Digital Signatures

Tibor Jager
tibor.jager@uni-wuppertal.de
Bergische Universität Wuppertal
Chair for IT Security and Cryptography

Updated on November 19, 2020

BERGISCHE
UNIVERSITÄT
WUPPERTAL

I would like to thank Florian Böhl, Peter Chvojka, Gareth Davies, Benny Fuhry, Kai Gellert, Felix Grün, Gunnar Hartung, Eduard Hauck, Max Hoffmann, Jan Holz, Björn Kaidel, Eike Kiltz, Evgheni Kirzner, Jessica Koch, Sebastian Lauer, Julia Rohlfing, Maik Schäfer, Christoph Striecks, and Sun Jing for helpful comments.

Contents

Chapter 1

Introduction

A sender wants to transmit a message to a receiver in such a way that the receiver can verify two things when receiving the message:

1. *Message integrity*: The recipient can be sure that the message has not been altered in transit. Neither maliciously (i.e. by an adversary that deliberately modifies the message), nor by chance (for example, by a transmission error).

2. *Authenticity of the message*: The recipient can determine whether the message really comes from a specific sender.

If the message is printed on paper, there is a classic solution to this problem. Each sender has a unique signature, which

1. can only be generated by the sender, and

2. is known to the recipient.

Upon receipt of the message, the recipient checks whether the received paper carries an unchanged message and the signature of the sender. If both are true, the signed message is accepted. However, we often want to transmit not only paper messages, but also digital messages. Digital messages are bit strings, i.e., elements from the set $\{0.1\}^*$. Unfortunately, the classic solution for paper messages is not applicable here. Therefore we use *digital signature schemes* (or in short: *digital signatures*).

Why do we need signatures on digital messages? There are numerous examples of applications of digital signatures, some of them we use almost daily.

Digital certificates on the Internet. Digital certificates are digitally signed cryptographic keys. Such certificates are an important basic building block of a central security mechanism on the Internet, used in so-called *public key infrastructures* (PKI), for instance. We use such certificates on an almost daily basis, for example, when we use our web browser to access a secure website whose address begins with `https://`. Then the web browser communicates with the server via the TLS protocol [DA99, RD08, Res18], which authenticates the communication partner using a certificate.

Electronic payments. When we make an electronic money transaction, for example by paying with a credit card, the authenticity and integrity of messages is also ensured by digital signatures. In particular, digital signatures are an important security component of the EMV (Europay/Mastercard/VISA) framework for secure credit card payment. When we withdraw money with our EC card at cash machines, the authenticity of the card is ensured by a protocol that uses digital signatures.

Operating system updates. When an update for our operating system, whether Windows, Linux or MacOS, is downloaded from the Internet, it is necessary to verify the its authenticity. This is often done automatically by the update or package management of the operating system. Digital signatures are used to ensure the authenticity of the downloaded software.

Electronic identity card. Electronic identity cards, such as the *Elektronische Personalausweis* in Germany, also contain a key pair for a digital signature algorithm. One feature used to promote the electronic identity card in Germany is the ability to prove one's identity by means of digital signatures on the Internet — for example, when shopping online or to comply with youth protection regulations.

Cryptographic theory. Digital signatures not only have practical applications, but are also essential for cryptographic theory. In particular, they are an important building block for the construction of other cryptosystems. This includes, for example, *public key* encryption algorithms with strong security features such as *adaptive chosen-ciphertext* (CCA) security or *simulation-sound zero-knowledge* proof systems [Sah99, CHK04, Lin03, Gro06].

What exactly are digital signatures? Digital signature schemes are a cryptographic equivalent to classical, handwritten signatures.

- The sender has a (in practice unique) cryptographic key pair (pk, sk). The *secret key* sk is used by the sender to generate a signature. This key is secret, so only the sender knows it. The *public key* pk is publicly known, especially the recipient knows pk. This key is used to verify digital signatures.

- To sign a message m, the sender creates a signature σ using the secret key:

$$\sigma \xleftarrow{\$} \mathsf{Sign}(sk, m)$$

- To verify whether σ is a valid signature for a message m, the recipient runs $\mathsf{Vfy}(pk, m, \sigma)$.

What are digital signature schemes *not*? A common misunderstanding is that digital signatures are directly related to *public key encryption* schemes. Occasionally, one has to read the following statement in books or lecture notes:

> *"Computing a digital signature means to encrypt the message with the secret key."*

That is *not* true in general, and in particular it does not apply to nearly all signature and encryption schemes. The so-called *textbook-RSA* construction of public-key encryption and digital signatures seems to be the only example to which the above statement applies.

1.1 Definition of Digital Signatures

We define digital signatures as follows:

Definition 1. A digital signature scheme is a triple $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ of (possibly probabilistic) polynomial time algorithms, where

- $\mathsf{Gen}(1^\lambda)$ receives as input a security parameter (*security parameter*) λ (in unary notation) and outputs a key pair (pk, sk).
- $\mathsf{Sign}(sk, m)$ receives as input the security parameter, the *secret key sk* and a message $m \in \{0,1\}^n$. It outputs a signature σ.
- $\mathsf{Vfy}(pk, m, \sigma)$ receives as input the security parameter, the *public key pk*, a message m and a signature σ. It outputs 0 or 1, where 1 means that σ is accepted as a signature for message m and *public key pk*. 0 means that the signature is not accepted.

Essential properties of a signature scheme. We expect a digital signature scheme to provide at least the two properties *correctness* and *soundness*.

Correctness. Correctness means essentially:

> *"The scheme works."*

Thus, for valid input values, the scheme should produce an output with correct distribution over the set of all possible output values. For digital signatures this means: if we

1. use $(pk, sk) \overset{\$}{\leftarrow} \mathsf{Gen}(1^\lambda)$ to create a key pair, and then
2. create a signature $\sigma \overset{\$}{\leftarrow} \mathsf{Sign}(sk, m)$ for some message m, and then
3. execute the verification algorithm $\mathsf{Vfy}(pk, m, \sigma)$,

then the signature is always accepted, that is, it holds that $\mathsf{Vfy}(pk, m, \sigma) = 1$. This must be true for all possible messages m and for all key pairs $(pk, sk) \overset{\$}{\leftarrow} \mathsf{Gen}(1^\lambda)$ that can be generated by algorithm Gen.[1]

Soundness. Soundness means essentially:

> *"The scheme is secure."*

This means that there should exist no efficient algorithm (the "adversary") that breaks the desired security property of the method with a non-negligible success probability. What the "desired security property" exactly is depends on the requirements of the application of the signature scheme. We will consider many different useful security definitions for digital signatures.

[1] Sometimes this is relaxed to all but a negligibly small fraction of all keys.

1.2 Security of Digital Signatures

The term *security* itself is quite simple, but also a little spongy and ambiguous. Therefore we have to make this term more precise.

1.2.1 Attacker Model and Attacker Target

To describe precisely what "security" means for a cryptographic protocol (e.g. a signature scheme), we need to specify two things:

1. The *attacker target* describes what an adversary has to accomplish to successfully "break" the security of the signature scheme. In the context of signatures, this could be, for example:

   (a) The adversary must forge a valid signature for a *prescribed* message m^*, where m^* is randomly selected. It is considered only successful if it outputs a valid signature σ^* for exactly this message, but not if it forges a signature for a different message m' — not even if m' message is very similar or even semantically identical (whatever this may mean for a given application) to m^*.

   A signature scheme that is secure against such attacks is called *secure against universal forgery* (or *universally unforgeable*, UUF).

   (b) The adversary is considered successful if it creates a valid signature for a *any* message. A signature scheme that is secure against such attacks is called *secure against existential forgery* (or *existentially unforgeable*, EUF).

   Note that EUF seems to be a much stronger security goal than UUF, since an adversary successfully breaks the security even if it is only able to forge a signature for a seemingly meaningless message that would probably never occur in a real-world application. Indeed, we will formalize this intuition later, when we study different security notions for signature schemes and their relation.

2. An *attacker model*, which describes which "abilities" an adversary has. In the context of signatures this can be for example:

   (a) The adversary gets only the *public key pk* as input. However, he has no access to valid signatures created by the honest sender. His goal is to forge a valid signature. This form of attack is called *no-message attacks* (NMA).

   This is not a particularly strong adversarial model, since it is assumed that the adversary does not have access to valid signatures, which could help him to forge a new signature.

   (b) The adversary may select a list $(m_1, \ldots, m_q)$ of messages. Then it will receive the *public key pk* as well as signatures $(\sigma_1, \ldots, \sigma_q)$ for the selected messages. The adversary wins only if it forges a valid signature for a *new* message m^*, which is not contained in the list $(m_1, \ldots, m_q)$, of course, as otherwise security is impossible to achieve.

   This attacker model is called *non-adaptive chosen-message attacks* (naCMA). It takes into account that an adversary may have access to valid signatures for certain messages.

(c) The adversary receives the *public key* as input. He can then trick the sender into issuing signatures $(\sigma_1, \ldots, \sigma_q)$ for arbitrary messages $(m_1, \ldots, m_q)$ selected by the adversary. The adversary may select these messages *adaptively*, e.g., depending on pk or signatures σ_j, $j < i$, already received. The adversary goal is to forge a valid signature for a *new* message, which is not contained in the list $(m_1, \ldots, m_q)$ of messages selected by the adversary.

Such attacks are called *adaptive chosen-message attacks* (CMA). This model takes into account that an adversary could also trick the users of the signature process to sign certain messages that the adversary has chosen depending on pk and on previously observed signed messages.

Security definitions. A security definition (*security notion*) is a combination of an attacker model with an attacker target.

Security definition = attacker model + attacker target

For example, the most important common security definition for digital signature schemes is *existential unforgeability under adaptive chosen-message attacks* (EUF-CMA), which was introduced by Goldwasser, Micali and Rivest [GMR84, GMR88]. This results from the combination of 1(b) with 2(c).

In this way, many more security definitions result from other combinations of attacker models with attack targets, see Table 1.1. Fortunately, we will not need all these security definitions in this lecture. It is probably now clear that there are many different ways to define *"security"* of digital signatures.

	2.(a)	2.(b)	2.(c)
1.(a)	UUF-NMA	UUF-naCMA	UUF-CMA
1.(b)	EUF-NMA	EUF-naCMA	EUF-CMA

Table 1.1: Various security definitions.

In addition to the definitions presented so far, there are numerous others in the literature. The commonly accepted standard security notion for digital signature schemes is *existential unforgeability under adaptive chosen message attacks*, i.e., EUF-CMA security. However, stronger notions such as *strong* EUF-CMA-security and weaker notions such as *existential unforgeability under non-adaptive chosen message attacks* (EUF-naCMA) will also turn out to be extremely useful.

1.2.2 Security Experiments

We want to prove later that a given signature scheme is *secure* (under certain assumptions). For this purpose, it is necessary to describe the attacker models and attacker targets very precisely. An elegant technique for a precise description of a security definition is to specify a *security experiment*.

Two parties participate in a security experiment:

1. The adversary $\mathcal{A}$.

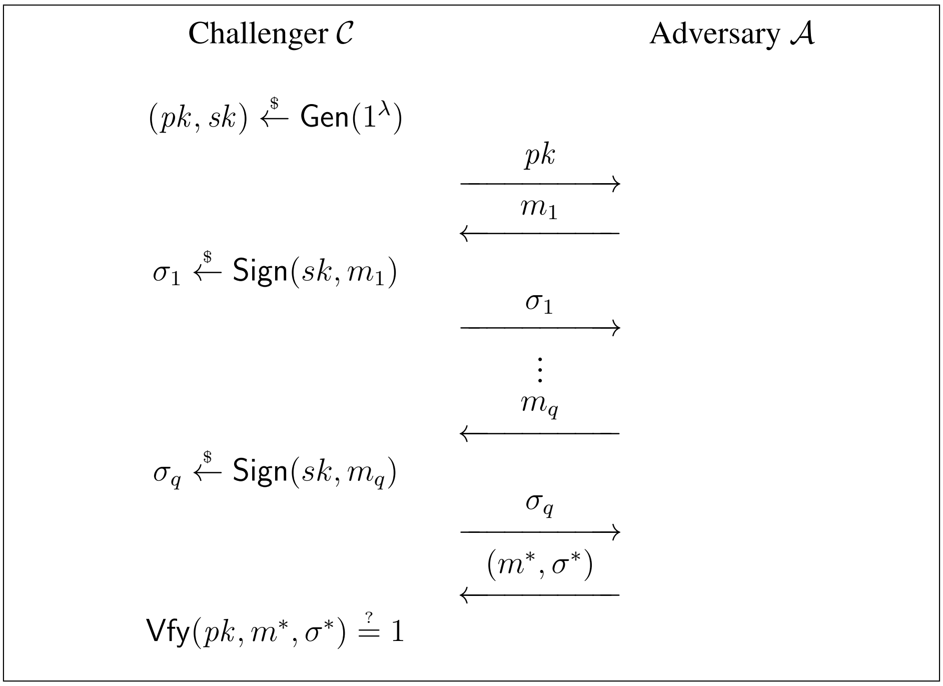
Challenger $\mathcal{C}$
Adversary $\mathcal{A}$
$(pk, sk) \xleftarrow{\$} \mathsf{Gen}(1^\lambda)$
$\xrightarrow{pk}$
$\xleftarrow{m_1}$
$\sigma_1 \xleftarrow{\$} \mathsf{Sign}(sk, m_1)$
$\xrightarrow{\sigma_1}$
$\vdots$
$\xleftarrow{m_q}$
$\sigma_q \xleftarrow{\$} \mathsf{Sign}(sk, m_q)$
$\xrightarrow{\sigma_q}$
$\xleftarrow{(m^*, \sigma^*)}$
$\mathsf{Vfy}(pk, m^*, \sigma^*) \stackrel{?}{=} 1$

Figure 1.1: The EUF-CMA security experiment.

2. A "challenger" $\mathcal{C}$.

In the experiment the adversary plays a game against the challenger. It wins the game if it breaks the security of the signature scheme.

The EUF-CMA security experiment. The best way to explain the idea of security experiments is to consider an example, see Figure 1.1. The EUF-CMA security experiment with adversary $\mathcal{A}$, challenger $\mathcal{C}$ and signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ runs as follows:

1. The challenger $\mathcal{C}$ generates a key pair $(pk, sk) \xleftarrow{\$} \mathsf{Gen}(1^k)$. The adversary gets pk.

2. Now the adversary $\mathcal{A}$ may have any messages $m_1, \ldots, m_q$ signed by the challenger.

   For this purpose it sends message m_i to the challenger. The challenger calculates $\sigma_i \xleftarrow{\$} \mathsf{Sign}(sk, m)$ and replies with σ_i.

   This step can be repeated by the adversary as often as it wants. If we consider adversaries with polynomially-bounded runtime, $q = q(k)$ is a polynomial in the security parameter.

3. At the end $\mathcal{A}$ outputs a message m^* with signature σ^*. It *wins* the game if

$$\mathsf{Vfy}(pk, m^*, \sigma^*) = 1 \qquad \text{and} \qquad m^* \notin \{m_1, \ldots, m_q\}.$$

   So $\mathcal{A}$ wins if σ^* is a valid signature for m^*, and it did not ask the challenger $\mathcal{C}$ for a signature on m^*. The second condition is of course necessary to exclude trivial attacks.

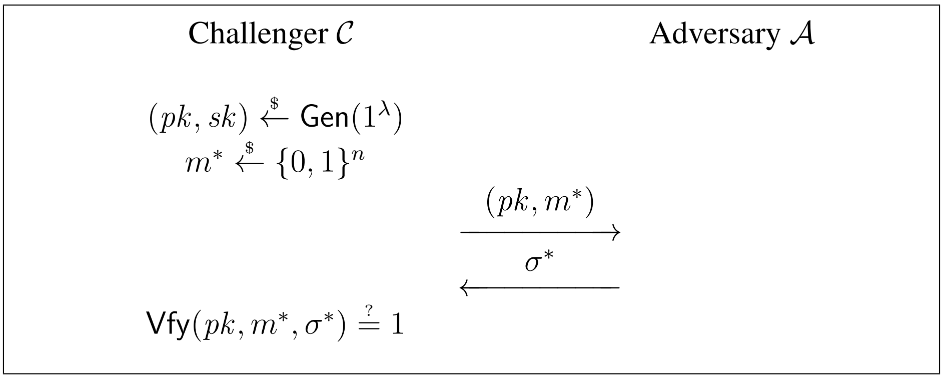
Challenger $\mathcal{C}$
Adversary $\mathcal{A}$
$(pk, sk) \xleftarrow{\$} \mathsf{Gen}(1^\lambda)$
$m^* \xleftarrow{\$} \{0,1\}^n$
(pk, m^*)
σ^*
$\mathsf{Vfy}(pk, m^*, \sigma^*) \overset{?}{=} 1$

Figure 1.2: The UUF-NMA security experiment.

Definition 2. We say that (Gen, Sign, Vfy) is secure in the sense of EUF-CMA, if for all PPT adversaries $\mathcal{A}$ in the EUF-CMA experiment it holds that

$$\Pr[\mathcal{A}^{\mathcal{C}}(pk) = (m^*, \sigma^*) : \mathsf{Vfy}(pk, m^*, \sigma^*) = 1 \land m^* \notin \{m_1, \ldots, m_q\}] \leq \mathsf{negl}(\lambda)$$

for a negligible function negl in the security parameter.

Remark 3. As a reminder, a function negl : $\mathbb{N} \to [0,1]$ is *negligible*, if for every constant $c \geq 0$ there exists a number λ_c, so for all $\lambda > \lambda_c$ it holds that

$$\mathsf{negl}(\lambda) \leq \frac{1}{\lambda^c}.$$

In other words, $\mathsf{negl}(\lambda)$ is a negligible function, if it converges to 0 faster than the inverse of any polynomial in λ, i.e. $\mathsf{negl}(\lambda) = o(1/\mathsf{poly}(\lambda))$.

Another example: The UUF-NMA security experiment. The UUF-NMA experiment runs as follows:

1. The challenger $\mathcal{C}$ generates a key pair $(pk, sk) \xleftarrow{\$} \mathsf{Gen}(1^\lambda)$. Furthermore it chooses a random message $m^* \xleftarrow{\$} \{0,1\}^n$. The adversary receives (pk, m^*).
2. At the end $\mathcal{A}$ outputs a signature σ^*. It wins the game, if

   $$\mathsf{Vfy}(pk, m^*, \sigma^*) = 1.$$

   So $\mathcal{A}$ wins if σ^* is a valid signature for m^*.

