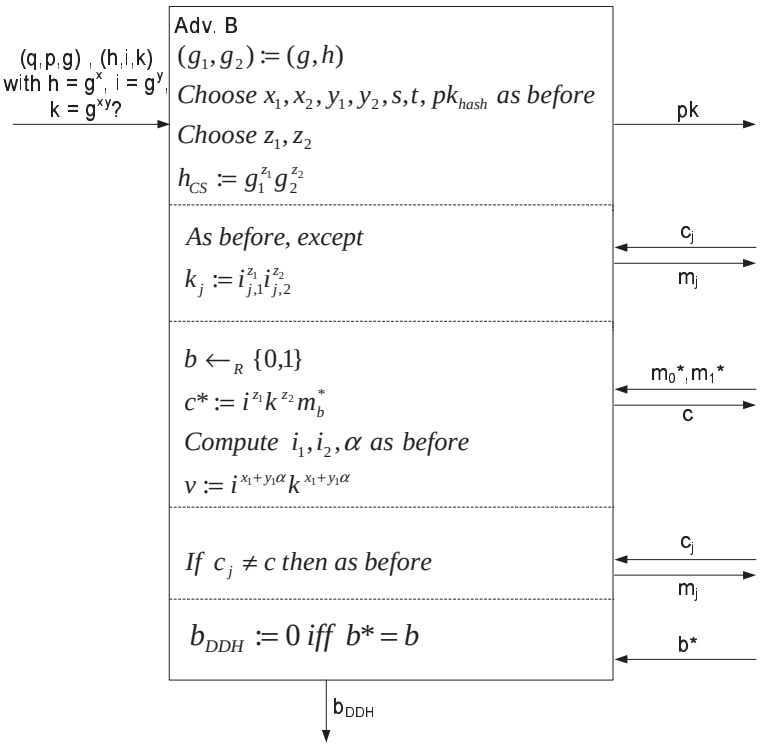


Figure 8.2: Summary of the Construction of $\mathsf{A}_{\mathsf{DDH}}$

good solution. The point here is that the verification equation is secret; otherwise this could not possibly work.)

Thus let us fix an arbitrary ciphertext $(i_{j,1}, i_{j,2}, c_j^*, v_j)$ where $i_{j,1} = g_1^{r_j}$ and $i_{j,2} = g_2^{r_j^*}$ with $r_j^* \neq r_j \bmod q$. Let $\alpha_j = \mathsf{H}(i_{j,1}, i_{j,2}, c_j^*)$ as usual. The ciphertext passes the verification if and only if

$$v_j = i_{j,1}^{x_1+y_1\alpha_j} \cdot i_{j,2}^{x_2+y_2\alpha_j} = g_1^{r_j(x_1+y_1\alpha_j)} \cdot g_2^{r_j^*(x_2+y_2\alpha_j)}.$$

Recall that $g_2 = h = g_1^x$. It will be helpful to compute entirely in the exponents (exploiting that $g_1^r = g_1^{r'}$ if and only if $r = r' \bmod q$). Hence we define β_j such that $v_j = g_1^{\beta_j}$. Then the verification is true if and only if (modulo q)

$$\beta_j = r_j(x_1 + y_1\alpha_j) + xr_j^*(x_2 + y_2\alpha_j). \tag{8.2}$$

If the attacker knew nothing at all about the secret key, it would be clear that he also had only an exponentially small chance to guess β_j, and thus v_j, correctly. However, the attacker gains knowledge about the secret key from several sources, and we have to show that this knowledge still leaves β_j unclear.

The secrets in Equation 8.2 are x_1, x_2, y_1, and y_2. (An unrestricted attacker could compute x from g_2.) The attacker gets the following information about them (follow Figure 8.2) to see that nothing is forgotten):

- The components $s = g_1^{x_1} \cdot g_2^{x_2}$ and $t = g_1^{y_1} \cdot g_2^{y_2}$ of pk. With unlimited computing power, he

could compute their discrete logarithms and thus get two linear equations

$$\sigma = x_1 + x \cdot x_2, \quad \tau = y_1 + x \cdot y_2. \tag{8.3}$$

- Decryption of correct ciphertexts. However, here the attacker only learns $k_j = i_{j,1}^{z_1} \cdot i_{j,2}^{z_2}$, which tells him nothing about (x_1, x_2, y_1, y_2).
- The encryption. Here c^* only depends on the values z, but $v = i^{x_1+y_1\alpha} \cdot k^{x_2+y_2\alpha}$. However, this is simply $s^y \cdot t^{y\alpha}$ in this correct case, where s and t are already known, and thus no new information about (x_1, x_2, y_1, y_2).
- Answers on incorrect ciphertexts. This is the case that we are just treating, i.e., we are showing that almost certainly, all answers will be the fixed error message (and that getting the fixed error message gives almost no information).

In summary, except for the error messages, the attacker obtains at most the two linear equations (Equation 8.3) in the four variables. Thus q^2 quadruples (x_1, x_2, y_1, y_2) are still possible from the attacker's point of view. We now show that Equation 8.2 for β_j is linearly independent of the two equations in Equation 8.3: The matrix (in the order x_1, x_2, y_1, y_2) is

$$\begin{pmatrix} 1 & x & 0 & 0 \\ 0 & 0 & 1 & x \\ r_j & xr_j^* & r_j\alpha_j & xr_j^*\alpha_j \end{pmatrix}$$

This can be transformed into

$$\begin{pmatrix} 1 & x & 0 & 0 \\ 0 & 0 & 1 & x \\ 0 & x(r_j^* - r_j) & 0 & x\alpha_j(r_j^* - r_j) \end{pmatrix}$$

We now exclude the case $x \neq 0$, which only occurs with exponentially small probability. We also know $r_j^* \neq r_j$. Thus $x(r_j^* - r_j) \neq 0$, and we can divide it out (recall that this is modulo q and thus in a field where every non-zero element has an inverse, hence division is well-defined). Exchanging Rows 2 and 3, we get

$$\begin{pmatrix} 1 & x & 0 & 0 \\ 0 & 1 & 0 & \alpha_j \\ 0 & 0 & 1 & x \end{pmatrix}$$

The rank of this matrix is 3, and thus it has exactly q solutions. This means that every β_j is correct for exactly q secret key quadruples (x_1, x_2, y_1, y_2) out of the q^2 that are possible from the point of view of the attacker. Thus any guess at β_j, and thus v_j, is correct with probability $\frac{1}{q}$. This is exponentially small (in the binary size of the group order G_q, i.e., in $\log q$). With each error message the attacker gets, one β_j and thus q keys are excluded. However, only a polynomial number of attempts is possible, and thus even at the end of the attack, the success probability is not significantly larger than $\frac{1}{q}$. ∎

Behavior if k randomly chosen in G_q: For this case, we want to show that the view of $\mathsf{A_{CS}}$, i.e., everything it sees, is essentially independent of b. Then it is clear that the output b^* of $\mathsf{A_{CS}}$ is also independent of b. Thus the probability of $b^* = b$ is almost exactly $\frac{1}{2}$, and thus also the probability that $b_{\mathsf{DDH}} = 0$. This will be the significant difference between the two cases.

Intuitively, we show that for this wrong scenario with a random k, the encryption is good even without the Diffie-Hellman assumption.

The primary source for $\mathsf{A_{CS}}$ to learn anything about b is the ciphertext c, which $\mathsf{A_{DDH}}$ constructs as

$$(i, k, c^*, v)$$

with

$$c^* = i^{z_1} \cdot k^{z_2} \cdot m_b^*, \quad v = i^{x_1+y_1\alpha} \cdot k^{x_2+y_2\alpha},$$

where $\alpha := \mathsf{H}(i, k, c^*)$ as usual. To show that this is independent of b, we show that $k^* = i^{z_1} \cdot k^{z_2}$ is as good as a multiplicative one-time pad for m_b^*. (It is easy to see that one-time pads can be made in any group; here is such a surrounding protocol where it makes sense not to use XOR.) In other words, we have to show that with all the knowledge the attacker has, all values of k^* are still equally likely. This can only be because z_1 and z_2 are secret. An attacker might obtain information about them from the following sources:

- The public key contains $h_{CS} = g_1^{z_1} \cdot g_2^{z_2}$. For a computationally unrestricted attacker, this means a linear equation
$$z = z_1 + xz_2. \tag{8.4}$$
- Decryption of correct ciphertexts. Here the attacker learns $k_j = i_{j,1}^{z_1} \cdot i_{j,2}^{z_2} = (g_1^{z_1} \cdot g_2^{z_2})^{r_j} = h_{CS}^{r_j}$. As he already knows h_{CS}, this is no new information about (z_1, z_2).
- The encryption is what we are just considering.
- Answers on incorrect ciphertexts. In Lemma 8.2, we will show that almost certainly all these answers will be the fixed error message. (Lemma 8.1 does not imply this because there we used that $k = g^{xy}$.)

In summary, except for the error messages, the attacker obtains at most one linear equation (Equation 8.4) about the two secret variables. Thus q pairs (z_1, z_2) are still possible from his point of view. The equation $k^* = i^{z_1} \cdot k^{z_2}$ can be rewritten as

$$\gamma = y \cdot z_1 + \delta \cdot z_2, \tag{8.5}$$

where γ and δ are defined such that $k^* = g_1^\gamma$ and $k = g_1^\delta$. Except in the one case where k, in spite of being randomly chosen, happens to be the correct Diffie-Hellman key g^{xy}, this equation is linearly independent from the one above. Hence, each $k^* \in G_q$ occurs for exactly one choice of (z_1, z_2). In other words, each k^* is equally probable from the point of view of the attacker, and thus it is a One-time Pad that perfectly hides m_b^* and thus b.

Lemma 8.2 *If k is randomly chosen from G_q, the probability that $\mathsf{A_{CS}}$ succeeds in sending any ciphertext c_j whose first two components are not of the form $i_{j,1} = g_1^{r_j}$, $i_{j,2} = g_2^{r_j}$ for some r_j, but which nevertheless passes the verification, is negligibly small.* □

Proof. We start the proof as for Lemma 8.1: We fix an arbitrary ciphertext $(i_{j,1}, i_{j,2}, c_j^*, v_j)$ where $i_{j,1} = g_1^{r_j}$ and $i_{j,2} = g_2^{r_j^*}$ with $r_j^* \neq r_j \bmod q$. It passes the verification if and only if

$$\beta_j = r_j(x_1 + y_1\alpha_j) + xr_j^*(x_2 + y_2\alpha_j), \tag{8.6}$$

where $v_j = g_1^{\beta_j}$. Again, the attacker $\mathsf{A_{CS}}$ can only obtain information about (x_1, x_2, y_1, y_2) from the following sources:

- The components s and t of pk, which give linear equations

$$\sigma = x_1 + xx_2, \quad \tau = y_1 + xy_2. \tag{8.7}$$

- The encryption, and only its component $v = i^{x_1+y_1\alpha} \cdot k^{x_2+y_2\alpha}$. Here is the difference from Lemma 8.1, because now this is not simply $s^y \cdot t^{y\alpha}$ and can therefore contain no new information about (x_1, x_2, y_1, y_2). We can rewrite this as

$$\epsilon = y(x_1 + y_1\alpha) + \delta(x_2 + y_2\alpha), \tag{8.8}$$

  where $k = g_1^\delta$ as in Equation 8.5 and $v = g_1^\epsilon$.

- Answers on incorrect ciphertexts. For these, we are just proving that they are almost certainly only the fixed error message.

In summary, except for the error messages, the attacker now obtains three linear equations about the four secret variables. Again we write them as a matrix, and in the last row we write the equation for β_j:

$$\begin{pmatrix} 1 & x & 0 & 0 \\ 0 & 0 & 1 & x \\ y & \delta & y\alpha & \delta\alpha \\ r_j & xr_j^* & r_j\alpha_j & xr_j^*\alpha_j \end{pmatrix}$$

We transform Rows 1, 2, and 4 as in Lemma 8.1 and move Row 3 down:

$$\begin{pmatrix} 1 & x & 0 & 0 \\ 0 & 1 & 0 & \alpha_j \\ 0 & 0 & 1 & x \\ y & \delta & y\alpha & \delta\alpha \end{pmatrix}$$

If we subtract Rows 1 and 3 from 4 with the appropriate coefficients, Row 4 becomes

$$\begin{pmatrix} 0 & \delta - xy & 0 & \alpha(\delta - xy) \end{pmatrix}$$

and finally, subtracting Row 2,

$$\begin{pmatrix} 0 & 0 & 0 & (\alpha - \alpha_j)(\delta - xy) \end{pmatrix}$$

If $(\alpha - \alpha_j)(\delta - xy) \neq 0$, the rank of the matrix is 4. This also implies that any three of the original equations have rank 3 and thus q solutions. Thus q secret key quadruples (x_1, x_2, y_1, y_2) are possible from the point of view of the attacker, and only for one of them, β_j is correct. This is an exponentially small probability $\frac{1}{q}$.

Now we have to show whether the attacker can achieve either $\alpha = \alpha_j$ or $\delta = xy$. (As to an intuitive idea why the case $\alpha = \alpha_j$ is special, recall the paragraph "some intuition" we gave you earlier in this chapter in Section 8.2.2.)

- $\delta = xy$ would mean that $k = g_1^{xy}$, i.e., k happens to be the correct Diffie-Hellman key. This only happens with exponentially small probability.
- $\alpha = \alpha_j$ would mean $\mathsf{H}(i, k, c^*) = \mathsf{H}(i_{j,1}, i_{j,2}, c_j^*)$. If $(i_{j,1}, i_{j,2}, c_j^*) \neq (i, k, c^*)$, then the attacker would have found a collision of the hash function, which is assumed to be infeasible.[3]

  Otherwise, the components i, k, c^* determine the only correct v uniquely, and thus $c_j = c$. If c_j is sent after c, $\mathsf{A}_{\mathsf{DDH}}$ would refuse to answer. And if c_j is sent before c, this would mean that the attacker A_{CS} guessed the random values i and k correctly although so far he obtained no information at all about them.

Thus in fact, at the beginning any β_j is only correct for one out of q pairs (z_1, z_2) that are possible from the point of view of the attacker, and each error message excludes only one such pair, so that the success probability of the attacker remains exponentially small. ∎

This finishes the proof that b is well hidden from A_{CS} in the case where k is random, and thus the overall proof of the security of the Cramer-Shoup system according to the basic idea explained at the beginning. ∎

[3] You might ask whether it was really the attacker, because (i, k, c^*) came from the (simulated) CCA2 challenger. However, the CCA2 challenger never exploited the fact that he himself chose pk_{hash}. Hence if A_{CS} could achieve this case, one could easily construct a successful attacker against the hash functions from it by letting that attacker play the roles of both $\mathsf{A}_{\mathsf{DDH}}$ with random i, k and A_{CS}.

9. One-way Functions and the RSA Trapdoor Permutation

In the last chapter we have seen that exponentiation in certain groups is conjectured hard to invert, i.e., the discrete logarithm in these groups is conjectured hard to compute. We now generalize this concept to an abstract concept called *one-way functions*. Closely related are so-called *trapdoor permutations*, which constitute an important foundation of many modern cryptographic systems. The most common trapdoor permutation is the RSA trapdoor permutation, which we will introduce in the following.

9.1 One-way Functions (OWFs) and Trapdoor Permutations

A *one-way function* is a deterministic function that is easy to compute, but (conjectured) hard to invert, i.e., an adversary gets a value $\mathsf{F}(x) = y$ for a randomly drawn x and tries to find a value x' such that $\mathsf{F}(x') = y$. The success probability of every efficient adversary should be negligible in $|x|$.

Definition 9.1 (One-way Functions) *An efficiently computable, deterministic function* $\mathsf{F}\colon \{0,1\}^* \to \{0,1\}^*$ *is a* one-way function *iff it is* hard to invert*, i.e., for any efficient* A*, we have that*

$$\mathsf{Pr}\left[\mathsf{F}(x) = \mathsf{F}(x');\ x \leftarrow_{\mathcal{R}} \{0,1\}^n,\ y := \mathsf{F}(x),\ x' \leftarrow \mathsf{A}(n,y)\right]$$

is negligible in n*. Here and in the following,* A *is given the parameter* n *in unary representation as usual, for reasons of complexity.* ◇

One could define one-wayness also for function ensembles indexed by a key-generation algorithm as follows: Let $\mathsf{Gen}\colon \mathbb{N} \to \mathcal{PK}$ be an efficiently computable key-generation algorithm and let $F = (\mathsf{F}(pk,\cdot))_{pk \in [\mathsf{Gen}(n)]}$ be a family of deterministic functions $\mathsf{F}(pk,\cdot)\colon \mathcal{X}_{pk} \to \mathcal{Y}_{pk}$ for finite domains and ranges $\mathcal{X}_{pk}$ and $\mathcal{Y}_{pk}$, respectively. Then F is said to be a *family of one-way functions*, or short *one-way*, iff for any efficient A, we have that

$$\mathsf{Pr}\left[\mathsf{F}(pk,x) = \mathsf{F}(pk,x');\ pk \leftarrow \mathsf{Gen}(n),\ x \leftarrow_{\mathcal{R}} \mathcal{X}_{pk},\ y := \mathsf{F}(pk,x), x' \leftarrow \mathsf{A}(n,pk,y)\right]$$

is negligible in n.

For building an encryption scheme, however, we often have to rely on some possibility to invert F, i.e., to recover the message again. This is accomplished by providing some extra information called a *trapdoor* (denoted by *sk* in the following), that enables people knowing the trapdoor to invert the function F. This is captured by the notion of *(keyed) trapdoor permutations*.

Definition 9.2 ((Keyed) Trapdoor Permutations) *Let* $\mathsf{Gen}\colon \mathbb{N} \to \mathcal{PK} \times \mathcal{SK}$ *be an efficiently computable key-generation algorithm and* $F = (\mathsf{F}(pk,\cdot)_{(pk,*) \in [\mathsf{Gen}(n)]}$ *be a family of efficiently computable, deterministic functions* $\mathsf{F}(pk,\cdot)\colon \mathcal{X}_{pk} \to \mathcal{Y}_{pk}$*. F is a* family of trapdoor permutations *iff*

- F *is one-way, i.e., for any efficient adversary* A*, we have that*

$$\mathsf{Pr}\left[\mathsf{F}(pk,x)=\mathsf{F}(pk,x');\ (pk,sk)\leftarrow\mathsf{Gen}(n),\ x\leftarrow_{\mathcal{R}}\mathcal{X}_{pk},\ y:=\mathsf{F}(pk,x),x'\leftarrow\mathsf{A}(n,pk,y)\right]$$

  is negligible, and
- F *has a* trap-door *sk, i.e., there exists an efficiently computable algorithm* G *that* inverts F *given sk: For all* $(pk,sk)\in[\mathsf{Gen}(n)]$ *and for all* $x\in\mathcal{X}_{pk}$*:*

$$\mathsf{G}(n,pk,sk,\mathsf{F}(pk,x))=x.$$

◇

Trapdoor permutations almost look like encryption schemes; however, the following example shows that they might fail completely in providing secrecy when applied in a straightforward manner. The reason is that one-way functions might leak substantial parts of the message, which is clearly not acceptable for encryption, and which does not even yield security against chosen-plaintext attack, i.e, against passive adversaries.

Proposition 9.1 *Let* F *be a OWF. Then* $\mathsf{H}(x\parallel y):=\mathsf{F}(x)\parallel y$ *(where* $|x|=\lceil\frac{|(x\|y)|}{2}\rceil$*) is a OWF as well.*

Proof. Assume that A constitutes a successful adversary against the one-wayness of H, i.e., upon receiving an input $\mathsf{F}(x)\parallel y$, it outputs $x'\parallel y'$ such that $\mathsf{F}(x)\parallel y=\mathsf{F}(x')\parallel y'$ holds with not-negligible probability. We then construct an adversary B against F as follows: It receives $\mathsf{F}(x)$ for a randomly chosen (and unknown) x, chooses a random y, and sends $\mathsf{F}(x)\parallel y$ to A. Upon receiving $x'\parallel y'$, it outputs x'.

B is obviously efficiently computable. Thus it is sufficient to calculate the success probability of B to invert $\mathsf{F}(x)$:

$$\begin{aligned}
&\mathsf{Pr}\left[\mathsf{F}(x)=\mathsf{F}(x');x\leftarrow_{\mathcal{R}}\{0,1\}^n,z:=\mathsf{F}(x),x'\leftarrow\mathsf{B}(n,z)\right] &(9.1)\\
=\ &\mathsf{Pr}\left[\mathsf{F}(x)=\mathsf{F}(x');x,y\leftarrow_{\mathcal{R}}\{0,1\}^n,\ z:=\mathsf{F}(x),\ x'\parallel y'\leftarrow\mathsf{A}(n,(z\parallel y))\right]\\
\geq\ &\mathsf{Pr}\left[\mathsf{F}(x)\parallel y=\mathsf{F}(x')\parallel y';x,y\leftarrow_{\mathcal{R}}\{0,1\}^n,\ z:=\mathsf{F}(x),\ x'\parallel y'\leftarrow\mathsf{A}(n,(z\parallel y))\right]\\
=\ &\mathsf{Pr}\left[\mathsf{H}(x\parallel y)=\mathsf{H}(x'\parallel y');x,y\leftarrow_{\mathcal{R}}\{0,1\}^n,\ z^*:=\mathsf{H}(x\parallel y),\ x'\parallel y'\leftarrow\mathsf{A}(n,z^*)\right] &(9.2)
\end{aligned}$$

We know by assumption that the probability given in Equation 9.2 is not negligible. Consequently the probability given in Equation 9.1 is not negligible which concludes the proof. ∎

Definition 9.3 (Hardcore Predicate) *A* hardcore predicate *of a function* $\mathsf{F}\colon\{0,1\}^*\to\{0,1\}^*$ *is an efficiently computable boolean predicate* $\pi\colon\{0,1\}^*\to\mathcal{BOOL}$ *such that for all efficient* A*, we have that*

$$\left|\mathsf{Pr}\left[z=\pi(x);\ x\leftarrow_{\mathcal{R}}\{0,1\}^n,\ y:=\mathsf{F}(x),\ z\leftarrow\mathsf{A}(n,y)\right]-\frac{1}{2}\right|$$

is negligible in n*.*

◇

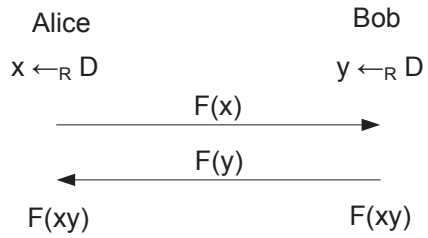


Figure 9.1: Generalized Diffie-Hellman Key Exchange

Goldreich and Levin showed the following theorem, which we give without proof: Let F be a length-preserving OWF (length-preserving means that $|\mathsf{F}(x)| = |x|$), and let $\mathsf{F}'(x \parallel r) := \mathsf{F}(x) \parallel r$ where $|x| = |r| = k$. Then

$$\pi(x \parallel r) := \sum_{i=1}^{k} x_i r_i \bmod 2$$

constitutes a hardcore predicate of F'.

Many functions currently exist that are conjectured one-way, e.g., based on assumptions such as the hardness of factoring, of the Diffie-Hellman function, or of computing the discrete logarithm, one can construct OWFs. Proving the existence of OWFs without such assumptions would however immediately imply the existence of secure encryption schemes, secure signature schemes, and would in particular prove $P \neq NP$.

We conclude by giving two simple constructions of OWFs.

Proposition 9.2 (OWFs from PRPs) *Assume $\mathsf{E} = (\mathsf{E}_n(K,\cdot))_{n\in\mathbb{N}}$ is a family of PRPs , where E_n has domain and range $\{0,1\}^n$. Then $\mathsf{F}^{\mathsf{E}}(x) = \mathsf{E}(x,0)$ for $x \in \{0,1\}^n$ is a one-way function.*

Proposition 9.3 (DLog is an OWF) *Let p be a prime and $g \in \mathbb{Z}_p^*$ an element of (large) prime order q. Then $\mathsf{F}^{\mathsf{Dlog}}(x) := g^x \bmod p$ is conjectured to be one-way. (Its inverse is $\mathsf{DLog}_g(\cdot)$ which is conjectured hard in $\mathbb{Z}_p^*$.)*

Note that $\mathsf{F}^{\mathsf{DLog}}$ has an additional property: Given $a \in \mathbb{Z}$ and $\mathsf{F}^{\mathsf{DLog}}(x)$, $\mathsf{F}^{\mathsf{DLog}}(y)$ one can easily compute $\mathsf{F}^{\mathsf{DLog}}(a\cdot x)$ and $\mathsf{F}^{\mathsf{DLog}}(x+y)$. One says that $\mathsf{F}^{\mathsf{DLog}}$ is an *additive function*. Due to additivity it can be used for Diffie-Hellman key exchange as well as for the ElGamal public-key encryption scheme, whereas one-way functions without such properties, e.g., F^{E} as constructed above, cannot be used for this purpose.

9.1.1 Generalized ELGamal and Diffie-Hellman

Diffie-Hellman key exchange and ElGamal encryption can be generalized to arbitrary additive one-way functions with domain D in an obvious manner. We only show the generalization of Diffie-Hellman key exchange in Figure 9.1 for simplicity. Note that given x and $\mathsf{F}(y)$ Alice can compute $\mathsf{F}(xy)$ by the additivity of F, similarly for Bob who holds y and $\mathsf{F}(x)$.

9.2 Arithmetic Modulo a Composite

In the following we are working in groups $\mathbb{Z}_N$ where $N \in \mathbb{N}$ is a composite. Unless otherwise stated N is the product $N = p \cdot q$ of two large (≈ 512 bit) primes that are of the same size. For a composite N let $\mathbb{Z}_N = \{0, 1, 2, \ldots, N-1\}$; addition and multiplication are defined modulo N as usual.

Extended Euclidean Algorithm Given two natural numbers a and b one can find the *greatest common divisor* $r = \gcd(a, b)$ by means of the *Euclidean algorithm*. In addition, integers x, y exist that satisfy $ax + by = r$, and these integers can be found by the *extended Euclidean algorithm* as follows:

1. Start with numbers a and b, as well as corresponding vectors $(1, 0)$ and $(0, 1)$.
2. Divide the larger of the two numbers a and b by the smaller using division with remainder. Call this quotient q.
3. Subtract q times the smaller from the larger number.
4. Subtract q times the vector corresponding to the smaller number from the vector corresponding to the larger number.
5. Repeat steps 2 through 4 until one of the numbers equals zero. The vector that was changed in the last step consist of the two desired numbers x, y.

The following examples shows the algorithm running on numbers $a = 53$ and $b = 30$:

53	30	$(1, 0)$	$(0, 1)$
$53 - 1 \cdot 30 = 23$	30	$(1, 0) - 1 \cdot (0, 1) = (1, -1)$	$(0, 1)$
23	$30 - 1 \cdot 23 = 7$	$(1, -1)$	$(0, 1) - 1 \cdot (1, -1) = (-1, 2)$
$23 - 3 \cdot 7 = 2$	7	$(1, -1) - 3 \cdot (-1, 2) = (4, -7)$	$(-1, 2)$
2	$7 - 3 \cdot 2 = 1$	$(4, -7)$	$(-1, 2) - 3 \cdot (4, -7) = (-13, 23)$
$2 - 2 \cdot 1 = 0$	1	$(4, -7) - 2 \cdot (-13, 23) = (30, -53)$	$(-13, 23)$

Thus we see that $-13 \cdot 53 + 23 \cdot 30 = -689 + 690 = 1$.

The *inverse* of $x \in \mathbb{Z}_N$ is an element $y \in \mathbb{Z}_N$ such that $x \cdot y = 1 \bmod N$. We write x^{-1} instead of y provided that the inverse exists.

Proposition 9.4 (Inverse) *An element $x \in \mathbb{Z}_N$ has an inverse if and only if* $\gcd(x, N) = 1$.

Proof. If $\gcd(x, N) = 1$ then there exist two integers $a, b \in \mathbb{Z}$ such that $ax + bN = 1$. Reducing this relation modulo N leads to $ax = 1 \bmod N$, hence $a = x^{-1} \bmod N$.

If, on the other hand, there exists a such that $ax = 1 \bmod N$, then there exists k such that $ax = kN + 1$ or equivalently $ax - kN = 1$. By definition $\gcd(x, N) | ax$ and $\gcd(x, N) | kN$, hence we have that $\gcd(x, N) | 1$, which is only possible if $\gcd(x, N) = 1$. ∎

This proof also yields an efficient method for constructing the inverse since the extended Euclidean algorithm can be used to efficiently construct the value a (which is the inverse of x modulo N). This inversion algorithm also works in $\mathbb{Z}_p$ for a prime p and is more efficient than inverting x by computing $x^{p-2} \bmod p$. The ability to compute an inverse in particular entails an algorithm for solving linear equations $a \cdot x = b \bmod N$. The solution is $x = b \cdot a^{-1} \bmod N$ where a^{-1} can now be computed using the extended Euclidean algorithm (provided that a^{-1} exists).

