TECHNISCHE
UNIVERSITÄT
DARMSTADT

Technical University Darmstadt

– Department of Computer Science –

Introduction to Cryptography

A Summary of the Course Contents

Author:

Jonas Weßner

Teaching professor	:	Prof. Dr. Marc Fischlin
Semester	:	Winter semester 2022/2023

CONTENTS

LIST OF ABBREVIATIONS

MAC message authentication code

1

INTRODUCTION

Cryptography is the science of the processes for securing information, data and systems.

Typically we talk about certain key concepts, which differ from book to book. In this course, we define them as the following:

C Confidentiality: Attackers cannot read a message.

I Integrity: Attackers cannot modify the content of a message (in a narrow sense), cannot fake the message's sender address (authenticity) and receivers cannot claim, they have received the message (non-repudiation).

A Availability: A system is always functional (less relevant in this course)

Often in this lecture, we take a three-step approach to cryptographic processes:

1. **Abstract Object:** We define what interface our cryptographic process should work with. Often, we define one process for encryption, one for decryption and maybe others like one for key generation. We do not yet define their implementation.
2. **Security Model:** We define how our abstract object is used and in which cases our abstract object is secure. Questions to be answered are e.g. when is an attack successful and what is information the attacker is allowed to possess without introducing insecurity?
3. **Security Proof a Specific process:** When given a concrete implementation of our security model, we check the security of that model. For example, we prove that an attacker must calculate at least n hash sums to compromise the system.

Cryptographic processes can be classified in a matrix as follows:

	Private-Key	**Public-Key**
Confidentiality	Symmetric processes	Asymmetric processes
Integrity	Message Authentication Codes (MACs)	Digital signatures

Furthermore, there are elementary processes serving as building blocks to build cryptographic processes:

- Pseudo random number generators (PRNGs)
- Hash functions

- Blockcipher
- Number theory

2

PRIVATE-KEY CRYPTOGRAPHY

2.1 ONE TIME PAD

The One Time Pad works as follows:

1. Alice has a message $m \in \{0,1\}^n$ to be sent to Bob. Furthermore, Alice and Bob possess a common key $k \in \{0,1\}^n, n \in N$. n is called the security parameter.
2. Alice computes a cipher text $c \in \{0,1\}^n$ as $c = m \oplus k$ and sends it to Bob.
3. Bob reconstructs the plain message as $m = c \oplus k = m \oplus k \oplus k = m \oplus \{0\}^n$.

Therefore, the abstract object for this cryptographic process consists of the following functions[1]:

- $kGen(1^n) \rightarrow k \in \{0,1\}^n$
- $enc(m,k) \rightarrow c, \quad |m| = |k| = |c| = n$
- $dec(c,k) \rightarrow m \quad || \quad \perp$

The functional correctness is then defined as follows:
For all security parameter $n \in N$, for all messages m, for all keys $k \leftarrow kGen(1^n)$, for all keys $k \leftarrow kGen(m,k)$ and for all ciphertexts $c \leftarrow enc(m,k)$ applies $dec(c,k) = m$. That means that we must be able to decrypt all encrypted messages for all possible input parameters.

Security of Shannon's OTP

Independently of a bit m_i in the message m, the OTP flips this bit with the probability of $\frac{1}{2}$ creating a distribution of ciphtertexts c that is independent of the messages m. And because m and c are independent, it is intuitive that knowing c cannot reveal anything about m. We can also prove this mathematically because the probability for two independent events to take place is the product of the individual probabilities: $Pr[M = m|C = c] = \frac{Pr[M=m \wedge C=c]}{Pr[C=c]} = \frac{Pr[M=m] \cdot Pr[C=c]}{Pr[C=c]} = Pr[M = m]$.

1 The symbol $\perp$ is indicating an error.

In practice, perfect security is not feasible because this would mean that for transferring each message m a key k with at least the same length $|k| \geq |m|$ would have to be transferred (see definition of perfect security in section 4.2). For this reason, we define a weakened security definition, which comes in two variants – Concrete security and asymptotic security. In this course, we look at asymptotic security.

Concrete Security:	**Asymptotic Security**
A process is $(t, \mathcal{E})$-secure if no attacker A can break it with at most t steps with a probability of $\mathcal{E}$.	A process is secure, if no efficient (polynomial limited by security parameter n) algorithm breaks it with non-negligible (less than $\frac{1}{poly}$) probability.

The relevant terms are defined as follows, where n is the security parameter (e.g. key size):

- *Efficient Algorithm:* Algorithm with polynomial runtime with regard to n.
- *Non-negligible Probability:* An inversely polynomial probability ($\frac{1}{poly}$) with regard to n.
- *Negligible Probability:* Smaller than an inversely polynomial function (i.e. less than non-negligible) $\leq \frac{1}{poly}$. Mathematically speaking, a function $\mathcal{E} := N \to R$ if there can be found a value $limit \in N$, such that for all polynomial functions $poly$ applies $\mathcal{E}(n) \leq \frac{1}{poly} \quad | \quad n \geq limit$. This means that, if the security parameter is high enough, its value is smaller than any polynomial function. Stating that a function $\mathcal{E}$ is negligible can be written in three ways: $\mathcal{E} \leq neg(n)$, $\mathcal{E} = neg(n)$ or $\mathcal{E} \approx 0$. Furthermore, if a the function $\mathcal{E}_1$ and the function $\mathcal{E}_2$ differ by a negligible difference, we can write $\mathcal{E}_1 \approx \mathcal{E}_2$.

Examples for Negligibility of probability functions

- $\mathcal{E} = 2^{-n}$ is *negligible* because exponential functions grow faster than polynomial functions.
- $\mathcal{E} = n^{-5}$ is *not negligible* because there are polynomials that have smaller values as n approaches infinity e.g. n^{-6}.
- $\mathcal{E} = \begin{cases} 2^{-n} & \text{if } n \text{ is even} \\ n^{-5} & \text{if } n \text{ is uneven} \end{cases}$ is *not negligible* because for some values (all uneven values) it behaves like a polynomial function, such that e.g. n^{-6} would have greater values regardless of a chosen $limit$.

- $\mathcal{E} = \frac{1}{8}$ is *not negligible* because it does not approach zero and therefore there are many polynomial functions that are smaller than $\mathcal{E}$ for high n

Arithmetic Laws for Negligible Functions

If $mathcal E_1$ and $mathcal E_2$ are negligible functions $mathcal E_1, mathcal E_2 \approx 0$ and q is a polynomial function $q = poly$ and ω ia a non-negligible $\omega \not\approx 0$ function, then applies:

$$\mathcal{E}_1(n) + \mathcal{E}_2(n) \approx 0 \tag{2.1}$$
$$\mathcal{E}_1(n) \cdot q(n) \approx 0 \tag{2.2}$$
$$\omega(n) - \mathcal{E}_1(n) \not\approx 0 \tag{2.3}$$

More intuitively phrased:

(2.1) Executing a negligible function and another negligible function after each other will still result in something negligible.

(2.1) Executing a negligible function for any polynomial number of times will still result in a negligible value in sum because the negligible function approaches zero faster than any polynomial function could grow towards positive values.

(2.1) Subtracting something negligible from something non-negligible does not make a difference – the result is still non-negligible.

2.2.1 *Semantic Security – A Simulation-Based Definition*

A cryptographic process is semantically secure if for all PPT attacker algorithms A, which calculate information $f(m)$ about a length-invariant message m (message of given length n) with the help of the message's ciphertext c with a certain probability, there is a simulator algorithm S, which can compute the same information with a negligibly different probability:

$$Pr[A(1^n, c) = f(m)] \approx Pr[S(1^n) = f(m)]$$

More intuitively speaking, everything which can be computed with the knowledge about the ciphtertext of a message can also be computed with nearly the same precision without knowledge about the ciphertext.

This is the simulation-based definition of asymptotic security.

2.2.2 *Indistinguishability (IND) Security – a Game-Based Definition*

This is a game-based definition of asymptotic security, which is equivalent to the semantic security definition (2.2.1).

A cryptographic process $\mathcal{E}$ is indistinguishable (IND) if for all PPT attackers A applies:

$$Pr[Exp^{IND}_{\mathcal{E},A}(n) = 1] \approx \frac{1}{2}$$

, where $Exp^{IND}_{\mathcal{E},A}(n)$ is defined as the following algorithm:

$$KGen(1^n) \to k \tag{2.4}$$
$$\{0,1\} \to b \tag{2.5}$$
$$A(1^n) \to (st, m_0, m_1)| \quad |m_0| = |m_1| = n \tag{2.6}$$
$$Enc(m_b, k) \to c \tag{2.7}$$
$$A(1^n, st, c) \to a \tag{2.8}$$
$$return \quad a == b \tag{2.9}$$

More intuitive worded, this means that (2.4) a secret key of length n is chosen (2.5) a secret random bit is chosen (2.6) the attacker selects two messages of his choice (2.7) one through b randomly selected message is encrypted (2.8) the attacker looks at the ciphertext of one of its messages and decides which messages ciphertext that is and (2.9) the experiment returns true if the attacker has been able to identify the message and otherwise false. If the output of this experiment is negligible close to $\frac{1}{2}$ for all attackers A, then the attackers have no significant advantage over simply guessing and, therefore, the cryptographic process $\mathcal{E}$ is secure.

An example of an insecure process $\mathcal{E}$ could be an algorithm, which simply flips all bits: $Enc(m,k) =\sim (m)$. The attacker could then just use the Enc algorithm (which is not secret) with any key and get the same result because the Enc algorithm does not use the key. Therefore, he could distinguish messages with the probability of $1 \not\approx \frac{1}{2}$.

An example of a secure algorithm is Shannon's OTP. The reason is that without knowing the key, the attacker cannot distinguish two messages, because the OTP flips bits with the probability of $\frac{1}{2}$, such that the cipthertext is completely random, just like the key.

2.3 Pseudo Random (Number) Generators (PR(N)G)

An algorithm G is a secure PRG, if for all PPT attackers A applies:

$$Pr[Exp^{PRG}_{G,A}(n) = 1] \approx \frac{1}{2}$$

, where $Exp^{PRG}_{G,A}(n)$ is defined as:

$$KGen(1^n) \rightarrow k \tag{2.10}$$
$$\{0,1\} \rightarrow b \tag{2.11}$$
$$G(k) \rightarrow y_0 \tag{2.12}$$
$$\{0,1\}^{|y_0|} \rightarrow \tag{2.13}$$
$$A(1^n, y_b) \rightarrow a \tag{2.14}$$
$$return \quad a == b \tag{2.15}$$

More intuitively speaking, this means if we generate a pseudo-random bitstring y_0 using G and a fully random bitstring y_1, no attacker algorithm can distinguish them.