Definition 4. We say that (Gen, Sign, Vfy) is secure in the sense of UUF-NMA, if for all PPT adversaries $\mathcal{A}$ in the UUF-NMA experiment it holds that

$$\Pr[\mathcal{A}^{\mathcal{C}}(pk, m^*) = \sigma^* : \mathsf{Vfy}(pk, m^*, \sigma^*) = 1] \leq \mathsf{negl}(\lambda)$$

for a negligible function negl in the security parameter.

When we want to analyze the security of a signature scheme in the future, we will always specify a security experiment to specify the considered security definition, that is, the attacker model and the what the adversary must accomplish to “win the game”.

Exercise 5. Describe the EUF-naCMA security experiment and provide a formal definition that specifies when a signature scheme is considered EUF-naCMA-secure.

1.2.3 Analyzing Relations between Security Definitions

By combining the attacker targets presented so far (EUF,UUF) with the presented attacker models (NMA,naCMA,CMA) we obtain 6 security definitions, see Table 1.1. A question that now arises is how these definitions relate to each other.

Intuition. Intuitively it seems clear that UUF security offers us lower security guarantees than EUF security because:

- To break a signature scheme in the sense of UUF, the adversary must forge a signature for a *prescribed* message, it is not allowed to choose it itself.

- To break a signature scheme in the sense of EUF, it is sufficient to forge a signature for an *arbitrary* message. In particular, the adversary can select the message for which it is "easiest" to forge a signature.

Equally, it seems clear that NMA security offers us lower security guarantees than naCMA security, and that naCMA security offers lower security guarantees than CMA security:

- The difference between the NMA experiment and the naCMA experiment is that in the latter the adversary has access to valid signatures, even for messages of its own choice. Thus, if a signature scheme is secure against such an adversary, then in particular it is secure if the adversary does not have access to any signatures.

- In the naCMA experiment, the adversary must determine a list of messages for which it wants to see signatures *before* it knows the public key with respect to which it has to forge. It receives the list of signatures then along with the public key. In contrast, in the CMA experiment it may request signatures at any time. Thus, if a signature scheme is secure against an adversary that can adaptively select its requests to the signature oracle, then it is in particular secure if the adversary cannot do so.

So there seems to be a hierarchy between the security definitions that we know so far:

$$\begin{array}{ccccc} \mathsf{UUF\text{-}NMA} & \leq & \mathsf{UUF\text{-}naCMA} & \leq & \mathsf{UUF\text{-}CMA} \\ \rotatebox{-90}{$\leq$} & & \rotatebox{-90}{$\leq$} & & \rotatebox{-90}{$\leq$} \\ \mathsf{EUF\text{-}NMA} & \leq & \mathsf{EUF\text{-}naCMA} & \leq & \mathsf{EUF\text{-}CMA} \end{array}$$

Thus, of the definitions presented so far, UUF-NMA seems to be the weakest, and EUF-CMA the strongest. These relations can be proved by *reduction*. To prove all above "$\leq$"-relationships individually would not be very interesting, because the proofs always follow the same recipe. Therefore we will now consider as a simple example only the statement

$$\mathsf{UUF\text{-}NMA} \leq \mathsf{EUF\text{-}CMA}$$

The other relations in the diagram above can be shown in the same way.

Theorem 6. *Let $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ be a signature scheme. If Σ is EUF-CMA-secure, then it is UUF-NMA-secure.*

Proof. We assume that Σ is $\mathsf{EUF\text{-}CMA}$-secure, but not $\mathsf{UUF\text{-}NMA}$-secure. If we can show this assumption to be contradictory, then it must be wrong.

If Σ is not $\mathsf{UUF\text{-}NMA}$-secure, then there exists an adversary $\mathcal{A}_{\mathsf{UUF\text{-}NMA}}$ that runs in polynomial time with

$$\Pr[\mathcal{A}_{\mathsf{UUF\text{-}NMA}}(pk, m^*) = \sigma^* : \mathsf{Vfy}(pk, m^*, \sigma^*) = 1] \geq \frac{1}{\mathsf{poly}(\lambda)}$$

for a polynomial $\mathsf{poly}(\cdot)$ in the security parameter λ. We show that we can construct a new adversary $\mathcal{A}_{\mathsf{EUF\text{-}CMA}}$ from $\mathcal{A}_{\mathsf{UUF\text{-}NMA}}$, successfully breaking the $\mathsf{EUF\text{-}CMA}$ security of Σ. The runtime of $\mathcal{A}_{\mathsf{EUF\text{-}CMA}}$ is approximately equal to the runtime of $\mathcal{A}_{\mathsf{UUF\text{-}CMA}}$, and the success probability is also at least $1/\mathsf{poly}(\lambda)$, and thus not negligible. This means that Σ is not $\mathsf{EUF\text{-}CMA}$-secure, which yields the desired contradiction.

Adversary $\mathcal{A}_{\mathsf{EUF\text{-}CMA}}$ uses $\mathcal{A}_{\mathsf{UUF\text{-}NMA}}$ as a subroutine, it proceeds follows (see also Figure 1.3).

1. $\mathcal{A}_{\mathsf{EUF\text{-}CMA}}$ receives the public key pk as input. It then chooses a random message $m^* \overset{\$}{\leftarrow} \{0,1\}^n$ and starts $\mathcal{A}_{\mathsf{UUF\text{-}NMA}}$ with input (pk, m^*).

2. If $A_{\mathsf{UUF\text{-}NMA}}$ outputs a signature σ^* with $\mathsf{Vfy}(pk, m^*, \sigma^*) = 1$, which happens with probability at least $1/\mathsf{poly}(\lambda)$ by assumption, then $\mathcal{A}_{\mathsf{EUF\text{-}CMA}}$ outputs the tuple (m^*, σ^*). Otherwise $\mathcal{A}_{\mathsf{EUF\text{-}CMA}}$ outputs an error symbol $\perp$ and terminates.

Obviously (m^*, σ^*) is a valid forgery in the sense of $\mathsf{EUF\text{-}CMA}$, because $\mathcal{A}_{\mathsf{EUF\text{-}CMA}}$ never asked his challenger for a signature for m^* (in fact, $\mathcal{A}_{\mathsf{EUF\text{-}CMA}}$ never asks for any signature, so it is actually an $\mathsf{EUF\text{-}NMA}$ adversary).

The runtime of $\mathcal{A}_{\mathsf{EUF\text{-}CMA}}$ is about the same as $\mathcal{A}_{\mathsf{UUF\text{-}NMA}}$, except for a minor overhead. The success probability of $\mathcal{A}_{\mathsf{EUF\text{-}CMA}}$ is identical to the success probability of $\mathcal{A}_{\mathsf{UUF\text{-}NMA}}$. Thus we have shown that if Σ is not $\mathsf{UUF\text{-}NMA}$-secure, then it is not $\mathsf{EUF\text{-}CMA}$-secure either. This applies to any signature scheme Σ. □

Exercise 7. How would you prove the statement "$\mathsf{UUF\text{-}CMA} \leq \mathsf{EUF\text{-}CMA}$"?

Exercise 8. Show that UUF security is *strictly* weaker than EUF security. Assume that you had an $\mathsf{UUF\text{-}CMA}$-secure scheme and show how you can modify it so that it remains provably $\mathsf{UUF\text{-}CMA}$-secure but is no longer $\mathsf{EUF\text{-}NMA}$-secure.

Exercise 9. Show that NMA security is *strictly* weaker than CMA security. Assume that you had a $\mathsf{EUF\text{-}NMA}$-secure scheme, and show how you can modify it so that it remains $\mathsf{EUF\text{-}NMA}$-secure, but is no longer $\mathsf{UUF\text{-}CMA}$-secure.

1.2.4 Perfectly Secure Digital Signatures Do Not Exist

For some cryptographic tasks, such as symmetric encryption or cryptographic secret sharing, there are *information-theoretically secure* solutions that are (often) unfortunately quite impractical, but have the strong characteristic that they cannot be broken, not even by unrestricted adversaries. A natural question is whether such strong signature schemes can also exist. Unfortunately, it is quite easy to see that they do not.

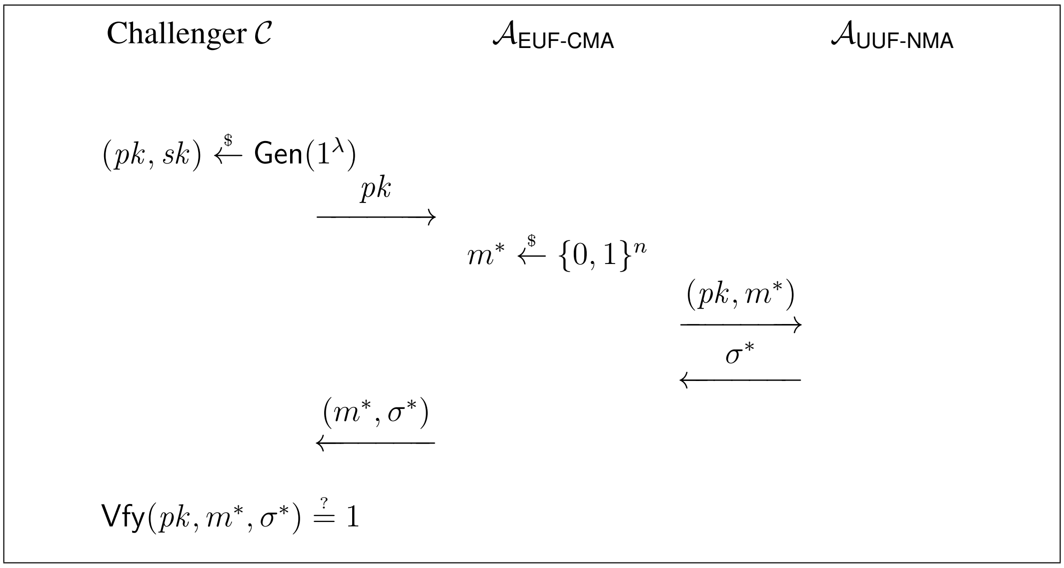
Challenger $\mathcal{C}$
$\mathcal{A}_{\mathsf{EUF\text{-}CMA}}$
$\mathcal{A}_{\mathsf{UUF\text{-}NMA}}$
$(pk, sk) \xleftarrow{\$} \mathsf{Gen}(1^\lambda)$
$\xrightarrow{pk}$
$m^* \xleftarrow{\$} \{0,1\}^n$
$\xrightarrow{(pk, m^*)}$
$\xleftarrow{\sigma^*}$
$\xleftarrow{(m^*, \sigma^*)}$
$\mathsf{Vfy}(pk, m^*, \sigma^*) \overset{?}{=} 1$

Figure 1.3: Construction of $\mathcal{A}_{\mathsf{EUF\text{-}CMA}}$ from $\mathcal{A}_{\mathsf{UUF\text{-}NMA}}$.

> *There exists no signature scheme that is secure (in the sense of a reasonable security definition) against an adversary with unlimited resources (especially computing time) at his disposal.*

Of course, the above statement is rather imprecise, since it is not clear what is meant by *"a reasonable security definition"*. However, it hits the core, as it applies to every common security definition, and seems that any meaningful definition can be used here. In particular, we can show that not even our weakest security definition $\mathsf{UUF\text{-}NMA}$ cannot be achieved if we allow unlimited adversaries.

Theorem 10. *Let $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ be any signature scheme. There exists an $\mathsf{UUF\text{-}NMA}$-adversary $\mathcal{A}$ breaking the security of Σ with running time $O(2^L)$ and success probability 1. Here L is the smallest integer such that for every message m there exists a valid signature of length L.*

The proof of this theorem is not very difficult, so we leave it as an exercise.

Exercise 11. Prove Theorem 10 by describing the adversary $\mathcal{A}$.

So we must always limit the runtime of the adversaries we consider. Therefore we model adversaries as (possibly probabilistic) Turing machine with polynomial runtime *(probabilistic polynomial-time, PPT)*.

Next, we can now ask ourselves if there exist at least signature schemes that are *perfectly secure* if we restrict the adversary to a polynomial runtime in the security parameter λ. "Perfectly secure" here means that the probability of success of the adversary is zero. Again, it is easy to show that this is not possible either.

Theorem 12. *Let $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ be any signature scheme. There exists a $\mathsf{UUF\text{-}NMA}$-adversary $\mathcal{A}$ breaking Σ with running time $O(L)$ and success probability 2^{-L}. Here L is the smallest integer such that for every message m there exists a valid signature of length L.*

So we have to accept that an adversary always have a non-zero (but possibly very small) probability of success. This proof is not very difficult either, so we again present it as an easy exercise.

Exercise 13. Prove Theorem 12 by describing the adversary $\mathcal{A}$.

Thus, the highest goal that we can achieve in the construction of secure digital signature schemes is to give adversaries with *polynomially bounded runtime* at most a *negligible probability of success* to break security. Note that the above theorems also give a lower bound on the bit-length of signatures of a secure digital signature scheme.

1.3 Extending the Message Space, Generically

We would like to build signature schemes that are able to sign arbitrary messages of *any length*. So the message space should be the set $\{0,1\}^*$ of all finite bit strings. However, many of the constructions we will discuss below have a *finite message space*, such as

- the set $\{0,1\}^n$ of all bit strings of length n or
- the set $\mathbb{Z}_p$ of residue classes modulo p, which we represent by the integers $\{0,\dots,p-1\}$.

Fortunately, it is very easy to extend a signature scheme with finite message space to a scheme that can be used to sign messages of any length. This is done with the help of a *collision resistant hash function*.

Collision-resistant cryptographic hash functions.

Definition 14. A *cryptographic hash function* H consists of two polynomial-time algorithms $H = (\mathsf{Gen}_H, \mathsf{Eval}_H)$.

$\mathsf{Gen}_H(1^\lambda)$ receives the security parameter λ as input and outputs a key t. This key t describes a function

$$H_t : \{0,1\}^* \to \mathcal{M}_t$$

where $\mathcal{M}_t$ is a finite set, which is also defined by t.

$\mathsf{Eval}_H(1^\lambda, t, m)$ receives as input the security parameter λ, a function description t, and a message m and computes the function value $H_t(m)$.

In the following the reference to the security parameter λ will usually be clear. Therefore, for the sake of simplicity of notation, we will write H and mean the description t of a function H_t, which was created by $t \xleftarrow{\$} \mathsf{Gen}_H(1^\lambda)$. We will also write $H(m)$ as a short form for $H_t(m)$.

Definition 15. We say that $H = (\mathsf{Gen}_H, \mathsf{Eval}_H)$ is *collision resistant*, if for $t \xleftarrow{\$} \mathsf{Gen}_H(1^\lambda)$ and for all polynomial time algorithms $\mathcal{A}$ it holds that

$$\Pr[\mathcal{A}(1^\lambda, t) = (x, x') : H_t(x) = H_t(x')] \leq \mathsf{negl}(\lambda)$$

for a negligible function negl in λ.

Remark 16. Cryptographic hash functions that are collision resistant in the above sense can be constructed from different complexity assumptions, like the discrete logarithm problem, for instance. In practice, however, dedicated hash functions such as SHA-256 are used for reasons of efficiency.

The collision resistance of these hash functions like SHA-256 cannot be formulated in terms of our asymptotic security definitions, because these functions are fixed, that is, there is no key t that depends on the security parameter. In particular, these hash functions have a constant output length, so there is of course a polynomial-time algorithm that finds collisions.

Nevertheless, it seems impossible to find collisions for these hash functions according to current knowledge. Therefore, any application of collision resistant hash functions that we will consider can be instantiated in practice with such a dedicated hash function. The parameterization of the hash function by the key t is merely an artifact of our asymptotic security definitions.

Extension of the message space. A cryptographic hash function can be used to extend the message space of a signature scheme. Let $\Sigma' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Vfy}')$ be a signature scheme with a finite message space $\mathcal{M}$ and let $H : \{0,1\}^* \to \mathcal{M}$ be a cryptographic hash function. We define a signature scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ with infinite message space $\{0,1\}^*$ as follows:

$\mathsf{Gen}(1^\lambda)$**.** The key generation algorithm Gen calculates (pk, sk) by running the key generation algorithm $(pk, sk) \overset{\$}{\leftarrow} \mathsf{Gen}'(1^\lambda)$ from Σ'.

$\mathsf{Sign}(sk, m)$**.** To sign message $m \in \{0,1\}^*$, first $H(m) \in \mathcal{M}$ is calculated and then signed by $\sigma \overset{\$}{\leftarrow} \mathsf{Sign}'(sk, H(m))$.

$\mathsf{Vfy}(pk, m, \sigma)$**.** The verification algorithm calculates $b = \mathsf{Vfy}'(pk, H(m), \sigma)$ and outputs b.

So the only modification is that the message is hashed before signing and before verification.

Theorem 17. *If Σ' is secure in the sense of* EUF-CMA *(resp.* EUF-naCMA*) and H is collision resistant, then Σ is secure in the sense of* EUF-CMA *(resp.* EUF-naCMA*).*

Exercise 18. Prove Theorem 17.

Remark 19. The above transformation introduces an additional complexity assumption, besides the assumptions necessary for the security of the signature scheme Σ', namely the existence of collision resistant hash functions. A similar extension of the message space is also possible *without* additional complexity assumptions, using *universal one-way* hash functions. However, we will not go into this further. A description can be found for example in the textbook by Jon Katz [Kat10].

Chapter 2

One-Time Signatures

One-time signature schemes are a basic and quite simple class of signature schemes. They only have to meet relatively weak security requirements, because it is sufficient if they are secure when used *once only*. One-time use means that only one signature is issued for every public key.

At first glance, the concept of one-time signatures does not seem to make much sense. Why should it be good to have a signature procedure that allows you to issue *only one signature*?

Interestingly, one-time signatures are a very powerful tool. In particular, we can use them to construct many-time signature schemes in a generic way, with which we can then issue a virtually unlimited number of signatures. The resulting signatures can have strong security properties, such as EUF-CMA security.

One-time signatures also can be constructed from very weak complexity assumptions, such as the difficulty of the discrete logarithm problem or, more generally, the existence of one-way functions. Together with the previously mentioned generic construction of many-time signatures from one-time signatures, this is interesting because it helps us to explore the minimal complexity assumptions needed for the construction of digital signatures. Furthermore, one-time signatures are an important building block for many cryptographic constructions.

In this chapter, we first adapt the well-known security definitions EUF-CMA and EUF-naCMA to one-time signatures. Afterwards, we consider a general construction of one-time signatures from one-way functions and two special constructions based on concrete complexity assumptions. Finally, we present some applications of one-time signatures. This includes a generic construction that allows to build an EUF-CMA-secure signature procedure from any EUF-naCMA-secure scheme. This transformation is very useful, and is used in the literature again and again. After that we describe tree-based signatures using *Merkle Trees*.