We note that if we know an element $x \in \mathbb{Z}_N$ with $0 \neq x \notin \mathbb{Z}_N^*$ then we can factor N, since $\gcd(x, N)$ constitutes a non-trivial factor of N.

We now investigate how many elements in $\mathbb{Z}_N$ are invertible. While in $\mathbb{Z}_p$ all elements except for 0 are invertible, this is no longer true for $\mathbb{Z}_N$.

Definition 9.4 (Eulers Totient Function) *Let $\varphi(N) := |\mathbb{Z}_N^*|$ the number of invertible elements in $\mathbb{Z}_N^*$.* ◇

The function $\varphi(N)$ turns out to be easily computable, provided that the factorization of N is known:

1. We already know that $\varphi(p) = p - 1$ for any prime p.
2. We have $\varphi(pq) = (p-1)(q-1)$ if p and q are relatively prime.
3. We have $\varphi(p^e) = p^e - p^{e-1}$ for prime p and $e \in \mathbb{N}$.

These rules can be summarized to the following compact form for computing $\varphi(N)$.

Proposition 9.5 (Computing $\varphi(N)$) *Let $N = p_1^{e_1} \cdot \ldots \cdot p_m^{e_m}$ denote the prime factorization of N. Then*

$$\varphi(N) = N \cdot \prod_{i=1}^{m} (1 - \frac{1}{p_i}).$$

The following theorem is a generalization of Fermat's Little Theorem to groups of arbitrary order:

Theorem 9.1 (Euler) *For any $a \in \mathbb{Z}_N^*$ we have that $a^{\varphi(N)} = 1 \bmod N$.* □

Theorem 9.2 (Chinese Remainder Theorem (CRT) for $N = pq$) *Let $p \neq q$ be primes and let $N = pq$. Given $x_p \in \mathbb{Z}_p$ and $x_q \in \mathbb{Z}_q$, there exists a unique element $s \in \mathbb{Z}_N$ such that $s = x_p \bmod p$ and $s = x_q \bmod q$. Furthermore, s can be computed efficiently by the following algorithm: On input (p, q, x_p, x_q) with p, q prime, $p \neq q$, and $x_p \in \mathbb{Z}_p, x_q \in \mathbb{Z}_q$:*

- *Use the extended Euclidean algorithm to compute $u, v \in \mathbb{Z}$ with $up + vq = 1$.*
- *Output $x := upx_q + vqx_p \bmod N$.*

□

The CRT shows that each element $s \in \mathbb{Z}_N$ can be viewed as a unique pair (x_p, x_q) where $x_p = s \bmod p$ and $x_q = s \bmod q$. The uniqueness guarantee shows that each pair $(x_p, x_q) \in \mathbb{Z}_p \times \mathbb{Z}_q$ corresponds to exactly one element of $\mathbb{Z}_N$. For example, the pair $(1, 1)$ corresponds to $1 \in \mathbb{Z}_N$.

We conclude with the following simple proposition; a simplified version thereof was already given on one of the exercise sheets.

Proposition 9.6 *Let G be a group and $g \in G$. For all $x, y \in \mathbb{Z}$ it holds that*

$$g^x = g^y \quad \textit{iff} \quad x = y \bmod \mathsf{ord}(g)$$

Proof. Let $\Omega := \mathsf{ord}(g)$.
"⇐": Let $x = y \bmod \Omega$. Then there exists $k \in \mathbb{N}$ such that $y = x + k \cdot \Omega$. Consequently,

$$g^y = g^{x+k\cdot\Omega} = g^x \cdot (g^\Omega)^k = g^x \cdot 1^k = g^x.$$

"⇒": Let $x - y = q \cdot \Omega + r$ where $0 \leq r < \Omega$. Then

$$1 = g^{x-y} = g^{q\cdot\Omega+r} = g^r.$$

Consequently $r = 0$, otherwise we had a contradiction to the minimality of Ω. Thus $x = y \bmod \Omega$.
■

9.3 The RSA Trapdoor Permutation

In this section we will investigate the RSA trapdoor permutation, one of the most famous and most widely deployed one-way functions. The RSA trapdoor permutation was invented by Rivest, Shamir, Adleman in 1977, and can be used to construct various cryptographic schemes. However, we will see that one has to be careful when using this function for constructing encryption schemes since naive constructions result in insecure encryption schemes (even if they are sometimes claimed to be good in several books). We will see later in this chapter how RSA can be used to construct good encryption schemes.

Definition 9.5 (RSA Trapdoor Permutation) *Pick random n-bit primes p and q (in practice often 512 bits) and let $N = pq$. Choose an arbitrary value e such that $\gcd(e, \varphi(N)) = 1$ and compute d such that $ed = 1 \bmod \varphi(N)$.*

- *The public information is $pk = (N, e)$, and the RSA trapdoor permutation $\mathsf{F}^{\mathsf{RSA}} \colon \mathbb{Z}_N \to \mathbb{Z}_N$ for this pk is defined as $\mathsf{F}^{\mathsf{RSA}}(x) := x^e \bmod N$.*
- *The trapdoor information is $sk = d$, and $\mathsf{D}^{\mathsf{RSA}}(y) := y^d \bmod N$ is the inverse of $\mathsf{E}^{\mathsf{RSA}}$.*

◇

Before proving that $\mathsf{D}^{\mathsf{RSA}}$ is the inverse of $\mathsf{F}^{\mathsf{RSA}}$ we prove the following lemma:

Lemma 9.1 *For N, e, d as constructed in Definition 9.5 and for arbitrary $m \in \mathbb{Z}_N$ it holds that*

$$m^{ed} = m \bmod N.$$

□

Proof. By the Chinese Remainder Theorem, it is sufficient to show $m^{ed} = m \bmod p$ and $m^{ed} = m \bmod q$. For symmetry reasons, we only show this for p.

- If $m = 0 \bmod p$, then obviously $0^{ed} = 0 \bmod p$, thus nothing needs to be proved.
- If $m \neq 0 \bmod p$, then $m \in \mathbb{Z}_p^*$. Since $ed = 1 \bmod \varphi(N)$ and $\varphi(N) = (p-1)(q-1)$, there exists $k \in \mathbb{Z}$ such that $ed = k(p-1)(q-1) + 1$. Thus we obtain

$$m^{ed} = m^{1+k(p-1)(q-1)} = m \cdot (m^{p-1})^{k(q-1)} = m \cdot 1^{k(q-1)} = m \bmod p.$$

This finishes the proof. ■

Now simple calculation shows that $\mathsf{D}^{\mathsf{RSA}}$ is the inverse of $\mathsf{F}^{\mathsf{RSA}}$: For any $m \in \mathbb{Z}_N$

$$\mathsf{D}^{\mathsf{RSA}}(\mathsf{F}^{\mathsf{RSA}}(m)) = (m^e)^d = m^{ed} = m \bmod N.$$

As we did for discrete logarithms, we explicitly state the assumption that the RSA trapdoor permutation is one-way. This is known as the *RSA assumption*.

Definition 9.6 (RSA Assumption) *Given random n-bit primes p and q, $N := pq$, a value e such that $\gcd(e, \varphi(N)) = 1$, and a random number $c \in \mathbb{Z}_N$, there does not exist any efficient algorithm that computes the e'th root of c modulo N up to negligible probability.* ◇

One of the main open questions in cryptography is the following: Given an algorithm for computing e'th roots modulo N, i.e., for inverting RSA, does it imply a factoring algorithm? There is some evidence that this might not be the case, i.e., that breaking RSA is indeed easier than factoring (Boneh-Venkatesan 1998). However, the best algorithm which is known to invert the RSA trapdoor permutation without knowing d is to first factor $N = p \cdot q$ (this is conjectured hard) and subsequently to compute e-th roots modulo p and q (this is easily doable).

9.3.1 Naive Encryption with RSA

It is tempting to use the following construction for encryption: One generates (pk, sk) as in Definition 9.5, publishes pk, and keeps sk secret. Then encryption and decryption are realized by applying and inverting the RSA trapdoor permutation, respectively: Given a message $m \in \mathbb{Z}_N$ or a ciphertext $c \in \mathbb{Z}_N$, one computes $\mathsf{E}(pk, m) := m^e \bmod N$ or $\mathsf{D}(sk, c) := c^d \bmod N$, respectively. Naive RSA encryption is obviously not CPA-secure, as it is deterministic. Furthermore, it leaks concrete bits, e.g., the so-called Jacobi symbol of the message. It is also highly vulnerable to active attacks, e.g., chosen-ciphertext attacks. There are simple attacks that completely retrieve the message for a given ciphertext.

For the first such attack the attacker asks for the decryption of two ciphertexts that are constructed as follows. Denote the given ciphertext the adversary is trying to decrypt by c.

- Choose $c_1 \in \mathbb{Z}_N^*$ arbitrary and ask for its decryption, yielding a message m_1.
- Let $c_2 := c \cdot c_1^{-1}$ and ask for its decryption, yielding a message m_2.
- Output $m_1 \cdot m_2$.

An easy calculation shows that this attack retrieves the message $m = c^d \bmod N$ from the ciphertext c:

$$m_1 \cdot m_2 = c_1^d \cdot c_2^d = c_1^d \cdot c^d \cdot c_1^{-d} = 1 \cdot c^d = m^{ed} = m.$$

One can do even better: Even with only one decryption query one can decrypt an arbitrary ciphertext as the following attack shows. Again the adversary is given a ciphertext c.

- Choose $m_1 \in \mathbb{Z}_N^*$ arbitrary and let $c_1 := m_1^e \bmod N$.
- Let $c_2 := c \cdot c_1^{-1}$ and ask for its decryption, yielding a message m_2.
- Compute $m_1 \cdot m_2$.

Again a simple calculation shows that this attack retrieves the message $m = c^d \bmod N$ from the ciphertext c:

$$m_1 \cdot m_2 = m_1 \cdot c_2^d = m_1 \cdot c^d \cdot c_1^{-d} = m_1 \cdot c^d \cdot m_1^{-1} = m^{ed} = m.$$

9. The RSA Trapdoor Permutation (cont'd)

9.3.2 Improving Performance of RSA

Using Small Exponents e. A common way to speed up the computation of the RSA trapdoor permutation is to use small values of e, which leads to a fast evaluation of the RSA function. The minimal value e possibly satisfying $\gcd(e, \varphi(N)) = 1$ is obviously $e = 3$ since $\gcd(2, \varphi(N)) = 2$. However, relying on values e that small should be avoided since sending the same message encrypted with e different public keys (which all rely on the same public exponent e but different moduli N_i) allows an attacker already to retrieve the message from the ciphertexts. The recommended value of e is $e = 65537 = 2^{16} + 1$, as this allows for computing the RSA trapdoor permutation using only 17 modular multiplications.

Exploiting the Chinese Remainder Theorem One can exploit the Chinese Remainder Theorem to compute RSA inverses separately modulo p and q. For computing $c^d \bmod N$, one does the following:

- Compute u and v such that $up + vq = 1$.
- Compute $m_p := c^{d_p} \bmod p$ and $m_q := c^{d_q} \bmod q$ where $d_p = d \bmod p - 1$ and $d_q = d \bmod q - 1$.
- Set $m := upm_q + vqm_p \bmod N$.

The advantage is that all computations are carried out in significantly smaller groups, using exponents of only half the size, thus gaining roughly a factor of 4 in speed. The computational overhead is only two multiplications and one addition and thus can be ignored. Note that the values u and v can be computed once and for all and then used for inverting different values c.

Using Small Exponents d – Bad idea! A still pretty common efficiency improvement is to make the value d small in order to speed up the computation of the inverse of the RSA trapdoor permutation. However, Wiener proved in 1989 that, if $d < 1/3N^{0.25}$, then one can fully recover d given (N, e). We will describe this very nice attack in the next section. Boneh, Durfee, and Frankel improved this bound in 1998 to $d < N^{0.292}$. For $d < N^{0.5}$, it is expected that d can be recovered as well given (N, e), but it is still an open problem how this can be achieved.

9.4 Wiener's Attack Against Small Exponents d

In this section we describe Wiener's attack against RSA if the exponent d is small, more precisely, if we have $d < \frac{N^{0.25}}{3}$. Let $N = p \cdot q$ as usual and assume that $q < p < 2q$. As we have argued before, it is tempting for Bob to choose a small d, as this speeds up the inversion of RSA.

Since $ed = 1 \bmod \varphi(n)$, there is an integer k such that $ed - k\varphi(N) = 1$. Thus we have

$$\left|\frac{e}{\varphi(N)} - \frac{k}{d}\right| = \frac{1}{d\varphi(N)}.$$

From $N = pq > q^2$ it follows $q < \sqrt{n}$, hence

$$0 < N - \varphi(N) = p + q - 1 < 2q + q - 1 < 3q < 3\sqrt{n}.$$

Now we obtain

$$\begin{aligned}\left|\frac{e}{N} - \frac{k}{d}\right| &= \left|\frac{ed - kN}{dN}\right| \\ &= \left|\frac{1 + k(\varphi(N) - N)}{dN}\right| \\ &< \frac{3k\sqrt{N}}{dN} \\ &= \frac{3k}{d\sqrt{N}}.\end{aligned}$$

Since $k < d$ and $d < \frac{N^{0.25}}{3}$, we have that $3k < 3d < N^{0.25}$, and hence

$$\left|\frac{e}{N} - \frac{k}{d}\right| < \frac{1}{dN^{0.25}} < \frac{1}{3d^2}. \tag{9.1}$$

Note that the fraction $\frac{e}{N}$ can be computed based on public information. Thus, this means that the secret fraction $\frac{k}{d}$ is very close to the public fraction $\frac{e}{N}$ the adversary can compute. It turns out that in this case, the adversary can even compute $\frac{k}{d}$ efficiently using the so-called continued fraction expansion.

9.4.1 Continued Fractions

We give a brief introduction on continued fractions, only as far as we will need to show how Wiener's attack works. A *(finite) continued fraction (expansion)* is an m-tuple of non-negative integers

$$[q_1, \ldots, q_m],$$

which is an abbreviation of the following rational number:

$$q_1 + \cfrac{1}{q_2 + \cfrac{1}{q_3 + \ldots + \cfrac{1}{q_m}}}.$$

For $1 \leq j \leq m$, the tuple $[q_1, \ldots, q_j]$ is said to be the j-th convergent of $[q_1, \ldots, q_m]$.

Suppose a and b are positive integers such that $\gcd(a, b) = 1$, then the continued fraction expansion $[q_1, \ldots, q_m]$ of $\frac{a}{b}$ can be computed using the Euclidean algorithm. Here the quotients arising in the algorithm when applied to a and b yield the desired m-tuple. This is best demonstrated by means of an example. Assume we would like to compute the continued fraction expansion of $\frac{30}{53}$. Then applying the Euclidean Algorithm to 30 and 53 yields:

$$\begin{aligned} 30 &= 0 \cdot 53 + 30 \\ 53 &= 1 \cdot 30 + 23 \\ 30 &= 1 \cdot 23 + 7 \\ 23 &= 3 \cdot 7 + 2 \\ 7 &= 3 \cdot 2 + 1 \\ 2 &= 2 \cdot 1 + 0 \end{aligned}$$

Then the continued fraction expansion of $\frac{30}{53}$ is the tuple $[0, 1, 1, 3, 3, 2]$. Correctness of this method of computing the tuple can be checked by a simple calculation:

$$[0,1,1,3,3,2] = 0 + \frac{1}{1+\frac{1}{1+\frac{1}{3+\frac{1}{3+\frac{1}{2}}}}} = \frac{1}{1+\frac{1}{1+\frac{1}{3+\frac{2}{7}}}} = \frac{1}{1+\frac{1}{1+\frac{7}{23}}} = \frac{1}{1+\frac{23}{30}} = \frac{30}{53}$$

The following famous theorem, which we give without proof, will later allow us to break RSA for small values of d.

Theorem 9.1 *Let $a, b, c, d \in \mathbb{N}$. Assume that $\gcd(a, b) = \gcd(c, d) = 1$ and*

$$\left|\frac{a}{b} - \frac{c}{d}\right| < \frac{1}{2d^2}.$$

Then $\frac{c}{d}$ equals one of the convergents of the continued fraction expansion of $\frac{a}{b}$. □

9.4.2 Wiener's Attack

Now we exploit Theorem 9.1 to mount a successful attack against RSA if d is sufficiently small. Equation 9.1 immediately allows us to apply Theorem 9.1 since $\gcd(k, d) = 1$ by assumption and since $\gcd(e, N) \neq 1$ would immediately allow us to factor N. Thus, one of the convergents of the continued fraction expansion of the known fraction $\frac{e}{N}$ equals the unknown and searched fraction $\frac{k}{d}$. These prefixes can easily be computed, but it remains to define a suitable check to identify which convergent is the desired one.

For every convergent $\frac{x}{y}$, we compute $M := (ey-1)/x$. Note that if $\frac{x}{y} = \frac{k}{d}$, then $M = \varphi(N)$. We then try to factor N given N and M using the method for factoring N and $\varphi(N)$ which we currently explore on the exercise sheet. If we picked the correct convergent, i.e., if we indeed have $M = \varphi(N)$, this procedure will give us the correct prime factors p and q of N; for wrong convergents, no solution will exist.

Let us conclude with an illustration of this attack for artificially small parameters: Suppose $N = 160523347$ and $e = 60728973$. The continued fraction expansion of $\frac{e}{N}$ is given by $[0, 2, 1, 1, 1, 4, 12, 102, 1, 1, 2, 3, 2, 2, 36]$; it's first convergents are 0, $\frac{1}{2}$, $\frac{1}{3}$, $\frac{2}{5}$, $\frac{3}{8}$, $\frac{14}{37}$, and so on. For $\frac{14}{37}$, we obtain $M = \frac{60728973 \cdot 37 - 1}{14}$. This yields $p = 12347$, $q = 13001$, and indeed $p \cdot q = N$. Thus $d = 37$.

9.5 How to Securely Encrypt using RSA (or any OWF)

We have already seen that using RSA naively results in an insecure encryption scheme, as it is vulnerable to several attacks. The central idea to avoid these attacks is to pre-process the messages

before applying the RSA function to them. This is often called *padding* and is used for two main reasons. First, it adds randomness to the encryption, as otherwise distinguishing ciphertexts is obviously easy. Secondly, its partially fixed structure prevents blinding attacks, e.g., the ones we have seen for naive RSA and for ElGamal.

9.5.1 PKCS#1 V1.5

PKCS#1 V1.5 is a standard from 1991 which is widely deployed in practice. It uses an ad-hoc padding scheme, whose security was never deeply analyzed, and which unfortunately turned out to be insecure. Bleichenbacher came up with a successful attack against PKCS#1 V1.5 in 1998, which caused V1.5 to be replaced with V2.0 that relies on a more suitable padding.

Let F be a (family of) keyed trapdoor permutations with key generation Gen and domains $\mathcal{M}_{pk}$, e.g, the RSA trapdoor permutation. Let n be a natural number and let efficient and invertible embeddings $\nu_{pk} : \{0,1\}^n \to \mathcal{M}_{pk}$ be given for all $(pk, *) \in [\mathsf{Gen}(n)]$. We use these embeddings implicitly when applying $\mathsf{F}(pk, \cdot)$ to bitstrings of length n. Let a message $m \in \{0,1\}^{n_0}$ be given, where $n_0 < n - 96$. Then the scheme works as follows:

- Generate public and secret keys $(pk, sk) \leftarrow \mathsf{Gen}(k)$.
- To compute $c \leftarrow \mathsf{E}(pk, m)$ for $m \in \{0,1\}^{n_0}$, choose a string r of length 64 bits at random, and compute
$$c := \mathsf{F}(pk,\ 0002 \parallel r \parallel 0000 \parallel m),$$
where numbers are in hexadecimal notation (i.e., both 0002 and 0000 are of length 16 bits).
- To compute $\mathsf{D}(sk, c)$, compute $\mathsf{F}^{-1}(sk, c) =: x \parallel r \parallel y \parallel m$, where $|x| = |y| = 16$ and $|m| = n_0$. If $x \neq 0002$ output $\downarrow$, else output m.

Correctness is verified as follows:

$$\begin{aligned}
\mathsf{D}(sk, \mathsf{E}(pk, m)) &= \mathsf{D}(sk, \mathsf{F}(pk, 0002 \parallel r \parallel 0000 \parallel m)) \\
&= \begin{cases} m & \text{, if } x = 0002 \\ \downarrow & \text{, else} \end{cases} \\
&\quad \text{where } x \parallel r \parallel m := \mathsf{F}^{-1}(sk, \mathsf{F}(pk, 0002 \parallel R \parallel 0000 \parallel m)) \text{ for } |x| = 16, |m| = n_0 \\
&= m.
\end{aligned}$$

The Bleichenbacher Attack Bleichenbacher's attack (which is specific to SSL which uses PKCS#1 V1.5) exploits the way the above padding interacts with the SSL implementation: After some handshake the browser randomly chooses a secret key K, pads the key as explained above and applies the RSA trapdoor permutation to this padded value. This ciphertext is then sent to the server. The server decrypts the ciphertext, and if it notices that the resulting plaintext is misformed, i.e., if its 16 most significant bits do not equal 0002, then it returns $\downarrow$. Else it answers with a different message, indicating that the key exchange was successfully completed.

Consider the following attacker: Given an arbitrary ciphertext c that it would like to decrypt, the attacker computes $c' := r^e \cdot c \bmod N$ for a randomly chosen value r and the public encryption key (e, N) of the server, and sends c' to the server yielding either an error or a confirmation message. Thus the answer of the server implicitly reveals if the underlying plaintext was of the correct format. The attacker repeats this step roughly four million times for different values r and collects all the answers. Every such answer gives the attacker one bit of information about the various plaintexts corresponding to the ciphertexts constructed in the above manner, and it turns out that using the

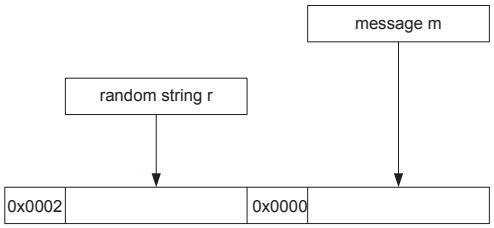


Figure 9.1: PKCS#1 V1.5

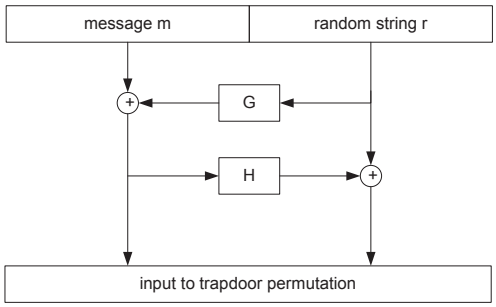


Figure 9.2: OAEP

homomorphic properties of RSA, an attacker can use this leaked information to fully reconstruct the plaintext corresponding to ciphertext c. Showing this calculation in detail is lengthy though, so that we refer to Bleichenbacher's original paper for details.

9.5.2 PKCS#1 V2.0 – OAEP

OAEP constitutes a generic method for constructing encryption schemes based on arbitrary keyed trapdoor permutations. Let a family of keyed trapdoor permutations F with key generation Gen and domains $\mathcal{M}_{pk}$ be given. Furthermore, let l, n be integers with $l < n$, and let $n_0 := n - l$. Let efficient and invertible embeddings $\nu_{pk}\colon \{0,1\}^n \to \mathcal{M}_{pk}$ be given for all $(pk, *) \in [\mathsf{Gen}(n)]$, which we again use implicitly when applying $\mathsf{F}(pk, \cdot)$ to bitstrings of length n. Let $G\colon \{0,1\}^{n_0} \to \{0,1\}^l$ and $H\colon \{0,1\}^l \to \{0,1\}^{n_0}$ be hash functions. The OAEP encryption scheme then works as follows:

- Generate public and secret keys $(pk, sk) \leftarrow \mathsf{Gen}(n)$.
- To compute $c \leftarrow \mathsf{E}(pk, m)$ for $m \in \{0,1\}^l$ first choose a random $r \leftarrow_{\mathcal{R}} \{0,1\}^{n_0}$ and let

$$c := \mathsf{F}(pk, m \oplus G(r) \parallel r \oplus H(m \oplus G(r))).$$

  This is illustrated in Figure 9.2.
- To compute $\mathsf{D}(sk, c)$ one computes $\mathsf{F}^{-1}(sk, c) =: x_1 \parallel x_2$, where $|x_1| = l$, and $|x_2| = n_0$. Then one calculates $m := x_1 \oplus G(x_2 \oplus H(x_1))$.

To see that decryption is correct one calculates

$$\begin{aligned}
\mathsf{D}(sk, \mathsf{E}(pk, m)) &= \mathsf{D}(sk, \mathsf{F}(pk, m \oplus G(r) \parallel r \oplus H(m \oplus G(r)))) \\
&= x_1 \oplus G(x_2 \oplus H(x_1)) \text{ for } x_1 \parallel x_2 = \mathsf{F}^{-1}(sk, \mathsf{F}(pk, m \oplus G(r) \parallel r \oplus H(m \oplus G(r)))) \\
&= (m \oplus G(r)) \oplus G((r \oplus H(m \oplus G(r))) \oplus H(m \oplus G(r))) \\
&= (m \oplus G(r)) \oplus G(r) \\
&= m.
\end{aligned}$$

The Random Oracle Model A proof of OAEP was only attempted in the so-called *Random Oracle model*, which constitutes an idealization of hash functions. Intuitively, instead of working with the correct properties of hash functions, e.g., one-wayness or collision-resistance, one assumed that the hash function is replaced by a truly random function. This random function is then written as a so-called *random oracle* as follows: (i) If a value is hashed for the first time, its hash value is

chosen randomly and independently of previous values, and (ii) if a value should be hashed that was already hashed before, then the (previously stored) same hash value is returned. The random oracle model thus constitutes an idealization of hash functions in order to allow for much simpler proofs. However, there is no theorem stating that these proofs remain valid if the random oracle is implemented with an actual hash function in reality, and there even exist results showing that in certain cases, a random oracle cannot be securely realized by any hash function. Thus proofs in the random oracle model should essentially be considered heuristics. (Recall that the Cramer-Shoup encryption scheme did not have to abstract its hash function into a random oracle which was one of the main reasons why this encryption scheme got a lot of attention.)

On the Security of OAEP There traditionally was a security proof of OAEP in the random oracle model. However, Shoup found a gap in this proof in 2000, and showed that this gap cannot be closed in general, i.e., that CCA2-security of OAEP cannot be proved for arbitrary trapdoor-permutations.[1] After discovering this gap, Shoup also proposed an improved construction OAEP+, which we will see below, and proved it CCA2-secure in the random oracle model. After Shoup's work, Okamoto et al. showed that OAEP based on the RSA trapdoor permutation is CCA2-secure. The resulting encryption scheme is often called RSA-OAEP, and its security could only be proven by exploiting properties similar to random self-reducibility that are specific to RSA.

9.5.3 OAEP+

OAEP+ constitutes a generic method for constructing encryption schemes based on arbitrary keyed trapdoor permutations, similar to OAEP. Let a family of keyed trapdoor permutations F with key generation Gen and domains $\mathcal{M}_{pk}$ be given. Let k, l, n be integers with $k + l < n$ such that both 2^{-k} and 2^{-l} are negligible in n. Let efficient and invertible embeddings $\nu_{pk} : \{0,1\}^n \to \mathcal{M}_{pk}$ be given for all $(pk, *) \in [\mathsf{Gen}(n)]$ as before. Let $G\colon \{0,1\}^k \to \{0,1\}^{n-k-l}$, $H'\colon \{0,1\}^{n-l} \to \{0,1\}^l$, and $H\colon \{0,1\}^{n-k} \to \{0,1\}^k$ be hash functions. The OAEP+ encryption scheme then works as follows:

- Generate public and secret keys $(pk, sk) \leftarrow \mathsf{Gen}(n)$.
- To compute $c \leftarrow \mathsf{E}(pk, m)$ for $m \in \{0,1\}^{n-k-l}$, choose a random $r \leftarrow_{\mathcal{R}} \{0,1\}^k$ and let

$$\begin{aligned} s &:= (G(r) \oplus m) \parallel H'(r \parallel m), \text{ and} \\ c &:= \mathsf{F}(pk, s \parallel (H(s) \oplus r)). \end{aligned}$$

  This is illustrated in Figure 9.3.
- To compute $\mathsf{D}(sk, c)$ one computes $\mathsf{F}^{-1}(sk, c) =: x_1 \parallel x_2 \parallel x_3$ where $|x_1| = n - k - l$, $|x_2| = l$, and $|x_3| = k$. Then one calculates

$$m = G(H(x_1 \parallel x_2) \oplus x_3) \oplus x_1.$$

  If $x_2 = H'(x_3 \oplus H(x_1 \parallel x_2) \parallel m)$ then output m as the decryption, otherwise reject the ciphertext.