There is another definition of PRG, where the secret bit b is set from the beginning. In this case the experiment will not return the expression $a == b$ but instead directly return the attacker's guess a. The IND security of the PRG is then defined as follows: The behavior of the attacker (i.e. its return value) is not significantly influenced by whether we give it the output of the PRG or a random bitstring (i.e. it does not depend on the value b):

$$Pr[Exp^{PRG}_{G,A,0}(n) = 1] \approx Pr[Exp^{PRG}_{G,A,1}(n) = 1]$$

As a side note, following Kerckhoff's principle, a potential attacker could also access our PRG. He could, therefore, calculate the pseudo-random strings for all possible input values in $|K_n|$. This means, that a generated value y only has the security of 2^k instead of $2^{|y|}$. However, as $|K_n|$ grows exponentially, such an attacker would not be efficient and is, therefore, asymptotically irrelevant.

2.4 PRGS FOR KEY EXPANSION

We have seen that the OTP is not practicable because for using it we must first exchange a key that is just as long as the message itself. Now that we have looked into PRGs from an abstract point of view, we could use PRGs to generate a long key from a small key and then use that for encryption:

$$KGen(1^n) \rightarrow k \quad | \quad n \in \{0,1\}^n \tag{2.16}$$

$$Enc(k,m) \rightarrow m \oplus G(k) \quad | \quad M_n = \{0,1\}^{l(n)} \tag{2.17}$$

$$Dec(k,c) \rightarrow c \oplus G(k) \quad or \ \perp if \ |c| \ \neq \ l(n) \tag{2.18}$$

Given the assumption that the Generator G is IND-secure, we must now prove that the encryption method $\mathcal{E}$, which utilizes it, is also secure. A suitable method of proof for this is reduction.
The main idea of using reduction for proving the IND security of a cryptographic method $\mathcal{E}$ is the following:

Assume that $\mathcal{E}$ is not IND-secure and, therefore, we could find an attacker algorithm $A_{\mathcal{E}}$ which breaks $\mathcal{E}$. Then construct a new attacker $A_{\mathcal{E}'}$ utilizing $A_{\mathcal{E}}$, which breaks another cryptographic method $\mathcal{E}'$, which is known to be secure. The existence of the newly constructed attacker would then contradict the knowledge about $\mathcal{E}'$. Hence, $\mathcal{E}$ must be IND-secure.

To prove the IND security of the newly constructed cryptographic method $\mathcal{E}$ we reduce it to the IND-security of the utilized PRG as shown in figure 2.1. We have a PRG challenger that challenges us to decide if its output is a random number or the output of the PRG. Then we take the challenger's output y_b and use it to compute a ciphertext c of a randomly chosen message. Assuming the PRG challenger gives us y_0, what we did until now is exactly what a challenger of an IND experiment for $\mathcal{E}$ would do. Because we assume that $\mathcal{E}$ is not IND-secure, the attacker $A_{\mathcal{E}}$ will then be able to efficiently decide what message has been encrypted by us (choose x == a). But in case the PRG challenger gives us y_1, the process of creating c is equivalent to an OTP, which we know is IND-secure, and can therefore not be decided by $A_{\mathcal{E}}$. That means, if there was an efficient attacker for the IND experiment for $\mathcal{E}$, this reduction would build an efficient IND attacker for the PRG. Hence, given the fact the PRG is IND secure, this would contradict. It follows that $\mathcal{E}$ is IND secure if the used PRG is IND secure and otherwise not.

2.5 Indistinguishability under chosen plaintext attack (IND-CPA)

CPA is an attack model in which the attacker can retrieve ciphertexts for arbitrarily chosen plain texts as many times as he wants. But because the attacker must be efficient, he can just retrieve a polynomial amount of ciphertexts. The game-based definition of IND-CPA security is identical to the definition of IND security but allows requesting c_b multiple times for different message pairs (m_0, m_1). But the encryption oracle will have to choose b once in the beginning and then keep it the same for each request.

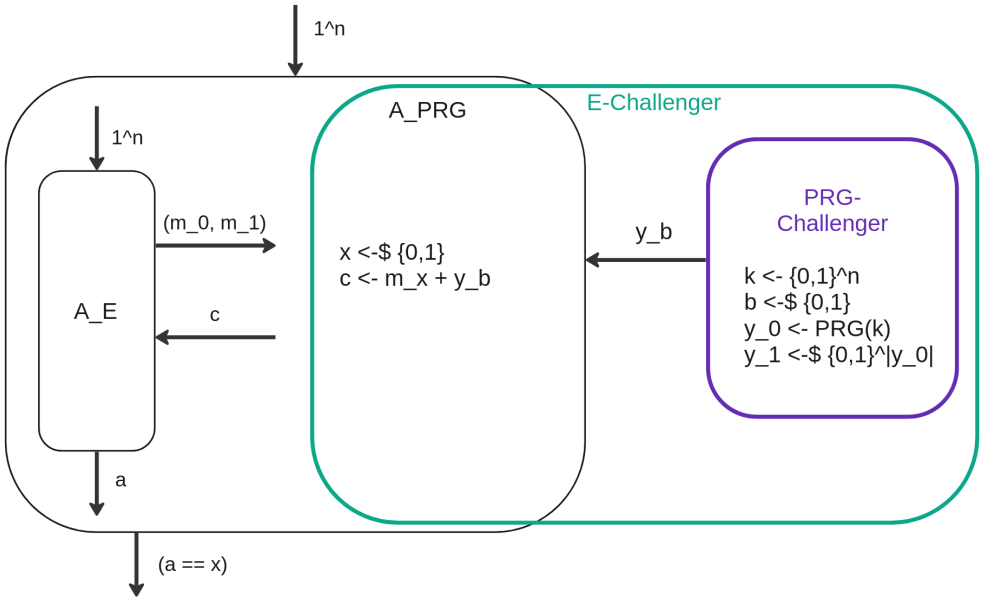

Figure 2.1: Reduction for proving the IND-security of $\mathcal{E}$ if the IND-Security of PRG is given.

Every IND-CPA-secure method is also IND-secure. But IND-secure methods are not automatically IND-CPA secure. For example, the OTP with the same key for each message is IND secure but not IND-CPA secure: An attacker could request the ciphertext for two message pairs $(m_0, m_1) \rightarrow c_1$ and $(m_0, m_2) \rightarrow c_2$. The attacker knows that if $b = 0$, the oracle has encrypted m_0 two times and will therefore return 0 if $c1 == c2$. If $b = 2$, the oracle must have encrypted two different messages, which must because of correctness be mapped to different ciphertexts. Therefore, the attacker returns 1 if $c_1 \neq c_2$.

Generally speaking, deterministic encryptions[2] can never be IND-CPA. The reason for this is that the adversary could e.g. query (m_0, m_0) at the ENC oracle at first to receive c_0 and then could query (m_0, m_1) at the ENC oracle to receive c_b. Then the adversary can simply compare c_b with c_0 to see whether ENC has encrypted m_0 or m_1 to get to c_b. Therefore, only non-deterministic encryptions, i.e. such that produce different ciphertexts for the same plaintext, can be used. It follows that the Enc operation must use some random component. To make it possible to compute Dec, the ciphertext (which might be a pair consisting of multiple bitstrings) must contain information about the random component used in Enc, for instance the IV used for block ciphers (see section 2.10).

An approach for making an encryption method using a secure PRG IND-CPA secure could be the following:
For each message to encrypt, call the PRG first to generate a key that is twice as long as the fixed-size message length. Then use the first half of the PRG

2 Such that return the same ciphertext if the same plaintext is queried

output for encryption and save the second part. The first seed for the PRG will be taken from a *KGen* algorithm. The following keys will be the second half of the PRG in the previous iteration. Because the PRG output is pseudo-randomized, it is also a secure seed.
This approach is nice, but it is not practicable, because the sender and the receiver must be synchronized, which might fail because of insecure connections with message retransmissions.

2.6 pseudo random functions (prf) (exercise 5)

A PRF $PRF := K \times \{0,1\}^{in(\gamma)} \rightarrow \{0,1\}^{out(\gamma)}$ is a family of functions. When choosing a concrete k, the resulting function $f \in F$ is a pseudo-random function. For different key values, the function behaves like a different pseudo-random function.

Another way to explain this is that a PRF is a function $F(k,x) \rightarrow y$, which takes a (secret) key and a seed an x-value as input and produces a pseudo-random output with the following characteristics:

- y_0 and y_1, generated as $F(k,x_0) \rightarrow y_0$ and $F(k,x_0) \rightarrow y_0$, are different, if $x_0 \neq x_1$ and are equal if $x_0 = x_1$.
- A CPA adversary cannot distinguish the output of a PRF from a truly random function, which would do the same thing but in a truly randomized fashion.

A PRF with equal input and output length is called length-preserving.

Defining a (secure) PRF using a game would involve creating a challenger which randomly chooses a hidden bit b and, depending on that bit, calls the PRF or a truly random function. For both, the truly random function and the PRF, the outputs must be deterministic (consistent), such that if the same x value is provided as input, the same y value will be emitted.

2.7 pseudo random permutations (prps) (exercise 6)

A pseudo-random permutation is, similar to a pseudo-random function, a family of functions, which a concrete function can be selected from by choosing a key. A pseudo-random permutation is defined as $\{0,1\}^s \times \{0,1\}^l \rightarrow \{0,1\}^l$ with the key length s, and the input and output length l. The input space and the output space of a PRP are equal and the PRP defines a bijective permutation on the set of input bit strings (not on the set of input bits), making a PRP an invertible function. In other words, every possible bitstring is mapped to another bitstring of the same length. Each output bitstring belongs to exactly one input bitstring. Every PRP is a length-preserving PRF.

A *secure* (weak *PRP*) is defined just like the definition for the secure PRF (as it is also a PRF).

In contrast to PRFs and PRGs, PRPs also have a *strong* security definition. A PRP is strong if is secure under chosen plain text attacks if the adversary can access the PRP and its inverted operation PRP^{-1}. The game-based CPA-IND definition states that, for a chosen bitstring, a distinguisher D cannot distinguish the PRPs output of $f(x)$ or $f^{-1}(x)$ from the output of a real random permutation or its inverse operation.

2.8 DATA ENCRYPTION STANDARD (DES)

DES is a block cipher, which was used until about the year 2000. Because of its short keys, it was later replaced with DES3 and eventually by AES.

DES defines an encryption method for encrypting 64-bit plain texts using 56-bit keys (the key is 64 bits long, but 8 bits are used for parity). It uses a so-called Feistel function in 16 rounds, where in each round a part of the key is used.