2.1 Security Definition

If we want to allow an attacker to make (non-adaptive) chosen-message signature queries, as we do in the CMA and naCMA attacker models, when considering one-time signature schemes we have to keep in mind that such schemes only have to guarantee security if only a single signature is issued. For this reason, we will weaken the CMA and naCMA attacker models for the consideration of one-time signatures. Instead of limiting the number of messages for which the attacker can request a signature by a polynomial $q = q(\lambda)$, we allow only a *single* signature

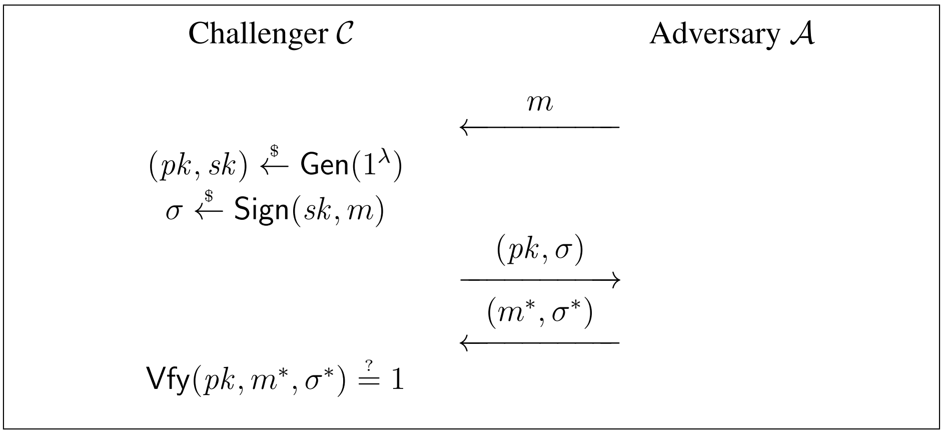
Challenger $\mathcal{C}$
Adversary $\mathcal{A}$
m
$(pk, sk) \xleftarrow{\$} \mathsf{Gen}(1^\lambda)$
$\sigma \xleftarrow{\$} \mathsf{Sign}(sk, m)$
(pk, σ)
(m^*, σ^*)
$\mathsf{Vfy}(pk, m^*, \sigma^*) \overset{?}{=} 1$

Figure 2.1: The EUF-1-naCMA security experiment.

request for one-time signatures. So we set $q = 1$. If we combine this attacker model with the attacker target *existential forgery* (EUF), this results in two new security definitions

- *existential unforgeability under one-time adaptive chosen-message attack* (EUF-1-CMA)
- *existential unforgeability under one-time non-adaptive chosen-message attack* (EUF-1-naCMA)

In this chapter, we will be particularly interested in EUF-1-naCMA security, since this is sufficient (as we will see later) to construct EUF-CMA secure signatures from it.

««««< HEAD ||||||| merged common ancestors (todo: Move to later paragraph Therefore we will only prove that Lamport's one-time signature scheme is EUF-1-naCMA-safe, assuming that the f function is a one-way function.[1]) ======= (todo: Move to later paragraph Therefore we will only prove that Lamport's one-time signature scheme is EUF-1-naCMA-secure, assuming that the f function is a one-way function.[2]) »»»»> 0b30b6e726abaccd5920935f883eefa0a3c578af TODO TODO

EUF-1-naCMA Security. The EUF-1-naCMA security experiment runs as follows (see also Figure 2.1):

1. The adversary $\mathcal{A}$ selects a message m for which he wants to request a signature from the challenger.
2. The challenger $\mathcal{C}$ generates a key pair $(pk, sk) \xleftarrow{\$} \mathsf{Gen}(1^\lambda)$ and a signature $\sigma \xleftarrow{\$} \mathsf{Sign}(sk, m)$, and sends (pk, σ) to $\mathcal{A}$.
3. $\mathcal{A}$ returns a message m^* with signature σ^*.

Definition 20. We say that $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ is a EUF-1-naCMA secure one-time signature scheme, if for all polynomial-time attackers $\mathcal{A}$ in the EUF-1-naCMA experiment holds that

$$\Pr[\mathcal{A}^{\mathcal{C}} = (m^*, \sigma^*) : \mathsf{Vfy}(pk, m^*, \sigma^*) = 1 \wedge m^* \neq m] \leq \mathsf{negl}(\lambda)$$

[1] It can also be shown that Lamport's signatures are EUF-1-CMA-safe. The proof is very similar, so it is a good exercise.

[2] It can also be shown that Lamport's signatures are EUF-1-CMA-secure. The proof is very similar, so it is a good exercise.

for a negligible function negl in the security parameter.

Exercise 21. Describe the EUF-1-CMA security experiment and define when the attacker wins in the experiment.

2.2 One-time Signatures from One-Way Functions

In this chapter we describe the construction of one-time signatures from one-way functions by Lamport [Lam79]. This construction is especially interesting for two reasons. First, the existence of one-way functions is one of the weakest assumptions in cryptography, and implied by many other computational hardness assumptions. Second, it establishes that secure digital signature schemes exist if and only if one-way functions exist.

2.2.1 One-Way Functions

The basic idea of one-way functions is:

- For a given function $f : \{0,1\}^* \to \{0,1\}^*$ and an input $x \in \{0,1\}^*$ it should be "easy" to calculate the function value $y = f(x)$. "Easy" means here that the runtime is limited by a polynomial in the length of x, as usual.
- Given a value $y = f(x)$, however, it should be "practically impossible" to calculate $f^{-1}(y) = x$, i.e., to find a preimage of y with respect to f. "Practically impossible" means here that there should not exist a polynomial time algorithm computing $f^{-1}(y)$ on input y.

An important basic assumption in cryptography is that one-way functions exist. Whether this assumption is correct is unknown. Nevertheless, the existence of one-way functions is one of the weakest assumptions in cryptography, and it is necessary to construct many interesting things, such as block ciphers or digital signatures. For many other cryptographic primitives, such as *public key* encryption or identity-based encryption, one (probably) needs much stronger assumptions.

Definition of one-way functions. Giving an abstract, general definition of one-way functions, which at the same time consistently covers all important examples of one-way functions used in cryptography (like cryptographic hash functions, the RSA function, or the discrete exponential function), is not that simple — but fortunately also not necessary.

For the sake of simplicity, we will therefore consider functions that can be described as a single function of the form

$$f : \{0,1\}^* \to \{0,1\}^*$$

So we consider functions whose input and output spaces contain *all* finite bit-strings. Furthermore, we assume that the function is *efficiently computable*, i.e., that it has a *description of constant length*, for example in the form of an algorithm, and that for an input x with $|x| = \lambda$ the length of the output $y = f(x)$ is bounded by a polynomial $\mathsf{poly}(\lambda)$ in λ. This is helpful, as it saves us a lot of irrelevant technical details and thus simplifies (todo: Strange conbination of words - the consideration considerably GD: how about 'our exposition considerably'). At the same time, the simplified formulation also brings the basic idea of the constructions, which is what we actually want to talk about, more to the fore.

TODO

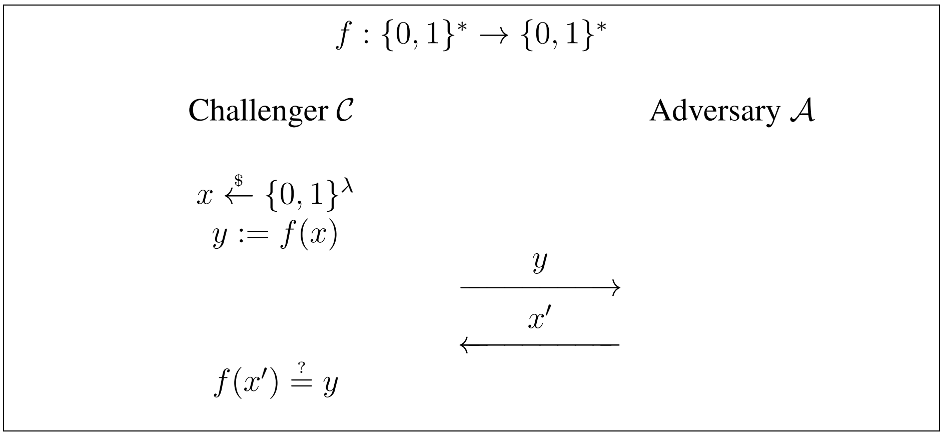

Figure 2.2: The security experiment for one-way functions.

Security experiment for one-way functions. The security experiment for a one-way function f runs as follows (cf. Figure 2.2).

1. The challenger $\mathcal{C}$ chooses $x \overset{\$}{\leftarrow} \{0,1\}^\lambda$ uniformly at random and computes $y = f(x)$.
2. The adversary receives the description of the function f and y as input. He "wins" the game if he outputs $x' \in \{0,1\}^*$ such that $f(x') = y$.

Definition 22. We say that f is a one-way function, if f can be computed efficiently and for all polynomial time algorithms $\mathcal{A}$ in the security experiment for one-way functions holds that

$$\Pr[\mathcal{A}(1^\lambda, f, y) = x' : f(x') = y] \leq \mathsf{negl}(\lambda)$$

for a negligible function negl.

Candidate one-way functions. Unfortunately we do not know whether one-way functions exist. But we do know some candidates for functions, which are assumed to be concrete instances of the abstract concept of one-way functions:

Cryptographic hash functions. A (rather mild) security requirement for a cryptographic hash function, such as MD5, SHA-1, RIPE-MD, or SHA-3, is that it is computationally hard to invert. Unfortunately, such functions are not secure one-way functions in the asymptotic formulation we have chosen, since the length of the output of these functions is constant and thus, of course, a polynomial time algorithm *exists* that finds preimages. Nevertheless, it seems practically impossible to compute preimages for these functions, according to the current state of knowledge, and these functions are suitable for instantiating the one-way-function-based procedures presented here in practice.

The discrete exponential function. Let $\mathbb{G}$ be a finite group with generator g and order p. For some such groups, the function $\mathbb{Z}_p \to \mathbb{G}$, $x \mapsto g^x$ is a candidate for a one-way function, namely, if the discrete logarithm problem in the group cannot be solved efficiently. This includes (probably) certain subgroups of the multiplicative group $\mathbb{Z}_p^*$ or certain elliptic curve groups.

The discrete exponential function is not immediately a function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ with domain and range $\{0,1\}^*$, but you can think of the discrete exponential function as a family of functions $(f_\lambda)_{\lambda \in \mathbb{N}}$. Part of the description of each function f_λ is the description of a group $\mathbb{G}$ with generator g and order p, where $p = p(\lambda)$ depends on λ.

The RSA Function. The RSA function $\mathbb{Z}_N \rightarrow \mathbb{Z}_N$, $x \mapsto x^e \bmod N$ is also assumed to be a one-way function (for suitably selected (N, e)).

The RSA construction can also be viewed as a family of functions $(f_\lambda)_{\lambda \in \mathbb{N}}$. Part of the description of each function f_λ are two integers (N, e), which define the function $f_\lambda : x \mapsto x^e \bmod N$.

The RSA function even represents a candidate for a *trapdoor one-way permutation*. This means that the function is efficiently and uniquely *invertible* if a matching "trapdoor" is known. In the case of RSA, the factorization of the modulus N serves as a trapdoor.

2.2.2 Lamport's One-Time Signature Scheme

In 1979, Lamport [Lam79] described a very simple and elegant one-time signature scheme based on one-way functions. In this chapter we describe Lamport's scheme and analyze its security.

To describe a signature scheme, we need to specify three algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$. In the case of Lamport's scheme these algorithms use a function $f : \{0,1\}^* \rightarrow \{0,1\}^*$, which we will assume to be a one-way function in the security analysis. We use n to denote the length of the messages to be signed, where $n = n(\lambda)$ is a polynomial in the security parameter λ. That is, the message space is $\{0,1\}^n$.

$\mathsf{Gen}(1^k)$. The key generation algorithm Gen selects $2n$ uniformly distributed random values

$$x_{1,0}, x_{1,1}, \ldots, x_{n,0}, x_{n,1} \overset{\$}{\leftarrow} \{0,1\}^\lambda$$

and computes

$$y_{i,j} = f(x_{i,j}) \qquad \forall i \in \{1, \ldots, n\}, \forall j \in \{0,1\}.$$

The keys pk and sk are defined as

$$pk := (f, y_{1,0}, y_{1,1}, \ldots, y_{n,0}, y_{n,1}) \quad \text{and} \quad sk := (x_{1,0}, x_{1,1}, \ldots, x_{n,0}, x_{n,1}).$$

$\mathsf{Sign}(sk, m)$. Given sk and a message $m \in \{0,1\}^n$, the message is signed by revealing certain preimages of the $y_{i,j}$ elements contained in pk. The individual bits of the message determine which preimages are revealed.

We denote the i-th bit of $m \in \{0,1\}^n$ as $m_i \in \{0,1\}$, so $m = (m_1, \ldots, m_n) \in \{0,1\}^n$. The signature σ for the message m consists of

$$\sigma = (x_{1,m_1}, \ldots, x_{n,m_n}).$$

$\mathsf{Vfy}(pk, m, \sigma)$. To verify a signature, it is checked whether the values $x_{i,j}$ contained in σ are indeed preimages of the elements $y_{i,j}$ contained in pk.

Given the public key pk, a message $m = (m_1, \dots, m_n) \in \{0,1\}^n$ and a signature $\sigma = (x'_{1,m_1}, \dots, x'_{n,m_n})$, it is checked whether

$$f(x'_{i,m_i}) = y_{i,m_i} \qquad \forall i \in \{1, \dots, n\}.$$

If this is true for all $i \in \{1, \dots, n\}$, the signature is accepted, i.e. 1 is output. Otherwise 0 is output.

The *correctness* of this scheme is obvious. In the following we will prove the *soundness* for a suitable security definition, assuming that f is a one-way function.

Theorem 23. *Let Σ be Lamport's one-time signature scheme, instantiated with one-way function f and message space $\{0,1\}^n$. For each PPT attacker $\mathcal{A}$ that breaks the $\mathsf{EUF\text{-}1\text{-}naCMA}$ security of Σ in time $t_\mathcal{A}$ with success probability $\epsilon_\mathcal{A}$, there exists a PPT attacker $\mathcal{B}$ that breaks the one-way function f in time $t_\mathcal{B}$ with success probability $\epsilon_\mathcal{B}$, such that*

$$t_\mathcal{A} \approx t_\mathcal{B} \qquad \text{and} \qquad \epsilon_\mathcal{B} \geq \frac{\epsilon_\mathcal{A}}{n}.$$

If $\mathcal{A}$ is a polynomial time attacker with a non-negligible success probability, $t_\mathcal{A}$ is bounded by $t_\mathcal{A} \leq \mathsf{poly}(k)$, and $\epsilon_\mathcal{A}$ is bounded by $\epsilon_\mathcal{A} \geq 1/\mathsf{poly}(k)$ for a polynomial poly in the security parameter. Then there is also an attacker $\mathcal{B}$ that breaks the one-way function f in polynomial time and with non-negligible probability. If one assumes that such an attacker $\mathcal{B}$ cannot exist, then $\mathcal{A}$ cannot exist either.

The idea of proof is quite simple. $\mathcal{B}$ uses $\mathcal{A}$ as "subroutine" by simulating the challenger for $\mathcal{A}$. The message m^* for which the adversary $\mathcal{A}$ forges the signature must differ from the message m for which $\mathcal{A}$ requests a signature in at least one bit. Adversary $\mathcal{B}$ guesses this location, and embeds there the image $y = f(x)$ of the one-way function, for which it is supposed to compute a preimage x' with $f(x') = y$. He generates the rest of pk in a way that it can create a valid signature for m.

Proof. We construct $\mathcal{B}$ as follows. $\mathcal{B}$ plays the security experiment for one-way functions, and receives as input a description of the function f and a value $y = f(x)$, where $x \xleftarrow{\$} \{0,1\}^\lambda$ is chosen uniformly at random.

$\mathcal{B}$ uses adversary $\mathcal{A}$ by simulating the $\mathsf{EUF\text{-}1\text{-}naCMA}$ challenger for $\mathcal{A}$. Therefore, $\mathcal{B}$ first receives a message $m = (m_1, \dots, m_n) \in \{0,1\}^\lambda$ from $\mathcal{A}$, for which $\mathcal{A}$ wants to see a signature.

Now $\mathcal{B}$ constructs a public key pk as follows:

1. $\mathcal{B}$ chooses a random index $\nu \xleftarrow{\$} \{1, \dots, n\}$ and defines $y_{\nu,1-m_\nu} := y$.

2. To simulate a complete pk, $\mathcal{B}$ now selects $2n - 1$ random values $x_{i,j}$ and calculates $y_{i,j} := f(x_{i,j})$ for all other $(i,j) \in \{1, \dots, n\} \times \{0,1\}$ with $(i,j) \neq (\nu, 1 - m_\nu)$. The pk generated by $\mathcal{B}$ is
$$pk = (y_{1,0}, y_{1,1}, \dots, y_{n,0}, y_{n,1}).$$

Notice that now $\mathcal{B}$ knows almost all preimages of the values $y_{i,j}$, with *one* exception: It does not know the preimage of $y_{\nu,1-m_\nu}$, because here it has embedded the challenge y. Thus $\mathcal{B}$ can now sign *any* message $m' = (m'_1, \dots, m'_n)$, *as long as* $m'_\nu = m_\nu$, so that the ν-th bit of the message m' is equal to the ν-th bit of the message m.

$\mathcal{B}$ can in particular simulate a valid signature σ for m, because it knows all secret key elements that are necessary to determine

$$\sigma = (x_{1,m_1}, \dots, x_{n,m_n})$$

$\mathcal{A}$ now receives (pk, σ) as input. The pk is perfectly indistinguishable from a public key in the real EUF-1-naCMA experiment and σ is a valid signature for m with respect to pk. Therefore, with probably $\epsilon_{\mathcal{A}}$, $\mathcal{A}$ will output a message m^* together with a forged signature σ^*, so that $m^* \neq m$ and $\mathsf{Vfy}(pk, m^*, \sigma^*) = 1$.

$\mathcal{B}$ now hopes that $\mathcal{A}$ forges a valid signature σ^* for a message $m^* = (m_1^*, \dots, m_n^*) \in \{0,1\}^n$, so that m^* differs from m at the ν-th position. Since m^* differs from m in at least one position and $\mathcal{A}$ receives absolutely no information about ν, this happens with probability $1/n$. In this case $\mathcal{B}$ learns a preimage of y, which it can use to win the security experiment for the one-way function f.

