Digital Signatures

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

Updated on December 8, 2020

I would like to thank Florian Böhl, Peter Chvojka, Gareth T. 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

1 Introduction **3**
- 1.1 Definition of Digital Signatures . . . 5
- 1.2 Security of Digital Signatures . . . 6
  - 1.2.1 Attacker Model and Attacker Target . . . 6
  - 1.2.2 Security Experiments . . . 7
  - 1.2.3 Analyzing Relations between Security Definitions . . . 10
  - 1.2.4 Perfectly Secure Digital Signatures Do Not Exist . . . 12
- 1.3 Extending the Message Space, Generically . . . 13

2 One-Time Signatures **16**
- 2.1 Security Definition . . . 16
- 2.2 One-time Signatures from One-Way Functions . . . 18
  - 2.2.1 One-Way Functions . . . 18
  - 2.2.2 Lamport's One-Time Signature Scheme . . . 20
- 2.3 Efficient One-Time Signatures from Concrete Complexity Assumptions . . . 23
  - 2.3.1 One-Time Signatures based on the Discrete Logarithm Problem . . . 23
  - 2.3.2 One-Time Signatures based on the RSA Assumption . . . 25
- 2.4 From EUF-naCMA-Security to EUF-CMA-Security . . . 27
- 2.5 Tree-based Signatures . . . 31
  - 2.5.1 q-Time Signatures . . . 31
  - 2.5.2 Trading Short Signatures for Small Public Keys . . . 32
  - 2.5.3 Clever Compression of Public Keys: Merkle Trees . . . 33
  - 2.5.4 Pseudorandom Functions . . . 35
  - 2.5.5 Secret Key Compression . . . 36
  - 2.5.6 More Efficient Variants . . . 37

3 Chameleon Hash Functions **38**
- 3.1 Motivation . . . 38
- 3.2 Definition of Chameleon Hash Functions . . . 39
- 3.3 Examples of Chameleon Hash Functions . . . 40
  - 3.3.1 Chameleon Hash Functions based on the Discrete Logarithm Problem . . . 40
  - 3.3.2 Chameleon Hash Functions based on the RSA Assumption . . . 41
- 3.4 Chameleon Signatures . . . 42

3.5 Chameleon Hash Functions are One-Time Signatures . . . 46
3.6 Strong Unforgeability from Chameleon Hashing . . . 47

4 RSA-based Signature Schemes **51**
4.1 Textbook RSA Signatures . . . 51
4.2 RSA Full-Domain Hash and the Random Oracle Model . . . 53
4.2.1 The Random Oracle Model . . . 54
4.2.2 Security Proof of RSA-FDH in the Random Oracle Model . . . 56
4.3 Gennaro-Halevi-Rabin Signatures . . . 59
4.3.1 The Strong RSA Assumption . . . 59
4.3.2 GHR Signatures . . . 60
4.3.3 Hash Functions Mapping to Primes . . . 62
4.4 Hohenberger-Waters Signatures . . . 63
4.4.1 Selective Security of GHR Signatures . . . 63
4.4.2 From SUF-naCMA-Security to EUF-naCMA-Security . . . 67
4.4.3 Clever "Compression" of GHR Signatures . . . 69
4.5 Open Problems . . . 70

5 Pairing-based Signature Schemes **71**
5.1 Pairings . . . 71
5.2 Boneh-Lynn-Shacham Signatures . . . 73
5.2.1 The Computational Diffie-Hellman Problem . . . 74
5.2.2 Security of BLS Signatures . . . 74
5.2.3 Aggregability of BLS Signatures . . . 76
5.2.4 Batch Verification of BLS Signatures . . . 77
5.3 Boneh Boyen Signatures . . . 77
5.3.1 The Signature Scheme . . . 78
5.3.2 Security Analysis . . . 78
5.3.3 The Fully Secure Scheme of Boneh and Boyen . . . 81
5.3.4 Similarity to GHR Signatures . . . 82
5.4 Waters Signatures . . . 83
5.4.1 Programmable Hash Functions . . . 83
5.4.2 The Signature Scheme . . . 86
5.4.3 Security of Waters Signatures . . . 86
5.4.4 The Programmable Hash Function of Waters . . . 88
5.5 Open Problems . . . 90

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 $(pk, sk) \xleftarrow{\$} \mathsf{Gen}(1^\lambda)$ to create a key pair, and then
2. create a signature $\sigma \xleftarrow{\$} \mathsf{Sign}(sk, m)$ 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 $(pk, sk) \xleftarrow{\$} \mathsf{Gen}(1^\lambda)$ 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 $\mathcal{A}$.
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.

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

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

for a negligible function negl in the security parameter.