The Feistel function uses another function F to operate on half of the 64-bit block at a time. Its most important characteristic is, that it is reversible, even if the internally used function F is not reversible (e.g. a one-way hash function, which also has collisions).
In each iteration of the Feistel function, the plain text will be split in half ($L_i||R_i$) and transformed to a new intermediate ciphertext $L_{i+1}||R_{i+1}$ as follows:

$$L_{i+1} := R_i$$
$$R_{i+1} := F(k_i, R_i) \oplus L_i$$

The exception is the last iteration, in which the sides will not be switched, such that $Li + 1 = F(k_i, R_i) + L_i$ and $R_{i+1} = R_i$.

Regardless of the function F, the Feistel function can be reversed by applying it again with reversed key order. That is possible because for each iteration we can compute L_i from $R_{i+1} + L_{i+1}$ and can take R_i directly from L_{i+1}.

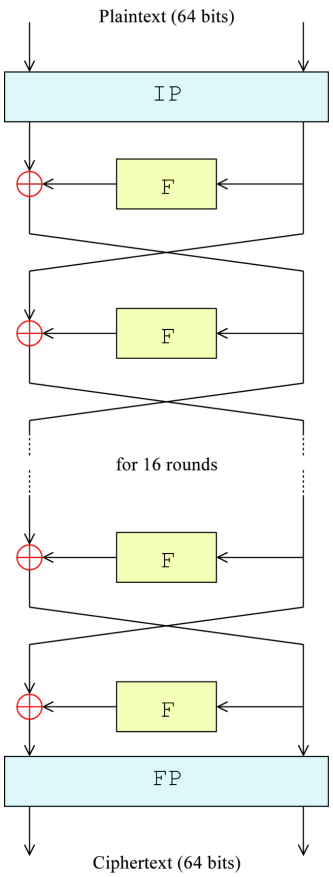

Figure 2.2: Feistel Function

The Feistel function provides the reversibility of the function, however, it does not provide the security itself. The security is provided by the internally utilized function F. In DES this function is using so-called S- and P-boxes, whose details go beyond the scope of this course. What is important, however, is that F implements Shannon's Concept of *confusion and diffusion* and that it takes a part of the key k_i and a bitstring as an input. The function F is, in fact, a non-reversible function.

56 bits (2^{56} possibilities) were soon not sufficient for security against brute-force adversary on modern hardware. What could have been done to increase its security would be creating a new encryption algorithm, which works like DES but uses a longer key. But to reduce the effort required to increase DES's security, another approach was used – applying DES multiple times with different keys.

The first approach would be to compute DES two times with two different keys to achieve the security of 2^{112}, because brute force a ciphertext, the adversary would have to compute all possible combinations of keys of the first DES and keys of the second DES $2^{56} \cdot 2^{56}$. But if an adversary would know the plain text m and the ciphertext c of a message, he could use a *meet-in-the-middle* attack to drastically reduce the computational effort for brute forcing. In the following, the computation of double DES is shown:

$$m \rightarrow DES_1(key_1, m) \rightarrow c' \rightarrow DES_2(key_2, c') \rightarrow c$$

An adversary could then first compute all possible intermediate ciphertexts c' for all possible values of k_1 from m and save the results in a hashmap. Then he could compute all possible intermediate ciphertexts c' with all possible values of k_2 from c using DES_2^{-1}. The computation required is equal to $2^{56} + 2^{56} = 2 \cdot 2^{56} = 2^{57}$ instead of 2^{112}. To obtain the correct combination of k_1 and k_2, the adversary would then search for collisions i.e. for pairs (k_1, k_2) that produced the same intermediate ciphertext c'.

For this reason, double DES is not useful. Instead, Triple DES is used, which has the security of 2^{112}, because the adversary can use a meet-in-the-middle attack but has to calculate two DES on one of the sides. Triple DES uses the inverse operation (reverse order key parts of k_2) of DES_2 for the second DES. The reason is that there was uncertainty about whether DES could be a *group*, such that there might be one DES_x operation, which would be equal to the combination of DES one to three. Later, however, it was discovered, that DES is not a group and inverting the middle DES was, therefore, not required:

$$m \rightarrow DES_1(key_1, m) \rightarrow c' \rightarrow DES_2^{-1}(key_2, c') \rightarrow c'' \rightarrow DES_3(key_3, c'') \rightarrow c$$

2.9 ADVANCED ENCRYPTION STANDARD (AES) (EXERCISE 6)

The security of DES (56-bit keys) and DES3 (112-bit security) is not sufficient. For this reason, the NIST started a public challenge for creating a cryptographic process replacing DES in 1979. The winner of this contest was decided to be Rijndael in 2000, which is the basis for the nowadays used advanced encryption standard (AES). AES is Rijndael, which can generally work with different input and output sizes (block sizes), with a block size of 128-bit. The key length for AES can be chosen to be 128, 192 or 256-bit.

AES is a substitution permutation network. This means it tries to replace bits with other bits (substitution) and change the order of bits (permutation)[3].

AES takes a 128-bit string as input and stores it as a 4×4 two-dimensional matrix, where each element is an 8-bit substring. Then it executes the following operations on the matrix for a minimum of 10 rounds[4]:

- *AddRoundKey:* The round key, which is a substring of the expanded input key, is added ($\oplus$) to the message.
- *SubBytes:* Every byte is replaced by another byte according to a fixed replacement scheme (S-box).
- *ShiftRows:* The bytes of each row are shifted depending on the row number (row 0 is not shifted, row 1 is shifted 1 element, ...).
- *MixColumns:* Same as ShiftRows but for columns.

In the last round, AddRoundKey is executed instead of MixColumns. SubBytes is the substitution and ShiftRows and MixColumns are the permutation of the substitution permutation network.

AES is an IND-CPA secure PRP (and therefore also an IND-CPA secure PRF).

2.10 BLOCK CIPHER MODES OF OPERATION (EXERCISE 6)

The main characteristic of block ciphers is that they can only encrypt bit strings of fixed size. However, the message size is usually variable and greater than the input size of the block cipher. Therefore, a method is needed to use block ciphers for messages with arbitrary lengths. Generally speaking, messages are divided into chunks that match the input length of the block cipher used. Then the mode of operation provides a schema for the encryption of the individual blocks utilizing a block cipher. If the message length is not a multiple of the block cipher's input length, padding must be added to the last block of the plain text message before encrypting. Usually, this will

3 Note that here permutation is a permutation on the set of bits, whereas the *permutation* in PRP refers to a permutation on the set of the different bit strings.

4 The number of rounds can be greater if longer keys are used.

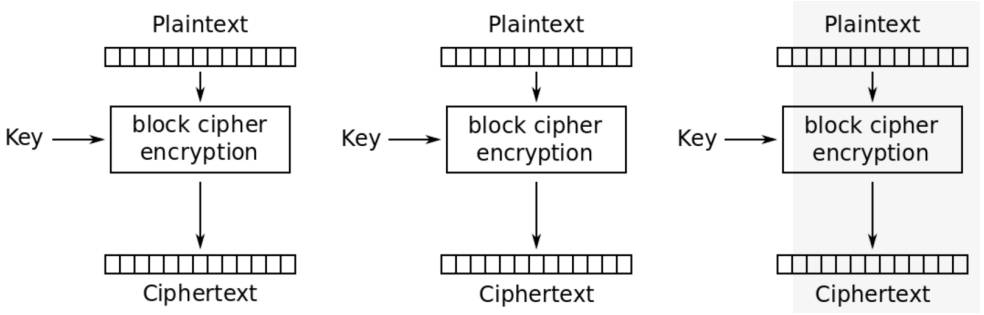

Figure 2.3: ECB Encryption

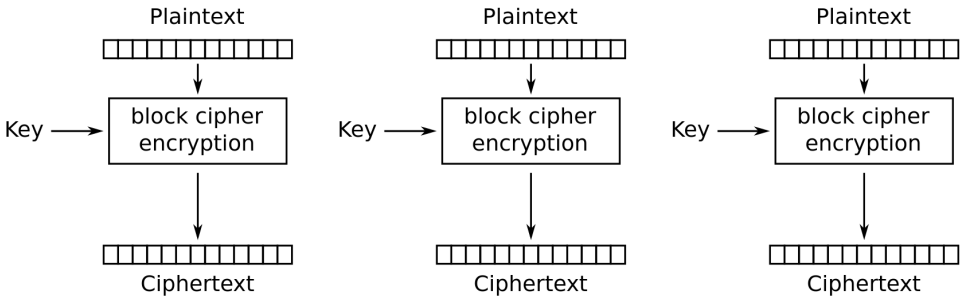

Figure 2.4: ECB Decryption

be a 1 followed by the required number of zeroes. This way, the padding can be identified after decrypting by removing all zeroes from the right side until a 1 appears.

2.10.1 *Electronic Code Book (ECB)*

This is the simplest mode of operation. It encrypts every part of the message independently with the same unmodified key. This is not IND-CPA secure, because equal plain text blocks are encrypted to equal ciphertext blocks.[5]

2.10.2 *Cipher Block Chaining (CBC)*

Cipher block chaining adds ($\oplus$) the previous cipher block to the message before applying the block cipher in each step. Because the cipher blocks are pseudo-random, the input to the block cipher is also pseudo-random, which makes it output pseudo-random. Because for the first block, there is no previous cipher text block, a random initialization vector IV is used. CBC is IND-CPA secure.

5 For formal proof of insecurity, see the solution of exercise sheet 6.

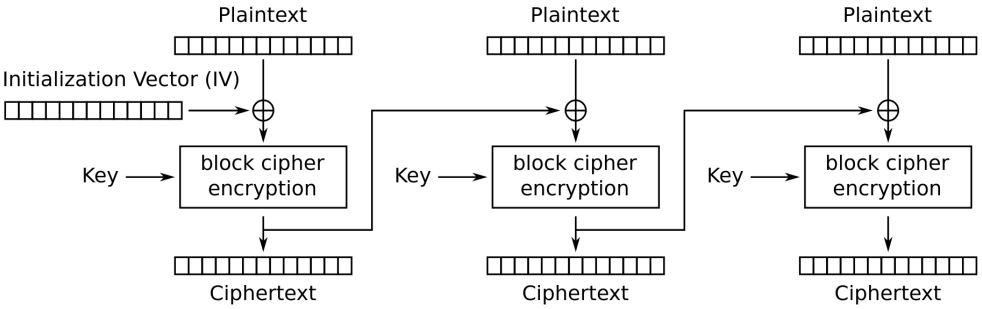

Figure 2.5: CBC Encryption

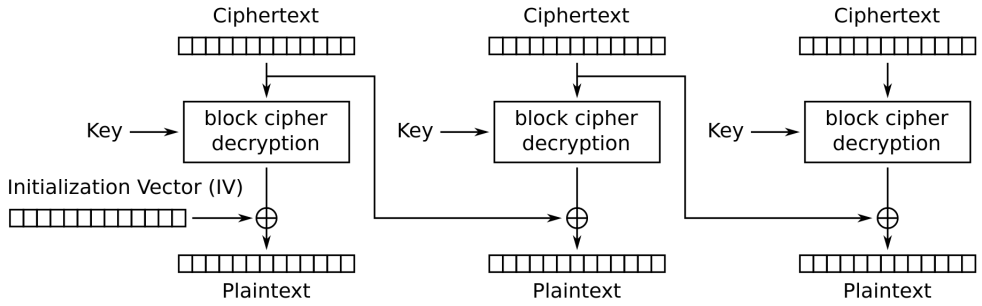

Figure 2.6: CBC Decryption

Advantages:

- Parallel Decryption is possible
- Loss of a cipher block c_i during transmission only affects plain text blocks m and $m+1$ after decryption (minor advantage)

Disadvantages:

- Encryption can only happen sequentially
- Decryption requires inverse block cipher

2.10.3 *Output Feedback Mode (OFB)*

The OFB uses the output of the block cipher of the previous iteration as input to the block cipher of each iteration. Because the previous output of the block cipher is pseudo-random, the output of each block cipher is pseudo-random. OFB is IND-CPA secure.