Thus $\mathcal{B}$ is successful if (i) $\mathcal{A}$ is successful and (ii) m^* differs from m at the ν-th place, i.e. $m_\nu^* \neq m_\nu$. Here (i) occurs with probability $\epsilon_{\mathcal{A}}$ by assumption, and (ii) with probability at least $1/n$. Since the attacker does not receive any information about ν both events are independent of each other, such that we get

$$\epsilon_{\mathcal{B}} \geq \epsilon_{\mathcal{A}} \cdot 1/n.$$

We also have $t_{\mathcal{A}} \approx t_{\mathcal{B}}$, because the runtime of $\mathcal{B}$ is about the same as the runtime of $\mathcal{A}$, plus a minor overhead. □

Using almost the same proof technique, it is easily possible to show directly that Lamport signatures are also secure in the sense of one-time *adaptive* chosen-message attacks (EUF-1-CMA). This is a simple extension of the proof presented here, which is a good exercise.

Remark 24. If you take a closer look at the proof of Theorem 23 and think a little bit about it, you can see that it consists essentially of two parts:

1. *Simulation:* First the algorithm $\mathcal{B}$, which would like to use the signature attacker $\mathcal{A}$ to solve a “difficult” problem (namely the inversion of a one-way function in the case of Theorem 23), has to make sure that $\mathcal{A}$ outputs a forged signature with the desired probability of success $\epsilon_{\mathcal{A}}$. Therefore, the “view” of $\mathcal{A}$ must be indistinguishable from the real security experiment — otherwise $\mathcal{A}$ might not work anymore, if its view differs from the real security experiment. This is because the only assumption that we make about $\mathcal{A}$ is that it is a valid adversary in the considered security experiment, but we do not know anything about the success probability of $\mathcal{A}$ if we provide $\mathcal{A}$ with a different view.

   For this reason, it was important that $\mathcal{B}$ created the pk in such a way that it is distributed *exactly as in the real security experiment*. Also the signature σ that $\mathcal{A}$ received from $\mathcal{B}$ was *perfectly indistinguishable* for $\mathcal{A}$ from a signature in the real experiment.

   If we manage to “convince” the adversary $\mathcal{A}$ that he actually takes part in the real security experiment, then that is already the first half of the battle in the security proof, because then he delivers us a forgery (m^*, σ^*) with probability of success $\epsilon_{\mathcal{A}}$.

2. *Extraction:* The second half is now that we still have to extract the solution for the difficult problem from the forged signature (m^*, σ^*).

   For example, in the case of Lamport signatures, we could hope that $\mathcal{A}$ would most likely give us the wanted preimage y as part of the signature.

These two tasks are the core of all security proofs for digital signatures. New signature schemes are usually constructed in such a way that one can later simulate the pk and, if necessary, valid signatures in the security proof on the one hand, and on the other hand extract the solution of a difficult problem from the forgery.

Exercise 25. Prove that Lamport's one-time signatures are EUF-1-CMA secure, assuming that f is a one-way function.

Remark 26. Winternitz signatures [Mer88] are an extension of Lamport's signatures. Here the message is not represented as a binary string, but generally in w-adic representation with $w \geq 2$. This allows a trade-off that achieves shorter signatures, (todo: Maybe at the cost would be better here - at the expense of more expensive calculations) and an additional checksum. See [Mer88] for details. TODO

2.3 Efficient One-Time Signatures from Concrete Complexity Assumptions

In this chapter, we will present two one-time signature schemes based on concrete complexity assumptions. These constructions are much more efficient than Lamport's generic scheme. We will use them later to construct efficient concrete signature schemes.

We assume basic knowledge about the discrete logarithm problem and the RSA problem, a comprehensive introduction can be found in [KL07].

2.3.1 One-Time Signatures based on the Discrete Logarithm Problem

Definition 27. Let $\mathbb{G}$ be a finite group with generator g and order $p \in \mathbb{N}$. The *discrete logarithm problem* in $\mathbb{G}$ is: Given a random group element $y \in \mathbb{G}$, find an integer $x \in \mathbb{Z}_p$ so that

$$g^x = y.$$

A common assumption in cryptography is that there exist groups in which the discrete logarithmic problem is "hard". Candidates for such groups are for example subgroups of the multiplicative group $\mathbb{Z}_q^*$ for a natural number $q \in \mathbb{N}$ (usually q is a prime number) or certain elliptic curve groups.

Based on the difficulty of the discrete logarithmic problem, a very simple one-time signature scheme can be constructed, which is closely related to the commitment scheme of Pedersen [Ped92]. Let Σ be the following signature scheme with message space $\mathbb{Z}_p$.

$\mathsf{Gen}(1^\lambda)$**.** The key generation algorithm selects $x, \omega \overset{\$}{\leftarrow} \mathbb{Z}_p$ and computes $h := g^x$ and $c := g^\omega$. The public key is $pk := (g, h, c)$, the secret key is $sk := (x, \omega)$.

$\mathsf{Sign}(sk, m)$**.** To sign the message $m \in \mathbb{Z}_p$, $\sigma \in \mathbb{Z}_p$ is computed as

$$\sigma := \frac{\omega - m}{x} \bmod p.$$

$\mathsf{Vfy}(pk, m, \sigma)$**.** For verification purposes it is checked whether the equation

$$c \overset{?}{=} g^m h^\sigma$$

is fulfilled.

The correctness of the procedure can be shown by simple insertion:

$$g^m h^\sigma = g^{m+x\sigma} = g^{m+x((\omega-m)/x)} = g^\omega = c.$$

Theorem 28. *For each PPT adversary $\mathcal{A}$ that breaks the* EUF-1-naCMA *security of Σ in time $t_\mathcal{A}$ with success probability $\epsilon_\mathcal{A}$, there exists a PPT adversary $\mathcal{B}$ that solves the discrete logarithm problem in $\mathbb{G}$ in time $t_\mathcal{B} \approx t_\mathcal{A}$ with success probability $\epsilon_\mathcal{B} \geq \epsilon_\mathcal{A}$.*

Proof. Adversary $\mathcal{B}$ receives as input the generator g and a random group element h, whose discrete logarithm to base g it aims to compute. It simulates the challenger for $\mathcal{A}$ as follows.

In the EUF-1-naCMA experiment $\mathcal{A}$ first outputs a message $m \in \mathbb{Z}_p$ for which it wants to see a signature. $\mathcal{B}$ selects a random value $\sigma \overset{\$}{\leftarrow} \mathbb{Z}_p$ and computes

$$c := g^m h^\sigma.$$

Then it sends the public key $pk = (g, h, c)$ and the signature σ to $\mathcal{A}$.

The pk consists of the generator g and two equally distributed random group elements (h, c) and is thus a correctly distributed public key. The value σ is a valid signature for m with respect to pk. Therefore $\mathcal{A}$ will most likely output $\epsilon_\mathcal{A}$ a fake (m^*, σ^*) with

$$c = g^{m^*} h^{\sigma^*}.$$

$\mathcal{B}$ can use this forgery to calculate the discrete logarithm of h to the base g, because

$$g^{m^*} h^{\sigma^*} = g^m h^\sigma \iff g^{m^*+x\sigma^*} = g^{m+x\sigma} \iff m^* + x\sigma^* \equiv m + x\sigma \bmod p.$$

Since $m \not\equiv m^* \bmod p$, it must also hold that $\sigma \not\equiv \sigma^* \bmod p$. Therefore x can be computed as

$$x := \frac{m - m^*}{\sigma^* - \sigma} \bmod p.$$

The running time of $\mathcal{B}$ is approximately equal to the running time of $\mathcal{A}$, plus a minimal overhead, and the success probability of $\mathcal{B}$ is at least as high as the success probability of $\mathcal{A}$. □

2.3.2 One-Time Signatures based on the RSA Assumption

Recap and notation. We denote with $\mathbb{Z}_N$ the set

$$\mathbb{Z}_N := \{0, \ldots, N-1\} \subset \mathbb{Z}.$$

$\mathbb{Z}_N$ with addition modulo N is an algebraic *group*, and an algebraic *ring* with respect to addition and multiplication modulo N. We write $\mathbb{Z}_N^*$ to denote the set of units modulo N, i.e. $\mathbb{Z}_N^* := \{a \in \mathbb{Z}_N : \gcd(a, N) = 1\}$. $\mathbb{Z}_N^*$ is a group with respect to the multiplication modulo N.

The group $\mathbb{Z}_N^*$ has order $\phi(N)$, where $\phi(N)$ is Euler's totient function. If $N = PQ$ is the product of two different prime numbers, then $\phi(N) = (P-1) \cdot (Q-1)$. For a natural number e with $\gcd(e, \phi(N)) = 1$ and for $d \equiv e^{-1} \bmod \phi(N)$ we have

$$(x^e)^d \equiv x^{ed \bmod \phi(N)} \equiv x^1 \equiv x \bmod N$$

for all $x \in \mathbb{Z}_N$.

Definition 29. Let $N := PQ$ be the product of two different prime numbers. Let $e \in \mathbb{N}$ such that $e > 1$ and $\gcd(e, \phi(N)) = 1$, and let $y \overset{\$}{\leftarrow} \mathbb{Z}_N$ be a random number modulo N. The *RSA problem* given by (N, e, y) is: Compute $x \in \mathbb{Z}_N$ so that

$$x^e \equiv y \bmod N.$$

It is well-known that the RSA problem is at most as hard as the problem of factoring N. Whether the opposite holds as well, i.e. whether the RSA problem is equivalent to the factoring problem, is unknown and one of the most fundamental open problems in cryptography. We currently do not know any algorithm that solves the RSA problem without essentially computing the secret RSA key d with $d \equiv 1/e \bmod \phi(N)$. Calculating the secret key d is known to be as difficult as factoring N [RSA78, May04]. It is unknown whether there is an easier way to computes e-th roots modulo N, but a common assumption in cryptography is that this is not the case.

RSA-based one-time signatures. The following signature scheme Σ is based on a construction from [HW09]. The scheme has message space $[0, 2^n - 1]$.

$\mathsf{Gen}(1^\lambda)$**.** The key generation algorithm generates an RSA modulus $N = PQ$ and a prime number $e > 2^n$ with $\gcd(e, \phi(N)) = 1$ and calculates $d := e^{-1} \bmod \phi(N)$. In addition, two numbers $J, c \overset{\$}{\leftarrow} \mathbb{Z}_N$ are randomly selected. The public key is $pk := (N, e, J, c)$, the secret key is $sk := d$.

$\mathsf{Sign}(sk, m)$**.** A signature for a message $m \in [0, 2^n - 1]$ is computed as

$$\sigma \equiv (c/J^m)^d \bmod N.$$

$\mathsf{Vfy}(pk, m, \sigma)$**.** The verification algorithm will output 1 if

$$c \equiv J^m \sigma^e \bmod N$$

applies, and otherwise 0.

It is very easy to prove *correctness* of this scheme:

$$c \equiv J^m \sigma^e \equiv J^m \cdot ((c/J^m)^d)^e \equiv J^m \cdot c/J^m \equiv c \bmod N.$$

Theorem 30. *Let (N, e, y) be an instance of the RSA problem so that $e > 2^n$ is a prime number. For each PPT attacker $\mathcal{A}$ that breaks the* $\mathsf{EUF\text{-}1\text{-}naCMA}$ *security of Σ in time $t_\mathcal{A}$ with success probability $\epsilon_\mathcal{A}$, there exists a PPT attacker $\mathcal{B}$ that computes $x \in \mathbb{Z}_N$ with $x^e \equiv y \bmod N$. The runtime of $\mathcal{B}$ is $t_\mathcal{B} \approx t_\mathcal{A}$, the probability of success is at least $\epsilon_\mathcal{B} \geq \epsilon_\mathcal{A}$.*

As a technical tool for the proof we will use the following lemma from [Sha83].

Lemma 31 (Shamir's trick)**.** *Let $J, S \in \mathbb{Z}_N$ and $e, f \in \mathbb{Z}$ so that $\gcd(e, f) = 1$ and*

$$J^f \equiv S^e \bmod N.$$

There is an efficient algorithm that, given $N \in \mathbb{Z}$ and $(J, S, e, f) \in \mathbb{Z}_N^2 \times \mathbb{Z}^2$, computes $x \in \mathbb{Z}_N$, so that $x^e \equiv J \bmod N$.

Proof. The algorithm first computes two integers $(\alpha, \beta) \in \mathbb{Z}^2$ so that

$$\alpha f + \beta e = 1.$$

using the extended Euclidean algorithm. These numbers exist because $\gcd(e, f) = 1$. Then it computes

$$x \equiv S^\alpha \cdot J^\beta \bmod N.$$

It remains to show that $x \in \mathbb{Z}_N$ actually satisfies the equation $x^e \equiv J \bmod N$:

$$J^f \equiv S^e \iff J^{f\alpha} J^{e\beta} \equiv S^{e\alpha} J^{e\beta} \iff J^{f\alpha + e\beta} \equiv (S^\alpha J^\beta)^e \iff J \equiv x^e$$

where all congruences are modulo N. □

Now we are ready to prove Theorem 30.

Proof of Theorem 30. Algorithm $\mathcal{B}$ receives as input an instance (N, e, y) of the RSA problem. It starts the adversary and receives a message $m \in [0, 2^n - 1]$. The pk is computed by $\mathcal{B}$ as (N, e, J, c) with

$$J := y \qquad \text{and} \qquad c := J^m \sigma^e \bmod N$$

for a random number $\sigma \xleftarrow{\$} \mathbb{Z}_N$. Because σ and x are equally distributed randomly over $\mathbb{Z}_N$, c and J are also distributed randomly. Thus pk cannot be distinguished from a real public key. Furthermore, σ fulfills the verification equation and is therefore a valid signature. Therefore, with probability $\epsilon_\mathcal{A}$ adevrsary $\mathcal{A}$ will output a forgery (m^*, σ^*) with $m^* \neq m$ and

$$c \equiv J^{m^*} (\sigma^*)^e \bmod N.$$

We obtain the equation

$$J^m \sigma^e \equiv c \equiv J^{m^*} (\sigma^*)^e \bmod N$$

which can be transformed into the two equations

$$J^{m - m^*} \equiv (\sigma^*/\sigma)^e \qquad \text{and} \qquad J^{m^* - m} \equiv (\sigma/\sigma^*)^e,$$

if $\sigma \in \mathbb{Z}_N^*$ and $\sigma^* \in \mathbb{Z}_N^*$.

Since we have $m \neq m^*$, we either have $m - m^* \in [1, 2^n - 1]$ (namely when $m > m^*$), or $m^* - m \in [1, 2^n - 1]$ (namely when $m^* > m$).

The case $m - m^* \in [1, 2^n - 1]$. If we now define $f := m - m^* \in [1, 2^n - 1]$ and $S := \sigma^*/\sigma \in \mathbb{Z}_N$,[3] then we obtain the equation

$$J^f \equiv S^e \bmod N.$$

Here we have $\gcd(e, f) = \gcd(e, m - m^*) = 1$, because e is a prime number and greater than 2^n. $\mathcal{B}$ can therefore apply Shamir's trick (**??**) to (N, J, S, e, f) to calculate x so that $x^e \equiv J \bmod N$. This is exactly the e-th root of y we are looking for, because $J = y$.

[3] We are a bit imprecise here, because we implicitly assume that σ is invertible modulo N, which is not necessarily the case. However, note that $\sigma = 0$ implies $c = 0$, which happens only with negligibly small probability. If $\sigma \not\equiv 0 \bmod N$ and is nevertheless non-invertible, then we can factorize N by calculating $\gcd(N, \sigma)$ and thus solve the given RSA problem instance.

The case $m^* - m \in [1, 2^n - 1]$**.** The argument in this case is identical, except that we set $f := m^* - m \in [1, 2^n - 1]$ and $S := \sigma/\sigma^* \in \mathbb{Z}_N$. □

2.4 From EUF-naCMA-Security to EUF-CMA-Security

An important application of one-time signatures is to reinforce the security of signature schemes. In this chapter we will describe a transformation that uses a EUF-1-naCMA secure one-time signature scheme to transform a EUF-naCMA secure scheme $\Sigma' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Vfy}')$ into an EUF-CMA secure scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$. The idea of the transformation presented here was described first by Even, Goldwasser and Micali [EGM96].

This transformation is extremely useful. Usually, our actual goal when constructing signatures is EUF-CMA security. However, this goal is not easily achieved. The transformation simplifies our task, since it yields that it is sufficient to construct an EUF-1-naCMA-secure one-time signature and an EUF-naCMA-secure signature scheme. Both are much easier to achieve than direct EUF-CMA security, because

- EUF-1-naCMA-secure one-time signatures are easy to design. We have already seen some simple and efficient constructions, based on generic assumptions (the existence of one-way functions) and from rather weak, concrete complexity assumptions.

- Proving a signature scheme EUF-naCMA-secure is much easier than proving EUF-CMA-security. The advantage is that in the EUF-naCMA security experiment the adversary has to tell the challenger for which messages he wants to see a signature *before* seeing the public key. Therefore a reduction simulating this experiment in a security proof can set up the public key in a way that signatures can be created for all messages selected by the adversary. Recall that this is already half the battle.

  Therefore EUF-naCMA-secure signatures are easier to construct than EUF-CMA-secure ones. However, it is often still difficult enough, because the other half, namely the extraction of a solution to a computationally hard problem from the forged signature, must be achieved, too.

To obtain a EUF-CMA-secure procedure in the end, we thus can first construct a EUF-naCMA-secure scheme, and then simply apply the transformation to obtain an EUF-CMA-secure scheme.

The Transformation. In the following let $\Sigma_1 = (\mathsf{Gen}^{(1)}, \mathsf{Sign}^{(1)}, \mathsf{Vfy}^{(1)})$ be a (one-time) signature scheme and $\Sigma' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Vfy}')$ a signature scheme. We describe a new scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ which uses $\Sigma^{(1)}$ and Σ' as building blocks.

$\mathsf{Gen}(1^k)$**.** For key generation a key pair $(pk', sk') \xleftarrow{\$} \mathsf{Gen}'(1^k)$ is generated with the key generation algorithm of Σ'. We define $pk := pk'$ and $sk := sk'$.

$\mathsf{Sign}(sk, m)$**.** A signature for message m is computed in two steps.

1. First a temporary key pair $(pk^{(1)}, sk^{(1)}) \xleftarrow{\$} \mathsf{Gen}^{(1)}(1^k)$ is generated with the key generation algorithm of $\Sigma^{(1)}$. The message m is signed with the temporary secret key $sk^{(1)}$:
$$\sigma^{(1)} := \mathsf{Sign}^{(1)}(sk^{(1)}, m).$$

2. Then $pk^{(1)}$ is signed with the long-lived secret key sk:

$$\sigma' := \mathsf{Sign}'(sk, pk^{(1)}).$$

The signature σ for message m is $\sigma := (pk^{(1)}, \sigma^{(1)}, \sigma')$.