Correctness of decryption can be shown as usual. Shoup showed that if F is a family of keyed trapdoor permutations, then OAEP+ is CCA2-secure in the random oracle model.

[1] This gap essentially arose from confusions with different security definitions that will not matter in the sequel. We only point out that OAEP is CCA1-secure, which is a strictly weaker notion of security than CCA2-security.

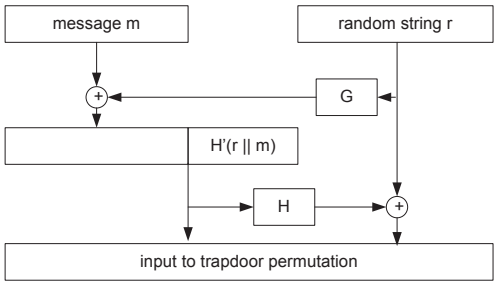


Figure 9.3: OAEP+

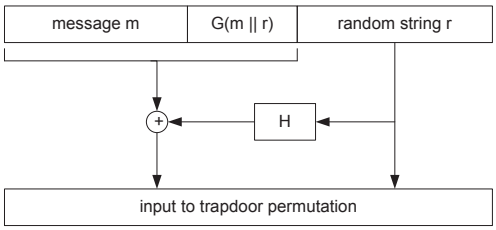


Figure 9.4: SAEP+

9.5.4 RSA-SAEP+

SAEP+ was introduced by Boneh in 2001 as a construction that is simpler than OAEP+ while additionally providing tighter security proofs. As a minor drawback, it can only be used with the RSA and the so-called Rabin trapdoor permutations. We will concentrate on RSA-SAEP+ in the sequel, i.e., SAEP+ based on the RSA trapdoor permutation.

Let a family of keyed trapdoor permutations F with key generation Gen and domains $\mathcal{M}_{pk}$ be given. Let j, k, l, n be integers with $n = j + k + l$ and $j > n/2$, and let efficient and invertible embeddings $\nu_{pk} : \{0,1\}^n \to \mathcal{M}_{pk}$ be given for all $(pk, *) \in [\mathsf{Gen}(n)]$ as before. Let $G\colon \{0,1\}^{k+l} \to \{0,1\}^j$, and $H\colon \{0,1\}^l \to \{0,1\}^{j+k}$ be hash functions. The RSA-OAEP+ encryption scheme then works as follows:

- Generate public and secret keys $(pk, sk) \leftarrow \mathsf{Gen}(n)$.
- To compute $c \leftarrow \mathsf{E}(pk, m)$ for $m \in \{0,1\}^k$, choose a random $r \leftarrow_{\mathcal{R}} \{0,1\}^l$ and let

$$c := \mathsf{F}(pk, H(r) \oplus (m \parallel G(m \parallel r)) \parallel r).$$

  This is illustrated in Figure 9.3.
- To compute $\mathsf{D}(sk, c)$ one computes $\mathsf{F}^{-1}(sk, c) =: x_1 \parallel x_2$, where $|x_1| = j + k$ and $|x_2| = l$. Then one calculates

$$m \parallel y_2 := x_1 \oplus H(x_2) \text{ where } |m| = k \text{ and } |y_2| = j.$$

  If $y_2 = G(m \parallel x_2)$ then output m as the decryption, otherwise reject the ciphertext.

Correctness can be shown as usual. Similar to OAEP+, RSA-SAEP+ is CCA2-secure provided that RSA is indeed a trapdoor permutation.

9.5.5 Implementation Attacks on Encryption Schemes

The attacks we have investigated so far all exploited the input/output behavior of the encryption scheme. In addition to this, measuring the time a decryption algorithm needs to perform a specific computation might leak information on the secret decryption key. Attacks that measure timing characteristics of cryptographic algorithms are called *timing attacks*. Exponentiations $c^d \bmod N$ are usually carried out by the repeated squaring technique, i.e., for each bit of the secret exponent d the algorithm either performs a squaring operation, or a squaring operation and a multiplication. Thus in the first case, the algorithm will clearly proceed faster so that simply measuring the execution time reveals the hamming-weight of the secret exponent d. Statistical analysis of a large number

of such timings then allows for recovering the complete value of d. Note that timing attacks can be carried out easily if one has access to a computer performing cryptographic operations, but sometimes even remote attacks are possible by measuring reaction times of servers provided that the network latency is sufficiently low.

A more sophisticated type of attacks are *power* and *emanation attacks*, where an attacker can measure the current energy a microprocessor is consuming while performing operations involving a secret key. For naive implementations of an algorithm one can sometimes directly see secret bits in the measured power consumption curves, e.g., when a register is shifted cyclically. This attack type is a severe threat against smartcards used, e.g., as signature cards, as the manipulation of the reader might remain undetected.

Fault attacks are slightly more esoteric in practice but very interesting from a conceptional point of view. By injecting errors into computations involving a secret key, an attacker might be able to provoke a different result that might be helpful for recovering secret information. These faults often are inserted by radiation such as X-ray, but also heating of microprocessors or memory can lead to errors that can be successfully exploited.

10. Digital Signature Schemes

Digital signature schemes are the electronic equivalent of handwritten signatures, i.e., they allow distinguished principles to sign messages in a way that everybody is able to check the validity of the signatures while additionally ensuring that nobody else is able to produce valid signatures. Technically, digital signatures can thus be seen as an asymmetric variant of MACs: While knowing the secret key is a necessary prerequisite to verify the validity of a MAC, signature schemes enable signature verification based on publicly available keys.

Digital signatures are used in various contexts and applications in practice, e.g., in electronic payment systems and for issuing certificates. In fact, digital signatures might even be of higher importance than encryption schemes in practice, as violations of integrity often turn out to be the most important concern.

10.1 Definition of Digital Signature Schemes

We now formalize the notion of digital signature schemes, or short *signature schemes*. Note that the definition closely resembles the notion of MACs, expressed in an asymmetric scenario.

Definition 10.1 (Signature Schemes) *A* signature scheme *consists of three efficient algorithms* $(\mathsf{Gen}, \mathsf{S}, \mathsf{V})$ *that are defined as follows:*

- *The randomized* key generation algorithm Gen *takes the security parameter n in unary representation as input and returns a pair (pk, sk) of keys, the* public key *pk and the corresponding* private key *sk.*
- *The* signing algorithm S *takes the secret key sk and a message m and returns a tag t. This algorithm may be randomized.*
- *The deterministic* verification algorithm V *takes the public key pk, a message m, and a tag t, and returns* true *or* false.
- *The* message space $\mathcal{M}_{pk}$ *for a public key pk is the set of all m such that $\mathsf{S}(sk, m)$ does not output a distinguished error symbol $\downarrow$ for all sk with $(pk, sk) \in [\mathsf{Gen}(n)]$. We require that, for all $n \in \mathbb{N}$, for all $(pk, sk) \in [\mathsf{Gen}(n)]$ and any message $m \in \mathcal{M}_{pk}$, if $t \leftarrow \mathsf{S}(sk, m)$ then $\mathsf{V}(pk, m, t) = \mathsf{true}$.*

$\diamond$

Most practical signature schemes rely on the message space $\mathcal{M}_{pk} = \{0,1\}^{n_0}$ for a value n_0 that only depends on n but not specifically on pk, or they allow the signing of arbitrary strings, i.e., $\mathcal{M}_{pk} = \{0,1\}^*$.

We again assume that all algorithms (including challengers and adversaries in the following) get the security parameter in unary representation as input so that their efficiency can be meaningfully measured in the security parameter. We do not write this explicitly for the sake of readability.

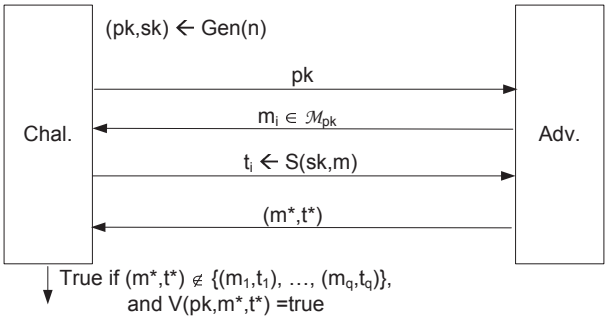


Figure 10.1: CMA-security of Signature Schemes

10.2 CMA-security of Signature Schemes

Security of signature schemes against existential forgery under chosen-message attack (CMA) is then defined similar to the corresponding notion for MACs. The corresponding challenger is called a CMA challenger for signatures, but we simply speak of a CMA challenger if confusion with a CMA challenger for MACs can be excluded from the context.

Definition 10.2 (CMA Challenger for Signatures) *Let* $\mathsf{I} = (\mathsf{Gen}, \mathsf{S}, \mathsf{V})$ *be a signature scheme and let* n *be the security parameter. The* CMA challenger (for signatures) *for* I *and* n *is defined as follows:*

- *It first creates keys* $(pk, sk) \leftarrow \mathsf{Gen}(n)$ *and outputs* pk*.*
- *It then receives messages* $m_i \in \mathcal{M}_{pk}$ *and outputs* $t_i \leftarrow \mathsf{S}(sk, m_i)$*. This stage may repeat arbitrarily but finitely often, thus yielding a sequence of pairs* $(m_1, t_1), \ldots, (m_q, t_q)$*.*
- *Finally, it receives a pair* (m^*, t^*)*. If* t^* *is a valid tag for* m^* *and this pair is not contained in the obtained sequence, i.e., if* $\mathsf{V}(pk, m^*, t^*) = \mathsf{true}$ *and* $\forall i \in \{1, \ldots, q\}\colon (m_i, t_i) \neq (m^*, t^*)$*, then the CMA challenger outputs* true*, and* false *otherwise.*

◇

The game between the CMA challenger and an adversary is depicted in Figure 10.1. The adversary wins the game if it is able to compute a tuple (m^*, t^*) such that the challenger outputs true. This tuple is called an *existential forgery*. Another type of forgery is a *selective forgery*, where an adversary has to be able to find a tag t' *for every* (given) message m'. In the following, $Exp_{\mathsf{A}}^{\mathsf{CMA}}$ denotes the experiment where the adversary A interacts with the CMA challenger, and we let $Exp_{\mathsf{A}}^{\mathsf{CMA}} = \mathsf{true}$ denote the event that the CMA challenger finally outputs true. The advantage can now be defined as a function of n as usual.

Definition 10.3 (CMA Advantage) *Let* $\mathsf{I} = (\mathsf{Gen}, \mathsf{S}, \mathsf{V})$ *be a signature scheme. The advantage of an adversary* A *against the CMA challenger for* I *is defined as follows (as a function of the security parameter* n *again since all algorithms are parametrized with* n*):*

$$Adv^{\mathsf{CMA}}[\mathsf{A}, \mathsf{I}](n) := \mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{CMA}} = \mathsf{true}\right].$$

◇

Definition 10.4 (CMA-secure Signature Schemes) *A signature scheme* $\mathsf{I} = (\mathsf{Gen}, \mathsf{S}, \mathsf{V})$ *is* secure against existential forgery under chosen-message attack (CMA) *if* $Adv^{\mathsf{CMA}}[\mathsf{A}, \mathsf{I}](n)$ *is negligible for all efficient adversaries* A*. We often say* CMA-secure signature scheme *for brevity instead of a signature scheme that is secure against existential forgery under chosen-message attack.* $\diamond$

10.3 On Signing Message Digests

Signature schemes are almost always used in conjunction with a hash function $\mathsf{H} : \{0,1\}^* \to \mathcal{D}$, where $\mathcal{D}$ is contained in the set of messages that can be signed. Given a message m, the signature algorithm first applies the hash function yielding $d := \mathsf{H}(m) \in \mathcal{D}$, and then signs this message digest yielding the signature. To verify a signature one computes $d := \mathsf{H}(m) \in \mathcal{D}$ and uses the verification algorithm for this digest.

The hash function used in this design necessarily has to be collision-resistant, as otherwise the following attack can be carried out. First, find a collision for the hash function, i.e., $m \neq m'$ with $\mathsf{H}(m) = \mathsf{H}(m')$, then let the challenger sign m. This signature is a valid signature both for m and m', as their hash-value is the same. Thus both pass verification regardless of the concrete verification algorithm. Note that this attack in practice would typically require that an attacker can come up with two *meaningful* messages that hash to the same value. On the other hand, for MD5 it took roughly one year from finding the first collision to efficiently finding two postscript files that hash to the same value. Thus it is unwise to hope that finding a collision of two meaningful messages is much harder than finding any collision.

Depending on the structure of the verification algorithm the hash function may also need to be one-way. This holds, e.g., for the PKCS#1 signature scheme that we will present below.

10.4 Well-known Signature Schemes

This section is devoted to several well-known signature schemes. We start with so-called one-time signatures which can only be used to sign a single message. While this clearly constitutes a severe constraint in practice, the corresponding schemes were and often still are interesting from a conceptual point of view, and they sometimes are even sufficient for practical applications. After that, we move on to the ElGamal signature scheme and variations thereof, whose security are based on the discrete logarithm problem. Finally, we consider signature schemes based on trapdoor permutations, e.g., based on RSA. This class of signature schemes in particular contains the full-domain hash, which constitutes an efficient scheme that can be proven CMA-secure provided that trapdoor permutations exist; however the proof is in the random oracle model and thus should only be considered a heuristic.

10.4.1 One-time Signatures

We review one of the first one-time signature schemes which is due to Lamport. It is interesting from a theoretical point of view since it shows that secure one-time signature schemes can be achieved only assuming the existence of one-way functions. We further sketch below how this scheme can be extended to allow signing of multiple messages. The drawback of this signature scheme is its bad expansion factor, i.e., the size of the signature compared to the size of the message to be signed.

Lamport's one-time signature scheme works as follows: Let F be a family of keyed one-way functions with key generation algorithm Gen and domains $\mathcal{M}_{pk}$. Let $n \in \mathbb{N}$ denote the security parameter as usual.

- The key generation algorithm creates a public key $pk' \leftarrow \mathsf{Gen}(n)$ of F, randomly chooses $y_{i,j} \leftarrow_{\mathcal{R}} \mathcal{M}_{pk}$, and sets $z_{i,j} := \mathsf{F}(pk', y_{i,j})$ for $1 \leq i \leq n$ and $1 \leq j \leq 2$. The public key pk is then defined as $pk := (pk', (z_{i,j})_{1 \leq i \leq n, 1 \leq j \leq 2})$; the private key sk is defined as $sk := ((y_{i,j})_{1 \leq i \leq n, 1 \leq j \leq 2})$.
- The signature sig of a message $m = m_1 \cdots m_n \in \{0,1\}^n$ with respect to a secret key $sk := ((y_{i,j})_{1 \leq i \leq n, 1 \leq j \leq 2})$ is computed as

$$sig := (y_{1,m_1}, \ldots, y_{n,m_n}).$$

- A signature $sig = (y_{1,m_1}, \ldots, y_{n,m_n})$ on a message $m = m_1 \cdots m_n \in \{0,1\}^n$ is verified with respect to a public key $pk := (pk', (z_{i,j})_{1 \leq i \leq n, 1 \leq j \leq 2})$ by computing

$$\mathsf{V}(pk, m, sig) := \begin{cases} \mathsf{true} & \text{if } \mathsf{F}(pk', y_{i,m_i}) = z_{i,m_i} \text{ for all } 1 \leq i \leq n, \\ \mathsf{false} & \text{otherwise.} \end{cases}$$

It can be easily seen that a correctly generated signature passes verification. Let $y_{i,j}$, $z_{i,j}$, and pk' be as constructed by the key generation algorithm, and let $(s_1 \cdots s_n)$ be a signature for the message $(m_1 \cdots m_n)$. But then by construction $s_i = z_{i,m_i}$, and thus by construction of the $z_{i,j}$ we have $s_i = \mathsf{F}(pk', y_{i,m_i})$, thus it passes the verification algorithm.

10.4.2 ElGamal Signatures

The ElGamal signature scheme was proposed in 1985, i.e., roughly at the same time the ElGamal encryption scheme was invented. A modified version of the ElGamal signature scheme was later adopted as the Digital Signature Algorithm (DSA) which we will describe in the next section. Similar to the ElGamal encryption scheme, the ElGamal signature is probabilistic, i.e., signing a message multiple times will yield different signatures. The ElGamal signature scheme is defined as follows:

- The key generation algorithm bears strong similarity to the key generation of the ElGamal encryption scheme: it randomly chooses an n-bit prime p, a generator g of $\mathbb{Z}_p^*$, and an integer $x \in \{1, \ldots, p-1\}$. It then sets $h := g^x$. The public key and secret key are $pk := (p, g, h)$ and $sk := (p, g, x)$, respectively. The message space is $\mathcal{M}_{pk} := \{1, \ldots, p-1\}$.
- A signature sig of a message $m \in \mathcal{M}_{pk}$ with respect to a secret key $sk = (p, g, x)$ is computed by choosing a random $r \leftarrow_{\mathcal{R}} \{1, \ldots, p-1\}$ and by then setting

$$\begin{aligned} s &:= g^r \bmod p, \\ t &:= (m - xs) r^{-1} \bmod (p-1), \\ sig &:= (s, t). \end{aligned}$$

- A signature $sig = (s, t)$ on a message m is verified with respect to a public key $pk = (p, g, h)$ by computing

$$\mathsf{V}(pk, m, (s,t)) := \begin{cases} \mathsf{true} & \text{if } h^s s^t = g^m \bmod p, \\ \mathsf{false} & \text{otherwise.} \end{cases}$$

The correctness of ElGamal signatures can be seen as follows: Let $(s,t) \leftarrow \mathsf{S}(sk,m)$ be a signature for pk and sk as defined above. Then we have

$$\begin{aligned}
\mathsf{V}(pk,m,(s,t)) = \mathsf{true} &\Leftrightarrow h^s s^t = g^m \bmod p \\
&\Leftrightarrow g^{xs}(g^r)^{(m-xs)r^{-1}} = g^{xs+r(m-xs)r^{-1}} = g^m \bmod p \\
&\Leftrightarrow xs + (m-xs) = m \bmod p-1 \\
&\Leftrightarrow m = m
\end{aligned}$$

Now we take a look at the security of the ElGamal signature scheme. Let us first note that the security of ElGamal is not proven, and in fact, the ElGamal signature scheme is not CMA-secure in the form it is stated above. We will first give some indication why selectively forging signatures could be as hard as computing discrete logarithms. Suppose Oscar tries to forge a message m without knowing x. If he chooses s first and then tries to find t such that (s,t) is a signature for m, then he has to find $t = \mathsf{DLog}_s g^a h^{-s}$, i.e., to solve a discrete logarithm. If, on the other hand, he chooses t first, then he has to solve the equation $h^s s^t = g^a \bmod p$ for s. For this no feasible solution is known, however, it cannot be reduced to the discrete logarithm problem. There might even be the possibility that one can compute s and t simultaneously; however, no one one has discovered a way to do this so far.

Things change very much if an attacker is satisfied with creating an existential forgery, which we consider as the standard threat for signature schemes we would like to defend against. We will see in the next paragraph that existential forgeries can be computed efficiently for the ElGamal signature scheme.

Existential Forgeries against the ElGamal Signature Scheme The ElGamal signature scheme as described above is existentially forgeable as follows. Suppose we know the public key only, i.e., we know g and h, and we want to find a signature $sig = (s,t)$ which passes the verification algorithm. Suppose i,j are integers such that $1 \leq i,j \leq p-2$, and suppose we express s in the form $s = g^i h^j$. Then the verification condition is $g^m = h^s(g^i h^j)^t \bmod p$, or, equivalently, $g^{m-it} = h^{s+jt} \bmod p$. The later congruence is satisfied if $m - it = 0 \bmod p-1$ and $s + jt = 0 \bmod p-1$. We are given i and j, thus, provided that $\gcd(j, p-1)$, we can easily solve these congruences for t and m, obtaining $t = -sj^{-1}$ and $m = it = -sij^{-1}$. Now from the construction is it immediately clear that (s,t) is a valid signature for m. This attack can be countered by applying a hash function to m before it is input to the ElGamal signature scheme.

10.4.3 Schnorr Signatures

Schnorr signatures have been proposed in 1989 and can be seen as a variant of ElGamal signatures while offering better performance. The ElGamal signature scheme (and its strengthening by using hash functions) rely on a prime p which has to be chosen large enough to prevent an attacker from computing discrete logarithms in $\mathbb{Z}_p^*$. Thus 1024 bit is a good choice; consequently, ElGamal signatures have a length of 2048 bits. Schnorr signatures can significantly decrease this size, typically down to 320 bits by performing computations in subgroups G_q of $\mathbb{Z}_p^*$. (Similar to the definition of ElGamal in subgroups G_q of $\mathbb{Z}_p^*$, we use n to denote the size of q and assume a function $n_p(\cdot)$ such that p is of size $n_p(n)$.)

Let $\mathsf{H}\colon \{0,1\}^* \to \mathbb{Z}_q$ be a hash function. Then the Schnorr signature scheme is defined as follows.

- The key generation algorithm randomly chooses an n-bit prime q and $n_p(n)$-bit prime p such that $q \mid p-1$. It picks an element $g \in \mathbb{Z}_p^*$ of order q and a random $x \leftarrow_{\mathcal{R}} \{1,\ldots,q\}$, and it computes $h := g^x$. The public key and secret key are then defined as $pk = (q,p,g,h)$ and $sk = (q,p,g,x)$, respectively. The message space is $\mathcal{M}_{pk} = \{0,1\}^*$.
- A signature sig on a message $m \in \{0,1\}^*$ with respect to a secret key $sk = (q,p,g,x)$ is computed by choosing a random $r \leftarrow_{\mathcal{R}} \{1,\ldots,q\}$ and by then setting

$$\begin{aligned} s &:= \mathsf{H}(m \parallel g^r), \\ t &:= r + xs \bmod q, \\ sig &:= (s,t). \end{aligned}$$

- A signature $sig = (s,t)$ on a message m is verified with respect to a public key $pk = (q,p,g,h)$ by computing

$$\mathsf{V}(pk,m,(s,t)) := \begin{cases} \mathsf{true}, & \text{if } \mathsf{H}(m \parallel g^t h^{-s}) = s \\ \mathsf{false} & \text{otherwise.} \end{cases}$$

The correctness of Schnorr signatures can be seen as follows: Let $(s,t) \leftarrow \mathsf{S}(sk,m)$ be a signature for pk and sk as defined above. Then we have

$$\begin{aligned} \mathsf{V}(pk,m,(s,t)) = \mathsf{true} &\Leftrightarrow \mathsf{H}(m \parallel g^t h^{-s}) = s \\ &\Leftrightarrow \mathsf{H}(m \parallel g^{r+xs} h^{-s}) = \mathsf{H}(m \parallel g^r) \\ &\Leftrightarrow \mathsf{H}(m \parallel g^{r+xs-xs}) = \mathsf{H}(m \parallel g^r) \\ &\Leftrightarrow \mathsf{H}(m \parallel g^r) = \mathsf{H}(m \parallel g^r) \\ &\Leftrightarrow \mathsf{true} \end{aligned}$$

An attacker that can compute discrete logarithms can clearly forge Schnorr signatures. The converse is, however, not known.

10.4.4 Digital Signatures Algorithm (DSA)

The digital signature algorithm (DSA) was proposed by the NIST in 1991 for the Digital Signature Standard (DSS). Some criticisms however arose when the standard was proposed: One (yet minor) issue was the "closed-door" approach that was employed in the process which did not involve U.S. industry. The more severe point of criticism was however that the size of the modulus p was initially fixed to 512 bits. The standard was later updated to allow moduli p of larger size, and in 2001 the NIST itself recommended to use 1024 bit primes p.

The Digital Signature Algorithm is defined as follows:

- The key generation algorithm randomly chooses an L-bit prime p where L has to satisfy $L = 0 \bmod 64$ and $512 \leq L \leq 1024$, and a 160-bit prime q such that $q \mid p-1$. It then randomly chooses $g \in \mathbb{Z}_p^*$ of order q and $x \leftarrow_{\mathcal{R}} \{0,\ldots,q-1\}$. Finally it sets $h := g^x$. The public key and secret key are then defined as $pk = (q,p,g,h)$ $sk = (q,p,g,x)$, respectively.

- A signature sig on a message $m \in \{0,1\}^*$ with respect to a secret key $sk = (q,p,g,x)$ is computed by choosing a random $r \leftarrow_{\mathcal{R}} \{1,\ldots,q\}$ and by then setting

$$\begin{aligned} s &:= (g^r \bmod p) \bmod q, \\ t &:= (\mathsf{SHA\text{-}1}(m) + xs) r^{-1} \bmod q, \\ sig &:= (s,t), \end{aligned}$$

where SHA-1 denotes the respective hash function. If either s or t equals 0, a new r is chosen and the signature is recomputed. (This happens only with very small probability and thus does not raise a problem in practice).

- A signature $sig = (s, t)$ on a message m is verified with respect to a public key $pk = (q, p, g, h)$ by computing

$$\mathsf{V}(pk, m, (s,t)) := \begin{cases} \mathsf{true} & \text{if } (g^{\mathsf{SHA\text{-}1}(m)t^{-1} \bmod q} h^{st^{-1} \bmod q} \bmod p) \bmod q = s, \\ \mathsf{false} & \text{otherwise.} \end{cases}$$

Correctness can be shown as usual.

10.4.5 RSA-based Signature Schemes

In this section we describe a very simple construction of a signature scheme based on the RSA trapdoor permutation. This construction can be found in various books, however, it can easily be broken by various attacks. Later we will see how these attacks can be countered by using hash functions and including redundancy inside the signature, as it is done for example in the PKCS#1 standard.

The naive RSA signature scheme works as follows:

- Keys are generated exactly as for (naive) RSA encryption: Randomly choose two n-bit primes p and q, let $N := pq$, and pick e, d such that $ed = 1 \bmod \varphi(N)$. Let $pk = (N, e)$ be the public key, and let $sk = d$ be the secret key.
- To sign a message $m \in \{1, \ldots, N-1\}$ with respect to the secret key $sk = d$ one computes

$$\mathsf{S}(sk, m) := m^d \bmod N.$$

- A signature sig of a message m with respect to the public key pk is verified by computing

$$\mathsf{V}(pk, m, s) := \begin{cases} \mathsf{true} & \text{if } sig^e = m \bmod N, \\ \mathsf{false} & \text{otherwise.} \end{cases}$$

Correct verification is a direct consequence of Lemma 9.1.

Attacks against Naive RSA Signatures There are several attacks against the above signature scheme. The first attack is an existential forgery under a passive attack: One starts by picking an arbitrary $sig \in \mathbb{Z}_N$ and computes $m := sig^e \bmod N$. Then sig is a signature for the message m, as one can easily verify.

A more sophisticated attack is the following, which can find selective forgeries under an active attack. It uses blinding techniques similar to those we have seen earlier. Suppose the adversary wants to get a signature for a message m. He chooses a random $r \in \mathbb{Z}_N^*$ and computes $m' := m \cdot r^e \bmod N$. He asks the signer to sign m', yielding a signature $s' = (m')^d \bmod N$. Finally we compute $s := s'/r$. Then s is a valid signature for m, as the following computation shows:

$$s^e = \left(\frac{s'}{r}\right)^e = \left(\frac{(m')^d}{r}\right)^e = \left(\frac{(mr^e)^d}{r}\right)^e = \left(\frac{m^d \cdot r}{r}\right)^e = \left(m^d\right)^e = m \bmod N.$$

Note that the ability to let a message be signed blindly can even be a desirable feature which is used in a cryptographic primitive called *blind signature*. This is used whenever it is desirable that the signer should not learn what he is actually signing, e.g., this preserves anonymity in electronic payment systems. We will see this in more detail later in the course.

Adding Redundancy One possibility to counter these attacks is to add redundancy to the message before applying the RSA signature operation, i.e., one computes $sig := (\mathsf{red}(m), m)^d \bmod N$, and verifies by testing $sig^e = (\mathsf{red}(m), m) \bmod N$. Now what should this padding look like? A bad way to do it is to prepend zeroes, as this enables mainly the same attacks as before. A good thing is to prepend something "chaotic", e.g., the encryption of the message m under DES with a fixed (public) key. Then a signature needs to be rejected if the padding is not of the correct form. This padding is used in an ISO standard. The padding in the PKCS#1 standard is similar: There a fixed prefix of the following form is prepended to the message (in hexadecimal notation)

$$0001\ 0101\ 0101\ \ldots\ 0101\ 0000\ ||\ \mathsf{H}(m).$$

10.4.6 Full-domain Hash

The full-domain hash (FDH) signature scheme, or short full-domain hash, uses the common design principle *hash-then-sign*, cf. Section 10.3. It is defined over an arbitrary trapdoor permutation and an arbitrary hash function. The only requirement is that it is important that the range of the hash function is the same as the domain of the trapdoor permutation (thus the name "full-domain").