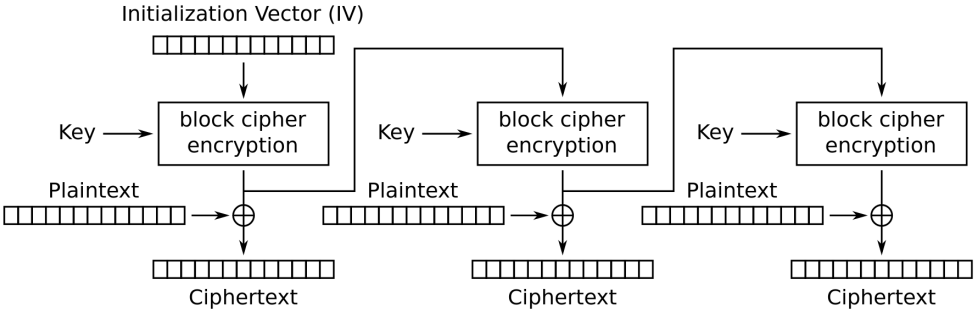

Figure 2.7: OFB Encryption

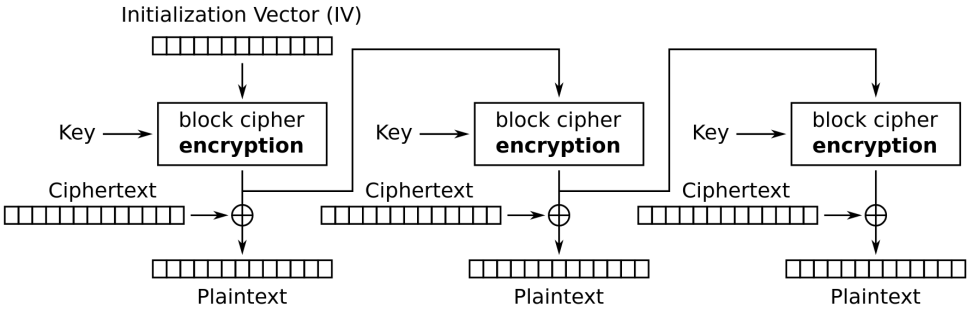

Figure 2.8: OFB Decryption

Advantages:

- No inverse operation of the block cipher is required
- No padding is required
- The block cipher output can be precalculated because it does not depend on the message. As soon as a message arrives, the block-cipher output can be added ($\oplus$) to the message (in parallel possible).

Disadvantages:

- Encryption and Decryption can only happen sequentially

2.10.4 *Cipher Feedback (CFB)*

CFB is a variant of OFB. Instead of using the output of the block cipher as the input of the next block cipher, the ciphertext block is taken as the next input to the block cipher. The advantages and disadvantages are the same as those of OFB.

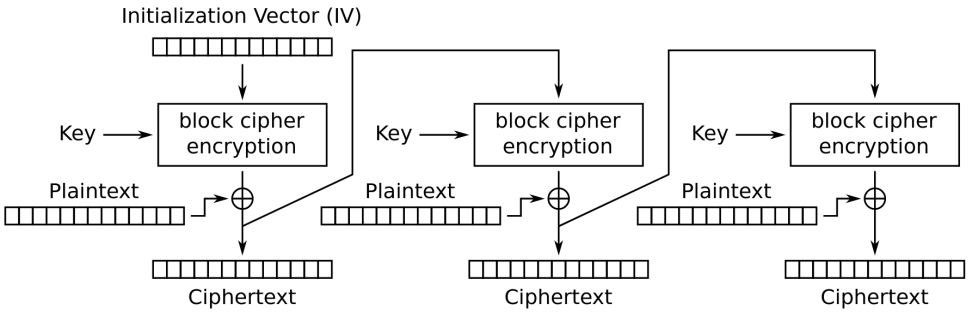

Figure 2.9: CFB Encryption

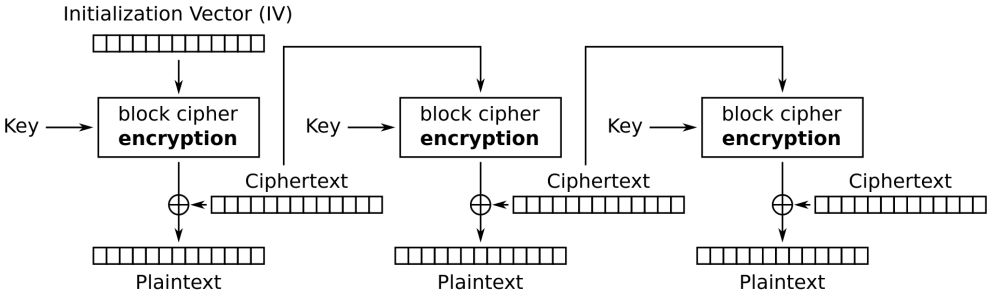

Figure 2.10: CFB Decryption

2.10.5 *Counter Mode (CTR)*

The counter mode uses a combination of a key (here called initialization vector or Nonce) and a counter.

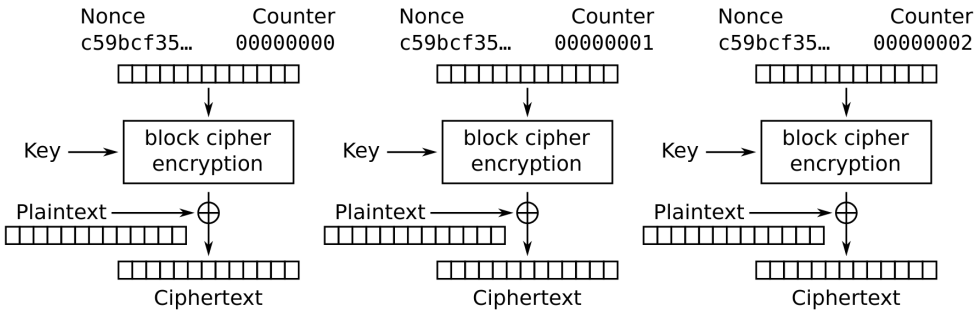

Figure 2.11: CTR Encryption (Decryption analogous, switch plain text and cipher text)

Advantages:

- Fully parallelizable encryption and decryption
- Transmission error in cipher text block c_i only affects plain text block m_i

- No padding for last plain text block necessary
- No inverse operation of the block cipher is required

Disadvantages:

- Security is reduced by the number of bits that are used as counter bits.

2.11 MESSAGE AUTHENTICATION CODES AND EUF-CMA

Encryption methods only ensure *confidentiality* and are not enough to ensure secure communication. What we want to do is to complement confidentiality with *integrity* and *authenticity*. These properties can be achieved by using message authentication codes (MACs). MACs are checksums that include a secret key in their calculation. That way, if the MAC is secure, the receiver of a message can be sure that the message has not been modified if the provided checksum is valid since no adversary can create a matching checksum for a modified message without knowing the secret key.

The security of a MAC is defined through the game-based definition of EUF-CMA, which stands for existential unforgeability under chosen message attacks. The adversary A can query as many MACs from the MAC oracle as it wants. Eventually, A must emit a pair (m, τ) of a message and a corresponding MAC. The adversary A is successful if τ matches the message m and if the message m has not been queried before.

A slightly stronger variant of EUF-CMA is sEUF-CMA (*strong* EUF-CMA). A MAC is sEUF-CMA if the adversary A also wins the game if it finds another MAC that is valid for a queried message m. That means that the requirements for the adversary's output are less strict. It is not required that the message m has not been queried before, but only that the pair (m, τ) has not been queried before. Since this gives the adversary additional opportunities to break the MAC, the MAC must be more secure.

2.12 PRF-MAC

Every pseudo-random function is also a MAC for messages *with the size of its input length*. To realize that, the MAC for a message m is calculated by using the PRFs function value at position m using the key k as $MAC := PRF(k, m)$. To break this MAC an adversary would have to guess the value of a strong PRF without knowing the key, which is not possible. Hence, the security of PRF-MACs can be proved using a reduction to the IND-CPA security of the used PRF.

2.13 CBC-MAC

To overcome the limitations that PRF-MAC has for the input message's length, MACs built based on block ciphers can be used. The block cipher is calculated for the entire message. Then the last ciphertext block is emitted. We had seen earlier that the security of block ciphers requires a *random* initialization vector IV. This is different when building MACs based on block ciphers. Since the first ciphertext block is not transmitted, the receiver would not have a chance to recalculate the MAC as the information about the IV is not transmitted. One could decide to transmit the IV as well. This, however, would allow the adversary to modify the message and the IV, which may allow him to find a matching MAC for the modified message when using the modified IV. Hence, CBC-MACs using random IVs and transmitting them are not secure. Instead, CBC-MACs should use a *constant* IV (usually just 0^λ, which is equivalent to no IV).

CBC-MAC (see section 2.10.2 for details on CBC) is not EUF-CMA if messages with different lengths are allowed because it is vulnerable to length-extension attacks. In this kind of attack, the adversary chooses a message m_1 with the same length as the output of the MAC. Then the adversary receives τ_1 from the MAC oracle. Then the attacker sends a second request with $m_2 = \tau_1 \oplus m_1$ to the oracle and receives τ_2. Now the attacker emits the pair $(m_3 = m_1||m_1,\ \tau_2)$. τ_2 is a valid MAC for m_3, because the CBC-MAC computes

$$\tau_3 = PRF(PRF(m_1) \oplus m_1) = PRF(\tau_1 \oplus m_1) = \tau_2$$

Also, the pair (m_3, τ_2) is new, because m_3 has never been queried before. Only parts of m_3 have been queried.