$\mathsf{Vfy}(pk, m, \sigma)$**.** The verification algorithm checks whether both signatures contained in $\sigma = (pk^{(1)}, \sigma^{(1)}, \sigma')$ are valid. The signature is accepted and 1 is returned, if

$$\mathsf{Vfy}^{(1)}(pk^{(1)}, m, \sigma^{(1)}) = 1 \qquad \text{and} \qquad \mathsf{Vfy}'(pk, pk^{(1)}, \sigma') = 1.$$

Otherwise, 0 is returned.

So the idea of the transformation is first to sign a message with a freshly generated key of Σ_1. The freshly generated key is then "certified" with the secret key sk.

Theorem 32. *For any PPT adversary $\mathcal{A}$ breaking the* EUF-CMA *security of Σ in time $t_{\mathcal{A}}$ with success probability $\epsilon_{\mathcal{A}}$ that makes at most q signing queries there exists a PPT adversary $\mathcal{B}$ running in time $t_{\mathcal{B}} \approx t_{\mathcal{A}}$ that breaks*

- *either the* EUF-1-naCMA *security of $\Sigma^{(1)}$, with success probability at least*

$$\epsilon^{(1)} \geq \frac{\epsilon_{\mathcal{A}}}{2q},$$

- *or the* EUF-naCMA *security of Σ', with success probability at least*

$$\epsilon' \geq \frac{\epsilon_{\mathcal{A}}}{2}.$$

Hence, if we assume that both Σ' and $\Sigma^{(1)}$ are secure, then ϵ' and $\epsilon^{(1)}$ are negligibly small for all polynomial time adversaries $\mathcal{B}$. Hence, $\epsilon_{\mathcal{A}}$ must also be negligibly small, so that Σ is secure.

Proof. An EUF-CMA adversary $\mathcal{A}$ makes a series of adaptive signature queries $m_1, \ldots, m_q$, $q \geq 0$, for which it expects signatures $\sigma_1, \ldots, \sigma_q$. For the scheme considered here, every signature σ_i consists of three components $\sigma_i = (pk_i^{(1)}, \sigma_i^{(1)}, \sigma_i')$. We consider two different events:

- Event E_0 occurs if the adversary produces a valid forgery $(m^*, \sigma^*) = (m^*, (pk^{(1)*}, \sigma^{(1)*}, \sigma'^*))$ so that

$$pk^{(1)*} = pk_i^{(1)}.$$

  for at least one $i \in \{1, \ldots, q\}$.

  Thus, event E_0 occurs if the adversary reuses a temporary public key $pk_i^{(1)}$ from one of the signatures $\sigma_i \in \{\sigma_1, \ldots, \sigma_q\}$.

- Event E_1 occurs if the adversary outputs a valid forgery $(m^*, \sigma^*) = (m^*, (pk^{(1)*}, \sigma^{(1)*}, \sigma'^*))$ and it holds that

$$pk^{(1)*} \neq pk_i^{(1)}.$$

  for all $i \in \{1, \ldots, q\}$.

  Thus, event E_1 occurs if the adversary forges a signature containing a *new* temporary public key $pk^{(1)*}$.

Note that any successful adversary will cause either event E_0 or event E_1, so that we have

$$\epsilon_{\mathcal{A}} \leq \Pr[E_0] + \Pr[E_1].$$

This inequality in turn implies that at least one of the two inequalities

$$\Pr[E_0] \geq \frac{\epsilon_{\mathcal{A}}}{2} \quad \text{or} \quad \Pr[E_1] \geq \frac{\epsilon_{\mathcal{A}}}{2} \tag{2.1}$$

must be satisfied.

Breaking the $\mathsf{EUF\text{-}1\text{-}naCMA}$ security of $\Sigma^{(1)}$. Attacker $\mathcal{B}$ attempts to break the $\mathsf{EUF\text{-}1\text{-}naCMA}$ security of $\Sigma^{(1)}$ by simulating the $\mathsf{EUF\text{-}CMA}$ Challenger for $\mathcal{A}$. It proceeds as follows.

$\mathcal{B}$ generates a key pair $(pk, sk) \xleftarrow{\$} \mathsf{Gen}'(1^k)$ and guesses an index $\nu \xleftarrow{\$} \{1, \ldots, q\}$. Attacker $\mathcal{A}$ receives pk as input. If $\mathcal{A}$ requests a signature for message m_i, then $\mathcal{B}$ signs this message as follows:

- If $i \neq \nu$, then $\mathcal{B}$ generates a key $(pk_i^{(1)}, sk_i^{(1)}) \xleftarrow{\$} \mathsf{Gen}^{(1)}(1^k)$ and uses this key to create a signature $\sigma_i = (pk_i^{(1)}, \sigma_i^{(1)}, \sigma_i')$ for m_i:

  $$\sigma_i^{(1)} := \mathsf{Sign}^{(1)}(sk_i^{(1)}, m_i) \qquad \text{and} \qquad \sigma_i' := \mathsf{Sign}'(sk, pk_i^{(1)}).$$

  This is obviously a valid signature for message m_i.

- To sign message m_ν $\mathcal{B}$ proceeds differently. In this case it outputs m_ν to his $\mathsf{EUF\text{-}1\text{-}naCMA}$ challenger $\mathcal{C}$. In response it receives a public key $pk_\nu^{(1)}$ and a valid signature $\sigma_\nu^{(1)}$ from the challenger. $\mathcal{B}$ now signs $pk_\nu^{(1)}$ by computing

  $$\sigma_\nu' := \mathsf{Sign}'(sk, pk_\nu^{(1)})$$

  and outputs $\sigma_\nu = (pk_\nu^{(1)}, \sigma_\nu^{(1)}, \sigma_\nu')$ as signature for m_ν. This signature is obviously valid as well.

Note that $\mathcal{B}$ perfectly simulates the $\mathsf{EUF\text{-}CMA}$ experiment for $\mathcal{A}$. When event E_0 occurs, then $\mathcal{A}$ outputs a message $m^* \notin \{m_1, \ldots, m_q\}$ with valid signature σ^* so that an index $i \in \{1, \ldots, q\}$ exists with

$$\sigma^* = (pk_i^{(1)}, \sigma^{(1)*}, \sigma_i'^*).$$

So the signature σ^* contains the i-th temporary key $pk_i^{(1)}$. Since the random index ν is perfectly hidden from $\mathcal{A}$, with probability $1/q$ this is exactly the public key $pk_\nu^{(1)}$ that $\mathcal{B}$ received from the $\mathsf{EUF\text{-}1\text{-}naCMA}$ challenger. Since $m^* \neq m_\nu$ is valid, $\mathcal{B}$ can then output the tuple $(m^*, \sigma^{(1)*})$ to its challenger $\mathcal{C}$ as a valid forgery. Hence, it breaks the $\mathsf{EUF\text{-}1\text{-}naCMA}$ security with a success probability of at least

$$\epsilon^{(1)} \geq \frac{\Pr[E_0]}{q} \tag{2.2}$$

The running time of $\mathcal{B}$ is essentially the same as that of $\mathcal{A}$, plus a small overhead.

Breaking the $\mathsf{EUF\text{-}naCMA}$ security of Σ'. Attacker $\mathcal{B}$ attempts to break the $\mathsf{EUF\text{-}naCMA}$ security of Σ' by again simulating the $\mathsf{EUF\text{-}CMA}$ challenger for $\mathcal{A}$. This time $\mathcal{B}$ generates q key pairs

$$(pk_i^{(1)}, sk_i^{(1)}) \xleftarrow{\$} \mathsf{Gen}^{(1)}(1^k), \qquad i \in \{1, \ldots, q\}.$$

Then $\mathcal{B}$ outputs the list $(pk_1^{(1)}, \ldots, pk_q^{(1)})$ to its $\mathsf{EUF\text{-}naCMA}$ challenger. In response $\mathcal{B}$ gets a public key pk together with signatures $(\sigma'_1, \ldots, \sigma'_q)$ so that for all $i \in \{1, \ldots, q\}$ the signature σ'_i is a valid signature for "message" $pk_i^{(1)}$.

Now $\mathcal{B}$ starts the $\mathsf{EUF\text{-}CMA}$ attacker $\mathcal{A}$ on input pk. If $\mathcal{A}$ requests a signature for message m_i, then $\mathcal{B}$ signs this message using the known temporary key $sk_i^{(1)}$ by computing

$$\sigma_i^{(1)} := \mathsf{Sign}^{(1)}(sk_i^{(1)}, m_i).$$

Then it outputs $\sigma_i = (pk_i^{(1)}, \sigma_i^{(1)}, \sigma'_i)$ as signature for m_i. This is a valid signature.

$\mathcal{B}$ perfectly simulates the $\mathsf{EUF\text{-}CMA}$-Challenger for $\mathcal{A}$ again. If event E_1 occurs, then $\mathcal{A}$ outputs a message $m^* \notin \{m_1, \ldots, m_q\}$ with valid signature $\sigma^* = (pk^{(1)*}, \sigma^{(1)*}, \sigma'^*)$ where σ^* contains a new value $pk^{(1)*}$ together with valid signature σ'^*. If this is the case, then $\mathcal{B}$ can use the tuple

$$(pk^{(1)*}, \sigma'^*)$$

to break the $\mathsf{EUF\text{-}naCMA}$ security with a probability of at least

$$\epsilon' \geq \Pr[E_1]. \tag{2.3}$$

The running time of $\mathcal{B}$ is again essentially the same as that of $\mathcal{A}$, plus a small overhead.

Wrapping up. By inserting the inequalities (2.2) and (2.3) into the inequalities of (2.1), we obtain that at least one of the two inequalities

$$\epsilon^{(1)} \geq \Pr[E_0] \geq \frac{\epsilon_\mathcal{A}}{2q} \quad \text{or} \quad \epsilon' \geq \Pr[E_1] \geq \frac{\epsilon_\mathcal{A}}{2}$$

must hold. □

Exercise 33. Construct an $\mathsf{EUF\text{-}1\text{-}CMA}$-secure one-time signature schemes based on the discrete logarithm assumption, and another one based on the RSA assumption. To this end, apply the above transformation to the schemes from Section 2.3.1 and Section 2.3.2, respectively. Give complete descriptions of the resulting $\mathsf{EUF\text{-}1\text{-}CMA}$-secure one-time signature schemes.

2.5 Tree-based Signatures

Unfortunately, one-time signatures can only be used once without becoming insecure. In practice this is of course not sufficient for most applications, since we would often like to use one public key for many signatures.

Unfortunately, constructing many-time signatures is more difficult than constructing one-time signatures. Fortunately, however, there is a generic construction that constructs a many-time signature scheme from any given one-time signature scheme without becoming too inefficient. The idea goes back to Ralph Merkle [Mer88], so such schemes are also called *Merkle-Signatures*. The construction is based on *Merkle-trees*, which have other interesting applications beyond digital signatures. For instance, Merkle-trees are very popular in the context of blockchains.

2.5.1 q-Times Signatures

From a one-time signature scheme $\Sigma^{(1)} = (\mathsf{Gen}^{(1)}, \mathsf{Sign}^{(1)}, \mathsf{Vfy}^{(1)})$ it is easy to construct a (quite inefficient) q-times signature scheme by simply using q schemes in parallel, as we describe in this chapter. However, we remark that we consider this scheme only to be able to better explain the idea of Merkle signatures later. It is not a scheme that needs to be remembered very well.

The signature scheme that we will consider in the following is *stateful*. This means that the signer has to save a state st, which is updated every time a signature is generated. Of course, such a state is not always very practical, especially if one wants to perform signature generation on several devices that do not share a common state (for example, on several servers in a cloud). This is why stateless schemes are preferred in practice.

$\mathsf{Gen}(1^k)$**.** The public key of the q-times scheme consists of q public keys of the one-time signature scheme. A counter $st := 1$ is initialized. The secret key consists of the corresponding q secret keys and the counter st:[4]

$$pk = (pk_1, \ldots, pk_q), \quad sk = (sk_1, \ldots, sk_q, st), \quad (pk_i, sk_i) \overset{\$}{\leftarrow} \mathsf{Gen}^{(1)}(1^k) \, \forall \, i \in \{1, \ldots, q\}.$$

$\mathsf{Sign}(sk, m)$**.** If st q, then the signing algorithm fails. Otherwise we have $st = i$ for some $1 \leq i \leq q$. To create the i-th signature for a given message m_i, the signer calculates

$$\sigma_i := \mathsf{Sign}^{(1)}(sk_i, m_i).$$

The signature $\sigma = (\sigma_i, i)$ consists of the signature σ_i and the number $i \in \{1, \ldots, q\}$, which tells the verifier to use the key pk_i for verification. Finally the state is incremented, $st := st + 1$.

$\mathsf{Vfy}(pk, m, \sigma)$**.** When the verifier receives a signature $\sigma = (\sigma_i, i)$, it checks if

$$\mathsf{Vfy}^{(1)}(pk_i, m_i, \sigma_i) \overset{?}{=} 1$$

If this holds, then it accepts the signature and outputs 1. Otherwise it outputs 0.

So the state of the above scheme consists of a counter that counts how many signatures have been created so far. The purpose of the counter is that each secret key is only used once. We will later explain a technique to avoid the state.

Exercise 34. Show that the above scheme is a EUF-naCMA secure q-times signature scheme, if $\Sigma^{(1)}$ is EUF-1-naCMA-secure. Show that it is EUF-CMA-secure, if $\Sigma^{(1)}$ is EUF-1-CMA-secure.

The above scheme has some major disadvantages. In particular, the size of the public and secret keys is *linear* in the number of signatures ever issued. The advantage, however, is that the size of a signature σ is quite small, as it is constant.

$$|pk| = O(q), \qquad |sk| = O(q), \qquad |\sigma| = O(1).$$

[4] Actually, the state st is not a secret, we just have to store it somewhere at the signer. Since the sk has to be stored anyway, we also put the state there.

2.5.2 Trading Short Signatures for Small Public Keys

Using a collision resistant hash function $H : \{0,1\}^* \to \{0,1\}^n$ the size of the public key pk can be reduced easily. The idea is to simply publish a hash of pk instead of the full list $pk = (pk_1, \dots, pk_q)$ of all q public keys. So

$$pk := (H, y),$$

where $y = H(pk_1, \dots, pk_q)$.

However, in order for a verifier to check the validity of a given signature, the list of public keys pk must be sent along with the signature. So a signature in this scheme consists of $\sigma = (\sigma_i, i, (pk_1, \dots, pk_q))$. The verifier now checks whether

$$\mathsf{Vfy}^{(1)}(pk_i, \sigma_i, m_i) \overset{?}{=} 1 \qquad \text{and} \qquad y \overset{?}{=} H(pk_1, \dots, pk_q).$$

If both hold, then it outputs 1. Otherwise 0.

Exercise 35. It can be proven that this scheme is an EUF-naCMA-secure (EUF-CMA-secure) q-times signature scheme, if $\Sigma^{(1)}$ is EUF-1-naCMA-secure (EUF-1-CMA-secure) and the hash function is collision resistant. However, we leave this out.

This "hashed" variant of the original q-times signature scheme now has a public key of constant size, but larger signatures.

$$|pk| = O(1), \qquad |sk| = O(q), \qquad |\sigma| = O(q).$$

So obviously it is possible to trade a short signature size for short public keys. *Could this be done a bit more clever than by sending the entire list* $(pk_1, \dots, pk_q)$ *along with* (σ_i, i)*?*

2.5.3 Clever Compression of Public Keys: Merkle Trees

The idea of Merkle trees [Mer88] is essentially to "compress" the public keys in a smarter way than we did in the previous section. The ingenious idea is that the hash function is not only used once, but recursively several times in a clever way. In the following we assume that $q = 2^t$ for a natural number t.

Let's consider the problem of cleverly compressing a list $(pk_1, \dots, pk_{2^t})$ of public keys in such a way that we end up with a small public key pk, but at the same time the size of the signatures doesn't increase too much. One way to compress the list is by the following computation:

1. First the hash value $h_{t,i} := H(pk_i)$, $i \in \{1, \dots, 2^t\}$, is computed for all keys in the list.

2. Then one proceeds recursively:

$$h_{j-1,i} := H(h_{j,2i-1} || h_{j,2i}) \quad \forall j \in \{t, \dots, 1\} i \in \{1, \dots, 2^{j-1}\}$$

This computation is illustrated with $t = 3$ in Figure 2.3. Note here that we build a *binary tree* whose nodes are the hash values $h_{j,i}$.

In the sequel it will be useful to define some terms about trees.

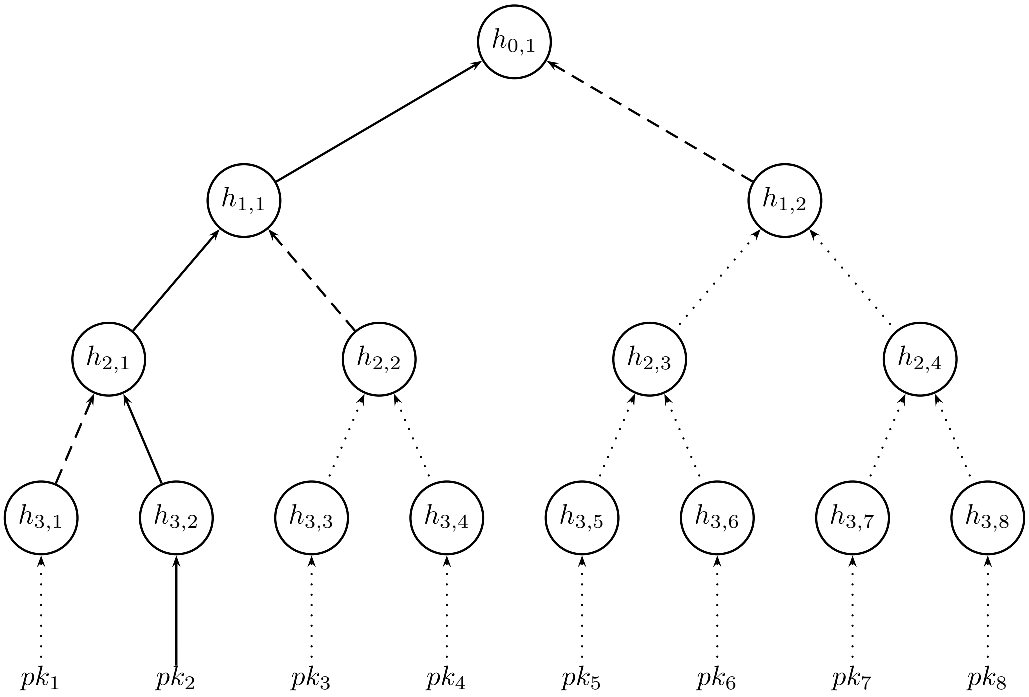

Figure 2.3: Merkle-tree with depth 3.