Let a family of keyed trapdoor permutations F with key generation Gen and domains $\mathcal{M}_{pk}$ be given. Furthermore, let $\mathsf{H}\colon \{0,1\}^* \to \mathcal{M}_{pk}$ be a hash function. Then the full-domain hash (FDH) signature scheme is defined as follows:

- The key generation algorithm simply generates the key for the underlying trapdoor permutation: Let $(pk, sk) \leftarrow \mathsf{Gen}(n)$; the public and the secret key of the signature scheme are then pk and sk, respectively.
- The signature sig of a message $m \in \mathcal{M}_{pk}$ with respect to the secret key sk is computed as

$$sig := \mathsf{F}^{-1}(sk, \mathsf{H}(m)).$$

- A signature sig on a message m is verified with respect to a public key pk by computing

$$\mathsf{V}(pk, m, sig) := \begin{cases} \mathsf{true} & \text{if } \mathsf{F}(pk, sig) = \mathsf{H}(m), \\ \mathsf{false} & \text{otherwise.} \end{cases}$$

Theorem 10.1 (Bellare, Rogaway'94) *The FDH signature scheme is existentially unforgeable under chosen-message attack assuming that F is one-way. This holds in the random oracle model, i.e., H is considered an ideal hash functions.* □

Proof. Given an adversary A that breaks the FDH signature scheme with not negligible probability, we construct an adversary B that inverts the trapdoor permutation F with not negligible probability.

Assume the attacker A makes a polynomially bounded number of q_{hash} queries $a_1, \ldots, a_{q_{hash}}$ to the random oracle and a polynomially bounded number of q_{sign} queries $m_1, \ldots, m_{q_{sign}}$ to the signature challenger and finally outputs a candidate forgery (m, sig). Without loss of generality we may assume that a message m_j was queried to the random oracle before it is queried to the signature oracle, and similarly, that the forged message m was queried to the random oracle. (Otherwise we can construct an adversary A' which behaves identical as A, but additionally makes these queries, and this A' still breaks the CMA-security of FDH.)

In the first step the adversary B gets the public key pk and a value $y = \mathsf{F}(x)$ for some unknown, randomly chosen value $x \in \mathcal{M}_{pk}$. Subsequently he does the following:

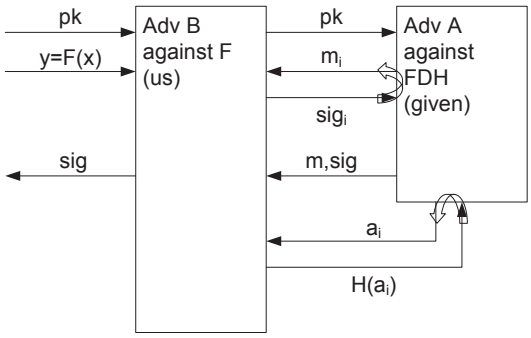


Figure 10.2: Attack game against the FDH

- It first chooses a random $k \leftarrow_{\mathcal{R}} \{1, \ldots, q_{hash} + q_{sign} + 1\}$.
- Upon receiving any hash query a_j, it tests if this value was queried before, in which case it answers the same value as before (all queries and the corresponding results are stored by B). Otherwise, if $j = k$ then it returns y, else it chooses a random $s_j \leftarrow_{\mathcal{R}} \mathcal{M}_{pk}$ and returns $\mathsf{F}(pk, s_j) =: h_j$ (Note that h_i is random because s_i is random and F is a permutation).
- Upon receiving a signature query m_l, it determines the first hash query a_j for this m_l (we assumed that m_l was queried to the random oracle before, so such an a_j will exist). If $j = k$ then output $\downarrow$, else output s_j.
- Receiving the (candidate) forgery (m, sig) find the first hash query a_j for m (again, we assumed this always exists). If $j = k$ then output sig, else abort with $\downarrow$.

Note first that in order for (m, sig) being a forgery, we must have $m \neq m_i$ for all remaining messages m_i that occurred in sign queries: Since signing is deterministic in FDH, this would imply $sig = sig_i$ so that the CMA challenger would output false upon receiving (m, sig) as a candidate forgery. Let $(a_1, \ldots, a_{q_{hash}+q_{sign}+1})$ denote the sequence of pairwise different messages that were used as hash queries (also containing the messages that we would have to add if we replaced A by A'). Then the adversary B gives a perfectly correct simulation to A provided that the two following conditions are satisfied. First, for all messages m_i that were used as sign queries, the corresponding hash queries $a_j = m_i$ satisfies $j \neq k$. Secondly, we have $a_k = m$. Since $m \neq m_i$ as discussed above, since k is randomly chosen from $\{1, \ldots, q_{hash} + q_{sign} + 1\}$, and since $y = \mathsf{F}(pk, x)$ is random because x is random and F is a permutation, these two conditions are simultaneously satisfied with probability at least $\frac{1}{q_{hash}+q_{sign}+1}$. If A succeeds in forging the signature for m in this case, then he outputs a signature sig such that $\mathsf{F}(pk, sig) = \mathsf{H}(m)$; thus sig is indeed a pre-image of y by construction. One finally obtains

$$\begin{aligned} \mathsf{Pr}[\mathsf{A} \text{ wins}] &= \mathsf{Pr}[\mathsf{F}(sig) = \mathsf{F}(x);\ x \leftarrow_{\mathcal{R}} \{0,1\}^n,\ y := \mathsf{F}(x),\ sig \leftarrow \mathsf{B}(n, y)] \\ &\geq \frac{1}{q_{hash} + q_{sign} + 1} \cdot Adv^{\mathsf{CMA}}[\mathsf{A}, FDH]. \end{aligned}$$

■

11. Certificates

We already saw in Chapter 5 that the secure exchange of symmetric keys is a difficult task, since both privacy and authenticity of the keys have to be ensured. In particular, this served as our major motivation for coming up with public-key cryptography, where secure exchange of public keys only requires authentic transmission of the keys. This chapter is finally dedicated to showing how such an authentic transmission can be achieved.

The main idea can be roughly summarized as follows: One considers a central authority, a so-called *certification authority (CA)*, that is trusted by every user. We assume that this authority correctly checks the identity of every person that is willing to create a key pair, and then issues a so-called certificate. This certificate contains a signature over the public key as well as the identity of the key's owner, thus binding the public key to a specific real-life person or organization. This signature is created using the secret signing key of the CA. Ideally, every user knows the corresponding public key of the CA so that the validity of every certificate can be locally tested by everybody. The resulting infrastructure is often called a *public-key infrastructure* (PKI).

11.1 (Key-) Certificates

A certificate binds a public key to a real-life person or an organization. Intuitively, certificates constitute signed statements of the form

> "I, Max Mueller, take responsibility for the secret key which belongs to the public key 100101100101010...0101. This is affirmed by the certification authority CA."

Technically speaking, a certificate consists of a digital signature over the user's name and additional unique identification information, the issuer identification, a serial number, an expiration date, the user's public key, as well as additional technical information and various optional extensions. We show two examples of certificates in Figure 11.1; both examples are taken from `www.wikipedia.org`.

The certification authority itself owns a certificate which is pre-distributed to all participants, and which is considered trusted. Certificates of important CAs are usually included immediately in web browsers and in other relevant software. Note that the CA itself is an offline entity in the sense that it only takes action when issuing a certificate, but it does not participate in the actual distribution and verification of the certificate in the future. Note further that this constitutes an important advantage over key distribution centers for symmetric keys, as these centers have to be online all the time.

The general idea how certificates are used can be summarizes as follows: A user Alice first creates a key pair, presents her public key to the CA, and proves her identity (either using her passport, a notarial document, etc.). If the CA can successfully verify Alice's identity, the CA issues a certificate on her public key and identity, and hands this certificate over to Alice. If Bob wants to send a message to Alice that is encrypted with this public key, Bob and Alice can achieve this as follows:

1. Alice first sends her certificate to Bob. (As it is Bob that would like to send a message to Alice, this step is typically triggered by Bob in practice, i.e., Bob would request the certificate first.)
2. Upon receiving the certificate, Bob verifies the validity of the certificate's signature with respect to the public key of the CA (recall that we assumed so far that the public key of the CA is known to everybody) and checks if the signature is over the correct data, i.e., that the identifying information obtained from the certificate indeed fit to Alice.
3. If this verification succeeds, Bob retrieves the public key of Alice from the certificate, encrypts the desired message using this key, and sends the ciphertext to Alice.

11.1.1 Single Domain Certification Authority

So far we assumed that there exist a single CA whose public key is known to everybody. This assumption turns out to be justified in practice if certification is only performed within a single enterprise; we sketch this scenario in Figure 11.2. While the existence of a single CA is indeed the easiest conceivable case, an authority that is globally trusted by everybody and that verifies the identity of every human seems hardly achievable if we consider global certification and communication over the Internet. These and similar practical concerns can be tackled by the following constructions.

11.1.2 Hierarchical CAs

A possible solution is to structure CAs hierarchically as sketched in Figure 11.3. There is still one central authority, here called *Root-CA*. However, this Root-CA does not certify users but it certifies other certification authorities. These secondary CAs may then either certify users or again other CAs, thereby further expanding the hierarchy. In such an hierarchical scenario, each user needs to own a valid certificate of the public key of the Root-CA. All other certificates are sent along with the user certificates when they are needed, thus enabling the deployment of a large number of secondary CAs without decreasing performance.

Suppose Bob wants to verify the certificate of Alice in the example presented in Figure 11.3. Along with her certificate $cert_{\mathsf{Alice}}$, she sends the certificate $cert_1$ to Bob. Bob then first verifies the validity of the certificate $cert_1$ using the public key of the Root-CA (i.e., using the root certificate he owns). If this verification succeeds, he verifies the validity of the certificate $cert_{\mathsf{Alice}}$ with the key obtained from $cert_1$. If the hierarchy spans more then two layers, this step is applied iteratively.

11.1.3 Cross-Certification

One does not necessarily need a single root-CA for linking two CAs. Instead one can allow several CAs to cross-certify each other. For instance, two CAs cross-certifying each other yield certificates $cert_{12}$ and $cert_{21}$, cf. Figure 11.4.

If Bob wants to verify the validity of the certificate of Alice, Alice first has to determine Bob's CA. Then she sends her certificate $cert_{\mathsf{Alice}}$ along with the right cross-certificate $cert_{12}$ to Bob. Bob then verifies the validity of $cert_{12}$ using the public key from CA_2's root certificate, which he knows, and then verifies the validity of Alice's certificate with the key obtained from $cert_{12}$.

```
Certificate:
   Data:
       Version: 1 (0x0)
       Serial Number: 7829 (0x1e95)
       Signature Algorithm: md5WithRSAEncryption
       Issuer: C=ZA, ST=Western Cape, L=Cape Town,
               O=Thawte Consulting cc,
               OU=Certification Services Division,
               CN=Thawte Server CA/Email=server-certs@thawte.com
       Validity
           Not Before: Jul  9 16:04:02 1998 GMT
           Not After : Jul  9 16:04:02 1999 GMT
       Subject: C=US, ST=Maryland, L=Pasadena, O=Brent Baccala,
                OU=FreeSoft,
                CN=www.freesoft.org/Email=baccala@freesoft.org
       Subject Public Key Info:
           Public Key Algorithm: rsaEncryption
           RSA Public Key: (1024 bit)
               Modulus (1024 bit):
                   00:b4:31:98:0a:c4:bc:62:c1:88:aa:dc:b0:c8:bb:
                   33:35:19:d5:0c:64:b9:3d:41:b2:96:fc:f3:31:e1:
                   66:36:d0:8e:56:12:44:ba:75:eb:e8:1c:9c:5b:66:
                   70:33:52:14:c9:ec:4f:91:51:70:39:de:53:85:17:
                   16:94:6e:ee:f4:d5:6f:d5:ca:b3:47:5e:1b:0c:7b:
                   c5:cc:2b:6b:c1:90:c3:16:31:0d:bf:7a:c7:47:77:
                   8f:a0:21:c7:4c:d0:16:65:00:c1:0f:d7:b8:80:e3:
                   d2:75:6b:c1:ea:9e:5c:5c:ea:7d:c1:a1:10:bc:b8:
                   e8:35:1c:9e:27:52:7e:41:8f
               Exponent: 65537 (0x10001)
   Signature Algorithm: md5WithRSAEncryption
       93:5f:8f:5f:c5:af:bf:0a:ab:a5:6d:fb:24:5f:b6:59:5d:9d:
       92:2e:4a:1b:8b:ac:7d:99:17:5d:cd:19:f6:ad:ef:63:2f:92:
       ab:2f:4b:cf:0a:13:90:ee:2c:0e:43:03:be:f6:ea:8e:9c:67:
       d0:a2:40:03:f7:ef:6a:15:09:79:a9:46:ed:b7:16:1b:41:72:
       0d:19:aa:ad:dd:9a:df:ab:97:50:65:f5:5e:85:a6:ef:19:d1:
       5a:de:9d:ea:63:cd:cb:cc:6d:5d:01:85:b5:6d:c8:f3:d9:f7:
       8f:0e:fc:ba:1f:34:e9:96:6e:6c:cf:f2:ef:9b:bf:de:b5:22:
       68:9f
```

```
Certificate:
   Data:
       Version: 3 (0x2)
       Serial Number: 1 (0x1)
       Signature Algorithm: md5WithRSAEncryption
       Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
               OU=Certification Services Division,
               CN=Thawte Server CA/Email=server-certs@thawte.com
       Validity
           Not Before: Aug  1 00:00:00 1996 GMT
           Not After : Dec 31 23:59:59 2020 GMT
       Subject: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
                OU=Certification Services Division,
                CN=Thawte Server CA/Email=server-certs@thawte.com
       Subject Public Key Info:
           Public Key Algorithm: rsaEncryption
           RSA Public Key: (1024 bit)
               Modulus (1024 bit):
                   00:d3:a4:50:6e:c8:ff:56:6b:e6:cf:5d:b6:ea:0c:
                   68:75:47:a2:aa:c2:da:84:25:fc:a8:f4:47:51:da:
                   85:b5:20:74:94:86:1e:0f:75:c9:e9:08:61:f5:06:
                   6d:30:6e:15:19:02:e9:52:c0:62:db:4d:99:9e:e2:
                   6a:0c:44:38:cd:fe:be:e3:64:09:70:c5:fe:b1:6b:
                   29:b6:2f:49:c8:3b:d4:27:04:25:10:97:2f:e7:90:
                   6d:c0:28:42:99:d7:4c:43:de:c3:f5:21:6d:54:9f:
                   5d:c3:58:e1:c0:e4:d9:5b:b0:b8:dc:b4:7b:df:36:
                   3a:c2:b5:66:22:12:d6:87:0d
               Exponent: 65537 (0x10001)
       X509v3 extensions:
           X509v3 Basic Constraints: critical
               CA:TRUE
   Signature Algorithm: md5WithRSAEncryption
       07:fa:4c:69:5c:fb:95:cc:46:ee:85:83:4d:21:30:8e:ca:d9:
       a8:6f:49:1a:e6:da:51:e3:60:70:6c:84:61:11:a1:1a:c8:48:
       3e:59:43:7d:4f:95:3d:a1:8b:b7:0b:62:98:7a:75:8a:dd:88:
       4e:4e:9e:40:db:a8:cc:32:74:b9:6f:0d:c6:e3:b3:44:0b:d9:
       8a:6f:9a:29:9b:99:18:28:3b:d1:e3:40:28:9a:5a:3c:d5:b5:
       e7:20:1b:8b:ca:a4:ab:8d:e9:51:d9:e2:4c:2c:59:a9:da:b9:
       b2:75:1b:f6:42:f2:ef:c7:f2:18:f9:89:bc:a3:ff:8a:23:2e:
       70:47
```

Figure 11.1: A Certificate and the Corresponding Root Certificate in Informal Notation

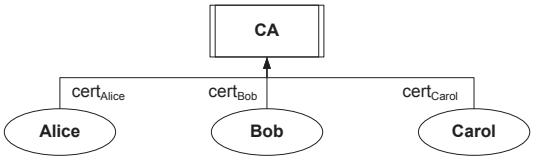


Figure 11.2: Single Domain CA

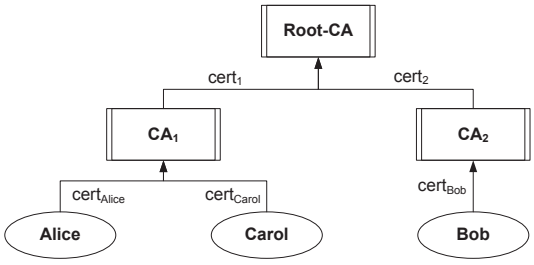


Figure 11.3: Hierarchical CAs

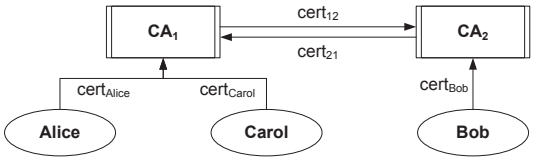


Figure 11.4: Cross-certification

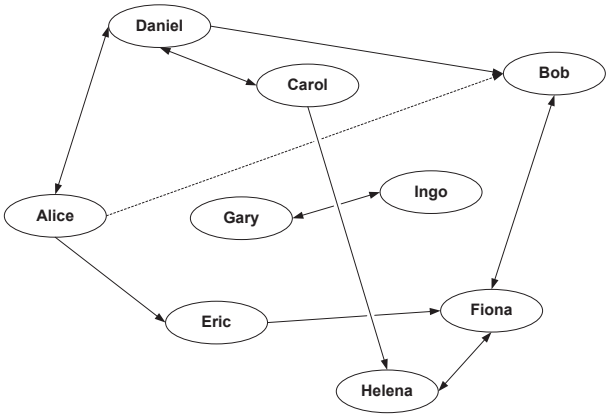


Figure 11.5: An Example Web of Trust

11.1.4 Web of Trust (PGP)

This idea can be further developed to yield a so-called *web of trust* as follows: Each user typically trusts several other users, say friends or colleagues. These in turn trust other users, and so on. Now one can hope (and in practice this seems to work pretty well) that if Bob can identify one or even multiple paths between Alice and himself, the public key of Alice can be trusted. This is illustrated in Figure 11.5, where a (solid) arrow from a node A to a node B means that B certified the public key of A. Suppose Bob wants to verify Alice's certificate. There exist two paths from Alice to Bob:

$$\text{Alice} \to \text{Daniel} \to \text{Bob},$$

and

$$\text{Alice} \to \text{Eric} \to \text{Fiona} \to \text{Bob}.$$

Consider the first path. Bob trusts Daniel who himself certified Alice's key. Then Bob verifies the certificate $cert_{\mathsf{Daniel}\to\mathsf{Alice}}$ with Daniel's public key. This gives him Alice's public key. To counter the case that one party in such a path is corrupted, one requires to have at least two (or better even multiple) different paths to establish trust in a user's public key.

11.1.5 Maurer's Scheme

Maurer's scheme provides a general model for determining "whom we trust". For every user B consider the following four predicates:

- $\mathsf{Aut}(X)$: B believes that the public key of X is authentic,
- $\mathsf{Cert}(X, Y)$: B has a certificate issued by X for the public key of Y,
- $\mathsf{Trust}(X, 1)$ (trust of level 1): B trusts X to correctly verify every user's identity before issuing certificates for them,
- $\mathsf{Rec}(X, Y, 1)$ (recommendation of level 1): B has an (attribute-) certificate where X says that he/she trusts Y to correctly verify every user's identity before issuing certificates for them.

Inductively, one defines predicates for higher levels of trust:

- $\mathsf{Trust}(X, i)$ (trust of level i): B trusts X to correctly verify recommendations of level $i-1$,
- $\mathsf{Rec}(X, Y, i)$ (recommendation of level i): the corresponding (attribute-) certificate.

Initially a user knows public keys for some users X ($\mathsf{Aut}(X) = \mathsf{true}$), he knows some certificates ($\mathsf{Cert}(X, Y) = \mathsf{true}$), he trusts some users X for some level i ($\mathsf{Trust}(X, i) = \mathsf{true}$) and got some recommendations for some level i ($\mathsf{Rec}(X, Y, i) = \mathsf{true}$). With the following rules he can deduce further trust:

- $\mathsf{Aut}(X) \wedge \mathsf{Trust}(X, 1) \wedge \mathsf{Cert}(X, Y) \rightarrow \mathsf{Aut}(Y)$
- $\mathsf{Aut}(X) \wedge \mathsf{Trust}(X, i+1) \wedge \mathsf{Rec}(X, Y, i) \rightarrow \mathsf{Trust}(Y, i)$.

One says that B trusts the keys of user Q if he can deduce $\mathsf{Aut}(Q)$ with the above rules from the initially given set of predicates. (However, this presupposes that trust is indeed transitive which is questionable.)

11.1.6 Certificate Revocation

Eventually it may happen that a secret key gets compromised. We then need a mechanism to invalidate the corresponding certificates; one says that these certificates are *revoked*. Such a revocation can be seen as a statement, signed by the CA, that certain explicitly specified certificates should be considered invalid. This statement needs to be signed, as otherwise an attacker could easily revoke any certificate, resulting in a denial-of-service attack. A crucial point is how these revocation statements reach the users, which should be prevented from accepting revoked certificates.

Certificate Revocation Lists (CRL) The conceptually simplest method is to publish a list of all revoked certificates, a so-called *certificate revocation list* (CRL). This list is signed by the CA and contains serial numbers of all certificates that are revoked. The main problem with this approach is that these lists can get very long. In practice, this is circumvented by using so-called *incremental CRLs*, where only new revocations are added.

Online Certificate Status Protocol (OCSP) A suitable way to circumvent the problem of having long CRLs is to offer an online verification service for certificates, i.e., an *online verification authority* (VA) is provided that checks, upon a user's request, if a given certificate was revoked or not. Provided that the number of VAs is small, they can be synchronized in little time. The main problem with this solution is the high load of the VAs, and as all these VAs have to be completely trustworthy, replicating VAs in a significantly large number to even out the load is hardly possible.

Certificate Revocation Trees (CRT) A nice remedy to this problem is the following one: We modify the scheme such that there is only one Root-VA, which everybody has to trust, and a large number of VAs that do not have to be trusted. Consequently, one may have a large number of VAs, which results in a scalable solution. However, we somehow have to guarantee that a malicious VA cannot trick an honest user into accepting revoked certificates.

This can be achieved as follows. Remember that certificates have a serial number, which is used to address the certificate. Let $C_1, \ldots, C_4$ be the serial numbers of the revoked certificates, ordered such that $C_i < C_{i+1}$. The Root-CA builds a tree as depicted in Figure 11.6, i.e., it hashes C_i obtaining H_i, and then constructs a hash tree yielding a top node H_7. Finally, the Root-VA signs the value H_7 of the root node and the depth of the tree yielding a signature $\mathsf{S}(sk_{\mathsf{Root\text{-}VA}}, (\text{tree-depth}, H_7))$. Then the following data is sent to the VAs, including the potentially malicious ones:

$$CRT = (C_1, C_2, C_3, C_4, \mathsf{S}(sk_{\mathsf{Root\text{-}VA}}, (\text{tree-depth}, H_7))).$$

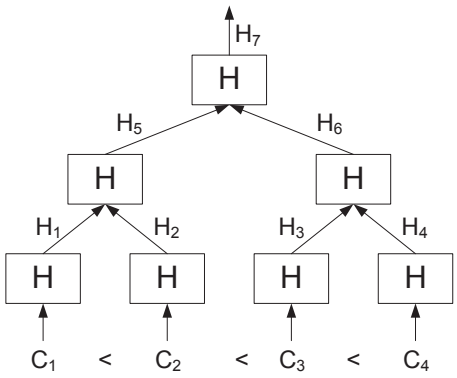


Figure 11.6: CRT

Now assume that a user queries a VA for a certificate with serial number C. Let us first consider the case that the certificate was revoked, for example $C = C_2$. Then the VA proves this as follows. Let us first note that given $C_1, \ldots, C_4$, the VA can recompute the entire tree of hash values. The VA sends $(\mathsf{S}(sk_{\text{Root-VA}}, (\text{tree-depth}, H_7)), H_1, H_6)$ to the user, which is enough information for the user to recompute the root node, which he can then compare to the root node contained in the signature of the Root-VA.

If a user asks for a certificate with serial number C which was not revoked, then the VA proves this as follows. It identifies the largest serial number which is smaller than C, and the smallest serial number which is larger than C, e.g., $C_1 < C < C_2$. It sends $(\mathsf{S}(sk_{\text{Root-VA}}, (\text{tree-depth}, H_7)), C_1, C_2, H_6)$ to the user. Again the user can recompute the root node of the hash tree and test if it equals the value H_7 in the signature of the Root-VA.

However, if the VA tries to cheat, e.g., it tries to prove that the certificate was not revoked although it is contained in the CRT, the VA had to construct a different hash tree with root element H_7. One can easily prove that such an adversary could be used to break the collision-resistance of the hash function. The proof is quite similar to the proof of Merkle Hash Trees and is left as an exercise.

12. Authentication Methods, SSL, and other Important Protocols

This chapter investigates various methods for letting a user authenticate itself to a server. Furthermore we will discuss several important protocols that are frequently used in practice, especially concentrating on SSL/TLS.

12.1 Authentication

Authentication is one of the crucial aspects of computer security, especially since it is used very often in daily life, e.g., one authenticates when logging into a computer, when checking emails, when doing online banking, and so on.

Let us first note that authentication is typically complemented with a key exchange protocol as otherwise an attacker can simply hijack a session, i.e., take over your session after your authentication to the server has been completed, e.g., by cutting off your connection and taking over your IP.

12.1.1 Password Authentication

The probably most common method used for authentication are passwords. They are easy to handle, easy to carry with you, and the principle is easy to understand. On the other hand, they often do not provide sufficiently strong security guarantees: common passwords in practice tend to have low entropy ("`password`" as a password is much more common than "`k34-hN(a1W*`"), and some protocols even further decrease entropy by not allowing special characters, by truncating the password, or by even converting the whole password to uppercase letters.

The most intuitive way to realize password authentication is to store the password on the server's harddisk and to test if the password provided by the user matches the stored password. While this solution seems natural, it bears the risk that an attacker that successfully gains access to the server's harddisk can simply read the password and use it for later authenticating himself as the user.

Instead of storing the password in clear, servers typically store a list containing entries of the form $(Alice, \mathsf{H}(P_{Alice}))$, where P_{Alice} is Alice's password and H is a one-way (hash) function. Then a password presented by a user P is verified by first applying H and then comparing $\mathsf{H}(P)$ with the stored hash of the password password. Thus even if an attackers gets access to the password file, it still needs to find a password P that maps to a fixed value $\mathsf{H}(P_{Alice})$ under H, thus an attacker needs to break one-wayness of the function H.

Early versions of Windows (before Windows 2000 SP3) used a password verification called LANMAN. It accepted 14 bytes for each password P, converted these bytes into uppercase letters, then split them into two halves of equal size $P = P_1 \parallel P_2$, and used both halves as DES keys to encrypt a fixed message IV. This method had various weaknesses: Most importantly, converting all letters to upper-case significantly reduces the passwords entropy, thus paving the way to successful dictionary attacks. Additionally, truncating passwords to 14 bytes is not good, but much worse

is splitting the password into two independent halves. Thus an attacker can attack both halves independently, which makes successful dictionary or brute-force attacks much easier to mount. Nowadays, Windows relies on MD4 hashes, which constitutes a much better primitive for storing password compared to LANMAN.

Early versions of Linux allowed passwords of 7 bytes only (but it additionally uses salting to protect against dictionary attacks, see below), and uses the password as key for several consecutive DES encryptions of a fixed message IV.

Dictionary attacks A generic attack against hashed passwords stored in a file are so-called *dictionary attacks*. The attack does not depend on the concrete hash function used, as long as the function is publicly known. The adversary starts with a dictionary of common words $W_1, W_2, \ldots$ (such dictionary are freely available on the net) and simply hashes each of them, i.e., he computes $\mathsf{H}(W_1), \mathsf{H}(W_2), \ldots$, until he found a hash H which is stored in the file. If this does not succeed then he tries to modify these words as a user might have done, i.e., by appending or inserting numbers and special characters, by reversing the order, and so on.

This is of course a heuristic attack, which however turns out to be successful in practice if an attacker is satisfied with finding *any* password stored in the file (in practice even most passwords, depending on the sophistication of the users to choose passwords, of course).

This attack can be made much more difficult to mount by *salting* the passwords: instead of hashing the password P_A, one picks a random $salt_A \in \{0,1\}^l$ for every user A (a common choice is $l = 12$), and stores

$$Alice, salt_A, \mathsf{H}(salt_A \parallel P_A)$$

in the file. Verifying the correctness of a given password can easily be done, as the salt is known to the server, but an attacker cannot attack a large number of passwords simultaneously, as they use different salts. So he has to attack each password individually, resulting in a much longer computation.