CMAC is an adapted implementation of CBC-MAC developed by the US government that *is secure* with different message sizes. To achieve that, CMAC XORs another key to the last plain text block. This way, the MAC of one message is *not* equal to what the PRF would expect as input for an extended version of the message.

2.14 HMAC

Another way to construct a MAC is by using hash functions (which will be covered in more detail in a later chapter). The kind of hash function that we want to use is a pseudo-random function (PRF) with much greater output than input length (to compress effectively) and a key length equal to the output length (to chain hash calls by using the output of one iteration as the key for next iteration). This construction is illustrated in Figure 2.12. In order to not be vulnerable to length extension attacks it is important to add a *final* step that involves using a different key than the previous steps.

If f is a good PRF, the construction is also a good PRF. Since every good PRF is also a good MAC for a fixed input length, the construction is secure for input messages of the same length.

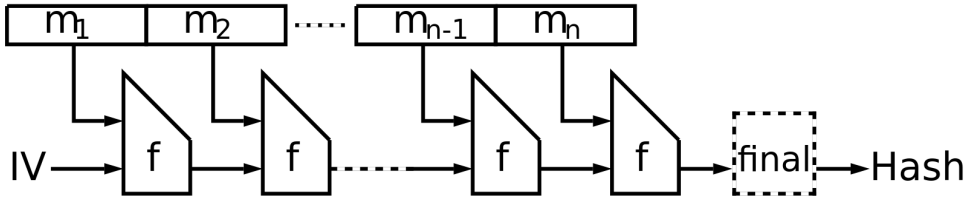

Figure 2.12: Construction of HMAC using Merkle-Damgard construction.

2.15 IND-CCA SECURITY

IND-CCA security is a stronger security definition compared to IND-CPA security. Its security is defined through a game shown in Figure 2.13.

$\underline{\mathcal{G}^{\text{cca}}_{\mathcal{E},\mathcal{A}}(1^\lambda)}$:
1 $k \leftarrow\!\!\$\ \mathsf{KGen}(1^\lambda)$
2 $b \leftarrow\!\!\$\ \{0,1\}$
3 $\mathcal{Q} \leftarrow \emptyset$
4 $d \leftarrow\!\!\$\ \mathcal{A}^{\textsc{Enc}(\cdot,\cdot),\textsc{Dec}(\cdot)}(1^\lambda)$
5 **return** $[\![b = d]\!]$

$\underline{\textsc{Enc}(m_0, m_1)}$:
6 **if** $|m_0| \neq |m_1|$
7 **return** $\perp$
8 $c \leftarrow\!\!\$\ \mathsf{Enc}(k, m_b)$
9 $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{c\}$
10 **return** c

$\underline{\textsc{Dec}(c)}$:
11 **if** $c \in \mathcal{Q}$
12 **return** $\perp$
13 **return** $\mathsf{Dec}(k, c)$

Figure 2.13: Game-based definition of IND-CCA security

The attacker has access to two oracles. The *ENC* oracle can be queried with two messages m_0 and m_1 and returns the ciphertext c_b of one of those messages. Which of the messages is encrypted by a secretly chosen bit b that the attacker A has to be found out by the attacker to be successful. The *DEC* oracle can be queried with a ciphertext c and will return the corresponding plain text m. But *DEC* will only be useful if the same ciphertext c has never been returned by the *ENC* oracle before. Because that would make the attack trivial, because the attacker could then obtain a c_b from *ENC*, decrypt it using *DEC* and obtain m_b and see what message has been encrypted by looking at m_b.

Every IND-CCA secure system is also END-CPA secure because not using the *DEC* oracle is also a valid IND-CCA attack. Using reduction, it can easily be shown that an IND-CPA attacker for ϵ could be used to break IND-CAA for ϵ, if such an attacker would exist. Therefore, such an attacker cannot exist, which means that ϵ must be IND-CPA secure if it is IND-CCA secure.

A common way to *construct* an IND-CAA secure system is to use the *Encrypt-then-MAC* scheme. This scheme first uses an IND-CPA secure encryption mechanism to encrypt a message and then uses an sEUF-CMA secure MAC

to tag it. The encryption output is the combination of the encrypted message and the tag of the ciphertext. When decrypting, an error is produced if the tag does not match the message's ciphertext. For this reason, an attacker can not use the DEC oracle in a useful way since he will not obtain an answer unless he forges the tag, which is not possible. As a result, the attacker effectively only has access to the ENC oracle, which is not sufficient for breaking the system, since the encryption method is IND-CPA secure. The encrypt-then-MAC construction is shown in its algorithmic representation in Figure 2.14.

$\underline{\mathsf{KGen}_{EtM}(1^\lambda)\text{:}}$

1 $k_M \leftarrow\!\$\ \mathsf{Gen}(1^\lambda)$
2 $k_E \leftarrow\!\$\ \mathsf{KGen}(1^\lambda)$
3 **return** (k_M, k_E)

$\underline{\mathsf{Enc}_{EtM}((k_M, k_E), m)\text{:}}$

4 $c \leftarrow\!\$\ \mathsf{Enc}(k_E, m)$
5 $\tau \leftarrow\!\$\ \mathsf{Mac}(k_M, c)$
6 **return** (c, τ)

$\underline{\mathsf{Dec}_{EtM}((k_M, k_E), (c, \tau))\text{:}}$

7 **if** $\mathsf{Vf}(k_M, c, \tau) = 0$
8 **return** $\perp$
9 $m \leftarrow \mathsf{Dec}(k_E, c)$
10 **return** m

Figure 2.14: Secure encrypt-then-MAC construction based on an IND-CPA secure encryption Enc and decryption Dec and an sEUF-CMA secure MAC (Mac and Vf)

3

PUBLIC-KEY CRYPTOGRAPHY

3.1 INTRODUCTION

Public-key methods use the following functions for encrypting and decrypting:

$$KGen(1^n) \rightarrow (sk, pk)$$
$$Enc(pk, m) \rightarrow c$$
$$Dec(sk, c) \rightarrow m$$

Often, the public key can be computed efficiently from the secret key $f(pk) \rightarrow sk$. But for security, it is required that the reverse of this function f^{-1}, which computes the secret key from the public key is *not* efficiently computable. We call such a function, that can only be efficiently computed in one direction a *one-way function* (OWF).

3.2 ONE-WAY FUNCTIONS

A one-way function (OWF) is a function $f : \{0,1\}^* \rightarrow \{0,1\}*$ that can be efficiently computed but its inverse function cannot be efficiently computed.

Formally speaking, a function f is one-way if it has the following two properties:

1. There is an efficient algorithm PPT for computing values $f(x)$ for all allowed input parameters x.
2. For all efficient PPT algorithms A their success probability in the below security game $G^{inv}_{f,A}$ is negligible.

$G^{inv}_{f,A}$ is defined as follows:

$$\{0,1\}^{\lambda}\$ \rightarrow x$$
$$f(x) \rightarrow y$$
$$A(1^{\lambda}, y) \rightarrow x'$$
$$return f(x) == x'$$

Note 1: This game does not require the adversary A to find the value x used but merely requires it to find a value that produces the output for f i.e. a collision.

Note 2: This assumes that there are problems that are hard to compute (NP-hard). This is an assumption that holds so far but is not proven.

3.3 NUMBER THEORY

3.3.1 *The Modulo Operator*

The modulo operator allows us to calculate with residual rings, which means that all values will never exceed the number $m-1$ for a residual ring Z_m. We define addition and multiplication, such that they produce a value $x \in Z_m$ such that:

$$a + b = c \mod m \quad | \quad \exists i \in Z : \quad a + b = c + i \cdot m$$
$$a + b = c \mod m \quad | \quad \exists i \in Z : \quad a \cdot b = c + i \cdot m$$

3.3.2 *Groups*

A group is a combination $(G, \circ)$ of a set G and an operation $\circ$ with the following 4 characteristics:

- Closure: $a \circ b \in G \quad | \quad \forall a, b \in G$
- Associativity: $a \circ (b \circ c) = (a \circ b) \circ c \quad \forall a, b, c \in G$
- There exists an identity element (neutral element) n such that $a \circ n = n \circ a = a \quad \forall a \in G$
- There exists an inverse element a^{-1} for all elements a in G that can turn a into the neutral element if combined with the operation $\circ$: $\exists a^{-1} \in G : a \circ a^{-1} = a^{-1} \circ a = n \quad \forall a \in G$.

We call a group an abelian group if it is a group for which the operation $\circ$ over G is commutative for all elements in G: $a \circ b = b \circ a \quad \forall a, b \in G$.

Groups, or abelian groups, are nice to work with in cryptography. Unfortunately, residual rings Z_m are generally not groups if paired with the multiplication operator $(Z_m, \cdot)$ because they do not have inverse elements for all elements in Z_m. For example, each group Z_m contains the value 0, but no number can be multiplied by 0 to produce the neutral element, which is 1. Additionally, depending on m, other elements might have no inverse element as well.

For this reason, we would like to reduce the set Z to a set Z_m^* that only contains all invertible elements of Z_m, thus making it a group. The invertible elements of Z_m are exactly all elements for which the greatest common divisor (gcd) is equal to 0. For instance, for the residual ring $Z_6 = \{0, 1, 2, 3, 4, 5\}$, the reduced group would be $Z_6^* = \{1, 5\}$.

The number of elements in Z_m^* is well defined for a couple of cases for prime numbers p, q and positive integers k:

- $|Z_p^*| = p - 1$ (all elements of Z_m except 0)
- $|Z_{p^k}^*| = p^k - \frac{p^k}{p}$ (all elements except all multiples of p in Z_{p^k})
- $|Z_{p \cdot q}^*| = |Z_p^*| \cdot |Z_q^*|$[1]

Def: The *order* of a group $(G, \circ)$ is the number of the elements in G: $ord(G) = |G|$.

Def: The *order* of an element $a \in G$ of a group $(G, \circ)$ is the number of elements that can be generated by applying the operation $\circ$ to a and a in G.

Def: A group $(G, \circ)$ is *cyclic* there exists an element $g \in G$, which we call the generator of the group, that can generate all elements of the group. This means that $ord(g) = ord(G)$. It can be proven all groups $(Z_p^*, \cdot)$ with prime numbers p are cyclic.