- We say that $h_{0,1}$ is the *root*.
- The *path* from a node $h_{j,i}$ to the root consists of all nodes that are on the shortest connection between $h_{j,i}$ and the root. For instance, in Figure 2.3 the path from $h_{3,2}$ to the root is represented by a solid line.
- We say that a node $h_{j,i}$ is the *ancestor* of $h_{j',i'}$ if $j + 1 = j'$ and $i' \in \{2i - 1, 2i\}$.
- We say that two nodes are *siblings* if they have a common ancestor. For example, $h_{1,1}$ and $h_{1,2}$ are siblings because they have a common ancestor $h_{0,1}$.
- The *co-path* from $h_{j,i}$ to the root consists of all siblings of a node that lies on the path from $h_{j,i}$ to the root. In Figure 2.3 the co-path from $h_{3,2}$ to the root is represented by a dashed (not dotted) line.

The public key of the tree-based scheme now consists of the root $pk := h_{0,1}$ of the binary tree spanned by the $h_{j,i}$ hash values. Note that the root $h_{0,1}$ is essentially a compressed representation of the list $(pk_1, \ldots, pk_q)$, just like in the previous section, but now the hash value is not simply calculated as $H(pk_1, \ldots, pk_q)$, but in a somewhat more complicated way. This will turn out to be quite clever.

A signature for the i-th message now consists of the i-th key pk_i, along with all the values needed to verify that $h_{t,i} = H(pk_i)$ is actually a child of $h_{0,1}$ by recomputing the direct path from $h_{t,i}$ up to the root $h_{0,1}$ of the tree.

Observe that to recompute this path we need the starting value $h_{t,i}$, which can be computed from pk_i as $h_{t,i} = pk_i$. The other required values are exactly the values that lie on the *co-path* from $h_{t,i}$ up to the root $h_{0,1}$. *The amazing property of Merkle trees is that the length of this path is only t, and thus only logarithmic in the number 2^t of public keys.*

Example 36. As an example, consider the value $h_{3,2}$ in Figure 2.3. Then the path from $h_{3,2}$ up to the root consists of the list of hash values $P = (h_{3,2}, h_{2,1}, h_{1,1}, h_{0,1})$. The path from $h_{3,2}$ to $h_{0,1}$ can be recomputed as follows

1. The value $h_{3,2}$ can be computed by $h_{3,2} = H(pk_2)$.
2. To recompute the value $h_{2,1}$ we need the two values $(h_{3,1}, h_{3,2})$. The value $h_{3,2}$ is already known, therefore only $h_{3,1}$ has to be sent along with the signature.
3. To recompute the value $h_{1,1}$ we need the two values $(h_{2,1}, h_{2,2})$. Again one value is known, namely $h_{2,1}$, therefore only $h_{2,2}$ must be sent in the signature.
4. To finally recompute the value $h_{0,1}$ we need the two values $(h_{1,1}, h_{1,2})$. Again one value is known, namely $h_{1,1}$, therefore only $h_{1,2}$ is sent in the signature.

So if pk_2 was used to sign a message, then a signature consists of the one-time signature $\sigma_2 = \mathsf{Sign}(sk_2, m)$, the index “2” which tells the verifier the path used, and the values

$$\sigma = (pk_2, h_{3,1}, h_{2,2}, h_{1,2})$$

Note that the list $(h_{3,1}, h_{2,2}, h_{1,2})$ is exactly the list of elements on the co-path from $h_{3,2}$ to the root.

It can be proven that this signature scheme is an EUF-naCMA-secure (or EUF-CMA-secure) q-times signature scheme, if the underlying one-time signature scheme is EUF-1-naCMA-secure (or EUF-1-naCMA-secure) and the hash function H is collision resistant.

Note that the size of signatures is bounded by the depth t of the tree. So in order to create $q = 2^t$ signatures, one has to use a tree of depth t. The size of the public key is constant, only the size of the secret key is still linear in the number q of messages to be signed. So we get:

$$|pk| = O(1), \qquad |sk| = O(q), \qquad |\sigma| = O(\log q).$$

This is much better than the previously presented signature schemes from Section 2.5.1 and Section 2.5.2. These inefficient schemes can be forgotten now – we only introduced them only to make it easier to explain the idea behind and the value of Merkle trees.

2.5.4 Short Representation of Secret Keys

One problem we still have to solve is to reduce the size of the secret keys. Is it possible to compress them in a similar way as the public keys? The answer to this question is – perhaps somewhat surprisingly – yes!

2.5.5 Pseudo-Random Functions

As a tool for this we will use a pseudorandom function (PRF). Intuitively, a pseudorandom function is a function that is indistinguishable from a truly random function. So let

$$\mathsf{PRF} : \{0.1\}^k \times \{0.1\}^n \rightarrow \{0.1\}^l$$

be a function that takes as input a *seed* $s \in \{0,1\}^k$ and a bit string $\alpha \in \{0,1\}^n$ and outputs a bit string $\mathsf{PRF}(s,\alpha) \in \{0,1\}^l$. Let

$$F : \{0,1\}^n \to \{0,1\}^l$$

be randomly chosen from all possible functions from $\{0,1\}^n$ to $\{0,1\}^l$.

We will consider an adversary having black-box access to a function, which is either the PRF with a hidden random seed s or a truly random function. This means that the adversary may only query values to the black box and get back the function evaluated on these values. Fur security, we require that such an adversary is not able to distinguish the function $\mathsf{PRF}(s,\cdot)$ from the function $F(\cdot)$. We write $\mathcal{A}^{\mathsf{PRF}(s,\cdot)}$ or $\mathcal{A}^{F(\cdot)}$ to indicate that an adversary $\mathcal{A}$ has black-box access to the function $\mathsf{PRF}(s,\cdot)$ or $F(\cdot)$, respectively.

Definition 37. We say that PRF is a *pseudo-random function* if

$$\left|\Pr\left[\mathcal{A}^{\mathsf{PRF}(s,\cdot)}(1^k) = 1\right] - \Pr\left[\mathcal{A}^{F(\cdot)}(1^k) = 1\right]\right| \leq \mathsf{negl}(k)$$

where $s \xleftarrow{\$} \{0,1\}^k$ is randomly selected and negl is a negligible function, holds for all polynomial-time adversaries $\mathcal{A}$.

2.5.6 Secret Key Compression

Our goal is to "compress" a list of secret keys $(sk_1, \dots, sk_q)$, so that we do not have to store the full list, but only some shorter information. Ideally, this would be a random string of length k, where k is the security parameter. Note that these keys are generated by

$$(pk_i, sk_i) \xleftarrow{\$} \mathsf{Gen}^{(1)}(1^k) \quad \forall i \in \{1, \dots, q\} \tag{2.4}$$

The key generation algorithm $\mathsf{Gen}^{(1)}(1^k)$ is a *probabilistic* algorithm (this is of course necessary to ensure security). However, we can think of any probabilistic algorithm equivalently as a deterministic algorithm

$$\mathsf{Gen}^{(1)}(1^k) = \mathsf{Gen}^{(1)}_{\mathsf{det}}(1^k, r)$$

which gets a random bit string r as additional input. We now compute all key pairs with this deterministic algorithm, where we generate the random string r for every key with a pseudo-random function PRF as

$$(pk_i, sk_i) \xleftarrow{\$} \mathsf{Gen}^{(1)}_{\mathsf{det}}(1^k, \mathsf{PRF}(s,i)) \quad \forall i \in \{1, \dots, q\}. \tag{2.5}$$

where $s \xleftarrow{\$} \{0,1\}^k$ is a randomly selected seed. To generate the i-th key pair, the PRF is evaluated at the position i. The observation is that, due to the security of the PRF and the fact that $s \xleftarrow{\$} \{0,1\}^k$ is randomly chosen, the keys generated are indistinguishable from

$$(pk_i, sk_i) \xleftarrow{\$} \mathsf{Gen}^{(1)}_{\mathsf{det}}(1^k, F(i)) \quad \forall i \in \{1, \dots, q\},$$

where F is a truly random function. And thus they are "just as good" as keys generated as in Equation (2.4).

This means that we no longer have to store the full list $sk = (sk_1, \dots, sk_q)$ of keys explicitly. Instead, it is sufficient to select a random seed $s \xleftarrow{\$} \{0,1\}^k$, store it in sk, and regenerate all key pairs as needed using s as shown in Equation (2.5). In combination with the Merkle tree-based signature procedure, this results in a signature scheme whose secret key also has a constant size, i.e.

$$|pk| = O(1), \qquad |sk| = O(1), \qquad |\sigma| = O(\log q).$$

Exercise 38. Prove that the signature scheme with PRF-based key derivation described in this section is $\mathsf{EUF\text{-}CMA}$ secure, assuming that the PRF is secure in the sense of Definition 37 and the underlying signature scheme is $\mathsf{EUF\text{-}1\text{-}CMA}$-secure.

The general technique for "compression" of secret keys presented in this section is applicable to many cryptographic protocols, especially tree-based signatures and similar constructions. It goes back to Oded Goldreich [Gol87]. This technique can sometimes also be used to make stateful signature schemes completely stateless. However, it is not (or only limitedly) applicable in the case of hash-function-based Merkle trees.

2.5.7 More Efficient Variants

One major disadvantage of all the methods presented here must still be pointed out. In each example *all* key pairs $(pk_1, sk_1), \dots, (pk_q, sk_q)$ have to be computed at once during key generation, in order to determine the root of the tree, which determines the public key. Especially if q is very large, this is of course extremely inefficient.

This problem does not occur if one builds the complete tree (not just the lowest level) using one-time signatures instead of a collision resistant hash function. Such schemes can even be constructed completely stateless, using the technique of Goldreich [Gol87]. We will not go into this further here, and refer to [Kat10] for details. In particular, such schemes are candidates for signature schemes secure against attacks with quantum computers, such as for instance [BDH11, BHK+19].

Exercise 39. Let $\Sigma^{(1)} = (\mathsf{Gen}^{(1)}, \mathsf{Sign}^{(1)}, \mathsf{Vfy}^{(1)})$ be a one-time signature scheme, and let $q = 2^t$. Use $\Sigma^{(1)}$ to construct a stateless, tree-based q-times signature scheme. Discuss how you would prove security of this scheme.

Chapter 3

Chameleon Hash Functions

Even if the name suggests it: chameleon hash functions are not hash functions that can change their color. But they have a very similar property, which remotely reminds of the versatility of a chameleon. They were introduced in 2000 by Krawczyk and Rabin [KR00], and are now an important building block for many cryptographic constructions — in particular, but not only, for digital signature schemes.

In this chapter we first explain the idea of chameleon hash functions and give two concrete constructions based on the discrete logarithm problem (DL) and the RSA problem. These constructions are very similar to the DL- and RSA-based one-time signatures we already know. After that we describe two interesting applications of chameleon hash functions in the context of signatures. As a first application we introduce are so-called *deniable signatures*. Afterwards we show that one can always construct a one-time signature from a chameleon hash fuction. Finally we show that one can construct signature schemes that are *strongly existential unforgeable*. This is, as the name suggests, a stronger security notion than *existential unforgeability*, which is necessary in some applications.

In addition to the examples presented here, chameleon hash functions have many other important applications in cryptography. In particular, the construction of CCA-secure encryption schemes from so-called identity-based encryption schemes can be mentioned here [Zha07] (an alternative to the one-time-signature-based construction from [CHK04]), and the relationship to certain identification schemes (Sigma protocols) [BR08].

3.1 Motivation

Let's say a customer K wants to buy a product. There are two competing dealers H_1 and H_2 who can supply this product in the same quality. Both dealers do not publish their prices, but negotiate individually with the customer each time. Each dealer prefers to be a little bit cheaper than his competitor – but unfortunately one does not the price the other dealer has just offered to the customer.

The customer first asks H_1 by e-mail for a price for the product. It replies with an e-mail offering a real bargain price of, say, € 100 per piece. In order for the message to arrive authentically at K, it is digitally signed by H_1.

Now the customer wants to contact the dealer H_2 and ask him for a lower price. If the customer simply claims that H_1 only asks for € 100 per item, then H_2 would not believe him – the price is really a bargain price (but still slightly profitable for the seller). Since H_1's message is digitally signed, K can simply forward it. The signature convinces H_2 that H_1 actually offered a price of only € 100 – and undercuts it by asking only € 99 per item.

In this case, a signature is useful on the one hand (because K can be sure that the message from H_1 has not been altered), but also harmful, because K can use the signature to convince H_2 that H_1 actually offered an incredibly low price. H_1 cannot *deny* this, because the signature is publicly verifiable.

It seems, therefore, that authenticity and deniability are two contradictory qualities that are mutually exclusive. Is this really necessary? Can we construct a signature scheme that

1. allows a recipient K to verify the authenticity of a received message m,

2. but at the same time enables a sender H_1 to *plausibly deny* to third parties (like H_2 for example) that a certain message m was sent, even if K claims this and provides a valid signature of H_1 over m?

The answer is – perhaps somewhat surprisingly – yes! Signature schemes that have such a property are called *chameleon signature schemes* [KR00]. The tool that we need to construct such schemes are chameleon hash functions.

3.2 Definition of Chameleon Hash Functions

A chameleon hash function consists of two algorithms $(\mathsf{Gen}_{\mathsf{ch}}, \mathsf{TrapColl}_{\mathsf{ch}})$.

$\mathsf{Gen}_{\mathsf{ch}}(1^k)$**.** The generation algorithm receives the security parameter as input. It outputs (ch, τ), where ch is a description of a function

$$\mathsf{ch} : \mathcal{M} \times \mathcal{R} \to \mathbb{N}$$

where $\mathcal{M}, \mathcal{N}, \mathcal{R}$ are sets that depend on the concrete construction of the chameleon hash function. The value τ is a *trapdoor* for ch.

$\mathsf{TrapColl}_{\mathsf{ch}}(\tau, m, r, m^*)$**.** Using the trapdoor τ, the *trapdoor collision algorithm* $\mathsf{TrapColl}_{\mathsf{ch}}$ can find collisions of ch. That is, the algorithm receives the trapdoor τ and $(m, r, m^*) \in \mathcal{M} \times \mathcal{R} \times \mathcal{M}$ as input. It calculates a value r^* so that

$$\mathsf{ch}(m, r) = \mathsf{ch}(m^*, r^*).$$

The only security requirement for a chameleon hash function is that it is collision resistant. An efficient algorithm that does *not* know the trapdoor should not be able to find collisions for a given function ch.

Definition 40. We say that a chameleon hash function $(\mathsf{Gen}_{\mathsf{ch}}, \mathsf{TrapColl}_{\mathsf{ch}})$ is *collision resistant* if

$$\Pr\left[\begin{array}{r}(\mathsf{ch}, \tau) \xleftarrow{\$} \mathsf{Gen}_{\mathsf{ch}}(1^k) \\ \mathcal{A}(1^k, \mathsf{ch}) = (m, r, m^*, r^*)\end{array} : \mathsf{ch}(m, r) = \mathsf{ch}(m^*, r^*) \land (m, r) \neq (m^*, r^*)\right] \leq \mathsf{negl}(k).$$

for all PPT algorithms $\mathcal{A}$.

Note that $\mathcal{A}$ does not get the trapdoor τ of ch as input – otherwise it could easily find collisions.

3.3 Examples of Chameleon Hash Functions

3.3.1 Chameleon Hash Functions based on the Discrete Logarithm Problem

We use the same definition of the discrete logarithm problem and the same notation as in Section 2.3.1. So let $\mathbb{G}$ be a finite abelian group with generator g and prime order p. In the following we describe a construction from [KR00]. We construct algorithms $(\mathsf{Gen}_{\mathsf{ch}}, \mathsf{TrapColl}_{\mathsf{ch}})$ defining a chameleon hash function

$$\mathsf{ch} : \mathbb{Z}_p \times \mathbb{Z}_p \to \mathbb{G}$$

as follows.

$\mathsf{Gen}_{\mathsf{ch}}(1^k)$**.** The generation algorithm chooses $x \xleftarrow{\$} \mathbb{Z}_p^*$ and computes $h := g^x$. The description of the chameleonq hash function consists of $\mathsf{ch} := (g, h)$. Given $(m, r) \in \mathbb{Z}_p \times \mathbb{Z}_p$ the function value is computed as

$$\mathsf{ch}(m, r) = g^m h^r.$$

The trapdoor is $\tau := x$.

$\mathsf{TrapColl}_{\mathsf{ch}}(\tau, m, r, m^*)$**.** Given (m, r, m^*), the value r^* is computed as the unique solution of the equation

$$m + xr \equiv m^* + xr^* \bmod p \iff r^* \equiv \frac{m - m^*}{x} + r \bmod p.$$

Then of course it holds that

$$g^m h^r = g^{m^*} h^{r^*}.$$

Theorem 41. *For any PPT adversary $\mathcal{A}$ that receives (g, h) as input, runs in time $t_{\mathcal{A}}$, and computes four values $(m, r, m^*, r^*) \in \mathbb{Z}_p^4$ such that $(m, r) \neq (m^*, r^*)$ and*

$$g^m h^r = g^{m^*} h^{r^*}$$

with success probability $\epsilon_{\mathcal{A}}$, there exists a PPT adversary $\mathcal{B}$ that solves the discrete logarithm problem in $\mathbb{G}$ in time $t_{\mathcal{B}} \approx t_{\mathcal{A}}$ with success probability $\epsilon_{\mathcal{B}} \geq \epsilon_{\mathcal{A}}$.

Exercise 42. Prove Theorem 41. The proof is very similar to the security proof of the one-time signature procedure from Section 2.3.1.

3.3.2 Chameleon Hash Functions based on the RSA Assumption

We use the same definition of the RSA problem and the same notation as in Section 2.3.2. We construct algorithms $(\mathsf{Gen}_{\mathsf{ch}}, \mathsf{TrapColl}_{\mathsf{ch}})$ defining a chameleon hash function

$$\mathsf{ch} : [0, 2^n - 1] \times \mathbb{Z}_N \setminus \{0\} \to \mathbb{Z}_N.$$

$\mathsf{Gen}_{\mathsf{ch}}(1^k)$**.** Let $n \in \mathbb{N}$. The generation algorithm generates an RSA modulus $N = PQ$ and a prime number $e > 2^n$ with $\gcd(e, \phi(N)) = 1$ and calculates $d := e^{-1} \bmod \phi(N)$. In addition, a number $J \xleftarrow{\$} \mathbb{Z}_N$ is randomly selected. The function description consists of (N, e, J). Given $(m, r) \in \{0,1\}^n \times \mathbb{Z}_N \setminus \{0\}$ the function value is computed as

$$\mathsf{ch}(m, r) = J^m r^e.$$

The trapdoor is the secret RSA key $\tau := d$.