The idea of adding some form of uncertainty to the hash value can be further extended by using a so-called *secret salt* or *pepper*. One chooses $salt_A \in \{0,1\}^l$ and $salt_A^* \in \{0,1\}^k$ (a common choice is $k = 8$), and stores

$$Alice, salt_A, \mathsf{H}(salt_A \parallel salt_A^* \parallel P_A).$$

To verify a password the server has to try all 2^k values for $salt_A^*$. If the server is given a correct password, this only increases the computation time from micro-seconds to milli-seconds, which is usually doable. The attacker's time, however, is as well increased by a factor of 2^k; however this is typically is from, say, one day to 2^k days.

12.1.2 Biometric Passwords

Biometric authentication methods are a pretty novel research and application area. One can for example use finger prints, iris scans, or speech-recognition to identify a person. Using biometrics as passwords however suffers from two main problems: First, the biometric passwords are not secret, e.g., copies of a person's fingerprints turn out to be easily achievable in practice. Secondly, there is no possibility to revoke a copied fingerprint (except by using surgery of course).

12.1.3 One-time Passwords

The following authentication scheme, which is due to Lamport, exploits so-called one-time passwords. Alice generates a password P_A, and computes $W_A := \mathsf{H}^n(P_A) := \mathsf{H}(\mathsf{H}(\ldots(\mathsf{H}(P_A))\ldots))$ for some one-way function H. The server stores W_A, Alice stores P_A and sets $cnt := n$. If Alice wants to authenticate to the server, she decreases $cnt := cnt - 1$, sends $A := \mathsf{H}^{cnt}(P_A)$ to the server. The server verifies that $\mathsf{H}(A) = W_A$, and if so, sets $W_A := A$. This scheme in particular avoids eavesdropping attacks since merely listening on the channel will only reveal passwords that would have been accepted by the server before he changed its state and hence the value of W_A. Moreover, breaking into the server does not weaken the scheme since no secrets are stored on the server. The drawback however is that Alice can only authenticate herself a limited number of times, namely n times. After that, the chain originating at her password has been used up so that she has to generate a new passwords, hash it n times, and let the server know the new value W_A.

Another system relying on one-time passwords is SecureID, where the client owns a smartcard. Here the server and the smartcard start with the same random value K_0, and every minute, they compute a new value $K_{i+1} := \mathsf{H}(K_i)$. If Alice would like to authenticate to the server, the smartcard outputs $\mathsf{DES}(K_i, ct)$, where ct denotes the current time. While this system allows for unlimited password usage, the drawback is that the server is now required to store a secret; moreover, the smartcard has to be tamper-resistant as otherwise stealing the smartcard, copying its memory and handing it back unnotedly would cause a severe threat to this kind of system. Note that the protocol is secure against eavesdropping attacks as well.

12.1.4 Challenge-Response Protocols

Another common authentication method is to use challenge-response protocols.

Based on Symmetric keys Alice and the server share a common key K. The authentication procedure constitutes a three-round protocol: Alice initiates the protocol by sending a hello message to the server. The server picks a random r – the challenge – and sends it to Alice. Alice picks a random r' and sends $\mathsf{E}(K,\ r \parallel r')$ to the server. The server decrypts this encryption, and if the first component matches the previously sent challenge r, it considers Alice authenticated. This protocol resists eavesdropping attacks but requires the server to stores secrets.

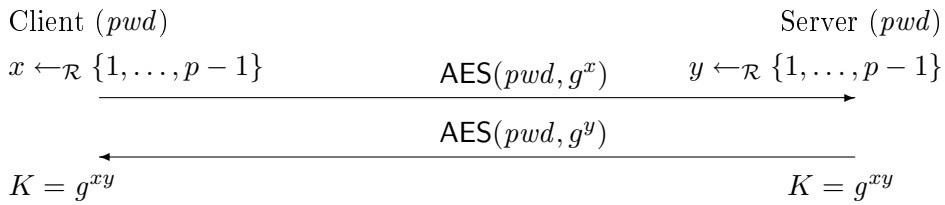
Based on Public-key Cryptography The protocol works exactly as the one before, but together with the hello message, Alice also sends her certificate for her public signature key. Later instead of encryption the message with a secret key, she signs the message with her secret signature key. The server then only tests the validity of the signature, which frees the server from storing secret information.

The Station-to-Station Protocol (STS) The STS protocol constitutes a widely known protocol for authentication and key exchange. It relies on a large prime p and a generator g of $\mathbb{Z}_p^*$ as publicly known parameters. Alice selects signature keys pk_A, sk_A, and obtains a certificate $cert_A$ for pk. Alice and Bob pick $x, y \in \{1, \ldots, p-1\}$, respectively. Then Alice sends $cert_A, g^x \bmod p$ to Bob, which in turn sends $cert_S, g^y, \mathsf{E}(K, \mathsf{S}(sk_S, (g^x, g^y)))$ where $K := g^{xy}$. Alice answers with $\mathsf{E}(K, \mathsf{S}(sk_A, (g^x, g^y)))$. K can be computed by both parties and is the shared key. Based on this

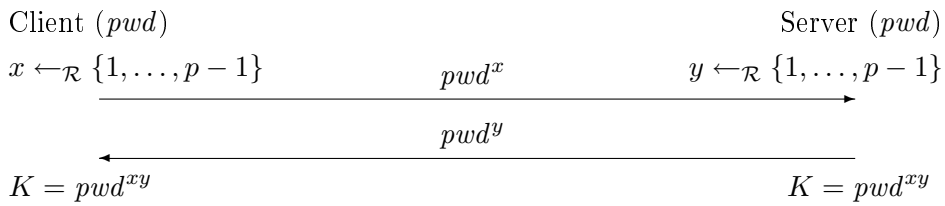
shared key, Alice and Bob can subsequently authenticate themselves to each other, e.g., by running the aforementioned challenge-response protocol.

12.1.5 Encrypted Key Exchange (EKE)

The encrypted key exchange protocol (EKE) works as follows. Both Alice and the server share a password pwd. They choose $x, y \in \{1, \ldots, p-1\}$, respectively. Alice sends $h := \mathsf{AES}(pwd, g^x)$ to the server, which in turn sends $i := \mathsf{AES}(pwd, g^y)$ to Alice. Then both compute $K := g^{xy}$ as usual. Then Alice sends r to the server, and the server answers with $\mathsf{E}(K, r \parallel r')$ as for the challenge-response protocol based on symmetric keys.



Jablon proposes a variation of EKE in 1996 which works as follows. Let pwd be the password, and let Alice and the server choose $x, y \leftarrow_{\mathcal{R}} \{1, \ldots, p-1\}$ respectively, Alice sends pwd^x to the server which answers with the message pwd^y. Then both share the key pwd^{xy}.



12.2 Protocols

Next we will see in some detail how authentication is done in practice.

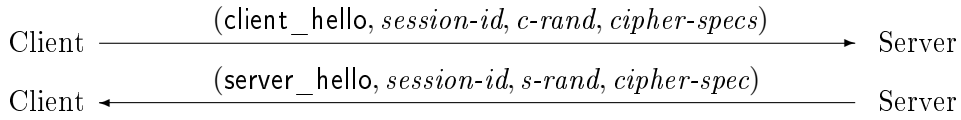
12.2.1 SSL/TLS

SSL/TLS is a protocol for providing secure (i.e., secret and authenticated) communications over the Internet. SSL 1.0 was first proposed in 1994 by Netscape Communications and subsequently improved. In 1999 the current version 3.0 was renamed (with some very small changes) to TLS 1.0, causing some confusion with names and versions. In the following, we use the term SSL, unless stated otherwise.

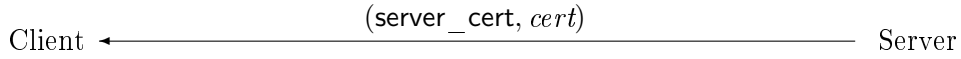
The SSL protocol first exchanges a symmetric encryption key using slow public key operations, then the payload data is encrypted using the faster symmetric encryption with the key they both parties previously agreed on. Depending on the protocol needs, SSL can provide single-sided authentication or mutual authentication. This is achieved using certificates; thus authenticated parties are required to maintain certificates. SSL supports, amongst other algorithms, PKCS#1 public-key encryption and digital signatures, Diffie-Hellman key exchange, the symmetric ciphers AES and 3DES, and the hash function SHA-1. It also supports primitives that are nowadays considered outdated such as (single) DES and MD5, but one can (and should) avoid using them. The SSL protocol has three phases:

1. Negotiation of supported algorithms,
2. Exchange of a session key by means of public-key encryption and authentication based on certificates, and
3. Payload encryption based on a symmetric cipher and the session key.

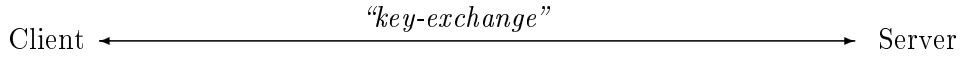
Overall Protocol A client starts a session by sending a client_hello message including a random string *c-rand* of length 28 bits, a session id, and specifying the cryptographic primitives it supports in *cipher-specs*. The server responds with a server_hello message, including another random string *s-rand*, the session id, and specifying the primitives that will be used in the sequel from the ones specified by the client.



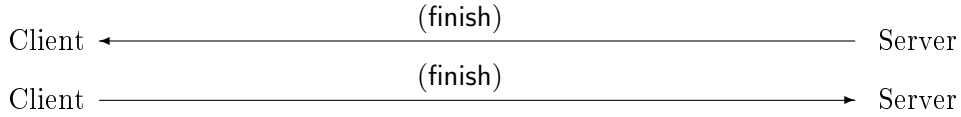
To be able to verify a certificate the parties need to support the cryptographic primitives that where used, e.g., RSA signatures or DSA signatures. Thus, it might be necessary to have a variety of certificates using different primitives. Now that both parties agreed on the parameters, they exchange a suitable certificate. For most applications it is sufficient that only the server authenticates to the user, in which case only the server sends its certificate to the user. Currently all certificates follow the X.509 standard, but there exists a draft for using OpenPGP-based certificates.



In the next step the key-exchange takes place. There are different options how to exchange a key; two of them we will discuss later. In either case, after finishing this step both parties share a so-called *pre-master secret* (PMS), a 40 bit secret key which is then used to derive the master key.

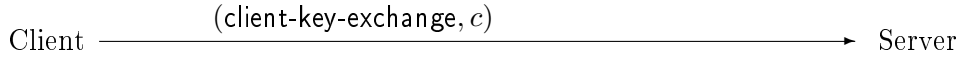


This pre-master secret is then used to derive a so-called master-secret as $\mathsf{H}(PMS \parallel c\text{-}rand \parallel s\text{-}rand)$. This master-secret is then used to derive the (symmetric) encryption and authentication keys as needed. Finally, both parties acknowledge the successful key-exchange.



Key Exchange I: Encrypted Secret A very simple variant for realizing the key exchange is the following: the client chooses a random bit string of 48 bits, encrypts it with the servers public key (which is known from the certificate), and sends the resulting ciphertext to the server.

This is reasonable fast, as only one public-key operation needs to be performed by each party. However, it does not provide forward secrecy, i.e., if at some point the servers key is compromised, an attacker can use it to obtain all past session keys and thus to read any communication of the server with any client. This is of course not optimal, and there are better ways to do it.



Key Exchange II: Ephemeral Diffie-Hellman (EDH) The following key-exchange provides forward secrecy, i.e., if at some time the servers key gets compromised, then an adversary can only read future communication using this key. Hence, if the attack was noticed, one can simply avoid using this key and almost no harm is done. This improvement, however, comes along with a running time which is three times slower than the previous method.

First, the server chooses a random 1024 bit prime p and a generator g of $\mathbb{Z}_p^*$. It furthermore chooses $a \leftarrow_{\mathcal{R}} \{1, \ldots, p-1\}$, and computes $z_1 = g^a \bmod p$. Then it signs (p, g, z_1) yielding sig, and sends this to the client. The client in turn verifies the signature sig, picks a random $b \in \{1, \ldots, p-1\}$, computes $z_2 = z_1^b \bmod p$, and sends it back to the server

$$\text{Client} \xleftarrow{(\mathsf{server\text{-}key\text{-}exchange}, p, g, z_1, sig)} \text{Server}$$

$$\text{Client} \xrightarrow{(\mathsf{client\text{-}key\text{-}exchange}, z_2)} \text{Server}$$

Then, both the client and the server can compute the pre-master secret $PMS := g^{ab}$. This is almost a Diffie-Hellman key exchange, but the server's signature prevents an attacker from doing a man-in-the-middle attack, as he cannot sign the value z_1.

Improving Performance: SSL Resume A nice feature of SSL is the so-called *SSL resume*, which improves performance by avoiding unnecessary key-exchange operations. If a client and a server have communicated before using the session id *CSID*, then the client simply sends a *client-hello* message with this session id:

$$\text{Client} \xrightarrow{(\mathsf{client\_hello}, \mathit{CSID}, \mathit{c\text{-}rand}, \mathit{cipher\text{-}specs})} \text{Server}$$

If the server can find this session in his session cache, then he answers with the same *CSID*, otherwise with a fresh session id.

$$\text{Client} \xleftarrow{(\mathsf{server\_hello}, \mathit{CSID}, \mathit{s\text{-}rand}, \mathit{cipher\text{-}spec})} \text{Server}$$

$$\text{Client} \xleftarrow{(\mathsf{server\_hello}, \mathit{new\text{-}session\text{-}id}, \mathit{s\text{-}rand}, \mathit{cipher\text{-}spec})} \text{Server}$$

If resume was successful, both the user and the server reuse their previous PMS and send an acknowledge message.

$$\text{Client} \xleftarrow{(\mathsf{finish})} \text{Server}$$

$$\text{Client} \xrightarrow{(\mathsf{finish})} \text{Server}$$

The master-secret, which is then used to actually derive keys from, is generated as $\mathsf{H}(PMS \parallel \mathit{c\text{-}rand} \parallel \mathit{s\text{-}rand})$.

SSL in Practice Let us briefly talk about networking problems of SSL that occur in practice. Commonly, big web sites are designed as follows: First, a server side cache tries to answer common queries to static data. Then a load balancer balances traffic between different web servers; the web servers are finally connecting to a database.

First of all, for using SSL one needs to install the keys on all web servers, which is not optimal from a security point of view, as this increases the risk of key exposure. Furthermore, the prevalent choice of RSA encryption is not optimal, as the web servers have to perform RSA decryption operations that are more time-consuming than RSA encryption operations. Currently, there are strong efforts in speeding up SSL, e.g., by using specialized hardware etc.

Another point is that caching as described is not possible if the content is encrypted, as the cache cannot see what is inside the packets. This, however, cannot be avoided with encrypted content.

One more problem occurs for web servers that host different domains at the same IP address. In this setting, the web server distinguishes the domains by the HTTP headers. However, with SSL the user and server exchange certificates before any HTTP request was made, i.e., the server cannot know the certificate for which domain it should provide. This was fixed with TLS 1.1 by including the host address in the client_hello message.

Another problem arises when encrypted communication passes a proxies server. Here get-requests are first sent to a proxy, which forwards them to the server. If SSL is turned on, the proxy does not know where to connect to, as the server's name is encrypted. For this reason HTTPS permits an additional first message which contains the target hostname as parameter; then normal SSL messages are exchanged.

User Authentication If user authentication is desired, the protocol slightly deviates from what was stated above: In the server-hello message the server includes a directive cert-req and a list of accepted certificates.

$$\text{Client} \xleftarrow{(\mathsf{server\_hello}, \ldots, \mathsf{cert\text{-}req},\ \textit{list of accepted CAs})} \text{Server}$$

Then the client sends the appropriate certificate to the server.

$$\text{Client} \xrightarrow{(\mathsf{client\_cert}, \textit{cert})} \text{Server}$$

Then, after the remaining protocol, the client signs all protocol data with his secret key and sending it to the server, which verifies the signature.

$$\text{Client} \xrightarrow{(\mathsf{cert\_verify}, \mathsf{S}(sk, \textit{All protocol data}))} \text{Server}$$

12.2.2 Kerberos

Kerberos is a well-known system for symmetric key exchange over a central authority. It originates from the "Athena" project at MIT for secure distributed systems, from around 1987, and is more or less deprecated nowadays. It is the name for a protocol standard as well as for a library of commands. It uses the term "tickets" for server messages, that can be used to derive session keys for the users.

12.2.3 Pretty Good Privacy (PGP)

Probably the best-known program for encrypting mails and files in the private sector, as it is free-of-charge for private use. It has various applications for encrypting emails and files, and also provides transparent encryption of the file system (PGP-Disk).

It was published first in 1991 by Phil Zimmermann. At that time strict US export regulations prevented the export of cryptography with keys larger than 40 bits. To circumvent this restriction he printed a book which contained the whole source-code of the program, and distributed this book around the world, which surprisingly was legal. Then other people scanned in the source code and re-compiled the program. Fortunately, nowadays the export regulations are gone, and the program is available freely. PGP can be used with a variety of algorithms, including RSA and ElGamal for asymmetric encryption, RSA and DSA digital signatures, symmetric encryption algorithms IDEA and 3DES, and hash functions MD5, SHA-1, and SHA-256.

12.2.4 IPSEC (for IPv6)

IPSEC is a cryptographic protocol for securing IP communication and key exchange. It operates on the network layer, i.e., lower than the aforementioned protocols, so it is more generally usable. Even the usual headers can be authenticated. However, header fields such as hop-count may change when a packet is sent trough the Internet; these fields are set to 0 before headers are authenticated. Two modes are supported by IPSEC: *Transport mode*, where only payload data is encrypted, and *tunnel mode*, where the whole package is encrypted, even the header, and new headers are added.

12.2.5 SSH ("Secure Shell")

The first version of SSH, also called SSH-1, was designed in 1995 by Tatu Ylönen from the University of Helsinki as a response to a password-sniffing attack at his university. It provides a replacement for protocols such as telnet and rlogin, which transmit passwords in cleartext and which are thus highly vulnerable to sniffing attacks. In 1996 SSH-1 was improved significantly, yielding the currently used version SSH-2. Some of these changes, however, caused both versions to be incompatible. However, given that successful attacks against SSH-1 have been discovered, this version should not be used any more anyway. While SSH is known best for its secure shell, it also provides other tools such as SCP as a replacement for the insecure "remote copy" (rcp) command, and options for tunneling, e.g., X11 sessions.

13. Commitment Schemes

Commitment schemes constitute a core cryptographic primitive for building larger protocols. They consist of two subprotocols, a *commit* protocol and an *open* protocol: In the commit protocol, one participant commits to a value, i.e., it fixes it so that it cannot be changed later, while still keeping the value hidden from the other participants. In the open protocol, the value is shown to the other participants. Typically, there is only one other participant, i.e., both commit and open constitute two-party protocols. In addition to the obvious correctness property, commitment schemes should satisfy two security properties: They should be *binding*, i.e., the committer cannot change the value after the commit protocol, and *hiding*, i.e., the recipient does not learn anything about the value during the commit protocol.

A common non-digital example of commitments is the following: Alice writes something on a piece of paper (without allowing Bob to see what is written), puts the paper inside an envelope and puts the envelope on the table, which completes the commit protocol. Now she is bound to the written value, but Bob cannot look inside the envelope. Later they can open the envelope and verify the value. A special case of commitment schemes are bit commitment schemes which only allow for committing to one bit. Indeed the early constructions only worked bit by bit; now, however, many schemes are known for committing to longer messages more efficiently than by proceeding bit by bit; this is sometimes called string commitment.

13.1 Definition of Commitment Schemes

Commitment schemes are of the following structure: The parameters are a message space $\mathcal{M}$ and a security parameter n. There are two roles: a so-called *committer* and a *recipient*. There are two subprotocols: The *commit* protocol, where the committer program obtains one input $m \in \mathcal{M}$, the message to be committed to. The recipient program obtains no input, except possibly for a start signal. Neither program makes an output except for a value $acc \in \{ok, \downarrow\}$ indicating whether the commit protocol terminated correctly. (But they store some values for the later opening protocol.) In the *open* protocol the same committer and recipient as in a previous commit protocol take part. The programs do not need inputs, but the recipient program will output either $(accept, m^*)$ for a message $m^* \in \mathcal{M}$ or *reject*. This means that it either accepts that the commitment contained the message m^*, or decides that the committer did not open the commitment correctly. There are three requirements. The first one is simply that everything should work in the faultless case; the other two are the real security requirements.

(a) **Correctness:** If both the committer and the recipient are honest and execute commit and open in succession, then $m^* = m$. More precisely, both programs output $acc = ok$ in the commit protocol, and the output of the recipient program in open is $(accept, m)$, where m is the message that was input to the committer program in the commit protocol.

(b) **Binding property** (or "committing property"): Even if the committer is dishonest, he cannot open one commitment in two different ways. More formally, the notions of "can open" can be represented as follows: One first executes the commit protocol with the cheating committer and the correct recipient. If the recipient program outputs $acc = ok$, the opening protocol is executed twice, both times starting from the state after commit. The cheating committer gets an additional bit $b = 0$ the first time and $b = 1$ the other time to distinguish how it should open. We say that the cheating committer was successful in a particular run if the recipient makes outputs $(accept, m)$ and $(accept, m')$ for two different messages $m \neq m'$ in the two executions of *open*.

(c) **Hiding property** (or "secrecy"): The commit protocol should not give the recipient any information about the message m. This is further formalized in the same way as for an encryption scheme.

The binding and hiding property can each be required with either information-theoretic or computational security, i.e., for computationally unbounded adversaries, or for adversaries that are constraint to running in probabilistic polynomial-time in the security parameter. The binding property is a requirement of the recipient, the hiding property a requirement of the committer. Consequently, the recipient is assumed to be honest in (b), and the committer is assumed to be honest in (c), whereas the other party may potentially deviate from the protocol to gain an advantage.

Some Special Cases and Notation

- Some constructions additionally have a subprotocol *initialization* that can be executed once for many commitments to improve the efficiency of the actual commit protocol.
- A commitment scheme is said to be with *non-interactive opening* if only one message is sent in the open protocol (from the committer to the recipient).
- A scheme is said to be with *public verification* if the final decision of the recipient whether he accepts a message, and if yes which, only depends on the traffic, i.e., the messages sent and received, and not on internal secrets. In this case, other parties who saw the traffic can also decide whether the recipient should accept.
- A commitment scheme is called *non-interactive* if committing is non-interactive in the same sense as opening. The one message sent in commit is then called the commitment.

  In this case a committer can, e.g., make a commitment and broadcast it, so that he is committed with respect to many recipients.

Almost all well-known commitment schemes are with non-interactive opening and public verification. Non-interactive committing is not so prevalent, but many people nevertheless tend to think primarily of non-interactive schemes.

13.2 An Impossibility Result

All the following protocols are either information-theoretically binding or information-theoretically hiding, but not both. It is not hard to see that one cannot construct commitment schemes that satisfy both binding and hiding in an information-theoretic manner.

Theorem 13.1 *No protocol can fulfill both hiding and binding in an information-theoretic manner.* □

Proof. (Sketch) The intuitive reason is that after an execution of commit, there are either two possibilities to open the commitment, then it is not information-theoretically binding, or only one, then it is not information-theoretically hiding. In the non-interactive case, this intuitive argument can simply be formalized by considering the messages that the committer can send in open. In the general case, "the commitment" is modeled by the traffic in the commitment protocol. We only need to consider resulting values *traffic* from correct executions of the commit protocol. Information-theoretic secrecy of the message m implies in particular that it should be impossible to guess m significantly better than with a-priori probability from *traffic* (because *traffic* is part of the recipient's view). In other words, this means that any message m' could lead to the same traffic in an execution of the commit protocol, and the probability for this is equal for all messages. Now we consider a cheating committer exploiting this fact: He can first execute the commit protocol correctly with one message m, then choose another message m' and find an execution that is consistent with both m' and the traffic from the first execution. By executing open starting with his final state from either the m-execution or the m'-execution, he can then (by the correctness) get the recipient to accept either m or m'. ■

13.3 Information-Theoretically Binding Schemes

We will see three well-known commitment schemes in the following, two information-theoretically binding ones in this section and an information-theoretically hiding one in the next section.

13.3.1 Commitment Schemes from Asymmetric Encryption

Any CPA-secure asymmetric encryption scheme can be used to build a commitment scheme in the following way. We assume that the message space of the encryption scheme comprises the desired one of the commitment scheme. Note that for this scheme, the random bits used by the key generation algorithm Gen and by the encryption algorithm E are important, thus we equip these functions with an additional argument representing this random string. Note that then the encryption scheme is secure only if the randomness was chosen uniformly and if it does not become known to the adversary.

- In *commit*, the committer generates a key pair

  $$(sk, pk) := \mathsf{Gen}(n; r_1),$$

  where Gen is the key generation algorithm of the given encryption scheme, n the same security parameter as in the commitment scheme, and r_1 a random string. He encrypts his message m into a ciphertext

  $$c := \mathsf{E}(pk, m; r_2),$$

  using a second random string r_2, and sends

  $$com := (pk, c)$$

  to the recipient.

- In *open*, the committer sends
$$(m, r_1, r_2)$$
to the recipient. The recipient regenerates the key pair (sk, pk) using r_1 and recomputes the ciphertext $c := \mathsf{E}(pk, m; r_2)$. He verifies that his result (pk, c) equals the commitment com, and that m belongs to the message space.

Now we show that this scheme fulfills the three requirements.[1]

Theorem 13.2 *The commitment scheme just described is information-theoretically binding, and it is computationally hiding if the underlying asymmetric encryption scheme is CPA-secure.* □

Proof. Correctness of the above scheme is obvious, and the *hiding property* follows immediately from the CPA-security of the encryption scheme.

To see that the *binding property* holds we assume that a cheating committer can send values (m, r_1, r_2) and (m', r_1', r_2') such that recipient accepts in both cases for the same commitment $com = (pk, c)$. The recipient's first test guarantees that key generation using r_1 gives a key pair (sk, pk). The correctness of the encryption scheme now guarantees that
$$\mathsf{D}(sk, c) = \mathsf{D}(sk, \mathsf{E}(pk, m, r_2)) = m,$$
but also
$$\mathsf{D}(sk, c) = \mathsf{D}(sk, \mathsf{E}(pk, m', r_2')) = m'.$$
This is a contradiction. ■

Note that the encryption scheme must guarantee CPA-security, in particular if the message space of the commitment scheme is small. However, no security against active attacks is necessary, because each key is only used once.

13.3.2 Quadratic Residues

The scheme described in the subsequent section is a special case of the previous one. We present it nevertheless, as it has a homomorphism property which allows many extensions and applications that are not possible (at least not efficiently) for the aforementioned, generic scheme. The scheme relies on the so-called Quadratic Residuosity Assumption (QRA), which is stronger than the factoring assumption (i.e., if someone can factor, he can also break QRA). Once given the QRA, the scheme is quite simple, but to get there, we need a bit more number theory. The basic question is what numbers are squares modulo a potentially composite number N.

Squares and Roots Generally

Recall that we already defined square roots for groups of prime order in an earlier chapter. We now define square roots for arbitrary, potentially composite values N as
$$QR_N := \{x \in \mathbb{Z}_N^* \mid \exists y \in \mathbb{Z}_N^* : y^2 = x \bmod N\}$$
for any natural number N. The set QR_N are the *quadratic residues*, and the complement set $QNR_N := \mathbb{Z}_N^* \setminus QR_N$ the *quadratic non-residues*.

[1] In the proof, we will also see why the recipient recomputes the encryption instead of simply decrypting: It is a general way of checking that something is a possible correct ciphertext. However, one can save communication by not sending m, and instead letting the recipient first decrypt c and then re-encrypt the resulting m.

Lemma 13.1 (Basic Facts Squares and Roots) *A few simple rules hold for squares and roots modulo arbitrary N:*

(a) If y is a root of x modulo N, then so is $-y$.

(b) The set QR_N constitutes a subgroup of $\mathbb{Z}_N^*$, i.e., a multiplicative group.

(c) If $x_1 \in QR_N$ and $x_2 \notin QR_N$, then $x_1 x_2 \notin QR_N$.

□

Proof. The proof of part (a) follows directly from $(-1)^2 = 1$.

For proving part (b), let x_1 and x_2 be elements of QR_N and y_1 and y_2 respective roots. Then $y_1 \cdot y_2$ and y_1^{-1} are roots of $x_1 \cdot x_2$ and x_1^{-1}, respectively, because $(y_1 \cdot y_2)^2 = y_1^2 \cdot y_2^2 = x_1 \cdot x_2 \bmod n$ and $(y_1^{-1})^2 = (y_1^2)^{-1} = x_1^{-1} \bmod N$.