In cryptography, we would like to have a group $(G, \cdot)$ that is cyclic *and* has a prime number of elements. If we choose Z_p^* with a prime p, we have a cyclic group. But the group has $(p - 1)$ elements, which itself is not a prime[2]. To find a group that is cyclic and has a prime order, we first find a group (Z_p^*) for a prime $p = wq + 1$[3], where p and q are both primes. Since p is prime, there is a generator a for Z_p^*. We can now find a subgroup with the desired characteristics by finding a generator g that generates a subgroup $< g >$ of Z_m^* that has a prime number of elements. We do that by defining the generator as $g := a^w mod p$. g is a generator that generates all w-th elements of the initial group because if it is multiplied with itself it skips multiples of

1 This can be proven as follows: $\Phi(p \cdot q) = p \cdot q - (p - 1) - (q - 1) - 1 = p \cdot q - (p - 1) - q + 1 - 1 = (p - 1)q - (p - 1) = (p - 1) \cdot (q - 1) = \Phi(p) \cdot \Phi(q)$. These are all elements contained in $Z_{p,q}$ except all multiples of p (there are $q - 1$ of those and q (there are $p - 1$ of those). And also the element 0 has to be subtracted.

2 Since all prime numbers > 3 are uneven numbers, $p - 1$ for $p > 2$ is even, therefore divisible by two and, hence, not prime.

3 Example: $11 = 2 \cdot 5 + 1$, w can be any positive integer.

a, e.g. g^3 is equivalent to $(a^w)^3 = a^3w$ and a^1, a^2 are skipped. This means that a group $< g >$ generated by g has one w-th of the elements that the initial group Z_m^* had: It follow that: $| < g > | = \frac{|Z_m^*|}{w} = \frac{wq+1-1}{w} = q$. Thus, the order of $< g >$ is also prime (because we defined q as a prime earlier).

3.3.3 *Fermat's Little Theorem*

Fermat's little theorem states that for any positive integer a and for any prime number p the following holds:

$$a^p \equiv a \ mod \ p$$

If the greatest common divisor of a and p is 1[4], it also follows that:

$$a^{p-1} = 1 \ mod \ p$$

because if we multiplied both sides by a, we would end up at the original equation.

3.4 DISCRETE LOGARITHM PROBLEM

Let $(G, \cdot)$ be a subgroup of Z_p^* with prime order q, g be a generator for the group and further $y \in G$ an element in the group. The logarithm problem describes the task to find the (smallest) exponent for which g generates y:

$$g^x \equiv y \text{ in } G$$

While solving this in the set of real numbers R is easy, it is not easy for some groups with prime order.

The discrete logarithm assumption, which means that the discrete logarithm is hard to compute, is given for a group generator algorithm $Gen(1^\lambda)$ if all PPT algorithms win the below-defined game $Exp^{DL}_{Gen,A}$ only with negligible probability: $Pr[Exp^{DL}_{Gen,A}(1^\lambda) = 1] = neg(\lambda)$.

The game $Exp^{DL}_{Gen,A}(1^\lambda)$ is defined as follows:

4 Which is true for at least all $a < p$ because p is prime

$$
\begin{aligned}
(G, q, g) &\leftarrow Gen(1^\lambda) \\
x &\leftarrow^{\$} G \\
y &\leftarrow g^x \text{ in } G \\
x' &\leftarrow A(1^\lambda, G, q, g, y)
\end{aligned}
$$

return $[g^x \equiv y \text{ in } G]$

3.5 ATTACKS AGAINST THE DISCRETE LOGARITHM

In this section, we show the known attacks against the discrete logarithm on groups G, q, p, where q is the order of the (sub)group and p is the order of the (super)group, in which we are calculating the modulo.

3.5.1 *Brute Force*

Try all values $g^0, g^1, g^2, g^3(...)$ until the value $g^i = y$ is found.

This trivial attack is on average not successful in an acceptable time because the adversary needs to try $\frac{1}{2} \cdot 2^\lambda$ of the possible 2^λ values on average. Therefore the runtime complexity is $O(\lambda)$

3.5.2 *Giant-step-baby-step algorithm*

1. *Giant-steps:* set $t = \sqrt{ord(G)}$. Then compute t giant step values from $k = 0$ to $k = (t - 1)$ that can be generated from G with the distance of t to each other as:
   $g^{0t}, g^{1t}, g^{2t}, ...g^{t-1}$.
2. The value y, of which we would like to find the discrete log to the base of g is g^x in G and is located between two of the precomputed giant steps. Also, its distance to the next giant step will be less than t, because the giant steps all have the distance t. So we now compute all values that can be generated from g starting at y until we reach the next giant step:
   $y \cdot g^0, y \cdot g^1, y \cdot g^2(...)$ until $y \cdot g^i$ matches one of the giant steps
3. Since we then know the next giant step and in how many steps it can be generated from g, we know that y can be generated in i fewer steps from g:
   $x = k \cdot t - i$

This algorithm has a memory complexity of $= (\sqrt{\lambda})$ and a computational complexity of $= \sqrt{\lambda}$

3.5.3 *Pollard's Roh-Algorithm*

This algorithm is very similar to the giant-step-baby-step algorithm but has constant memory complexity $O(1)$

3.5.4 *Index Calculus*

This attack only works on Z_p^* i.e. it does not work on elliptic curves. Its runtime complexity is:

$$2^{O(\sqrt{log(p)log(log(p))})}$$

3.5.5 *Number Field Sieve*

This attack also only works for Z_p^* and not for elliptic curves. Its runtime complexity is:

$$2^{O(log(p)^{1/3} \cdot log(log(p))^{2/3})}$$

3.5.6 *Implications of Attacks and Motivation for Elliptic Curves*

From the attacks 3.5.2 and 3.5.3, we see that the order of the subgroup q must be chosen high enough, because the attacks have a square root runtime based on q. Today, we would say $q \geq 2^265$ is sufficient.

From the attacks 3.5.4 and 3.5.5, we see that the range of numbers that we allow through the selection of p must be even significantly greater than q because the algorithms can break the DL with logarithmic runtime based on p. Today, we would say $q \geq 3072$ would be sufficient.

This means if we work on subgroups with prime order q of Z_p^*, we have to allow bit strings with the length of $p = 3072$ bits even though we only use $2^q = 2^{256}$ of the possible 2^{3072} values. So this is not ideal.

For this reason, today the discrete logarithm based on elliptic curves is used instead of on Z_p^*, prohibiting the index calculus (3.5.4) and the number field sieve (3.5.5) attacks. Then we can use values of q that are close to the value of p and therefore use much shorter bit strings, which speeds up calculations.

Elliptic curves are additive groups E over groups like Z_p^*. All values of E are tuples (x, y) with x and y as elements in Z_p^* that fulfill the equation of the elliptic curve for given a and b. Additionally, the *point at infinity* $\mathcal{O}$ is part of E:

$$E_{a,b}(Z_p^*) = \{\forall x, y \in Z_p^* \mid y^2 = x^3 + ay + b\} \cup \{\mathcal{O}\}$$

We define an addition operation for points on the elliptic curve. The addition can be illustrated as shown in Figure 3.1. When two points A and B are added, first, the line that crosses both points will be found. This line will cross the curve at another point, which is the point $-A + B$. If we mirror the point on the x-axis, we get the point $A + B$ i.e the point resulting from the addition operation.

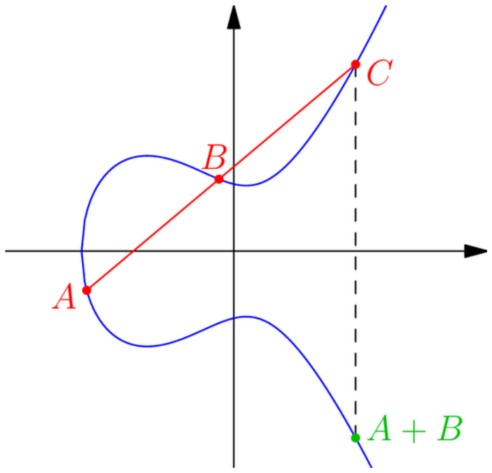

Figure 3.1: Illustration of addition on elliptic curves

In case both points are the same, the line will be the tangent of the curve. In case A is added to $-A$, the line would be horizontal, thus not crossing the curve again. The result of this operation is defined as the point at infinity $\mathcal{O}$.

When calculating with an elliptic curve, the concept is very similar to calculating in *mod*. We have a generator Point $\mathcal{G}$ on E, which we will add to itself to produce new points in E. The discrete logarithm problem on elliptic curves expresses *how many times must* $\mathcal{G}$ *be added to itself to produce a Point* Y*?*: $DLog_{\mathcal{G}}^{E}(Y)$ is the value x for which $x\mathcal{G} = Y$.

If we had a given Z_p^* and we could find an elliptic curve E with the number of distinct elements q, such that q is very close to p, this would be better than using a prime subgroup of Z_p^*. The reason is that a prime subgroup of Z_p^* would require us to choose $q << p$ because of the attacks shown in Section 3.5.4 and **??** work against subgroups of Z_p^*. These smaller values are much faster to compute than bigger values. The question now is whether given Z_p^* it is actually possible to find an elliptic curve with its number of elements close to p. Hasse's theorem shows that it is possible to find an elliptic curve with the number of elements relatively close to p (with "only" up to $2\sqrt{p}$ distance to p):

$$p + 1 - 2\sqrt{p} \leq |E_{a,b}| \leq p + 1 + 2\sqrt{p}$$

In practice, the value for p and the elliptic curve is given. Those are defined through standards published by e.g. the BSI.

The Youtube channel https://www.youtube.com/@thenativeweb also explains elliptic curves very nicely in the videos *Architecture's Daily* episodes 91 to 95.

3.7 DIFFIE-HELLMAN KEY EXCHANGE

The Diffie-Hellman key exchange claims a way to generate a common secret key for two communication partners over a public channel.

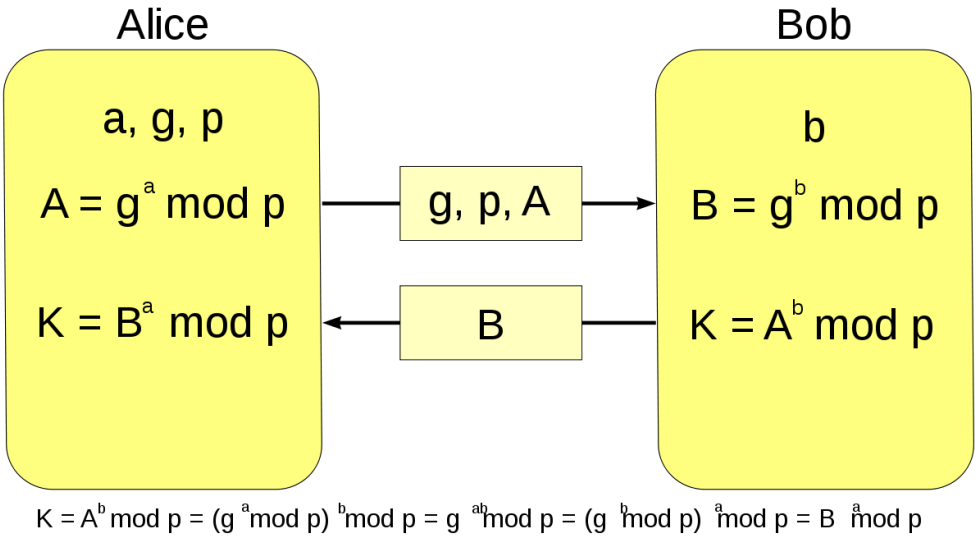

Figure 3.2: Illustration of the Diffie-Hellman key exchange. Note that g and p can be generated by Alice, like shown here, or delivered by a third party (since both values are public).