$\mathsf{TrapColl}_{\mathsf{ch}}(\tau, m, r, m^*)$. Given $(m, r, m^*) \in [0, 2^n - 1] \times \mathbb{Z}_N \setminus \{0\} \times [0, 2^n - 1]$, the value r^* is computed as the solution of the equation

$$J^m r^e \equiv J^{m^*}(r^*)^e \bmod N \iff r^* \equiv \left(J^{m-m^*} \cdot r^e\right)^d \bmod N.$$

If this value does not exist, because J^{m^*} is not invertible, an error symbol $\perp$ is returned. (However, this happens "almost never". Can you explain why?)

Exercise 43. To run $\mathsf{TrapColl}_{\mathsf{ch}}$ efficiently, one does not necessarily need to know the factorization of N (or, equivalently, the secret exponent d belonging to e). Instead, the e-th root of J is sufficient, i.e., the number j so that $j^e = J \bmod N$. Explain how $(\mathsf{Gen}_{\mathsf{ch}}, \mathsf{TrapColl}_{\mathsf{ch}})$ are defined in this case.

This chameleon hash function is collision resistant under the RSA assumption. The following theorem states that we can construct an algorithm to solve the RSA problem from an algorithm that breaks the collision resistance.

Theorem 44. *For each PPT adversary $\mathcal{A}$ that receives (N, e, J) as input, runs in time $t_{\mathcal{A}}$, and outputs four values*

$$(m, r, m^*, r^*) \in [0, 2^n - 1] \times \mathbb{Z}_N \setminus \{0\} \times [0, 2^n - 1] \times \mathbb{Z}_N$$

so that $(m, r) \neq (m^*, r^*)$ and

$$J^m r^e \equiv J^{m^*}(r^*)^e \bmod N$$

with success probability $\epsilon_{\mathcal{A}}$, there exists a PPT adversary $\mathcal{B}$ which, given (N, e, y) with $e > 2^n$ prime and $y \xleftarrow{\$} \mathbb{Z}_N$, calculates $x \in \mathbb{Z}_N$ such that $x^e \equiv y \bmod N$. Adversary $\mathcal{B}$ runs in time $t_{\mathcal{B}} \approx t_{\mathcal{A}}$ and has success probability $\epsilon_{\mathcal{B}} \geq \epsilon_{\mathcal{A}}$.

Exercise 45. Prove Theorem 44. The proof is very similar to the security proof of the one-time signature scheme from Section 2.3.2.

3.4 Chamäleon-Signaturen

With chameleon signature methods, a message is signed slightly differently than with normal digital signatures. For signature creation as well as for verification, a chameleon hash function *of the recipient* is additionally used. Therefore both the signature algorithm and the verification algorithm get an additional input, namely the chameleon hash function of the recipient.

Instead of giving a general definition of a chameleon signatures first, we will directly describe a scheme. In the following let

$$\mathsf{ch} : \mathcal{M} \times \mathcal{R} \to \mathcal{N}$$

be a chameleon hash function generated by the recipient by $(\mathsf{ch}, \tau) \xleftarrow{\$} \mathsf{Gen}_{\mathsf{ch}}(1^k)$.[1] Let $\Sigma' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Vfy}')$ be a signature scheme. We define a chameleon signature scheme as follows.

$\mathsf{Gen}(1^k)$**:** The key generation algorithm receives the security parameter as input. It computes the key pair by $(pk, sk) \xleftarrow{\$} \mathsf{Gen}'(1^k)$ and outputs (pk, sk).

$\mathsf{Sign}(sk, m, \mathsf{ch})$**:** The signature algorithm chooses $r \xleftarrow{\$} \mathcal{R}$ and first calculates $m' := \mathsf{ch}(m, r)$ and then $\sigma' := \mathsf{Sign}'(sk, m')$. The signature consists of

$$\sigma := (\sigma', r).$$

$\mathsf{Vfy}(pk, m, \sigma, \mathsf{ch})$**:** The verification algorithm receives $\sigma = (\sigma', r)$, and checks if

$$\mathsf{Vfy}'(pk, \mathsf{ch}(m, r), \sigma') \stackrel{?}{=} 1.$$

If this is true, it outputs 1. Otherwise 0.

Now we first have to show that using the chameleon hash function does not make this scheme insecure, provided that the chameleon hash function is collision resistant. We consider adversaries that do *not* know the trapdoor τ of ch – otherwise the scheme becomes trivially insecure.

The EUF-CMA security experiment with adversary $\mathcal{A}$, challenger $\mathcal{C}$, signature procedure $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ and chameleon hash function $(\mathsf{Gen}_{\mathsf{ch}}, \mathsf{TrapColl}_{\mathsf{ch}})$ is identical to the EUF-CMA game, except that the attacker receives the chameleon hash function of the recipient as an additional input. For the sake of completeness we recall the whole experiment (see Figure 3.1):

1. The challenger $\mathcal{C}$ generates a key pair of the sender $(pk, sk) \xleftarrow{\$} \mathsf{Gen}(1^k)$ and a chameleon hash function of the receiver $(\mathsf{ch}, \tau) \xleftarrow{\$} \mathsf{Gen}_{\mathsf{ch}}(1^k)$. The adversary receives (pk, ch).

2. Now $\mathcal{A}$ may ask for signatures for arbitrary messages $m_1, \ldots, m_q$. For this purpose it sends message m_i to the challenger. The challenger computes $\sigma_i = \mathsf{Sign}(sk, m_i, \mathsf{ch})$ and replies with σ_i. This step can be repeated by the adversary as often as desired. Since we consider adversaries with polynomially bounded runtime, then $q = q(k)$ is a polynomial in the security parameter.

3. At the end $\mathcal{A}$ outputs a message m^* with signature σ^*. It *wins* the game if

$$\mathsf{Vfy}(pk, m^*, \sigma^*, \mathsf{ch}) = 1 \qquad \text{and} \qquad m^* \notin \{m_1, \ldots, m_q\}.$$

So $\mathcal{A}$ wins if σ^* is a valid signature for m^* and the challenger $\mathcal{C}$ did not ask for a signature for m^*.

Remark 46. Despite modeling EUF-CMA security, this is a relatively weak security experiment for chameleon signatures, since the adversary can only obtain signatures that are intended for a third party, the honest recipient. In particular, the attacker must not generate a chameleon hash function (possibly maliciously) himself and request a signature from the challenger with respect to this chameleon hash function. However, this could help him to forge a signature that is also accepted by a third party, the honest recipient. We consider this security model, which was also used in [KR00], for simplicity.

[1] A recipient could also maliciously generate ch so that he can later plausibly prove that he can find *no* collision for ch. We first assume that this is not the case and argue later how this can be ensured.(todo: explained?) TODO

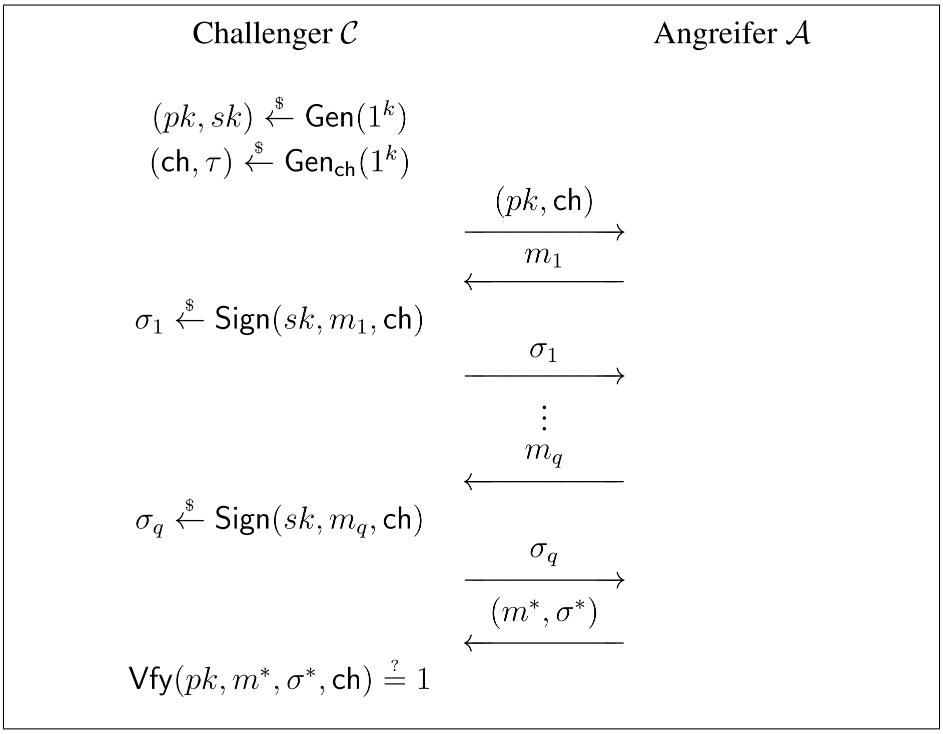
Challenger $\mathcal{C}$
Angreifer $\mathcal{A}$
$(pk, sk) \overset{\$}{\leftarrow} \mathsf{Gen}(1^k)$
$(\mathsf{ch}, \tau) \overset{\$}{\leftarrow} \mathsf{Gen}_{\mathsf{ch}}(1^k)$
$\xrightarrow{(pk, \mathsf{ch})}$
$\xleftarrow{m_1}$
$\sigma_1 \overset{\$}{\leftarrow} \mathsf{Sign}(sk, m_1, \mathsf{ch})$
$\xrightarrow{\sigma_1}$
$\vdots$
$\xleftarrow{m_q}$
$\sigma_q \overset{\$}{\leftarrow} \mathsf{Sign}(sk, m_q, \mathsf{ch})$
$\xrightarrow{\sigma_q}$
$\xleftarrow{(m^*, \sigma^*)}$
$\mathsf{Vfy}(pk, m^*, \sigma^*, \mathsf{ch}) \overset{?}{=} 1$

Figure 3.1: The EUF-CMA Security Experiment for Chameleon Signatures.

Theorem 47. *For every PPT adversary $\mathcal{A}(pk, \mathsf{ch})$ that breaks the EUF-CMA security of Σ in time $t_{\mathcal{A}}$ with success probability $\epsilon_{\mathcal{A}}$, there exists a PPT adversary $\mathcal{B}$ that runs in time $t_{\mathcal{B}} \approx t_{\mathcal{A}}$ and*

- *either breaks the collision resistance of ch with a success probability of at least*

$$\epsilon_{\mathsf{ch}} \geq \frac{\epsilon_{\mathcal{A}}}{2},$$

- *or breaks the EUF-naCMA security of Σ' with a success probability of at least*

$$\epsilon' \geq \frac{\epsilon_{\mathcal{A}}}{2}.$$

Remark 48. Note that the theorem asserts the EUF-CMA security of Σ, even if Σ' is only EUF-naCMA secure.

Proof. Any EUF-CMA adversary $\mathcal{A}$ makes a series of adaptive signature queries $m_1, \ldots, m_q$, $q \geq 0$, in response to which it receives signatures $\sigma_1, \ldots, \sigma_q$, where each signature σ_i consists of two components (σ_i', r_i). We consider two different events:

- We say that event E_0 occurs if the adversary outputs $(m^*, \sigma^*) = (m^*, (\sigma'^*, r^*))$, so

$$\mathsf{ch}(m^*, r^*) = \mathsf{ch}(m_i, r_i)$$

for at least one $i \in \{1, \ldots, q\}$.

- We say that event E_1 occurs if the adversary outputs $(m^*, \sigma^*) = (m^*, (\sigma'^*, r^*))$ so that

  $$\mathsf{ch}(m^*, r^*) \neq \mathsf{ch}(m_i, r_i)$$

  for all $i \in \{1, \ldots, q\}$.

Any successful adversary will cause either event E_0 or event E_1. So we have

$$\epsilon_{\mathcal{A}} \leq \Pr[E_0] + \Pr[E_1].$$

The above inequality also implies that at least one of the two inequalities

$$\Pr[E_0] \geq \frac{\epsilon_{\mathcal{A}}}{2} \quad \text{or} \quad \Pr[E_1] \geq \frac{\epsilon_{\mathcal{A}}}{2} \tag{3.1}$$

must be fulfilled.

Attack on the chameleon hash function. Adversary $\mathcal{B}$ tries to break the collision resistance of the chameleon hash function as follows. $\mathcal{B}$ receives ch as input and generates a key pair $(pk, sk) \xleftarrow{\$} \mathsf{Gen}'(1^k)$. Then it starts $\mathcal{A}$ with input pk. $\mathcal{B}$ can simulate the $\mathsf{EUF\text{-}CMA}$ experiment, since it knows the secret key sk and thus can answer all signature queries from $\mathcal{A}$.

With probability $\epsilon_{\mathcal{A}}$ $\mathcal{A}$ will produce a forgery $(m^*, \sigma^*) = (m^*, (\sigma'^*, r^*))$. If this occurs, $\mathcal{B}$ checks if Event E_0 occurs. $\mathcal{B}$ thus checks if there exists $i \in \{1, \ldots, q\}$, so that $\mathsf{ch}(m^*, r^*) = \mathsf{ch}(m_i, r_i)$. If yes, then $\mathcal{B}$ outputs (m^*, r^*, m_i, r_i).

Obviously $\mathcal{B}$ can use the forgery of $\mathcal{A}$ to break the collision resistance of the chameleon hash function when E_0 occurs. So $\mathcal{B}$ is successful with a probability of at least

$$\epsilon_{\mathsf{ch}} \geq \Pr[E_0]. \tag{3.2}$$

attack on Σ'. $\mathcal{B}$ attempts to break the $\mathsf{EUF\text{-}naCMA}$ security of Σ' as follows. It first generates a chameleon hash function $(\mathsf{ch}, \tau) \xleftarrow{\$} \mathsf{Gen}_{\mathsf{ch}}(1^k)$. Then $\mathcal{B}$ q chooses random values $(\tilde{m}_i, \tilde{r}_i) \xleftarrow{\$} \mathcal{M} \times \mathcal{R}$ and computes $y_i = \mathsf{ch}(\tilde{m}_i, \tilde{r}_i)$ for all $i \in \{1, \ldots, q\}$.

The list $y_1, \ldots, y_q$ is output by $\mathcal{B}$ as messages to its $\mathsf{EUF\text{-}naCMA}$ challenger. In response he receives a public key pk as well as signatures $\sigma'_1, \ldots, \sigma'_q$, so that for each $i \in \{1, \ldots, q\}$ the value σ'_i is a valid signature for y_i with respect to pk. Now $\mathcal{B}$ starts the adversary $\mathcal{A}$ on input pk.

The i-th signature request m_i from $\mathcal{A}$ is answered by $\mathcal{B}$ like this:

1. Using the trapdoor τ of the chameleon hash, $\mathcal{B}$ computes

   $$r'_i = \mathsf{TrapColl}_{\mathsf{ch}}(\tau, \tilde{m}_i, \tilde{r}_i, m_i)$$

   so that $y_i = \mathsf{ch}(m_i, r'_i)$.

2. Then $\mathcal{B}$ answers $\mathcal{A}$'s signature query by $\mathcal{A}$ using the signature σ'_i for y_i received from the challenger. $\mathcal{B}$ thus returns $\sigma_i = (\sigma'_i, r_i)$ to $\mathcal{A}$. This is a valid signature for m_i.

With success probability $\epsilon_{\mathcal{A}}$ $\mathcal{A}$ outputs a forgery $(m^*, \sigma^*) = (m^*, (\sigma'^*, r^*))$. We write $y^* := \mathsf{ch}(m^*, r^*)$. If event E_1 occurs, then $y^* \neq y_i$ for all $i \in \{1, \dots, q\}$. Then $\mathcal{B}$ can issue the tuple (y^*, σ'^*) as a valid forgery to its challenger. If E_1 occurs then $\mathcal{B}$ can win against his challenger. So we get

$$\epsilon' \geq \Pr[E_1].. \tag{3.3}$$

In summary, by inserting the inequalities (**??**) and (3.3) into the inequalities from (3.1), it follows that at least one of the two inequalities

$$\epsilon_{\mathsf{ch}} \geq \Pr[E_0] \geq \frac{\epsilon_{\mathcal{A}}}{2} \quad \text{or} \quad \epsilon' \geq \Pr[E_1] \geq \frac{\epsilon_{\mathcal{A}}}{2}$$

must apply. □

So we know now that the additional chameleon hash function does not make the scheme insecure. On the contrary, it makes it even more secure, since EUF-naCMA security of Σ' is enough to show EUF-CMA security of Σ.

Confidentiality. It remains to consider whether a recipient K who receives a signature from sender H_1 can convince a third party H_2 that H_1 has signed a specific message.

If H_1 uses the above chameleon signature scheme, then it uses the chameleon hash function ch of the recipient K to create the signature. For this chameleon hash function, K is able to find arbitrary collisions with the help of the trapdoor – i.e. open a given hash value to *any message*.

From the point of view of a third party H_2, any of these messages could be the message that H_1 originally signed. So K cannot credibly claim that H_1 signed a *specific* message. Only K knows which message H_1 has actually signed.

Maliciously generated chameleon hash function. Of course, it could be that a malicious recipient generates his chameleon hash function in such a way that he can later plausibly explain that he does *not* know the trapdoor. So far, we have made the assumption that the recipient has honestly generated the chameleon hash function with $(\mathsf{ch}, \tau) \xleftarrow{\$} \mathsf{Gen}_{\mathsf{ch}}(1^k)$.

However, we can *"force"* the recipient to always know the τ trapdoor. For example, the recipient can do this by publishing a cryptographic proof that it knows the corresponding trapdoor together with the description ch of its chameleon hash function. This can be done with cryptographic protocols, such as (non-interactive) *zero-knowledge proofs of knowledge*. We do not explain here what this is exactly – we merely want to point out that the problem has a solution.

3.5 Chameleon Hash Functions are One-Time Signature Schemes

We have already seen that some constructions of chameleon hash functions are very similar to the one-time signature schemes we already know. This gives reason to suspect that both primitives are related to each other.

In fact it is true that chameleon hash functions imply one-time signature schemes. This means that any chameleon hash function can be used to build a one-time signature scheme in a generic way. The opposite is not necessarily true. The construction was published in 2010 in [Moh11].