Part (c) holds for any group: If $x_1 x_2$ were in QR_N, then x_2 would also have to be in QR_N because it can be expressed as $(x_1 x_2) \cdot (x_1)^{-1}$. ■

However, some other rules that are well-known from $\mathbb{N}$ or $\mathbb{R}$ are not always true in residue rings for composite moduli. In particular, a number can have more than two roots modulo N, for example for $N = 8$ we have $1 = 1^2 = 3^2 = 5^2 = 7^2 \bmod 8$.

Squares and Roots modulo N with Known Factors

Now we consider squares and roots modulo numbers $N = pq$ as in the usual setting for cryptographic systems that rely on the hardness of factoring. As usual, we use the Chinese Remainder Theorem to exploit the facts and algorithms known for the prime factors p and q.

Lemma 13.2 (Squares and Roots mod pq) *Let $N = pq$ with p, q prime and $p \neq q$. Then the following holds:*

(a) For all $x \in \mathbb{Z}_N^*$, we have

$$x \in QR_N \Leftrightarrow x \in QR_p \wedge x \in QR_q.$$

Note that the two conditions on the right-hand side can be tested efficiently with Euler's criterion if one knows p and q.

(b) Every $x \in QR_N$ has exactly 4 square roots (except if p or q equals 2).

□

Proof. Recall that from the Chinese Remainder Theorem we know that a congruence modulo N holds exactly if it holds modulo both p and q, i.e., for all $x, y \in \mathbb{Z}_N^*$:

$$x = y \bmod N \Leftrightarrow x = y \bmod p \ \wedge \ x = y \bmod q. \quad (*)$$

(a) "⇒": Let $x \in QR_N$ be given, and let y be a root, i.e., $y^2 = x \bmod N$. Then $y^2 = x \bmod p$ and $y^2 = x \bmod q$ follow immediately from (*), and therefore $x \in QR_p$ and $x \in QR_q$.
"⇐": Now let $x \in QR_p$ and $x \in QR_q$. This means that x has a square root modulo each of p and q, say $y_p^2 = x \bmod p$ and $y_q^2 = x \bmod q$. We cannot immediately apply (*) because the two congruences are different. However, by the Chinese Remainder Theorem itself, there exists a (unique) value y modulo N with

$$y = y_p \bmod p \text{ and } y = y_q \bmod q.$$

This implies

$$y^2 = y_p^2 = x \bmod p \text{ and } y^2 = y_q^2 = x \bmod q.$$

Now we can apply (*) and obtain

$$y^2 = x \bmod N.$$

(b) We know from (a) that x belongs to QR_p and QR_q. We know that it has two roots in each group, say $\pm y_p$ and $\pm y_q$. By the proof of part (a), each of the 4 possible combinations can be combined to a root of x with the Chinese Remainder Algorithm.

■

We now extend the Legendre symbol to the so-called *Jacobi symbol* modulo $N = pq$. First this is once again pure notation:

$$\left(\frac{x}{N}\right) := \left(\frac{x}{p}\right) \cdot \left(\frac{x}{q}\right)$$

More generally, the Jacobi symbol is defined for arbitrary $N \in \mathbb{N}$ in the same way: If $N = p_1^{e_1} \cdot \ldots \cdot p_t^{e_t}$ is the prime factorization of N, then

$$\left(\frac{x}{N}\right) := \left(\frac{x}{p_1}\right)^{e_1} \cdot \ldots \cdot \left(\frac{x}{p_t}\right)^{e_t}.$$

The Jacobi symbol is always 1 or -1 (except for values $x \in \mathbb{Z}_N \setminus \mathbb{Z}_N^*$, for which the Jacobi symbol is sometimes defined to be 0; we excluded these values by definition). However, the Jacobi symbol does not denote exactly whether x is a quadratic residue modulo N! Therefore we also introduce notation for all the elements with a certain Jacobi symbol:

$$\mathbb{Z}_N^{(+1)} := \{x \in \mathbb{Z}_N^* \mid \left(\frac{x}{N}\right) = 1\}$$

and $\mathbb{Z}_N^{(-1)}$ analogously.

Lemma 13.3 (Jacobi Symbol) *Let $N = pq$ with p, q prime and $p \neq q$.*

(a) We have

$$\left(\frac{x}{N}\right) = 1 \Leftrightarrow (x \in QR_p \wedge x \in QR_q) \text{ or } (x \notin QR_p \wedge x \notin QR_q).$$

Note that the first conjunction means $x \in QR_N$. Thus all squares have the Jacobi symbol 1, but not all numbers with Jacobi symbol 1 are squares.

(b) The Jacobi symbol is multiplicative, i.e., for all $x, y \in \mathbb{Z}_N^*$:

$$\left(\frac{xy}{N}\right) = \left(\frac{x}{N}\right) \cdot \left(\frac{y}{N}\right).$$

In other words, it is a homomorphism from $\mathbb{Z}_N^*$ to $\{1, -1\}$.

(c) $\mathbb{Z}_N^{(+1)}$ is a multiplicative subgroup of $\mathbb{Z}_N^*$.

□

Proof.

(a) The product in the definition of the Jacobi symbol is 1 if either both factors are 1 or both are -1. Now the claim follows immediately from the definition of the Legendre symbol.

(b) This follows immediately from the definition of the Jacobi symbol and the multiplicativity of the Legendre symbol, which easily follows from Euler's theorem.

(c) This follows immediately from Part (b): In the language of algebra, one can simply say that $\mathbb{Z}_N^{(+1)}$ is the kernel of the homomorphism. Otherwise, it is clear that for $x, y \in \mathbb{Z}_N^{(+1)}$ also $xy \in \mathbb{Z}_N^{(+1)}$, and one can easily derive that $\left(\frac{x^{-1}}{N}\right) = \left(\frac{x}{N}\right)^{-1}$ with $\left(\frac{x^{-1}}{N}\right) \in \mathbb{Z}_N^{(+1)}$.

■

Special Case $p, q = 3 \bmod 4$

If $N = pq$ for two primes $p \neq q$, and if additionally both primes are congruent to 3 modulo 4, we obtain the following lemma.

Lemma 13.4 *For $N = pq$ with p, q prime, $p \neq q$, and $p = q = 3 \bmod 4$:*

(a) The number -1 is a quadratic non-residue modulo N with Jacobi symbol 1, i.e.,

$$-1 \notin QR_N \wedge \left(\frac{-1}{N}\right) = 1.$$

(b) The non-squares with Jacobi symbol 1 are exactly the negatives of the squares:

$$\mathbb{Z}_N^{(+1)} \setminus QR_N = -QR_N.$$

(c) Anyone who knows p and q can compute square roots of a value $x \in QR_N$ by applying the algorithm from Section 6.1.3 to obtain roots y_p and y_q modulo p and q, respectively, and then applying the Chinese Remainder Algorithm to all 4 combinations of $\pm y_p$ with $\pm y_q$.

□

Proof.

(a) Obviously -1 is not a square modulo either p or q, as $\left(\frac{-1}{p}\right) = (-1)^{\frac{p-1}{2}} = (-1)^{2k+1} = -1 \bmod p$ for $p = 4k + 3$, and similar for q. Thus $-1 \notin QR_N$ and $\left(\frac{-1}{N}\right) = 1$.

(b) By Lemmas 13.2 (a) and 13.3 (a), a value x lies in $\mathbb{Z}_N^{(+1)} \setminus QR_N$ exactly if $x \notin QR_p$ and $x \notin QR_q$, i.e., if $\left(\frac{x}{p}\right) = \left(\frac{x}{q}\right) = -1$. This is equivalent to $\left(\frac{-x}{p}\right) = \left(\frac{-x}{q}\right) = 1$ by the multiplicativity and the proof of part (a), and thus to $-x \in QR_N$. This in turn is equivalent to $x \in -QR_N$.

(c) Clear by the proof of Lemma 13.2.

■

Squares and Roots Modulo N Without Knowledge of the Factors

The basic use of all the algorithms for testing quadratic residuosity and extracting square roots in cryptography is that one hopes that some of these tasks are hard for someone who does not know the factorization of N. We summarize some known facts in the following lemma:

Lemma 13.5 *For numbers $N = pq$ with p, q prime and $p \neq q$:*
(a) Extracting square roots is infeasible under the factoring assumption.
(b) Anybody can efficiently choose elements of QR_N randomly and uniformly.
(c) Anyone can efficiently compute Jacobi symbols modulo N. (This holds for all N.)

□

Proof.

(a) This will be proved on the exercise sheet.

(b) One can simply choose a random element of $\mathbb{Z}_N^*$ and square it. This gives a uniform distribution because every element of QR_N has exactly 4 roots.

(c) The algorithm is not very complicated and its efficiency quite similar to that of the Euclidean algorithm. However, the proof uses the so-called quadratic reciprocity law which is a fairly hard theorem in elementary number theory that we do not present it here.

■

It is not known whether deciding quadratic residuosity is as hard as factoring, thus we have to make a separate assumption for it. The assumption must in particular take into account that it is not hard to compute the Jacobi symbol for arbitrary N (Lemma 13.5). This means that one can efficiently see that certain numbers x are not squares, namely those with $\left(\frac{x}{N}\right) = -1$, whereas the remaining numbers x with $\left(\frac{x}{N}\right) = 1$ might either be squares, or non-squares modulo both p and q. Moreover, the assumption cannot state that no polynomial-time algorithm can find out whether $x \in QR_N$ with non-negligible probability, because mere random guessing will already succeed with probability 1/2. Finally, we will immediately make the assumption for the class of numbers we need. The assumption is therefore formulated as follows:

Definition 13.1 (QRA Challenger) *The QRA challenger for security parameter n is defined as follows:*

- *Firstly, it randomly chooses n-bit primes p, q with $q \neq q$ and sets $N := pq$.*
- *Secondly, it chooses a value x as follows, depending on the bit b:*
  - *If $b = 0$ let $x \leftarrow_{\mathcal{R}} \mathbb{Z}_N^{(+1)} \setminus QR_N$,*
  - *If $b = 1$ let $x \leftarrow_{\mathcal{R}} QR_N$.*
- *Finally, it outputs (N, x).*

◇

An adversary wins the game against the QRA challenger if it is able to deduce the bit b significantly better than by pure guessing. In the following, $Exp_{\mathsf{A}}^{\mathsf{QRA}}(b)$ denotes the experiment where the adversary A interacts with the QRA challenger with input bit b. Furthermore $Exp_{\mathsf{A}}^{\mathsf{QRA}}(b) = 0$ and $Exp_{\mathsf{A}}^{\mathsf{QRA}}(b) = 1$ denote the event that the adversary outputs 0 and 1 in the respective experiment. The advantage is defined as usual.

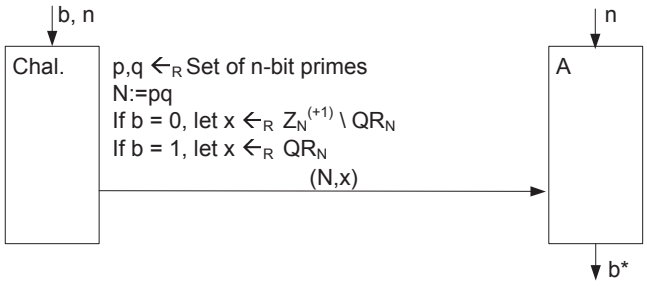


Figure 13.1: The QRA Game

Definition 13.2 (QRA Advantage) *The advantage of an adversary* A *against the QRA challenger is defined as follows (as a function of the security parameter* n *again):*

$$Adv^{\mathsf{QRA}}[\mathsf{A}](n) := \left|\mathsf{Pr}\left[Exp^{\mathsf{QRA}}_{\mathsf{A}}(0) = 1\right] - \mathsf{Pr}\left[Exp^{\mathsf{QRA}}_{\mathsf{A}}(1) = 1\right]\right|.$$

◇

Assumption 1 (Hardness of the QRA Problem) *The QRA problem is conjectured to be hard, i.e., it is conjectured that* $Adv^{\mathsf{QRA}}[\mathsf{A}](n)$ *is negligible in the security parameter* n *for all efficient adversaries* A*.*

More precisely, we will only use primes congruent 3 modulo 4 in the following, thus we might need a similar assumption 3-QRA where this additional restriction is made on the choice of p and q. However, 3-QRA follows from QRA because of Dirichlet's prime number theorem, which states that approximately half of all primes are congruent 3 mod 4. Thus about $1/4$ of all pairs (p, q) are of this form, and if an attacker A could guess quadratic residuosity with significant probability P for these, the success probability of A on arbitrary numbers would be at least $P/4$.[2]

13.3.3 The Quadratic-Residuosity Commitment Scheme

The following scheme is a bit commitment scheme. Essentially, one commits to 0 by a quadratic residue and to 1 by a quadratic non-residue. The scheme can be seen as a special case of the construction using an encryption scheme we have seen earlier, using a certain encryption scheme from Goldwasser and Micali, which encrypts 0 as a quadratic residue and 1 as a quadratic non-residue. In detail, the construction looks as follows:

- In *commit*, the committer generates n-bit primes p and q with $p = 3 \bmod 4$ and $q = 3 \bmod 4$ randomly and uniformly. (This can be guaranteed within the normal prime generation algorithm, also note that approximately half of all primes fulfill this property.) He verifies that $p \neq q$ (if not, he repeats the generation of q), and sets $N := pq$. He now chooses $y \leftarrow_{\mathcal{R}} \mathbb{Z}_N^*$ and computes
$$x := (-1)^m y^2 \bmod N,$$

[2]This was of course a sketch. One would need to state the "approximately" in Dirichlet's theorem precisely and proceed from there.

where m is his message bit. In other words, this part is y^2 or $-y^2$ depending on m. He sends the recipient

$$com := (N, x).$$

- In *open*, the committer sends (m, p, q, y) to the recipient. The committer verifies that N was generated correctly, i.e., that p and q are primes with $p = 3 \bmod 4$ and $q = 3 \bmod 4$ and $p \neq q$, and that $N = pq$. Then he verifies the main commitment, i.e., that

$$y \in \mathbb{Z}_N^* \text{ and } x = (-1)^m y^2 \bmod N.$$

Theorem 13.3 *The quadratic-residuosity commitment scheme just described is information-theoretically binding, and it is computationally hiding under the Quadratic Residuosity Assumption.* □

Proof. The binding property holds information-theoretically because no number x can be both a quadratic residue and a quadratic non-residue. We have to verify that the recipient's tests guarantee that opening x to $m = 0$ or 1 is in fact only possible if $x \in QR_N$ or $x \notin QR_N$, respectively. The former is clear. The verification of N guarantees that Lemma 13.4 applies and therefore $-y^2 \notin QR_N$.

The hiding property follows from the Quadratic Residuosity Assumption because this assumption says that one cannot distinguish quadratic residues and quadratic non-residues. More precisely, we have to check whether the commitments are chosen precisely as in the assumption.

For the modulus N this is clear, at least under the assumption 3-QRA, which was mentioned to follow from QRA.

If $m = 0$, then x is uniformly distributed in QR_N by the proof of Lemma 13.5(b).

If $m = 1$, then x is uniformly distributed in $-QR_N$, which equals $\mathbb{Z}_N^{(+1)} \setminus QR_N$ by Lemma 13.4(b). Thus if m is chosen randomly, the resulting x is a uniformly random element of $\mathbb{Z}_N^{(+1)}$. (In both cases for m, each of the possible values x occurs for 4 values y.) Thus we have shown that if m is chosen randomly, secrecy is exactly equivalent to the QRA. For our special case where the message space has only 2 elements, this immediately implies CPA-security: The attacker cannot find two messages m_1, m_2 such that he can distinguish their encryptions (here commitments) if the honest participant selects one of m_1 and m_2 randomly and encrypts it. In our case, m_1 and m_2 can only be 0 and 1. ■

If one wants to commit to several bits, it seems intuitively clear that one can commit to each bit separately with the given construction, and that secrecy of each bit will imply secrecy of the entire message. With the computational definition of secrecy, this statement nevertheless needs a proof, which we omit here.

Improved multi-bit commitment scheme For a commitment to a message m that is treated as a whole, i.e., whose bits will all be opened at the same time, the committer can choose one modulus N that is valid for all bits and only compute a separate value x_i for each bit m_i. Note that the committer cannot use the same modulus N for messages m and m' that will be opened at different times because revealing p and q would open both commitments. We can later extend this by using zero-knowledge proofs for the partial correctness of N. Then the choice of N and the proof of its correctness could be a subprotocol initialization that is used for many commitments, and the individual commitments would be opened by only showing y.

Special Properties

The quadratic-residuosity commitment scheme is obviously non-interactive and with public verification. Several other nice properties make the scheme especially useful in larger protocols. These are captured in he following lemma, whose proof will be omitted.

Lemma 13.6 *The following properties hold for the improved multi-bit commitment scheme, where several bits are committed to using the same modulus N.*

- *Once N is known, anyone (i.e., not only the committer) can make commitments with respect to this N. The committer can open them all.*
- *The commitments are homomorphic: Given two bit commitments x_1 and x_2 with respect to the same modulus N, their product $x_1 \cdot x_2$ is a commitment to $m_1 \oplus m_2$, where m_i is the content of x_i. If one assumes that correctness of N is shown in other ways, the product commitment can be opened without revealing additional information about the individual contents.*
- *Once N is known, anyone can blind commitments, i.e., transform a given commitment into a random commitment to the same bit.*

□

13.4 Information-Theoretically Hiding Schemes

This section is dedicated to an information-theoretically hiding commitment scheme based on discrete logarithms that can be used to commit to many bits efficiently. Committing is interactive. This is natural because the binding property should hold under a computational assumption, i.e., something should be infeasible for a computationally restricted committer. Thus someone else must choose the particular instance of the hard problem, e.g., a number that a cheating committer hopefully cannot factor.

13.4.1 Commitment Schemes based on Discrete Logarithms

The following scheme can be used in any family of groups of prime order where computing discrete logarithms is conjectured to be infeasible. For concreteness, we use subgroups G_q of $\mathbb{Z}_p^*$, making the same discrete-logarithm assumption as in Chapter 7.

The Discrete-Logarithm Commitment Scheme

We start immediately with the construction:

- As for the ElGamal encryption scheme we assume a function $n_p \colon \mathbb{N} \to \mathbb{N}$ which determines the size of the group $\mathbb{Z}_p$ containing the subgroup G_q. The message space is $\mathbb{Z}_q$.
- In commit, the recipient randomly chooses an n-bit prime q and an $n_p(n)$-bit prime p with $q|(p-1)$. The recipient also selects an elements g of order q, and a second generator h of $\langle g \rangle$. The recipient sends
$$pk_{Rec} := (p, q, g, h)$$
to the committer. The committer verifies that p and q are prime and that g and h are of order q. The committer chooses an integer $k \leftarrow_{\mathcal{R}} \mathbb{Z}_q$ and sends
$$com := g^m h^k \bmod p.$$

to the recipient.

- In open, the committer sends (m, k), and the recipient verifies that $com = g^m h^k \bmod p$.

Theorem 13.4 *The discrete-logarithm commitment scheme just described is information-theoretically hiding, and it is computationally binding under the Discrete Logarithm assumption for the chosen family of groups.* □

Proof. For the information-theoretical hiding property, we essentially show that k serves as a one-time pad on m.

First note that the committer verifies that g and h generate the same group of order q, where q is prime. Hence each element of G_q has exactly one representation as h^k with $k \in \{0, \ldots, q-1\}$. Thus h^k is a uniformly random element of G_q. Multiplying by h^k is therefore a one-time pad operation in the group G_q and hides g^m perfectly, and thus also m.

Now assume for contradiction that a cheating committer could break the binding property, using a probabilistic polynomial-time algorithm C. Thus C gets an input (p, q, g, h) and should output values (com, m, k, m', k') with $m \neq m'$ and $com = g^m h^k = g^{m'} h^{k'} \bmod p$. From such outputs of C, one can easily compute $\mathsf{Dlog}_g(h) \bmod p$ as follows:

$$\begin{aligned} g^m h^k = g^{m'} h^{k'} \bmod p \quad &\Rightarrow \quad g^{m-m'} = h^{k'-k} \bmod p \\ &\Rightarrow \quad g = h^{(k'-k)(m-m')^{-1}} \bmod p. \end{aligned}$$

The inverse in the exponent is taken in the group $\mathbb{Z}_q^*$: The precondition $m \neq m'$ implies $m - m' \neq 0 \bmod q$, and congruences mod q can be applied in the exponents. Thus $(k' - k)(m - m')^{-1} = \mathsf{Dlog}_g(h) \bmod p$. Hence we can use C as a subprogram to construct an almost equally efficient algorithm C^* that computes discrete logarithms with the same success probability as that of C. This contradicts the discrete-logarithm assumption. ∎

Note that this commitment scheme is similarly efficient as other asymmetric cryptographic primitives like RSA (in contrast to the quadratic-residue scheme): To commit to an n-bit message, one essentially only needs two exponentiations with exponents of length n. This means two multiplications per message bit. Moreover, the commitment is only one number modulo p.

Improved scheme for many commitments The recipient can generate pk_{Rec} once and for all in a subprotocol initialization. Then many committers can commit to many messages for this recipient non-interactively, always using this pk_{Rec}. Each of them verifies the correctness of pk_{Rec} before he uses it for the first time. One can easily see that the security proof still holds in this scenario.

Every recipient really needs his own pk_{Rec} (or needs to trust another recipient or a center for its generation) because he is only convinced that the commitments are binding if he is sure that nobody has generated g as h^x so that he would know $\mathsf{Dlog}_g(h)$.

Special Properties

The discrete-logarithm commitment scheme is obviously with non-interactive opening and public verification.[3] Similar to the quadratic-residuosity commitments, it has another nice property that

[3] Note that public verification does not contradict the fact that recipients cannot trust commitments made for other recipients, which was mentioned earlier.

is useful in larger protocols.

Lemma 13.7 *The discrete-logarithm commitment scheme is homomorphic if several messages are committed to using the same value pk_{Rec}: If a committer has made two commitments com_1 and com_2 with respect to the same value $pk_{Rec} = (p, q, g, h)$, he can open the product $com_1 \cdot com_2$ as a commitment to $m_1 + m_2 \bmod q$. Opening the product commitment does not reveal additional information about the individual contents.* □

Note that the homomorphism property was formulated in a different way than in Lemma 13.6: Now we said "he can open the commitment as" instead of talking about the content of a commitment. This is unavoidable with information-theoretically hiding commitments: The notion "the content" is not defined for such commitments by themselves because of the fact that they can have any content — this is exactly the property of being information-theoretically hiding. What the "actual" content is lies only in the knowledge of the committer, in our case which of the possible pairs (m, k) the committer knows.[4]

Proof. The committer knows values k_1 and k_2 such that $com_1 = g^{m_1} h^{k_1}$ and $com_2 = g^{m_2} h^{k_2} \bmod p$. This implies

$$com_1 \cdot com_2 = (g^{m_1} h^{k_1})(g^{m_2} h^{k_2}) = g^{m_1+m_2} h^{k_1+k_2} \bmod p. \qquad (*)$$

Thus the committer can open the product commitment by sending $(m_1 + m_2, k_1 + k_2)$. Both sums are taken modulo q.

We now show that after the product commitment has been opened, any pair of messages (m'_1, m'_2) with the correct sum is still equally likely. This is now an information-theoretic property and therefore simpler to prove than in Lemma 13.6: We show that for each such pair, there is exactly one pair (k'_1, k'_2) that the committer could have chosen in committing such that it fits both the secret messages (m'_1, m'_2) and the view of the attacking recipient. As the "pads" k_1 and k_2 are chosen uniformly, this is sufficient for secrecy.

Uniqueness: If such a pair (k'_1, k'_2) exists at all, then $com_1 = g^{m'_1} h^{k'_1}$ and $com_2 = g^{m'_2} h^{k'_2} \bmod p$ must hold. This uniquely defines $h^{k'_1}$ and $h^{k'_2} \bmod p$ and thus also k'_1 and k'_2 modulo q because h is a generator of G_q.

Existence: The recipient's view consists of the two commitments and the values (m_1+m_2, k_1+k_2) that opened the commitments. Thus we have to show that the only possible values k'_1 and k'_2 fulfill $k'_1 + k'_2 = k_1 + k_2 \bmod q$. We know that

$$com_1 com_2 = (g^{m'_1} h^{k'_1})(g^{m'_2} h^{k'_2}) = g^{m'_1+m'_2} h^{k'_1+k'_2} \bmod p.$$

We combine this with equation (*) and use that $m'_1 + m'_2 = m_1 + m_2 \bmod q$: Thus we get

$$h^{k_1+k_2} = h^{k'_1+k'_2} \bmod p.$$

This implies $k'_1 + k'_2 = k_1 + k_2 \bmod q$, which finishes the proof. ■

Remark: One can easily generalize this from sums to arbitrary linear combinations, i.e., if a_1, a_2 are two openly known numbers, the committer can open $com_1^{a_1} com_2^{a_2}$ as a commitment to $a_1 m_1 + a_2 m_2$ without revealing additional information about the individual contents.

[4] For the same reason, we cannot state an analog of Lemma 13.6 (c). Moreover, nobody can efficiently open commitments that he has not made, in contrast to Lemma 13.6 (a).

14. Secret Sharing

Sometimes it is necessary to store a secret in places that are not fully secure, e.g., we think it might be possible that a single place might be revealed, but it is very unlikely that all of them are revealed at the same time. This case can be effectively treated by using a secret sharing scheme to distribute the secret to several places. Secret sharing schemes prevent an attacker from learning any information if he revealed a single location, but still enables us to reconstruct the secret from a sufficiently large number of shares.

Note that this cannot be achieved in a trivial way by storing individual letters of the password in different places: On the one hand, the strong secrecy requirement would be violated even if an adversary found only one share. On the other hand, even the requirement that any sufficiently large number of shares enables us to reconstruct the secret would not be easy to fulfill because some shares would need to overlap in some letters.

14.1 Definition Sketch of Basic Secret Sharing

The following is called a sketch because we leave out the precise formalization of notions like protocol and participant. Nevertheless many cryptographers would not hesitate to call it a definition.

As in all following definitions, we will define parameters of a protocol (also called scheme), roles, i.e., different types of participants, subprotocols (also called phases or transactions or protocols again), and requirements with corresponding trust models.

The parameters are the total number of desired shares n, the minimum number k of shares from which reconstruction is possible, and a secret space S, i.e., a set of possible secrets. An instantiation with specific n and k is often called a k-of-n secret sharing scheme.

There are three roles: a so-called *dealer*, i.e., the initial owner of the secret, *shareholders*, i.e., participants who get shares, and a *reconstructor*, i.e., a party that reconstructs the secret. In a particular execution of secret sharing, one dealer, n shareholders, and one reconstructor will take part.

By the initial example, one may not see a need for shareholders and reconstructors — the shares could be in "places" instead of held by "participants", and the reconstructor is typically the dealer. However, to include more examples and some extensions, it is better to make this slightly more general definition at once.

There are two subprotocols:

- *Sharing.* This is a protocol between the dealer and all shareholders. Typically it is a very simple type of protocol with almost no interaction: The dealer has an input $s \in S$, i.e., the secret to be shared. The dealer makes some local computation given by the algorithm share (typically probabilistic) resulting in so-called *shares* $s_1, \ldots, s_n$. One share s_i is sent to each shareholder, who simply stores s_i.

- *Reconstruction.* This is a protocol between the reconstructor and k shareholders. Typically it is also a very simple type of protocol with almost no interaction: Each participating shareholder sends its share to the reconstructor. The reconstructor makes some local computation given by the algorithm $\mathsf{reconstruct}$ resulting in a value s^*. (This is supposed to be the secret s again, but it is better to use different notation and formally require equality in our availability requirement.) We will call the set of indices of shares used for reconstruction *Avail*. For instance, if s_2, s_3, and s_5 are used, we have $Avail = \{2, 3, 5\}$. More generally, this is the set of indices (or names) of the shareholders participating in reconstruction.

There are two requirements.

- *Availability*: Reconstruction works correctly, i.e., $s^* = s$, if the dealer, the reconstructor, and the k shareholders participating in this particular reconstruction are honest, and the adversary cannot modify any message.
- *Confidentiality*: Any $k-1$ shareholders gain no information about the secret s if the dealer and the reconstructor are honest, and the lines between honest participants are not tapped.

If several reconstructions of the same secret may be needed (e.g., the password is needed every three months), this requirement includes that shareholders who already participated in some reconstructions still do not learn the secret. In the non-interactive case as above it is clear that shareholders do not gain any information in reconstruction, but in some extensions this needs more care.

We can already write the two requirements more formally for the non-interactive case. For this, let share and $\mathsf{reconstruct}$ be the algorithms that the dealer and the reconstructor use. The former must be probabilistic, the latter is deterministic. We can assume that $\mathsf{reconstruct}$ does not only get the shares themselves as inputs, but also the indices of the participants who held them.