3.8 IND-CPA AND IND-CCA FOR PKE

IND-CPA and IND-CCA for PKE (public key encryption) are defined the same way as for symmetric encryption:

There are a couple of things that should be kept in mind when dealing with these definitions in comparison to the symmetric versions:

1. For PKE, it does not matter how many times the adversary queries ENC. So IND and $IND - CPA$ are equivalent.
2. An additional $ENC'(m) \rightarrow c$ oracle that only takes one message and encrypts it (without a bit b), does not help the adversary, since the ad-

$\underline{\mathcal{G}^{\text{cpa}}_{\mathcal{E},\mathcal{A}}(1^\lambda)}$:

1 $(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\$\ \mathsf{KGen}(1^\lambda)$
2 $b \leftarrow\!\$\ \{0,1\}$
3 $d \leftarrow\!\$\ \mathcal{A}^{\text{Enc}(\cdot,\cdot)}(1^\lambda, \mathsf{pk})$
4 **return** $[\![b = d]\!]$

$\underline{\text{Enc}(m_0, m_1)}$:

5 **if** $|m_0| \neq |m_1|$
6 **return** $\perp$
7 **return** $\mathsf{Enc}(\mathsf{pk}, m_b)$

Figure 3.3: Game-Based definition of IND-CPA for PKE. A PKE system $\epsilon = KGen$, Enc, Dec is secure if for all PPT adversaries A, there is a negligible function $neg(\lambda)$, such that $Pr[G^{cpa}_{A,\epsilon}(1^\lambda)] = \frac{1}{2} + neg(1^\lambda)$

$\underline{\mathcal{G}^{\text{cca}}_{\mathcal{E},\mathcal{A}}(1^\lambda)}$:

1 $(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\$\ \mathsf{KGen}(1^\lambda)$
2 $\mathcal{Q} \leftarrow \emptyset$
3 $b \leftarrow\!\$\ \{0,1\}$
4 $d \leftarrow\!\$\ \mathcal{A}^{\text{Enc}(\cdot,\cdot),\text{Dec}(\cdot)}(1^\lambda, \mathsf{pk})$
5 **return** $[\![b = d]\!]$

$\underline{\text{Enc}(m_0, m_1)}$:

6 **if** $|m_0| \neq |m_1|$
7 **return** $\perp$
8 $c \leftarrow\!\$\ \mathsf{Enc}(\mathsf{pk}, m_b)$
9 $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{c\}$
10 **return** c

$\underline{\text{Dec}(c)}$:

11 **if** $c \in \mathcal{Q}$
12 **return** $\perp$
13 **return** $\mathsf{Dec}(\mathsf{sk}, c)$

Figure 3.4: Game-Based definition of IND-CCA for PKE. A PKE system $\epsilon = KGen$, Enc, Dec is secure if for all PPT adversaries A, there is a negligible function $neg(\lambda)$, such that $Pr[G^{cca}_{A,\epsilon}(1^\lambda)] = \frac{1}{2} + neg(1^\lambda)$

versary can compute ciphertexts itself anyways because the Enc function does not involve a secret key.

3. Any deterministic algorithm cannot be $IND - CPA$ even if the adversary only queries ENC once (i.e. IND), because the adversary can compute ciphertexts itself. If Enc is deterministic, the attacker could query $ENC(m_0, m_1)$ and receive c_b. Then he could compute $Enc(pk, m_0) \rightarrow c_0$ itself and compare it with c_b.[5]

3.9 ELGAMAL ENCRYPTION

$\underline{KGen(1^\lambda):}$
$(G, g, q) \leftarrow GroupGen(1^\lambda)$
$sk \leftarrow \$G$
$pk \leftarrow g^{sk}$ in G
return $((G, g, q), (sk, pk))$

$\underline{Enc(pk, (G, g, q), m):}$
$y \leftarrow \$G$
return $(g^y, pk^y \cdot m)$

$\underline{Dec(sk, (Y, c)):}$
$return c \cdot Y^{-sk}$

ElGamal is IND-CPA if the DDH assumption (see section 3.7) holds. However, it is *not* IND-CCA because it has the following property:

5 Compare section 2.5

$$Dec(sk, (Y, c \cdot r)) = Dec(sk, (Y, c)) \cdot r$$

This holds because of $Dec(sk, (Y, c \cdot r)) = Y^{-sk} \cdot c \cdot r = g^{-sk \cdot y} \cdot g^{sk \cdot y} \cdot m \cdot r = m \cdot r = Dec(sk, (Y, c)) \cdot r$. This allows an adversary to modify the ciphertext with a known outcome. The adversary can then divide r back out. This way, ciphertexts c can be decrypted without querying c at the DEC oracle, allowing us to compare them with an earlier from ENC obtained c, this way breaking IND-CCA.

A further restriction of ElGamal is that all messages must be in the group G and therefore have a certain length and structure. Both problems are solved by ICIES.

3.10 ICIES ENCRYPTION

The ICIES (Elliptic Curve Integrated Encryption Scheme) encryption is a modern standardized version of the ElGamal encryption concept and uses Diffie-Hellman and the common key K that it generates as the input for a PRF to create a key for symmetric encryption and authentication.

ICIES uses Diffie-Hellman for key exchange to generate two keys (k_E, k_M), one for symmetric encryption and one for key exchange. It then uses encrypt and MAC, which we have learned earlier. Additionally, it uses a key derivation function (KDF). The purpose of this function is simply to transform points on the elliptic curve into a binary representation that can be used as a key. This means that KDF works like a hashing function. For simplicity, in the following, we use the notation above Z_p^* although ICIES in reality work with elliptic curves. The operations of ICIES are formally defined as follows:

$\underline{KeyGen(1^\lambda)}$:	$\underline{Enc(pk, m)}$	$\underline{Dec(sk, (Y, c, \tau))}$
$(G, g, q) \leftarrow GroupGen(1^\lambda)$	$y \leftarrow Z_p^*$	$(k_E, k_M) \leftarrow KDF(Y^{sk})$
$sk \leftarrow G$	$Y \leftarrow g^y$	*if* $MAC(k_M, c)$!= τ return $\perp$
$pk \leftarrow g^{sk}$	$(k_E, k_M) \leftarrow KDF(pk^y)$	return $SymDec(k_E, c)$
return $((G, g, q), (pk, sk))$	$c \leftarrow SymEnc(k_E, m)$	
	$\tau \leftarrow MAC(k_M, c)$	
	return (Y, c, τ)	

KeyGen is executed on the receiver side. Then the public key must be published. Afterwards, *Enc* can be executed by senders. The receiver can then decrypt the ciphertexts (Y, c, τ) using *Dec*.

This is a so-called KEM-DEM hybrid encryption. *KEM (Key Encapsulation)* refers to a process of not explicitly encrypting the key as one arbitrary message (in contrast to asynchronous encryption), but instead, only encrypting a key. Because optimizations can be applied, this can be faster than asynchronous encryption. *DEM (Data Encapsulation)* refers to using a symmetric key for encrypting data. *KEM-DEM Hybrid* describes using an asymmetric KEM system for generating a key and a faster symmetric DEM system for encrypting payload data.

KEM-DEM hybrid encryption combines the simple key management of asynchronous encryption with the high performance of synchronous encryption. On the other hand, we need two systems instead of one, an asynchronous one and a synchronous one.

ICIES is IND-CCA if a DH-like assumption holds and if the symmetric encryption used is IND-CPA and the MAC used is sEUF-CMA.

3.11 DISCRETE SECURITY ASSUMPTIONS

This chapter will explain the discrete security assumptions DL, CDH and DDH. It can be proven that if DDH holds, CDH must hold and if CDH holds, DL must hold. But we do not know the reverse. All these assumptions are unbroken for more than 40 years know, however, not been proven either.

3.11.1 *DL (Discrete Logarithm) Assumption*

The discrete logarithm assumption is that given a group (G, g, p) and a value $y \in G$, the discrete logarithm $DLog_g(y)$ is not efficiently computable. This can be defined through the game shown in Figure 3.7. DL then states the following regarding the game for all *PPT* adversaries A:

$$Pr[Exp^{dlog}_{GroupGen,A}(1^\lambda) = 1] \leq neg(\lambda)$$

$\underline{\mathcal{G}^{\mathsf{dlog}}_{\mathsf{GroupGen},\mathcal{A}}(1^\lambda)}$:

1 $(G, g, q) \leftarrow\!\$\ \mathsf{GroupGen}(1^\lambda)$
2 $h \leftarrow\!\$\ G$
3 $x \leftarrow\!\$\ \mathcal{A}(1^\lambda, (G, g, q), h)$
4 **return** $[\![h = g^x]\!]$

Figure 3.5: Game-based definition of the DL assumption.

3.11.2 *CDH (Computational Diffie-Hellman) Assumption*

The computational Diffie-Hellman assumption states that given a group (G, g, p) and two values g^x and g^y, g^{xy} cannot be com-

$\underline{\mathcal{G}^{\mathsf{cdh}}_{\mathsf{GroupGen},\mathcal{A}}(1^\lambda)}$:

1 $(G, g, q) \leftarrow\!\$\ \mathsf{GroupGen}(1^\lambda)$
2 $x, y \leftarrow\!\$\ \mathbb{Z}_q$
3 $h \leftarrow\!\$\ \mathcal{A}(1^\lambda, (G, g, q), g^x, g^y)$
4 **return** $[\![g^{xy} = h]\!]$

Figure 3.6: Game-based definition of the CDH assumption.

puted efficiently (of course without knowing x and y but only knowing the powers). If this holds, DL must also hold (can be shown by reduction). This can be defined through the game shown in Figure 3.7. CDH then states the following regarding the game for all PPT adversaries A:

$$Pr[Exp^{cdh}_{GroupGen,A}(1^\lambda) = 1] \leq neg(\lambda)$$

3.11.3 *DDH (Decisional Diffie-Hellman) Assumption*

The decisional Diffie-Hellman assumption states that given (G, p, g), g^x and g^y, the two values g^{xy} and g^z can not be distinguished efficiently. If this holds, CDH must also hold (can be shown by reduction). This can be defined through the game shown in Figure 3.7. DDH then states the following regarding the game for all PPT adversaries A:

$$Pr[Exp^{ddh}_{GroupGen,}(1^\lambda) = 1] \leq \frac{1}{2} + neg(\lambda)$$

$\underline{\mathcal{G}^{\mathsf{ddh}}_{\mathsf{GroupGen},\mathcal{D}}(1^\lambda)}$:

1 $(G, g, q) \leftarrow\!\$\ \mathsf{GroupGen}(1^\lambda)$
2 $x, y \leftarrow\!\$\ \mathbb{Z}_q$
3 $b \leftarrow\!\$\ \{0, 1\}$
4 **if** $b = 0$
5 $\quad z \leftarrow x \cdot y$
6 **else**
7 $\quad z \leftarrow\!\$\ \mathbb{Z}_q$
8 $d \leftarrow\!\$\ \mathcal{D}(1^\lambda, (G, g, q), g^x, g^y, g^z)$
9 **return** $[\![b = d]\!]$

Figure 3.7: Game-based definition of the DDH assumption.