Let $(\mathsf{Gen}_{\mathsf{ch}}, \mathsf{TrapColl}_{\mathsf{ch}})$ be a chameleon hash function. We construct a one-time signature scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ as follows.

$\mathsf{Gen}(1^k)$. The key generation algorithm generates a chameleon hash function

$$\mathsf{ch} : \mathcal{M} \times \mathcal{R} \to \mathcal{N}$$

with trapdoor τ by running $(\mathsf{ch}, \tau) \overset{\$}{\leftarrow} \mathsf{Gen}_{\mathsf{ch}}(1^k)$. Then it selects $(\tilde{m}, \tilde{r}) \overset{\$}{\leftarrow} \mathcal{M} \times \mathcal{R}$, and computes $c := \mathsf{ch}(\tilde{m}, \tilde{r})$. The public key is $pk := (\mathsf{ch}, c)$, the secret key is $sk := (\tau, \tilde{m}, \tilde{r})$.

$\mathsf{Sign}(sk, m)$. To sign a message $m \in \mathcal{M}$, a matching randomness r is computed so that $\mathsf{ch}(m, r) = c$. This can be done using $sk = (\tau, \tilde{m}, \tilde{r})$ by calculating

$$r := \mathsf{TrapColl}_{\mathsf{ch}}(\tau, \tilde{m}, \tilde{r}, m).$$

The signature is $\sigma := r$.

$\mathsf{Vfy}(pk, m, \sigma)$. Given a signature $\sigma = r \in \mathcal{R}$, the verification algorithm checks whether

$$c \overset{?}{=} \mathsf{ch}(m, r)$$

holds. If yes, 1 is output, otherwise 0.

Theorem 49. *For every PPT attacker $\mathcal{A}$ that breaks the $\mathsf{EUF\text{-}1\text{-}naCMA}$ security of Σ in time $t_{\mathcal{A}}$ with success probability $\epsilon_{\mathcal{A}}$, there is exists a PPT attacker $\mathcal{B}$ that breaks the collision resistance of the chameleon hash function in time $t_{\mathcal{B}} \approx t_{\mathcal{A}}$ with success probability $\epsilon_{\mathcal{B}} \geq \epsilon_{\mathcal{A}}$.*

Exercise 50. Prove theorem 49. Take advantage of the fact that in the $\mathsf{EUF\text{-}1\text{-}naCMA}$ experiment one must only generate the pk *after* having received the selected message m from the adversary, and that the adversary produces a solution (m^*, σ^*) to the equation

$$c = \mathsf{ch}(m^*, \sigma^*)$$

Exercise 51. Construct a $\mathsf{EUF\text{-}1\text{-}CMA}$ secure one-time signature scheme from chameleon hash functions. Here you can apply the transformation from Section 2.4.

Remark 52. The discrete logarithm problem (DLP) based and the RSA-based one-time signature schemes described in Chapter 2 are exactly the one-time signature schemes that result from applying the above transformation to the DLP- and RSA-based chameleon hash functions.

3.6 Strong Existential Unforgeability durch Chamäleon-Hashing

The strongest security definition for digital signatures presented so far is $\mathsf{EUF\text{-}CMA}$. For some applications, however, even stronger security is required.

In the case of $\mathsf{EUF\text{-}CMA}$ security (or actually in general EUF security, combined with any attacker model such as NMA, naCMA, CMA,...) we demand that no efficient adversary is able to create a valid signature for a *new* message $m^* \notin \{m_1, \ldots, m_q\}$. This models the goal that no adversary is able to create signatures for any messages that are not signed by the honest signer.

One attack that is not covered by this security definition is that the adversary might be able to create a *new* signature for a message $m^* \in (m_1, \dots, m_q)$ that was signed by the signer.

Such an attack may seem pointless at first, because the adversary already knows a signature for m^*, why would it want to create a new one? However, in fact there are applications in cryptography where such attacks must be considered. Perhaps the most important example are some constructions public key encryption schemes (IND-CCA secure), which use a (one-time) signature scheme, or alternatively a chameleon hash function, as a building block. These constructions are not trivial, so unfortunately we cannot go into details here. It is only important to say that EUF-CMA security is sometimes not enough, and stronger security features are needed.

A scheme where it is not possible to generate a *new* signature, even if another signature for the same message is already known, is called *strongly* unforgeable.

The sEUF-CMA Security Experiment . The sEUF-CMA security experiment with attacker $\mathcal{A}$, challenger $\mathcal{C}$ and signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ runs exactly like the EUF-CMA security experiment (see also Figure 3.2):

1. The challenger $\mathcal{C}$ generates a key pair $(pk, sk) \xleftarrow{\$} \mathsf{Gen}(1^k)$. The adversary gets pk.
2. Now the attacker $\mathcal{A}$ may have messages $m_1, \dots, m_q$ signed by the challenger.

   For this purpose it sends message m_i to the challenger. The challenger computes $\sigma_i = \mathsf{Sign}(sk, m_i)$ and answers with σ_i.

   This step can be repeated as often as desired by the adversary. If we consider adversaries with polynomial limited runtime, then $q = q(k)$ is a polynomial in the security parameter.
3. At the end $\mathcal{A}$ outputs a message m^* with signature σ^*.

Definition 53. We say that $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ is *secure* in the sense of sEUF-CMA, if for all PPT adversaries $\mathcal{A}$ in the sEUF-CMA experiment holds that

$$\Pr[\mathcal{A}^{\mathcal{C}}(pk) = (m^*, \sigma^*) : \mathsf{Vfy}(pk, m^*, \sigma^*) = 1 \wedge (m^*, \sigma^*) \notin \{(m_1, \sigma_1), \dots, (m_q, \sigma_q)\}] \leq \mathsf{negl}(k)$$

for a negligible function negl in the security parameter.

The difference between the definitions of EUF-CMA security and sEUF-CMA security is subtle. While in the case of EUF-CMA-security we only require that

$$m^* \notin \{m_1, \dots, m_q\}$$

the stronger requirement for sEUF-CMA is that

$$(m^*, \sigma^*) \notin \{(m_1, \sigma_1), \dots, (m_q, \sigma_q)\}.$$

The above definition of sEUF-CMA security can be naturally adapted to sEUF-naCMA, sEUF-1-CMA and sEUF-1-naCMA.

Exercise 54. Specify the definitions of sEUF-1-naCMA and sEUF-1-CMA security by describing the security experiments and the winning condition.

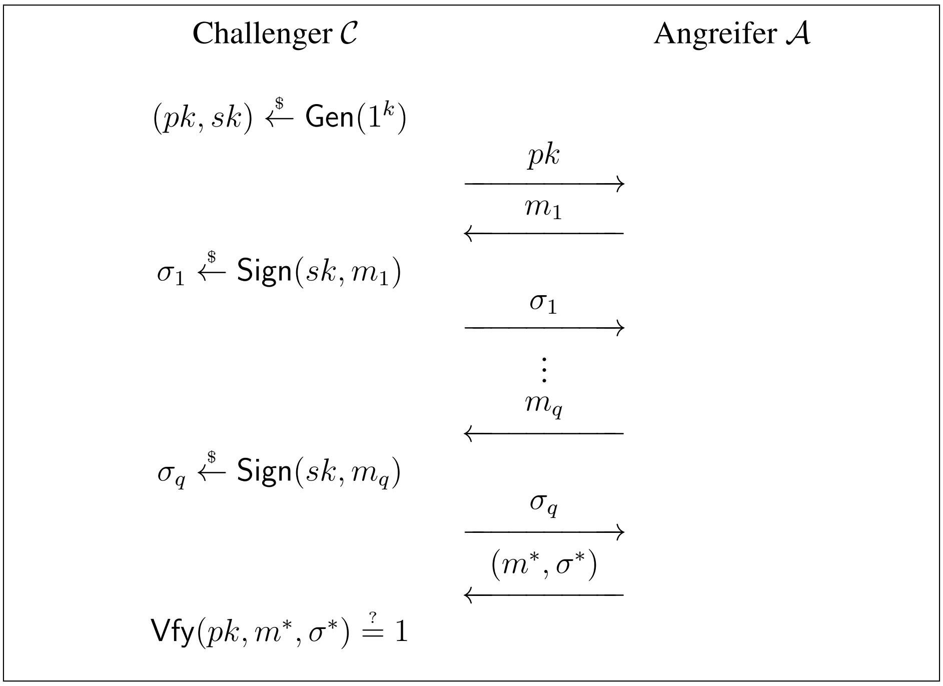
Challenger $\mathcal{C}$
Angreifer $\mathcal{A}$
$(pk, sk) \overset{\$}{\leftarrow} \mathsf{Gen}(1^k)$
pk
m_1
$\sigma_1 \overset{\$}{\leftarrow} \mathsf{Sign}(sk, m_1)$
σ_1
$\vdots$
m_q
$\sigma_q \overset{\$}{\leftarrow} \mathsf{Sign}(sk, m_q)$
σ_q
(m^*, σ^*)
$\mathsf{Vfy}(pk, m^*, \sigma^*) \overset{?}{=} 1$

Figure 3.2: The sEUF-CMA seurity experiment is identical to the EUF-CMA security experiment.

Exercise 55. Prove that one-time signature schemes created by applying the transformation from Section 3.5 to chameleon hash functions are sEUF-1-naCMA secure.

Exercise 56. Construct a sEUF-1-CMA secure one-time signature scheme from chameleon hash functions. If you have used the transformation from Section 2.4 to solve Exercise 51, you can show that the construction is already sEUF-1-CMA-secure.

Strong Existential Unforgeability from Chameleon Hashing. Interestingly, it is very easy to transform a EUF-naCMA secure signature scheme Σ' into a sEUF-CMA secure scheme. This is done by cleverly applying a sEUF-1-CMA-secure one-time signature scheme. Such a procedure was constructed in exercise 56 based on chameleon hash functions.

The Transformation. In the following $\Sigma^{(1)} = (\mathsf{Gen}^{(1)}, \mathsf{Sign}^{(1)}, \mathsf{Vfy}^{(1)})$ denotes a sEUF-1-CMA secure one-time signature scheme and $\Sigma' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Vfy}')$ denotes a EUF-naCMA secure signature scheme. We describe a new signature scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ which uses $\Sigma^{(1)}$ and Σ' as building blocks. The construction strongly resembles the transformation from EUF-naCMA secure signature schemes to EUF-CMA secure signatures from **??**.

$\mathsf{Gen}(1^k)$**.** For key generation a key pair $(pk', sk') \overset{\$}{\leftarrow} \mathsf{Gen}'(1^k)$ is generated with the key generation algorithm of Σ'. In addition, two chameleon hash functions are generated by $(\mathsf{ch}, \tau) \overset{\$}{\leftarrow} \mathsf{Gen}_{\mathsf{ch}}(1^k)$ and $(\mathsf{ch}', \tau') \overset{\$}{\leftarrow} \mathsf{Gen}_{\mathsf{ch}}(1^k)$.

The public key is $pk = pk'$, the secret key is $sk = sk'$.

$\mathsf{Sign}(sk, m)$**.** A signature for message m is computed in two steps.

1. First a key pair $(pk^{(1)}, sk^{(1)}) \overset{\$}{\leftarrow} \mathsf{Gen}^{(1)}(1^k)$ is generated for the one-time signature scheme.
2. The value $pk^{(1)}$ is then signed with the signature scheme Σ':
$$\sigma' := \mathsf{Sign}'(sk', pk^{(1)}).$$
3. Then a signature $\sigma^{(1)} = \mathsf{Sign}^{(1)}(sk^{(1)}, (m, \sigma'))$ is computed over (m, σ') using $sk^{(1)}$.

The signature σ for message m is $\sigma := (\sigma', pk^{(1)}, \sigma^{(1)})$.

$\mathsf{Vfy}(pk, m, \sigma)$. The verification algorithm will get $\sigma := (\sigma', pk^{(1)}, \sigma^{(1)})$ and m, and will output 1 if

$$\mathsf{Vfy}'(pk, pk^{(1)}, \sigma') = 1 \qquad \text{and} \qquad \mathsf{Vfy}^{(1)}(pk^{(1)}, (m, \sigma'), \sigma^{(1)}) = 1.$$

Otherwise, 0 is output.

So the idea of the transformation is to sign the signature σ' in addition to the message.

Theorem 57. *For every PPT adversary $\mathcal{A}$ that breaks the* $\mathsf{sEUF\text{-}CMA}$ *security of Σ in time $t_\mathcal{A}$ with probability of success $\epsilon_\mathcal{A}$, there is a PPT adversary $\mathcal{B}$ that runs in time $t_\mathcal{B} \approx t_\mathcal{A}$ and*

- *either breaks the collision resistance of* ch *with a success probability of at least*
$$\epsilon_{\mathsf{ch}} \geq \frac{\epsilon_\mathcal{A}}{2},$$
- *or breaks the* $\mathsf{EUF\text{-}naCMA}$ *security of Σ' with a success probability of at least*
$$\epsilon' \geq \frac{\epsilon_\mathcal{A}}{2}.$$

The proof of this theorem is very similar to the proofs of Theorem 32 and Theorem 47. Therefore we do not give it here, but leave it as a (somewhat more complex) exercise.

Exercise 58. Prove Theorem 57.

The construction we have presented in this chapter is basically from Steinfeld, Pieprzyk and Wang [SPW07], but we have described it somewhat differently. While we first constructed a $\mathsf{sEUF\text{-}1\text{-}CMA}$ secure signature scheme from the chameleon hash function, and then used this scheme for the generic transformation, chameleon hash functions were directly used in [SPW07]. A somewhat weaker generic transformation that only works for some signature schemes (so-called “separable signatures”) was introduced in [BSW06].

Bibliography

[BDH11] Johannes A. Buchmann, Erik Dahmen, and Andreas Hülsing. XMSS - A practical forward secure signature scheme based on minimal security assumptions. In Bo-Yin Yang, editor, *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*, pages 117–129, Tapei, Taiwan, November 29 – December 2 2011. Springer, Heidelberg, Germany.

[BHK$^+$19] Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The SPHINCS$^+$ signature framework. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 2129–2146. ACM Press, November 11–15, 2019.

[BR08] Mihir Bellare and Todor Ristov. Hash functions from sigma protocols and improvements to VSH. In Josef Pieprzyk, editor, *Advances in Cryptology – ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 125–142, Melbourne, Australia, December 7–11, 2008. Springer, Heidelberg, Germany.

[BSW06] Dan Boneh, Emily Shen, and Brent Waters. Strongly unforgeable signatures based on computational Diffie-Hellman. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006: 9th International Conference on Theory and Practice of Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 229–240, New York, NY, USA, April 24–26, 2006. Springer, Heidelberg, Germany.

[CHK04] Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 207–222, Interlaken, Switzerland, May 2–6, 2004. Springer, Heidelberg, Germany.

[DA99] Tim Dierks and Christopher Allen. *RFC 2246 - The TLS Protocol Version 1.0.* Internet Activities Board, January 1999.

[EGM96] Shimon Even, Oded Goldreich, and Silvio Micali. On-line/off-line digital signatures. *Journal of Cryptology*, 9(1):35–67, March 1996.

[GMR84] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A "paradoxical" solution to the signature problem (abstract) (impromptu talk). In G. R. Blakley and David Chaum, editors, *Advances in Cryptology – CRYPTO'84*, volume 196 of *Lecture*

Notes in Computer Science, page 467, Santa Barbara, CA, USA, August 19–23, 1984. Springer, Heidelberg, Germany.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[Gol87] Oded Goldreich. Two remarks concerning the Goldwasser-Micali-Rivest signature scheme. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 104–110, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany.

[Gro06] Jens Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In Xuejia Lai and Kefei Chen, editors, *Advances in Cryptology – ASIACRYPT 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 444–459, Shanghai, China, December 3–7, 2006. Springer, Heidelberg, Germany.

[HW09] Susan Hohenberger and Brent Waters. Realizing hash-and-sign signatures under standard assumptions. In Antoine Joux, editor, *Advances in Cryptology – EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 333–350, Cologne, Germany, April 26–30, 2009. Springer, Heidelberg, Germany.

[Kat10] Jonathan Katz. *Digital Signatures*. Springer-Verlag, 2010.

[KL07] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

[KR00] Hugo Krawczyk and Tal Rabin. Chameleon signatures. In *ISOC Network and Distributed System Security Symposium – NDSS 2000*, San Diego, CA, USA, February 2–4, 2000. The Internet Society.

[Lam79] Leslie Lamport. Constructing digital signatures from a one-way function. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory, October 1979.

[Lin03] Yehuda Lindell. A simpler construction of cca2-secure public-key encryption under general assumptions. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 241–254, Warsaw, Poland, May 4–8, 2003. Springer, Heidelberg, Germany.

[May04] Alexander May. Computing the RSA secret key is deterministic polynomial time equivalent to factoring. In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 213–219, Santa Barbara, CA, USA, August 15–19, 2004. Springer, Heidelberg, Germany.

[Mer88] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *Advances in Cryptology – CRYPTO'87*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378, Santa Barbara, CA, USA, August 16–20, 1988. Springer, Heidelberg, Germany.

[Moh11] Payman Mohassel. One-time signatures and chameleon hash functions. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *SAC 2010: 17th Annual International Workshop on Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 302–319, Waterloo, Ontario, Canada, August 12–13, 2011. Springer, Heidelberg, Germany.

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140, Santa Barbara, CA, USA, August 11–15, 1992. Springer, Heidelberg, Germany.

[RD08] Eric Rescorla and Tim Dierks. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, August 2008.

[Res18] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978.

[Sah99] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th Annual Symposium on Foundations of Computer Science*, pages 543–553, New York, NY, USA, October 17–19, 1999. IEEE Computer Society Press.

[Sha83] Adi Shamir. On the generation of cryptographically strong pseudorandom sequences. *ACM Trans. Comput. Syst.*, 1(1):38–44, 1983.

[SPW07] Ron Steinfeld, Josef Pieprzyk, and Huaxiong Wang. How to strengthen any weakly unforgeable signature into a strongly unforgeable signature. In Masayuki Abe, editor, *Topics in Cryptology – CT-RSA 2007*, volume 4377 of *Lecture Notes in Computer Science*, pages 357–371, San Francisco, CA, USA, February 5–9, 2007. Springer, Heidelberg, Germany.

[Zha07] Rui Zhang. Tweaking TBE/IBE to PKE transforms with chameleon hash functions. In Jonathan Katz and Moti Yung, editors, *ACNS 07: 5th International Conference on Applied Cryptography and Network Security*, volume 4521 of *Lecture Notes in Computer Science*, pages 323–339, Zhuhai, China, June 5–8, 2007. Springer, Heidelberg, Germany.