- *Availability*: For all $s \in S$, for all n-tuples $(s_1, \ldots, s_n) \in [\mathsf{share}(s)]$ (i.e., all possible outputs of $\mathsf{share}(s)$), and all k-tuples $(i_1, \ldots, i_k)$ where all i_j are distinct elements of $\{1, \ldots, n\}$, we have $\mathsf{reconstruct}(s_{i_1}, \ldots, s_{i_k}) = s$.
- *Confidentiality*: This is defined as follows: For all probability distributions D on the secret space, all specific secrets s, and all observations c of the attacker, the a-posteriori probability of s, given the observation c, equals the a-priori probability of s. The probability space in which we can speak of all these probabilities is given by the initial distribution D on the secret space and (the random bits used by) the probabilistic algorithm share: All shares are random variables in this space, and the observation c consists of the shares an attacker has. Thus, if Pr_{D^*} is the probability in this space, we can simply write for any $i_1, \ldots, i_{k-1} \in \{1, \ldots, n\}$ (defining the attacker)
$$\mathsf{Pr}_{D^*}(s \mid (s_{i_1}, \ldots, s_{i_{k-1}})) = \mathsf{Pr}_{D^*}.$$

14.2 Remarks on the Trust

First note that we have different trust models for availability and confidentiality. Recall that one generally speaks of a k-of-n trust model if one trusts that at most $k-1$ out of n given participants are dishonest.

- For confidentiality, we have exactly such a k-of-n model for the shareholders: If at most $k-1$ of them are dishonest, the dishonest ones have no information.

- For availability, however, the trust is different: We assume that at least k shareholders are honest. This means that at most $n-k$ are dishonest, which corresponds to an $(n-k+1)$-of-n model.

Note that there is *no integrity* at all. In particular, if one of the shareholders participating in a reconstruction contributes a wrong share, a wrong secret will be reconstructed.

Furthermore, to see that considering the definition and in particular the underlying trust model can be helpful in practice, consider an example that is a rather usual introduction to secret sharing: Consider a bank with a highly secret vault that no single employee is supposed to open, but only, say, 3 of 5 directors together. The vault is opened with a secret number, and this number is shared among the directors. This sounds ok at first, but if you look closely at the trust, there are problems: Who should be the trusted dealer and reconstructor? In particular, reconstruction is needed quite often. If the vault has a mechanical combination lock, i.e., the directors themselves must reconstruct the secret first and then input it, at least one of them will know it. On the other hand, if the vault is computerized and can serve as the reconstructor itself, there is no need for secret sharing: It would be much simpler if each of the directors got an independent password and the vault counted how many correct passwords had been entered. Thus basic secret sharing is in fact much better suited to the sharing of a personal password or a personal secret key, where the owner is both reconstructor and dealer, and where the secret is given a priori from another system.

14.3 Shamir's Construction

Secret sharing was invented independently by Shamir and Blakley in 1979, but Shamir's construction is simpler and better known nowadays. We first describe conditions on the parameters and then the two protocols, sharing and reconstruction. Both are of the simple, almost non-interactive structure described above.

14.3.1 Parameter Conditions

The secret space S must be a subset of a finite field $\mathbb{F}$, e.g., a prime field $\mathbb{Z}_p$. Moreover, $|\mathbb{F}| > n$ is needed.

14.3.2 Sharing

1. First, the dealer chooses a random polynomial pol of degree $k-1$ over $\mathbb{F}$ with constant term s. In other words, he chooses coefficients $a_1, \ldots, a_{k-1} \leftarrow_{\mathcal{R}} \mathbb{F}$ and sets

   $$pol(x) = a_{k-1}x^{k-1} + \ldots + a_1 x_1 + s,$$

   where s is the secret.

2. Second, the dealer fixes a different value $x_i \in \mathbb{F}^*$ (i.e., non-zero field elements) for each shareholder and gives the pair

   $$s_i := (x_i, pol(x_i))$$

   to this shareholder as her share. Thus each pair is a point on the polynomial.

   If the shareholders are a priori numbered, we can also assume a fixed mapping to values x_i. Then only the value $pol(x_i)$ is the actual share. For example, if $\mathbb{F} = \mathbb{Z}_p$, shareholder i can

simply have $x_i = i$. The precondition $|\mathbb{F}| > n$ is needed so that each shareholder can have a different value x_i, and $x_i \neq 0$ is needed because $pol(0) = s$, i.e., a share at the point 0 would be the secret itself.

14.3.3 Reconstruction

Before presenting the actual reconstruction algorithm, we can see that the secret can in principle be reconstructed uniquely from any k shares: It is a well-known theorem in algebra that over any field, there is at most one polynomial of degree $k-1$ through any k given points. (One can easily derive this as a corollary from an even better-known theorem that any non-zero polynomial of degree $k-1$ has at most $k-1$ zeros.) E.g., there is at most one straight line through any two points, and at most one parabola through any three points. Now the k shareholders have k points, thus their points determine a polynomial uniquely, and thus also the secret as the constant coefficient of this polynomial. An efficient algorithm for actually constructing the polynomial from the given points is Lagrange interpolation.

First one computes "basis polynomials" $bpol_{i^*}(x)$ which have value 1 at one point x_{i^*}, and value 0 at all the other points under consideration.

To do this, one starts with even simpler polynomials

$$cpol_{i^*}(x) := \prod_{i \in Avail \setminus \{i^*\}} (x - x_i).$$

Recall that *Avail* is the set of indices of shares used for reconstruction. Obviously, $cpol_{i^*}(x_i) = 0$ for all $i \in Avail \setminus \{i^*\}$ already, but the value at x_{i^*} is typically not yet 1 (but different from 0!). Thus we normalize these polynomials by dividing them through their value at the point x_{i^*} and obtain

$$bpol_{i^*}(x) := \frac{cpol_{i^*}(x)}{cpol_{i^*}(x_{i^*})} = \prod_{i \in Avail \setminus \{i^*\}} \frac{(x - x_i)}{(x_{i^*} - x_i)}$$

Now we simply use the appropriate linear combination of basis polynomials to obtain a polynomial *pol* through the points (x_i, y_i) for all $i \in Avail$:

$$pol(x) = \sum_{i \in Avail} y_i \cdot bpol_i(x).$$

14.3.4 Proof of Availability

Availability is the fact that reconstruction works correctly, and this was almost proven in the last section. Section However, we can still check it in the final formula: For all $i^* \in Avail$ we have

$$\begin{aligned} pol(x_{i^*}) &= \sum_{i \in Avail} y_i \cdot bpol_i(x_{i^*}) \\ &= 0 + \ldots + 0 + y_{i^*} \cdot bpol_{i^*}(x_{i^*}) + 0 + \ldots + 0 \\ &= y_{i^*}. \end{aligned}$$

Thus Lagrange interpolation gives in fact a polynomial through all the given points, and by uniqueness this is the polynomial the dealer had chosen.

14.3.5 Proof of Secrecy

We prove secrecy by the following lemma. Note that here we assumed that any attacker has in fact $k-1$ shares and not less; it is clear that any weaker attacker cannot have more information.

Lemma 14.1 *For any attacker indices $i_1, \ldots, i_{k-1} \in \{1, \ldots, n\}$, for all possible observations $c = (s_{i_1}, \ldots, s_{i_{k-1}})$, where each share s_{i_k} is a point (x_{i_j}, y_{i_j}), and for every secret $s \in S$, there exists exactly one polynomial pol that is consistent with both the observation and the secret.* □

Proof. The secret and the shares together fix k points on the polynomial (recall that the secret was the point at $x = 0$). By the algebraic theorem, there is at most one such polynomial, and by Lagrange interpolation, there is also at least one. ■

14.4 Efficiency Improvements and Further Properties

Now we consider some improvements and interesting facts about Shamir's scheme.

14.4.1 Simple Improvements

Not reconstructing the entire polynomial As we only need the constant coefficient of the polynomial at the end, it is not necessary to reconstruct all the others. Thus we actually only need to compute

$$s = pol(0) = \sum_{i \in Avail} y_i \cdot bpol_i(0)$$

where

$$bpol_{i^*}(0) = \prod i \in Avail \setminus \{i^*\} \frac{-x_i}{x_{i^*} - x_i}$$

Splitting long secrets For sharing large secrets exactly as above, one would have to compute in very large fields, and the complexity of multiplication is quadratic. However, the complexity becomes linear if we simply split long secrets into fixed-length blocks and share those independently.

Sharing several secrets For secrecy, one clearly needs a completely new polynomial for each secret to be shared. Nevertheless, there is a lot of work one only needs to do once, at least if the same subset of participants reconstructs several secrets: The basis polynomials depend only on the subset and values x_i assigned to the participants, which can be fixed. Thus this subset can compute the value $bpol_i(0)$ for each of its members once, and reconstruction is a simple linear combination with the current values y_i.

14.4.2 Optimality

As you may have noticed, each share is as long as the original secret. This may not correspond to an intuitive idea of "sharing". However, one can quite easily see that one cannot do better. The proof sketch goes as follows: With any $k-1$ shares, one should still have no information at all, in an information-theoretic sense, about the secret. However, with k shares, one suddenly knows the secret exactly, i.e., has the full information about it. Thus as much information as in the secret must

have been in the k-th share. If one knows elementary information theory, one can formalize this in a natural way. Here we prove a slightly weaker version of the optimality theorem in an elementary way: We show that each share must have at least as many possible values as the secret. Thus we show that

$$|S_i| \geq |S|,$$

where S_i is the set from which the i-th share is chosen, and S was the secret space.

Proof. (Sketch) First consider an attacker who has $k-1$ shares, not including share i, i.e., the attacker indices are $i_1, \ldots, i_{k-1} \in \{1, \ldots, n\} \setminus \{i\}$. From the point of view of this attacker, all secrets $s \in S$ must still be possible by the information-theoretic confidentiality. Thus for each s, the probabilistic algorithm $\mathsf{share}(s)$ must have an output in which all the attacker's shares occur. Let $s_i^{(s)}$ be share i in this same output. (I.e., it is a value for the i-th share that is consistent with both the secret s and the observation c.) Now we consider reconstruction by the k participants i and $i_1, \ldots, i_{k-1}$. By the availability requirement, $\mathsf{reconstruct}((i, s_i^{(s)}), (i_1, s_{i_1}), \ldots, (i_{k-1}, s_{i_{k-1}}))$ must be s. This implies that all the values $s_i^{(s)}$ for different s must be different: If $s_i^{(s)}$ were equal to $s_i^{(s*)}$ for $s \neq s^*$, then $\mathsf{reconstruct}$ would have two different values s, s^* as output on the same input tuple. But reconstruct is a deterministic function. Hence there are at least as many possible values in S_i as in S. ∎

14.4.3 Computing with Secrets

Sometimes we will want to compute with secrets that have been shared, and it is dangerous to put the secret together. In particular, this holds in group cryptography. For the moment, we will just consider the simplest case, which is to compute linear combinations of shared secrets. This is equivalent to being able to compute additions and scalar multiplications, i.e., multiplications with a value that is not hidden as a shared secret, but known to all participants. Assume that two secrets s and s' are shared using polynomials pol and pol'. Thus the secrets are

$$s = pol(0) \text{ and } s' = pol'(0),$$

and the shares of a participant with fixed value x_i are

$$y_i = pol(x_i) \text{ and } y_i' = pol'(x_i).$$

Now consider the sum pol'' of the polynomials, which is also a polynomial of degree at most $k-1$: We have

$$pol''(0) = pol(0) + pol'(0) = s + s'.$$

Hence this polynomial pol'' can be used to share the sum of the secrets. Moreover,

$$pol''(x_i) = pol(x_i) + pol'(x_i) = y_i + y_i'.$$

Hence the shares of this polynomial pol'' are the sums of the shares of the original polynomials. This last operation can be carried out by each shareholder locally: He just adds his two shares together. Thus they now hold shares to the sum $s + s'$ of the original two secrets. Similarly, one obtains the shares of a value ks, where $k \in \mathbb{F}$ is a known value and s is a shared secret, if each shareholder locally multiplies her share by k.

14.5 Sketch of Verifiable Secret Sharing

As the name says, verifiable secret sharing (abbreviated VSS) is an extension of secret sharing. The point is to add more robustness (availability) to the original secret sharing. This has two aspects:

- *Malicious shareholders*: So far we assumed that shares are either correct or get lost completely. In other words, each share that is used in a reconstruction is correct. However, a malicious shareholder might input a wrong share, and then a wrong secret would be reconstructed. (One can easily see that Shamir's scheme has no redundancy at all if exactly k shares are used for reconstruction, i.e., whatever one inputs, there is always some result.) Depending on the situation, one might notice that the reconstructed secret is incorrect (e.g., it is not a secret key that corresponds to a certain public key), and try other subsets of shareholders. But this procedure is cumbersome. Hence it would be nicer if one could verify each share individually.

- *Untrusted dealer*: The more complicated problem is if the person who originally holds the secret is not trusted by the others, and the others need a guarantee that they can put a consistent secret together. An example is that a person with an important job in a company has to share her private key with some others in case she dies or leaves the company in a quarrel. Now the others will want a guarantee that they hold shares of a private key that correctly fits the known public key. In other examples it is enough that any secret is consistently fixed in the shares. (For instance, in a multi-party coin flipping protocol the individual random values r_i might not be fixed by commitments, but by sharing them with a verifiable secret sharing system to avoid the disruption problem at the cost of a different trust model.)

We now present Pedersen's verifiable secret sharing scheme.We omit a few details about parameters, the choice of the group etc.

1. Basically the scheme uses Shamir's secret sharing to share the secret s. Thus the dealer chooses a polynomial
$$pol(x) = a_{k-1}x^{k-1} + \ldots + a_1 x^1 + s,$$
where the constant coefficient is the secret and the other coefficients are random. We assume that the x-values x_i of the shares for each participant P_i are fixed in advance, e.g., $x_i = i$.

2. Now, to fix everything, the dealer makes commitments to the whole polynomial, i.e., to all its coefficients. The discrete-logarithm commitments from Section 13.4.1 are used. Let us denote the commitments as follows:
$$\begin{aligned} com &= (com_{k-1}, \ldots, com_0) \\ &= (g^{a_{k-1}}h^{b_{k-1}}, \ldots, g^{a_0}h^{b_0}). \end{aligned}$$
Here, for simplicity, we also used the notation $a_0 := s$. The b_i's are all random. The dealer sends these commitments to everyone, i.e., the shareholders and the reconstructor. This should be done by broadcast, so that everybody agrees on these commitments.

   Note that so far, we have already made sure that there is really a polynomial of degree k, and that the secret is the content of com_0. Hence we can use the scheme for situations where an arbitrary but fixed secret is needed, and for situations where an external secret is given by such a commitment $g^s h^{b_0}$.

3. The dealer gives a normal Shamir share to each shareholder P_i, i.e.,

$$y_i = pol(x_i).$$

In addition (to help the verification in the next step), the dealer gives P_i a corresponding "helpshare" on the polynomial pol' defined by the values b_i. Thus, let

$$pol'(x) = b_{k-1}x^{k-1} + \ldots + b_1x^1 + b_0$$

and

$$z_i = pol'(x_i).$$

4. Now each shareholder verifies that his share s_i does in fact lie on the polynomial that is hidden in the published commitments. For this, we use the homomorphic property of the commitment scheme, i.e., the fact that we can perform additive operations on hidden values by performing the corresponding multiplicative operation on the commitments. If y_i and z_i are correct, we have

$$\begin{aligned} g^{y_i}h^{z_i} &= g^{pol(x_i)}h^{pol'(x_i)} \\ &= g^{a_{k-1}x_i^{k-1}+\ldots+a_1x_i^1+a_0} \cdot h^{b_{k-1}x_i^{k-1}+\ldots+b_1x_i^1+b_0} \\ &= (g^{a_{k-1}}h^{b_{k-1}})^{x_i^{k-1}} \cdot \ldots \cdot (g^{a_1}h^{b_1})^{x_i^1} \cdot (g^{a_0}h^{b_0}) \\ &= com_{k-1}^{x_i^{k-1}} \cdot \ldots \cdot com_1^{x_i^1} \cdot com_0 \\ &=: comshare_i. \end{aligned}$$

The last formula only contains publicly known values. Hence everybody can compute $comshare_i$. Thus P_i can simply verify whether the values y_i and z_i that the dealer gave him fulfil the equation $g^{y_i}h^{z_i} = comshare_i$.

5. During reconstruction, the reconstructor collects both the shares y_i and the help-shares z_i from k shareholders. He can also verify whether $g^{y_i}h^{z_i} = comshare_i$. If this is true, he knows that these shareholders gave him their correct shares, and he uses normal Shamir reconstruction on the values y_i to obtain the secret s.

In the full protocol, one must also specify what happens if some shareholders complain in Step 4 that their shares don't lie on the polynomial. (Either these shareholders or the dealer might be incorrect.) It should be clear from the explanations within the protocol, in particular in Step 4, that the scheme works in the correct case. Secrecy and availability should be plausible: All the commitments hide their contents information-theoretically, so they should not hurt secrecy. And the computational binding property of the commitments should guarantee that all the commitments have, for practical purposes, unique contents, so that in particular no shareholder can later open a commitment in different ways and thus define different polynomials, even if he colludes with a dishonest dealer. But if we had more time, a real proof should be given in this place.

15. Zero-Knowledge Proofs

Zero-knowledge proof systems are intensely studied in modern theoretical cryptography. Similar to commitment schemes, they mostly occur as subprotocols in other protocols, but there also exist direct applications. Examples for direct applications are as identification schemes and constructions of signature schemes from these identification schemes.

15.1 Overview of Types of Proof Systems and Zero-Knowledge

We first give an overview of possible goals with proof systems. The notions of a proof system and zero-knowledge are actually independent: A *proof system* is a 2-party protocol between a so-called *prover* and a so-called *verifier*. The goals are roughly:

- *Completeness*: The prover can convince the verifier of correct statements.
- *Soundness*: Not even a cheating prover can convince an honest verifier of wrong statements. Note that "completeness" in the terminology of proof systems is simply the property that everything works if nobody cheats, whereas "soundness" is the main integrity goal.

Being *zero-knowledge* is a property of one interactive machine, here the prover. It means that this machine, no matter with what other machines it interacts, does not give the other machines any new knowledge. The definition of "any new knowledge" is not trivial and takes some getting used to when one first sees it. For example, one might think that being convinced after a proof seems to be "knowledge" and thus the notion of a zero-knowledge proof would be contradictory. There are two major classes of proof systems with different goals:

- Proofs of *language membership*: Here a language L is given, i.e., a subset of the strings over some alphabet, typically $\{0,1\}$. The statements to be proved are that a certain string x, known to both the prover and the verifier, lies in L.
- Proofs of *knowledge*: Here a binary relation R is given. The statements to be proved are that for a certain string x, known to both parties, the prover "knows" a string w with $(w,x) \in R$. The string w is often called a *witness* for x.

Examples Proving that a number x is a quadratic residue is an example of a proof of language membership: The language L in this case is actually the set of pairs (N,x) with $x \in QR_N$. Proving that one knows a square root of a number x modulo N is an example of a proof of knowledge: The relation in this case is $R = \{(w,(x,N)) \mid w^2 = x \bmod N\}$.

It should be fairly clear at this point how the security of proofs of language membership will be formalized, whereas defining "knowledge" is non-trivial, just like the other extreme, "zero-knowledge". The goals can hold with information-theoretical or computational security. Proofs whose soundness only hold with computational security are often called arguments. Moreover, one can construct

proof systems either for one particular language L or for a large class of languages, and similarly for relations. Hence there is an even greater variety of constructions of zero-knowledge proof systems than of commitment schemes.

15.2 Defining Interactive Proofs

In this section, we do not bother about zero-knowledge yet, i.e., about the secrecy aspect of our protocols, but only about the proof properties, i.e., about integrity aspects. These aspects are in the interest of the verifier. Thus we assume that the verifier is honest, whereas the prover may try to cheat. We first define proofs of language membership and then proofs of knowledge.

15.2.1 Proofs of Language Membership

Before we can formalize the definition, we have to consider for what reason it makes sense to have an *interactive* proof at all: Why doesn't the verifier sit down at home and prove the statement for himself? The reasons are different in complexity theory and in practical cryptography, and this leads to two different definitions.

In *complexity theory*, interactive proofs are seen as an extension of the concept of NP: One can regard a language $L \in NP$ as a language with an interactive proof system where the prover is computationally unrestricted and sends only one message. This message is a witness for the NP statement, i.e., the string that the verifier would otherwise have to guess non-deterministically, e.g., a satisfying assignment for a Boolean formula if $L = SAT$. Given such a string, the verifier can decide in polynomial time whether it is correct or not. We will not follow this further, because honest provers in practical protocols cannot be computationally unrestricted. Let us just mention that a famous result says that $IP = PSPACE$, where IP is the set of all languages with an interactive proof system, i.e., the class is presumably larger than NP.

In *practical cryptography*, the reason why the prover can do more than the verifier is that he has originally chosen some secrets, i.e., he has auxiliary information at the start of the proof protocol. We will have to mention this auxiliary information in the definition, and the proofs will only work if this information is somehow correct.[1]

Example In an interactive proof that a number x_0 is a quadratic residue modulo some N, we will typically exploit that the prover knows the prime factors of N.[2] Most definitions of interactive proofs in the literature are in the first scenario, which makes them a bit simpler than the following one. However, the following definition is still a compromise: A definition that can really be used for most things one usually wants to do with it contains even more parameters.

Definition 15.1 (Interactive Proof System for Language Membership) *An* interactive proof system for language membership *with generation algorithm has the following components:*

[1] In these cases, if we would not bother about zero-knowledge, an interactive proof could always be transformed into one where only one message is sent: The prover would send his entire auxiliary information to the verifier, and the verifier could carry out the rest of the protocol alone (i.e., run both interactive algorithms).

[2] The auxiliary information here is similar to the witness in a proof of knowledge. However, here the goal of the proof is not to prove that the prover has some particular auxiliary information, in contrast to proofs of knowledge. For example, even if we construct a correct prover algorithm that needs the factors of N as auxiliary information, it is not forbidden that a cheating prover also succeeds if he only knows a root of x_0 or some other auxiliary information.

- *The parameters are the language L and certain security parameters, which we collectively call par.*
- *There are two roles, the* prover *and the* verifier.
- *A probabilistic algorithm* Gen. *It is meant for the prover to generate an element from the language together with suitable auxiliary information. The input is only the security parameters par, and the output is a pair (x, aux).*
- *The subprotocol* prove *is a 2-party protocol. The individual interactive algorithms of the prover and the verifier in it are called* P *and* V.
  - P *needs an initial input (par, x, aux) and does not make a final output.*
  - V *needs an initial input (par, x) and produces a Boolean output b, where 0 denotes that it is convinced that $x \in L$.*

  Let
  $$Exp_{\mathsf{P},\mathsf{V}}((par, x, aux), (par, x)) = b$$
  denote the event that V *finally outputs the bit b.*

These algorithms fulfill the following requirements:

- Correct generation: *For all valid parameters par, and all $(x, aux) \in [\mathsf{Gen}(par)]$, we have $x \in L$.*
- Completeness: *If x has been generated correctly, a correct prover can always convince a correct verifier. More precisely, for all valid parameters par:*
  $$\mathsf{Pr}\left[Exp_{\mathsf{P},\mathsf{V}}((par, x, aux), (par, x)) = 0;\ (x, aux) \leftarrow \mathsf{Gen}(par)\right] \ = \ 1.$$
- (Information-theoretical) Soundness*: If $x \notin L$, no cheating prover can convince a correct verifier with not with more than an exponentially small probability. For this, we will now assume that par, the tuple of parameters, contains a parameter σ that determines how small this probability should be. Then we can write: For all efficient interactive algorithms* P^**, all valid parameters par, all $x \notin L$, and all aux, the following holds:*
  $$\mathsf{Pr}\left[Exp_{\mathsf{P}^*,\mathsf{V}}((par, x, aux), (par, x)) = 0\right] \ \leq \ 2^{-\sigma}.$$

$\diamond$

To define *computational soundness*, one has to be more careful where x and *aux* come from than in the information-theoretic version. The best way to define this is to let them be chosen by the cheating prover, i.e., in polynomial time, but not with the correct generation algorithm. This gives the following definition:

Definition 15.2 (Computational Soundness) *Assume that we have a proof system according to Definition 15.1 except for the soundness, and we have only one parameter $par = n$. Then we say that this system has* computational soundness *if for all efficient algorithms* Gen^* *and* P^**, where* P^* *is interactive, the following holds:*

$$\mathsf{Pr}\left[x \notin L \ \wedge \ Exp_{\mathsf{P}^*,\mathsf{V}}((n, x, aux), (n, x)) = 0;\ (x, aux) \leftarrow \mathsf{Gen}^*(n)\right] \ \textit{is negligible in } n.$$

$\diamond$

15.2.2 Proofs of Knowledge

Before looking at the current definition of proofs of knowledge, it may be useful if you try for yourself for a while how you would define such a notion, so that you see the problems. Some approaches are discussed in the sequel, followed by the actual definition.

One approach at defining knowledge might be to say: If a machine knows a certain value w, this w should be written down somewhere in the machine's memory. Then soundness of a proof of knowledge would be something like "no machine P^* can convince V without having w written down correctly in a certain place on its tapes". This is fine for correct machines P, but difficult for cheating machines. For example, a cheating machine P^* might have w written down in reverse order, or encrypted with a simple encryption system. Then P^* will certainly still be able to convince a verifier by simulating P and always looking up or decrypting the appropriate parts of w. Thus soundness in this narrow sense would never be fulfilled.

The next approach might be to say: "Any P^* that can convince V has w written down in some encoding". But now, what is an arbitrary encoding? E.g., having primes p and q written as $N = pq$ is a valid encoding in the information-theoretic sense, but not what we want: We would not want to say that someone knows the factors p and q if in fact he only knows N.

Thus we want to add a computational aspect to the encoding, somehow like "P^* must have written down something from which w can be derived in polynomial time". Polynomial-time computability is defined for individual input-output pairs, only for an entire function. Here the inputs are what an arbitrary successful P^* has written down. This is still a rather vaguely defined set: E.g., is the program of P^* needed in the input in addition to the content of its tapes?

The following definition is slightly more restricted: It says that deriving w should be possible by using P^* as a black box. I.e., it requires that a polynomial-time algorithm K exists that finds out w from the initial state of P^*, but without examining the internal details of P^*. More precisely, K can interact with P^* in the normal way, and reset P^* to a previous state, including the state of its random tape. The algorithm K is called the *knowledge extractor*.

The final question is what to do with provers P^* that are only successful with a small probability. The intuition is that a prover that does not know a witness w should only be successful with negligible probability. Therefore the extractor K should be successful for all provers P^* whose success probability is at least the inverse of any polynomial. However, it is intuitively clear that the smaller the success probability of P^* is, the more difficult extracting will get: The extractor will not have much chance unless it observes at least one case where P^* is successful. The current definition handles this by allowing K a running time inversely proportional to the success probability of P^*.

The following definition is due to Bellare and Goldreich. Except for the soundness, it is quite similar to Definition 15.1, but we now have a relation R instead of the language L.

Definition 15.3 *An* interactive proof-of-knowledge system *has the following components:*

- *The parameters are a binary relation R and certain security parameters, which we collectively call par. Let L_R denote the language of all values x that have a witness w, i.e.,*

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

- *There are roles and subprotocols as in Definition 15.1.*[3]

[3] We could live without the generation algorithm Gen in this definition. (Compare how completeness is defined here and in Definition 15.1). However, it will be needed again in the computational zero-knowledge property.

Again we denote the event that V *outputs* b *by*

$$Exp_{\mathsf{P},\mathsf{V}}((par, x, aux), (par, x)) = b.$$

The requirements are:

- Correct generation*: For all valid parameters par, the output* (x, w) *of* $\mathsf{Gen}(par)$ *is always a value* x *with its witness, i.e.,* $(w, x) \in R$*.*
- Completeness*: A correct prover whose input* w *is a witness for the common input* x *can always convince a correct verifier. More precisely, for all valid parameters par we have*

$$\mathsf{Pr}\left[Exp_{\mathsf{P},\mathsf{V}}((par, x, w), (par, x)) = 0;\ (x, w) \leftarrow \mathsf{Gen}(par)\right] = 1.$$

- Weak soundness *(see the remarks below): We now assume that par, the tuple of parameters, contains a computational parameter* n*. There should exist a probabilistic polynomial-time algorithm* K*, called* knowledge extractor*, that can interact with and reset one other machine* P^**, and a polynomial pol such that the following holds: For all probabilistic polynomial-time interactive algorithms* P^**, all strings aux and all* $x \in L_R$ *we define* $p_{\mathsf{P}^*}(par, x, aux)$ *to be the probability that* V *accepts on input* (par, x) *in interaction with* P^* *using aux, i.e.,*

$$p_{\mathsf{P}^*}(par, x, aux) := \mathsf{Pr}\left[Exp_{\mathsf{P}^*,\mathsf{V}}((par, x, aux), (par, x)) = 0\right].$$

Then on input (par, x)*, and interacting with this* P^**, the knowledge extractor* K *should output a witness* w *for* x *in expected time*

$$\frac{pol(n)}{p_{\mathsf{P}^*}(par, x, aux)}.$$

◇

Remarks and Variants The soundness in the above definition is called weak because only strings $x \in L_R$ are considered. For instance, this means that we don't bother if a cheating prover can convince an honest verifier that he knows non-trivial factors p and q of a prime number N, or a square root of a number $x \notin QR_N$. Weak soundness is sufficient in several applications, in particular in identification schemes, where x and w are initially chosen by an honest participant. We define *strong soundness* to mean weak soundness plus the requirement that prove, together with a suitable generation algorithm, is a proof of language membership for L_R.[4]

15.3 Defining Zero-Knowledge

Zero-knowledge is, as mentioned, a property of an individual machine, in our case the prover. Just as with knowledge, the definition deserves some motivation. (But the two notions of knowledge and zero-knowledge have not much in common.)