3.12 TEXTBOOK-RSA

$\underline{KeyGen(1^\lambda)}$:
$(N, e, d) \leftarrow RSAGen(1^\lambda)$
return (N, e, d)

$\underline{Enc((N, e), m)}$
return $(m^e mod N)$

$\underline{Dec((N, d), c)}$
return $c^d mod N$

The correctness is assured because $RSAGen$ chooses two prime numbers p and q (of similar lengths) to compute N. Because we know that $\Phi(p \cdot q) = (p-1)(q-1)$, the algorithm can compute an exponent e that has a multiplicative inverse in $mod\ \Phi(N)$ (fulfilled if $gcd(e, \Phi(N)) = 1$). Then the $RSAGen$ algorithm can also compute that inverse d of e for which $d \cdot e = 1\ mod\ \Phi(N)$ since it knows $\Phi(N)$. Because we know that $\forall x \in Z^*_m : \ x^r = x^{r\ mod\ \Phi(m)}\ mod\ m$ the correctness of Textbook RSA follows as:

$$
\begin{aligned}
m &= Dec((N,d), Enc((N,e), m)) \\
&= Dec((N,d), m^d \ mod \ N) \\
&= m^{d \cdot e} \ mod \ N \\
&= m^{d \cdot e \ mod \ \Phi(N) \ mod \ N} \\
&= m^1 \ mod \ N \\
&= m
\end{aligned}
$$

Textbook-RSA is not IND-CPA and therefore also not IND-CCA because it is a deterministic encryption system (see Section 2.5).

Furthermore, another property of Textbook-RSA interferes with IND-CCA. Textbook-RSA has a *homomorphism* property (just like ElGamal). This means that $Enc(m_0) \cdot Enc(m_1) = Enc(m_0 \cdot m_1)$ because $Enc(m_0) \cdot Enc(m_1) = m_0^e \ mod \ N \cdot m_1^e \ mod \ N \ = (m_0 * m_1)^e \ mod \ N = Enc(m_0 \cdot m_1)$. Using this, an adversary against IND-CCA can query $ENC(m_0, m_1) \rightarrow c_b$ and then query $DEC(c_b \cdot c_b) \rightarrow m^*$ (because $c_b \cdot c_b$ has never been returned by ENC before). Then he could compare m^* with $m_0 \cdot m_0$ or with $m_1 \cdot m_1$ to find out b.

RSA is also called a *trapdoor function*, because it is not invertible if d is unknown, but if d is known (or something that gets us to d e.g. p and q), it is invertible (trapdoor opened by d).

3.13 RSA ASSUMPTION

RSA is based on the RSA assumption defined through the following security game $Exp^{RSA}A$:

$$
\begin{aligned}
&\underline{Exp^{RSA}(1^\lambda):} \\
&\quad (N,e,d) \leftarrow \$ \ RSAGen(1^\lambda) \\
&\quad x \leftarrow \$ \ Z_N^* \\
&\quad y \leftarrow x^e \ mod \ N \\
&\quad x* \leftarrow A(1^\lambda, N, e, y) \\
&\quad return \ [(x^*)^e = y \ mod \ N]
\end{aligned}
$$

For all PPT algorithms A there exists a negligible function $neg(\lambda)$ such that:

$$Pr[Exp^{RSA}(1^\lambda) = 1] \leq neg(\lambda)$$

In other words, there is no efficient algorithm that can compute the message x or an equivalent value without knowledge of d.

This assumption is tightly coupled with the assumption that it is difficult to factorize prime number products: Given a product of prime numbers $N = p \cdot q$, we assume there is no efficient algorithm that computes p and q. If this does not hold, an adversary to RSA could factorize N to p and q and then compute $Phi(N)$ as $(p-1) \cdot (q-1)$ and from that d in the same way that the $RSAGen$ algorithm does.

So we know if the RSA assumption holds, then prime factorization must also be difficult. However, if prime factorization is difficult, we do not know if the RSA assumption holds because there might be another way of breaking RSA that we do not know yet.

One of the best attacks known against RSA is by factorization e.g. the number field-sieve with the runtime complexity:

$$O(2^{log(N)^{1/3} \cdot log(log(N))^{2/3}})$$

3.14 Optimal asymmetric encryption padding (OAEP)

To build an IND-CCA secure encryption system utilizing the RSA function that does not have the drawbacks of textbook RSA (determinism, homomorphism), the OAEP system has been developed. It adds two different types of padding to the message and uses two hash functions G and H to combine the messages. Afterwards, the resulting new bit string $X||Y$ is used as input for the RSA function.

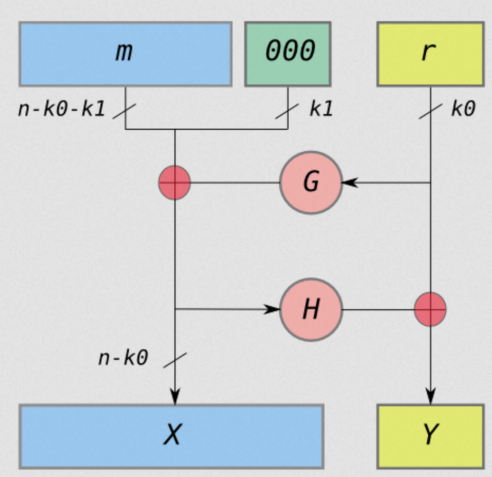

Figure 3.8: OAEP permutation before applying RSA function

The first type of padding is a random part r consisting of k_0 bits. This adds the probabilistic component to the encryption, therefore making it IND-CPA: Since G produces random bits based on r, the left side becomes a pseudo-random string. Afterwards, we use this as input for H and add the resulting pseudo-random string to the right part. Therefore, both, X and Y will look like random bit strings. Using then RSA on top of that makes it IND-CPA (full proof not discussed here.)

The second type of padding is a field of k_0 zero-bits. These are used for checking the integrity of decrypted ciphertexts and thereby ensuring IND-CCA: When decrypting a ciphertext, the *Dec* function first computes the inverse of the RSA function using d, thereby reconstructing $X||Y$. Then the Feistel network is traversed from the bottom up (hash functions do not need to be inverted, they will cancel out by XOR). If an adversary tries to modify a ciphertext c before passing it to DEC, the zero-bits will be modified (contain ones) with high probability due to the diffusion done by the Feistel. In this case, *Dec* will return an error, thereby preventing adversaries from modifying and then decrypting ciphertexts. This way, the IND-CCA attack against textbook RSA, which was based on the homomorphism of the RSA function, is no longer possible.

4

GLOSSARY

In this chapter, basic terms will be defined and explained.

4.1 KERCKHOFFS'S PRINCIPLE

Security should not require the system, but merely the key to be secret.

4.2 PERFECT SECURITY

A cryptographic process is perfectly secure if for all messages in the set of possibles messages $m \in M$ and all possibly producible ciphertexts $c \in C \quad | \quad Pr[C = c] > 0$ the probability that m occurs is equal no matter if the ciphertext is known or unknown: $Pr[M = m] = Pr[M = m|C = c]$. More simply put, this means that an attacker cannot gain any knowledge about the message when seeing its ciphertext regardless of the message's and the ciphertext's concrete values.

4.3 CONDITIONAL PROBABILITY

Conditional probability $Pr[A = a|B = b]$ is the probability that the event $a \in A$ occurs if it is already known that the event $b \in B$ occurs. It is calculated as $Pr[A = a|B = b] = \frac{Pr[A=a] \wedge Pr[B=b]]}{Pr[B=b]}$. An example is a probability rolling a 6-sided dice results in the number 6 if it is already known that the result is an even number: $Pr[is\,6|is\,even] = \frac{Pr[is\,6] \wedge Pr[is\,even]}{is\,even} = \frac{\frac{1}{6}}{\frac{1}{2}} = \frac{1}{3}$.

Perfect security can only be reached for processes with deterministic a decryption function if the key space is at least as big as the message space $|K| \geq |M|$. This is because then a given ciphtertext c could be decrypted to $|K|$ different messages m (determinism), which minimizes the search space for an attacker from $|M|$ possibilities to $|K|$ possibilities and therefore also decrease the attacker's uncertainty..

4.4 HIDING MESSAGE LENGTH

Completely hiding the length of a message is impossible if the length of possible messages is unlimited. Hiding messages' lengths would require changing their length for transmission. The message length cannot be decreased because this would result in data loss. Therefore, messages would have to be extended up to at least the size of the longest possible message to make all messages appear to have the same length. If the message size is unlimited, we cannot expand messages to that nonexisting limit.

4.5 INSECURITY OF SHIFT-CIPHERS

Shift-ciphers, like Ceasar, are not perfectly secure, which can be shown by contraposition: If the possible messages are defined as $M = \{aa, ab\}$, a shift cipher shifting each character by some number of characters in an alphabet would always create ciphertexts as follows $Dec(aa) \rightarrow c_0c0$ and $Dec(ab) \rightarrow c_0c_1 \quad | \quad c_0 \neq c_1$ Therefore, an attacker's probability when seeing the ciphertexts to guess the correct message $m \in M$ is equal to 1 and greater than the initial probability, which was $\frac{1}{2}$. A concrete example would be: $Pr[M = aa|C = c_0c_0] = 1 \neq Pr[M = aa] = \frac{1}{2}$. We see that limiting the set of possible messages can be a helpful method for contradictions of perfect correctness.

4.6 CRITERIA FOR CORRECT REDUCTION PROOFS

If using reduction for proving the security of a cryptographic process, the following points must be addressed:

1. Show the reduction algorithm as text, pseudo-code or picture
2. Show that the algorithm simulates the security game for the inner attacker, such that it cannot know that it is used in a reduction.
3. Show that the reduction algorithm is efficient
4. Show that the reduction algorithm wins the outer game with a non-negligible advantage. This often is done by showing that the success of the reduction depends on the success of the inner attacker, which we know will win with a non-negligible advantage. Here, a formal calculation of the probability with which the attacker wins is required.