Some Introduction

The first point to notice is that the definition is not about keeping a specific secret hidden, but it is supposed to capture that the prover gives the verifier "no knowledge at all". The reason for this is that one wants to use zero-knowledge proofs in a modular way:

[4] It is tempting to simply extend the definition of weak soundness to all x. However, for $x \notin L_R$, this does not work if there is any error probability because the knowledge extractor can never find a witness.

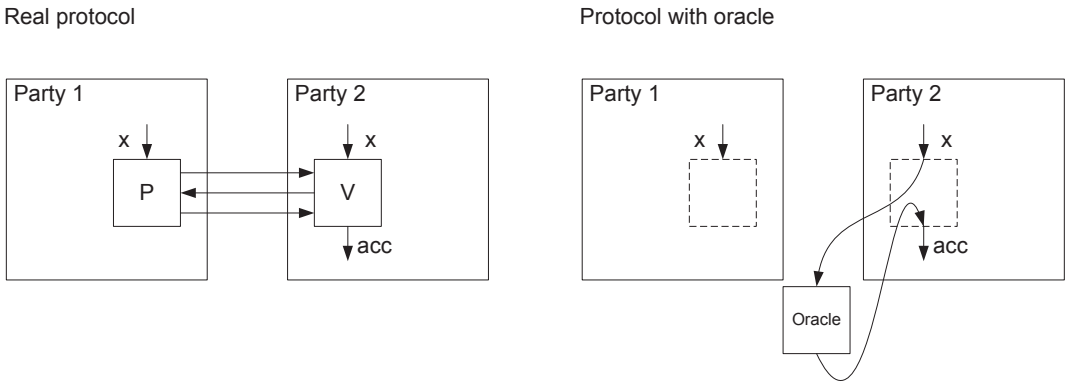


Figure 15.1: Sketch of modular proof of a protocol that uses a zero-knowledge proof as a subprotocol

First, one designs a surrounding protocol, in our examples a commitment scheme, and simply assumes that in the place where the zero-knowledge proof is needed, some trusted oracle tells the verifier $b = 0$ or $b = 1$, see Figure 15.1. It is typically much easier to prove security of this simplified surrounding protocol than of the real one, in particular for the party who plays the prover.

Second, one hopes that the real protocol, i.e., with the real zero-knowledge proof instead of the oracle, is also secure. Intuitively, it makes no significant difference whether the party who plays the verifier gets an entire view of a run of the zero-knowledge proof or only the final result.[5] So how can the verifier's view in a zero-knowledge proof be "no knowledge at all"? The basic idea is: Something is no new knowledge if one could easily have computed it alone. Thus we want to define that the verifier could easily have computed his view in the zero-knowledge proof alone, given only x and no interaction with the prover. This may sound strange: If one can compute such views without knowing the prover's secret, where is this secret used at all? Can't anyone else cheat the verifier by also computing such views? The point is in the order of computation: Typically a zero-knowledge proof is a game of questions (often called challenges) and answers. It will only be possible to answer arbitrary questions correctly if one knows a secret. However, one can somehow make up pairs of answers together with suitable questions even without the secret.

Example Imagine that the questions (asked by the verifier) were random quadratic residues x modulo a number N, and the correct answers were roots y of x. Then one should hope that only someone who knows the factors of N can answer all questions correctly. Nevertheless, the view, seen afterwards, only consists of pairs (x, y) with $y^2 = x \bmod N$, which one can easily make up oneself by choosing y first. Note, however, that this is not yet a real zero-knowledge proof because it only considers an honest verifier: A cheating verifier could choose quadratic residues where he already knows one root, and then use the second root obtained from the prover to factor N. We can already see from the example that the point of "computing the view alone" is not to compute one particular view, but to generate possible views with the correct probability distribution. Then an outside observer (or surrounding protocol) to whom the verifier later gives a view cannot distinguish

[5]Now one can also see why the fact that the verifier is convinced after the proof, but not before, is no knowledge at all: The zero-knowledge property is only defined for an honest prover. The honest prover should not try to prove wrong statements. Thus if the prover is honest, the verifier's result is always $b = 0$, i.e., it is not even one bit of new knowledge.

whether it is a real view or one computed by the verifier himself. This statement exists in three versions:

- *Perfect zero-knowledge*: This is the usual word for information-theoretical zero-knowledge without error probability. It is defined exactly as explained so far, i.e., the correct views and those that the verifier computes himself have exactly the same distribution.
- *Statistical zero-knowledge*: Here the probability distributions may have very small differences.
- *Computational zero-knowledge*: Here it is not required that the distributions are actually equal. It is only required that nobody can distinguish them in polynomial time. This roughly means that someone who gets either a correct view or a simulated view (each drawn according to their probability distributions) cannot guess much better than with probability 1/2 what he got.

Finally, we have to define what it means that "the verifier could compute something". This will be done by requiring that a machine S, the simulator, exists, which actually carries out this computation, given only the input of the verifier. We must also consider cheating verifiers V^*. Their views will usually have a different probability distribution than those of honest verifiers because already their own questions are distributed differently. Thus there will be a different simulator $\mathsf{S}_{\mathsf{V}^*}$ for each V^*, see Figure 15.2. However, when proving a zero-knowledge property in practice, one usually makes a black-box reduction. This means that one constructs one global algorithm S that performs the simulation for any verifier V^* by using V^* as a black box, including resetting V^* to a previous state (i.e., just like a knowledge extractor can use a cheating prover P^*).

Definition of Indistinguishability

To make the zero-knowledge definition modular, one typically defines indistinguishability of ensembles of probability distributions, like real views and simulated views, first. An ensemble of probability distributions is defined as a family

$$(Prob_i)_{i \in I}$$

of probability distributions. This means that a set I, called *index set*, is given, and for each $i \in I$, there is a probability distribution (over some other, unnamed set). In our case, all sets will be subsets of the bitstrings.

Example Let's see that we really need ensembles of probability distributions and not just individual distributions, even for a specific verifier V^*. In the above example the index set I would be the numbers $N = pq$, and one would try to distinguish the probability distribution of real views (x, y) for this N and simulated views (x, y) for the same N. The success probability hopefully gets very small if N is large enough. Indistinguishability is defined for two ensembles over the same index set.

Definition 15.4 *Let two ensembles $(Prob_i)_{i \in I}$ and $(Prob^*{}_i)_{i \in I}$ of probability distributions over bitstrings for one index set $I \subseteq \{0,1\}^*$ be given.*

- *They are called* perfectly indistinguishable *if they are equal.*

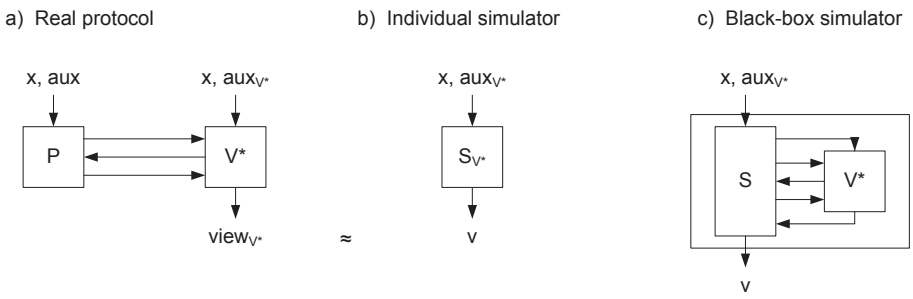


Figure 15.2: Notion of a simulator

- *They are called* computationally indistinguishable *for a certain generation algorithm* Gen^* *that generates indices* $i \in I$*, given parameters par, if the following holds for all probabilistic polynomial-time algorithms* Dist *(the distinguisher): For all* $i \in I$*, one defines probabilities*

$$\begin{aligned} Pr_i &:= \mathsf{Pr}\left[\mathsf{Dist}(i,v) = 0;\ v \leftarrow Prob_i\right], \\ Pr_i^* &:= \mathsf{Pr}\left[\mathsf{Dist}(i,v) = 0;\ v \leftarrow Prob^*{}_i\right], \end{aligned}$$

and their difference

$$\Delta_i := |Pr_i - Pr_i^*|.$$

Now one requires that the differences are negligible on average, i.e.,

$$\mathsf{E}(\Delta_i;\ i \leftarrow \mathsf{Gen}^*(par)) \quad \textit{is negligible.}$$

Here, $\mathsf{E}(x;\ \ldots)$ *denotes the expected value of a random variable* x *in the probability space defined after ";", similar to the notation* $\mathsf{Pr}\left[(pred;\ \ldots)\right]$.[6]

◇

Definitions of Zero-Knowledge

The final addition to be made to the explanation is that a cheating verifier V^* in a zero-knowledge protocol may have an auxiliary input aux_{V^*} even if an honest verifier has no such thing. This corresponds to a-priori knowledge of the verifier. The most important cases of such a-priori knowledge are that the prover uses the same secret in a zero-knowledge proof and a surrounding protocol, or in several successive zero-knowledge proofs. For arbitrary a-priori knowledge, the view of the verifier should give no new knowledge.

We now present the details of the definition.

Definition 15.5 *Let an interactive proof system with generation algorithm be given, i.e., a protocol according to Definition 15.1.*

- *It is called* perfectly zero-knowledge *if the following holds: For all probabilistic polynomial-time interactive algorithms* V^* *(the cheating verifier), there exists a probabilistic polynomial-time algorithm* $\mathsf{S}_{\mathsf{V}^*}$*, called* simulator*, such that the following two ensembles are* perfectly indistinguishable*:*

[6] There is also a definition for computational indistinguishability where the distinguisher actually tries to guess which of the two probability distributions v was drawn from. However, the above definition better fits our needs.

– *The index set is the set of possible inputs, i.e., the set I of tuples $i = (par, x, aux, aux_{\mathsf{V}^*})$ with $(x, aux) \in [\mathsf{Gen}(par)]$.*
– *The two probability distributions for such an i are V^*'s view in the experiment*

$$Exp_{\mathsf{P},\mathsf{V}^*}((par, x, aux), (par, x, aux_{\mathsf{V}^*}))$$

and the output of $\mathsf{S}_{\mathsf{V}^*}(par, x, aux_{\mathsf{V}^*})$.

- *For* computational indistinguishability*, we only consider auxiliary information aux_{V^*} generated from x and aux in a feasible way, i.e., by an arbitrary probabilistic polynomial-time algorithm G. This is natural because we assume that this information results from previous executions of protocols using x and aux, and both the honest party and the attacker are now computationally restricted. Given such an algorithm G, the entire generation algorithm $\mathsf{Gen}_{\mathsf{G}^*}$ that generates an index i from given parameters is defined by*

$$(x, aux) \leftarrow \mathsf{Gen}(par); aux_{\mathsf{V}^*} \leftarrow \mathsf{G}(x, aux).$$

Now we require that the two ensembles defined as above are computationally indistinguishable for any such generation algorithm $\mathsf{Gen}_{\mathsf{G}^*}$.

◇

15.4 Proving Quadratic Residuosity

In this section we look at a concrete zero-knowledge proof system for language membership. The prover P wants to prove to the verifier V that a number x is a quadratic residue modulo another number N. The language characterizing the goal of the proof is therefore

$$L = \{(N, x) | x \in QR_N\}.$$

Typically such a proof system is used for composite N, because if N is prime, the verifier can decide quadratic residuosity by himself, but the proof system works in all cases. The auxiliary information that the prover needs is a square root of x, i.e.,

$$aux = y \text{ with } y^2 = x \bmod N.$$

This is a rather week prerequisite: If a prover knows the factors of N, he can compute such a y for any quadratic residue and therefore use the proof system. However, the proof system even works if the prover does not know these factors and has generated x by choosing y first and squaring it.

15.4.1 Construction

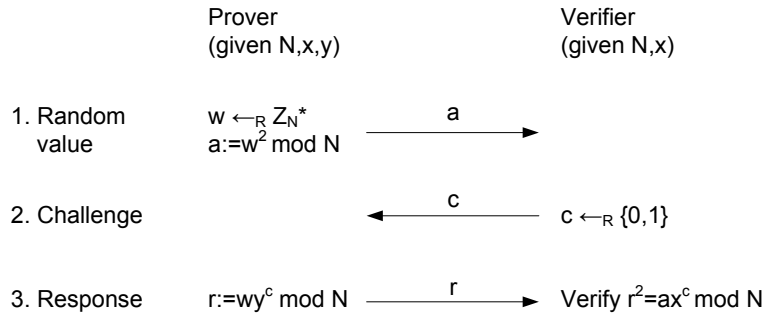
The scheme has the following parameters. The language L describing what is to be proved is

$$L = \{(N, x) | x \in QR_N\}.$$

There is a security parameter σ such that the error probability in the soundness will be at most $2^{-\sigma}$.

It has the following protocols:

- *Generation algorithm*: Any algorithm Gen can be used that in fact outputs pairs $((N, x), y)$ with $y^2 = x \bmod N$, no matter with what probability distribution.
- The *main protocol* consists of the following basic protocol, which is iterated σ times:



  In each iteration, new and independent values w and c are chosen. The verifier accepts if all σ verifications yield true.

The security of this protocol is based on the following ideas:

- *Perfect zero-knowledge*: The only thing the prover sends that depends on the secret y is the response r. This is either a random value w, which cannot possibly tell anything about y, or it is y encrypted with a multiplicative one-time pad w modulo N. In the formal version it will also become clear that a, the public version of the "one-time pad", does not give any new information in this case.
- *Soundness*: If x is not a quadratic residue, then at most one of the values a and ax can be a quadratic residue. Hence for at most one of them an acceptable response r exists. Thus in each iteration, the verifier will only accept with probability $1/2$, and overall only with probability $2^{-\sigma}$. Now we show the security in more detail.

15.4.2 Security of the Construction

Theorem 15.1 *The above construction is a* perfect zero-knowledge proof system. □

Proof. In the sequel we will prove that the above construction provides completeness, soundness, and perfect zero-knowledge.

Completeness Here both parties are honest, and the prover's input fulfills $y^2 = x \bmod N$. We have to show that the verifier always accepts. This can easily be seen if we substitute the definition of r into the verification equation:

$$r^2 = (wy^c)^2 = w^2(y^2)^c = ax^c \bmod N.$$

Soundness Now a pair (N, x) with $x \notin QR_N$ is given, and an arbitrary prover P^* with an arbitrary auxiliary input aux^*. In each iteration of the protocol, P^* must first send a value a.

- If $a \notin QR_N$, no value r with $r^2 = a$ exists, and thus no acceptable response for $c = 0$.

- If $a \in QR_N$, then $ax \notin QR_N$ (otherwise $x = (ax) \cdot x^{-1} \in QR_N$ would follow). Hence no value r with $r^2 = ax$ exists, and thus no acceptable response for $c = 1$.

Hence no matter how P^* chooses a, it can respond correctly to at most one of the two possible challenges c. Moreover, at the time when P^* chooses a, it cannot know which challenge it will get. Hence it passes each iteration with probability at most $\frac{1}{2}$, and all iterations together with probability at most $2^{-\sigma}$.

Perfect zero-knowledge Here we have to construct a simulator that can produce correct-looking views (a, c, r, v^*) of the protocol, where v^* denotes the final output of V^* (which can contain the entire internal view of V^* without loss of generality). We start with two simpler tasks to make the ideas clearer, but we will only show that the views have the correct probability distribution in Part (d) for our final simulator.

(a) Honest-verifier zero-knowledge for one iteration: Here we construct a specific simulator S_V that only works for the correct verifier V in one iteration of the protocol. The simulator's input is only (N, x). Hence it cannot compute the views in the correct order. Instead, we let it start at the end:

S_V:
1. It chooses $r \leftarrow_\mathcal{R} \mathbb{Z}_N^*$.
2. It chooses $c \leftarrow_\mathcal{R} \{0, 1\}$ just like the correct V.
3. Now it solves the verification equation for the remaining variable a: $a := r^2 \cdot x^{-c}$.

(b) Arbitrary verifier V^*, one iteration: Compared with S_V, we now have a problem in Step 2: We do not know how V * chooses c. In particular, it may choose c as a function of a. Our simulator cannot do that, because it has to compute a last. We deal with this as follows:

$\mathsf{S}_{\mathsf{V}^*}^{(1)}$
1. Choose r, c, a like S^V.
2. Start V^*, send a to it, and wait until it answers with a challenge c^*.
3. If $c^* = c$, then send r to V^* and get its final output v^*. Output $v = (a, r, c, v^*)$. Else try again (from Step 1).

This means that $\mathsf{S}_{\mathsf{V}^*}^{(1)}$ first simply guesses some c. Then, when it has computed a, it checks what value c^* the cheating verifier V^* would have output for this a. (Recall that the simulator can use V^* as a subprotocol and reset it.) If $c^* = c$, we have a consistent view and output it. Otherwise, this view is thrown away and another attempt is made.

(c) Overall simulator $\mathsf{S}_{\mathsf{V}^*}$: Essentially, it treats each iteration like $\mathsf{S}_{\mathsf{V}^*}^{(1)}$. It must only take care that V^* might act differently in each iteration, possibly depending on the results of the previous iterations. Hence at the end of each iteration i, it also stores the resulting state $state_i$ of V^*. In each attempt to produce a view for iteration $i+1$, it resets V^* to $state_i$. Moreover, it need not output intermediate values v_i^* for each round, but only the final v^*.

(d) Correct distribution: We have claimed perfect zero-knowledge. Thus we have to show that the outputs of $\mathsf{S}_{\mathsf{V}^*}$ have exactly the same probability distribution as the views of V^* in interaction with the real prover (for the same values N, x, y, and aux_{V^*}). We show this even for V^* with any fixed content rand of its random tape; then it obviously also holds on average over all values rand. Thus we only need to consider deterministic verifiers. Thus in each iteration of the real protocol, the only random choice is w_i, and $view_i$ and $state_i$ are a function of $state_{i-1}$

and w_i. (We have now added an index i to each variable.) Let the function be f. We first show that any simulated view v_i and its $state_i$ are also $f(state_{i-1}, w_i)$ for $w_i := r_i/y^{c_i}$. (It does not matter that the simulator does not know w_i; this is only our proof technique.)

1. a_i should be w_i^2, and in fact, $w_i^2 = (r_i/y^{c_i})^2 = r_i^2/x^{c_i} = a_i$ by the choice of a_i.
2. Now c_i is the correct challenge from V^* for this situation because the simulator explicitly verified this.
3. r_i should be $w_i y^{c_i}$. In our case, this is clear by the choice of w_i.
4. Now $state_i$ is produced by the same V^* in the simulation as in the real run, based on the same.

It remains to show that the values w_i, which determine the entire view for this iteration, occur with the correct probability in the simulation, i.e., that they are uniformly distributed. In the simulation, r_i/y^{c_i} where r_i and c_i are chosen uniformly. Thus each w_i occurs for exactly one pair $(r_i, c_i = 0)$ and one pair $(r_i, c_i = 1)$. One of these cases is that with $c_i^* = c_i$, so in one case the view is thrown away and in the other case it is be output. Hence we have both shown that each attempt at producing v_i is successful with probability $1/2$ (which shows that the running time of the simulator is on average twice that of the real protocol) and that the retained views have the correct probability distribution.

The overall proof is by induction: We have just shown that when $state_{i-1}$ is distributed correctly, then so are v_i and $state_i$. This clearly yields that the overall view and the final output of V^* are distributed correctly.

■

Remark This protocol is also a proof of knowledge that the prover knows y. The basic idea for the knowledge extractor is the following: If it can get correct responses to both $c = 0$ and $c = 1$ for the same value a, then these should be $r_0 = w$ and $r_1 = wy$, and thus he can compute y from them as r_1/r_0. More precisely, it is only verified that the responses fulfill $r_0^2 = a$ and $r_1^2 = ax$, but this also implies that $(r_1/r_0)^2 = ax/a = x$ and thus $y = r_1/r_0$ is the correct knowledge.

Cryptography (SS2006) Prof. Michael Backes

Errata

Version: 07/16/2006 Saarland University

Chapter 1: Historical Ciphers

none.

Chapter 2: Stream Ciphers

none.

Chapter 3: Block Ciphers

- **Page 5, caption of Figure 3.5:**
  Old: For a binary string $x_0x_1x_2x_3x_4x_5x_6x_7$ the ...
  New: For a binary string $x_0x_1x_2x_3x_4x_5$ the ...
- **Page 5, caption of Figure 3.5:**
  Old: ...row x_0x_7 and column $x_1x_2x_3x_4x_5x_6$. This ...
  New: ...row x_0x_5 and column $x_1x_2x_3x_4$. This ...
- **Page 6, Figure 3.8:**
  Old: Input of PC-2 should be 56 bits instead of 28 bits (2 times)
  New: Output of PC-2 should be 48 bits instead of 28 bits (2 times)
- **Page 11, Line 2:**
  Old: ... thus computing $\mathsf{E}(K_1, \mathsf{E}(K_2, m))$. While ...
  New: ... thus computing $\mathsf{E}(K_2, \mathsf{E}(K_1, m))$. While ...
- **Page 11, Line 22:**
  Old: $\ldots := \mathsf{E}(K_1, \mathsf{D}(K_2, \mathsf{E}(K_3, m)))$.
  New: $\ldots := \mathsf{E}(K_3, \mathsf{D}(K_2, \mathsf{E}(K_1, m)))$.
- **Page 11, Line 24:**
  Old: $\mathsf{D}^{\mathsf{3DES}}((K_1, K_2, K_3), m) := \mathsf{D}(K_3, \mathsf{E}(K_2, \mathsf{D}(K_1, m)))$.
  New: $\mathsf{D}^{\mathsf{3DES}}((K_1, K_2, K_3), c) := \mathsf{D}(K_1, \mathsf{E}(K_2, \mathsf{D}(K_3, c)))$.

Chapter 4: PRPs, PRFs, Semantic Security

- **Page 2, Line 18:**
  Old: ... for all (sequences of) adversaries A.
  New: ... for all (sequences of) efficient adversaries A.
- **Page 4, Line 16:**
  Old: $|\mathsf{Pr}[Coll]| = \left|\frac{1}{|\mathcal{X}|} + \ldots + \frac{q-1}{|\mathcal{X}|}\right|$
  New: $|\mathsf{Pr}[Coll]| \leq \left|\frac{1}{|\mathcal{X}|} + \ldots + \frac{q-1}{|\mathcal{X}|}\right|$
- **Page 6, Line 7:**
  Old: ... Deterministic Counter Mode (detCTR
  New: ... Deterministic Counter Mode (detCTR)

- **Page 7, Figure 4.1, A's input:**
  Old: Cb'
  New: $c_{b'}$

- **Page 7, Line -5:**
  Old: ... equals $\frac{1}{2} + Adv^{\mathsf{CT-only}}[\mathsf{A}, \mathsf{E}]$, which ...
  New: ... equals $\frac{1}{2} + \frac{1}{2} \cdot Adv^{\mathsf{CT-only}}[\mathsf{A}, \mathsf{E}^{\mathsf{detCTR}}]$, which ...

- **Page 7, Line -1:**
  Old: $\ldots Adv^{\mathsf{CT-only}}[\mathsf{A}, \mathsf{E}]$
  New: $\ldots Adv^{\mathsf{CT-only}}[\mathsf{A}, \mathsf{E}^{\mathsf{detCTR}}]$.

- **Page 8, Line -6::**
  Old: $Adv^{\mathsf{CPA}}[\mathsf{A}, \mathsf{E}] := \left| \mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{CPA}}(b) = 0\right] - \mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{CPA}}(b) = 1\right] \right|.$
  New: $Adv^{\mathsf{CPA}}[\mathsf{A}, \mathsf{E}] := \left| \mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{CPA}}(0) = 1\right] - \mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{CPA}}(1) = 1\right] \right|.$

Chapter 5: MACs and Hash Functions

- **Page 9, Line 13:**
  Old: $\ldots H_{i+1} := \mathsf{F}(m_i, H_i)$ for $i = 2, \ldots, r$,
  New: $\ldots H_{i+1} := \mathsf{F}(m_i, H_i)$ for $i = 1, \ldots, r$,

Chapter 5*: Secure Channels and Key Management

none.

Chapter 6: Basic Number Theory Facts

- **Page 4, Line 4:**
  Old: $h_1 = g^{p+1}/4 \bmod p$ and $h_2 = -g^{p+1}/4 \bmod p$
  New: $h_1 = g^{p+1/4} \bmod p$ and $h_2 = -g^{p+1/4} \bmod p$

Chapter 7: Public-key Encryption, Diffie-Hellman, ElGamal

none.

Chapter 8: The Cramer-Shoup Encryption Scheme

- **Page 2, Line -7:**
  Old: ..., he would also know the ciphertext.
  New: ..., he would also know the plaintext.

- **In Figure 8.1:**
  Old: Adv. B
  New: Adv. $\mathsf{A}_{\mathsf{DDH}}$

- **In Figure 8.1:**
  Old: Adv. A
  New: Adv. A_{CS}

- **Page 6, Line 6:**
  Old: ... does not know r
  New: ... does not know y

- **In Figure 8.2:**
  Old: Adv. B
  New: Adv. $\mathsf{A}_{\mathsf{DDH}}$

- **Page 10, Line 9:**
  Old: ... and can therefore contain no new information ...
  New: ... and can therefore contain new information ...

Chapter 9: One-way Functions and the RSA Trapdoor Permutation

- **Page 3, Line 14:**
  Old: (DLog is an OWF)
  New: ($\mathsf{F}^{\mathsf{DLog}}$ is an OWF)

- **Page 5, Line 5:**
  Old: $\varphi(pq) = (p-1)(q-1)$
  New: $\varphi(pq) = \varphi(p)\varphi(q)$

Chapter 9: The RSA Trapdoor Permutation (cont'd)

- **Page 2, Line 3 and 4 (2 times):**
  Old: $\sqrt{n}$
  New: $\sqrt{N}$

- **Page 4, Line 15:**
  Old: $\mathsf{Gen}(k)$
  New: $\mathsf{Gen}(n)$

- **Page 4, Line 26:**
  Old: $\ldots \| R \| \ldots$
  New: $\ldots \| r \| \ldots$

Chapter 10: Digital Signature Schemes

- **Page 4, Line 5:**
  Old: $1 \leq j \leq 2$
  New: $0 \leq j \leq 1$

- **Page 4, Line -7:**
  Old: $r \leftarrow_{\mathcal{R}} \{1, \ldots, p-1\}$
  New: $r \leftarrow_{\mathcal{R}} \mathbb{Z}^*_{p-1}$

- **Page 5, Line 12:**
  Old: $t = \mathsf{DLog}_s g^a h^{-s}$
  New: $t = \mathsf{DLog}_s g^m h^{-s}$

- **Page 5, Line 13:**
  Old: $h^s s^t = g^a$
  New: $h^s s^t = g^m$

- **Page 5, Line 27:**
  Old: $\gcd(j, p-1)$
  New: $\gcd(j, p-1) = 1$

- **Page 9, Line -2:**
  Old: $\mathsf{Pr}[\mathsf{A} \text{ wins}] = \ldots$
  New: $\mathsf{Pr}[\mathsf{B} \text{ wins}] = \ldots$

Chapter 11: Certificates

none.

Chapter 12: Authentication Methods, SSL, and other Important Protocols

- **Page 1, Line -9:**
  Old: ... if an attackers gets ...
  New: ... if an attacker gets ...
- **Page 3, Line -2:**
  Old: $cert_S, g^y, \mathsf{E}(K, \mathsf{S}(sk_S, (g^x, g^y)))$
  New: $cert_B, g^y, \mathsf{E}(K, \mathsf{S}(sk_B, (g^x, g^y)))$
- **Page 6, Line 9:**
  Old: $z_2 = z_1^b \bmod p$
  New: $z_2 = g^b \bmod p$

Chapter 13: Commitment Schemes

none.

Chapter 14: Secret Sharing

none.

Chapter 15: Zero-Knowledge Proofs

none.