Remark 3. As a reminder, a function $\mathsf{negl} : \mathbb{N} \to [0, 1]$ 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. $\mathsf{negl}(\lambda) = o(1/\mathsf{poly}(\lambda))$.

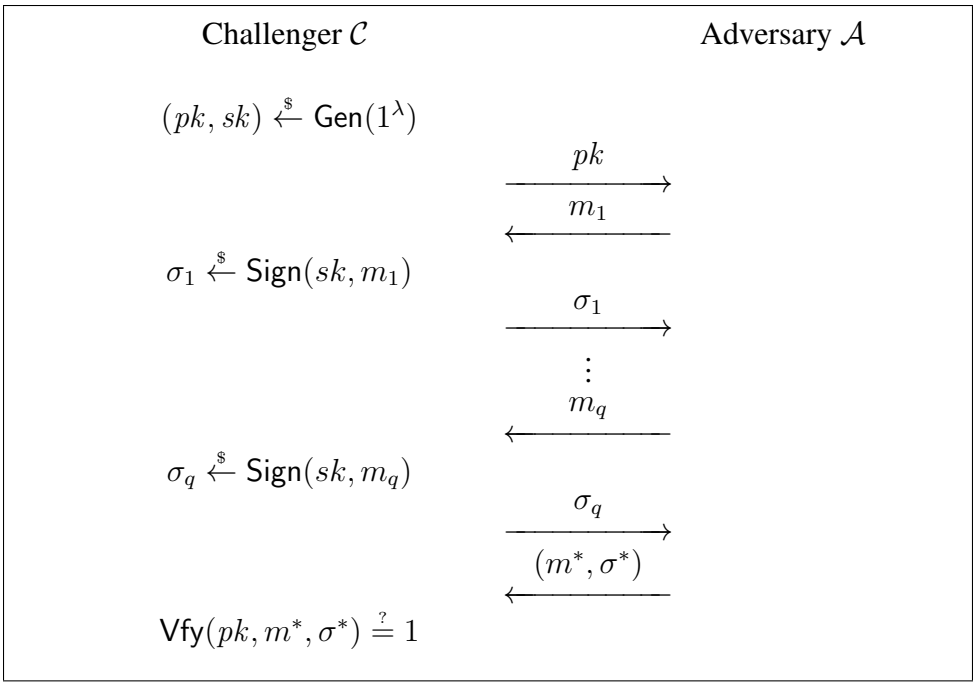

Figure 1.1: The EUF-CMA security experiment.

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 $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ 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.

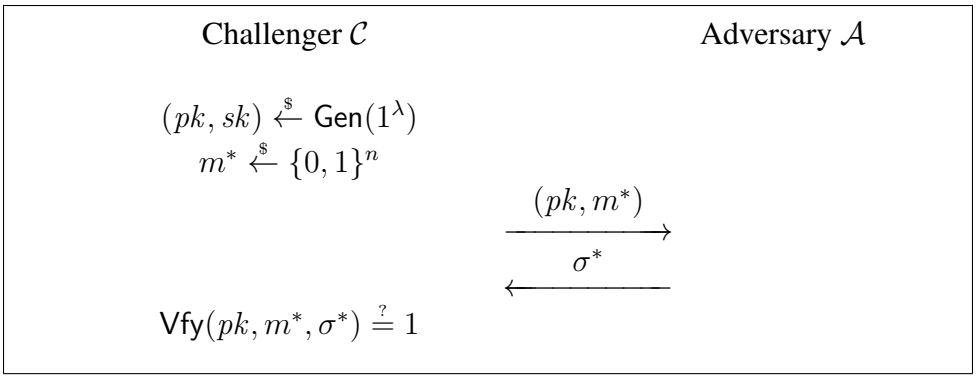

Figure 1.2: The UUF-NMA security experiment.

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:

$$\begin{array}{ccccc} \mathsf{UUF\text{-}NMA} & \leq & \mathsf{UUF\text{-}naCMA} & \leq & \mathsf{UUF\text{-}CMA} \\ \shortmid\!\wedge & & \shortmid\!\wedge & & \shortmid\!\wedge \\ \mathsf{EUF\text{-}NMA} & \leq & \mathsf{EUF\text{-}naCMA} & \leq & \mathsf{EUF\text{-}CMA} \end{array}$$

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 EUF-CMA-secure, but not UUF-NMA-secure. If we can show this assumption to be contradictory, then it must be wrong.

If Σ is not UUF-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 EUF-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 EUF-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 $m^* \xleftarrow{\$} \{0,1\}^n$ 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 EUF-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 EUF-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 UUF-NMA-secure, then it is not EUF-CMA-secure either. This applies to any signature scheme Σ. □

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

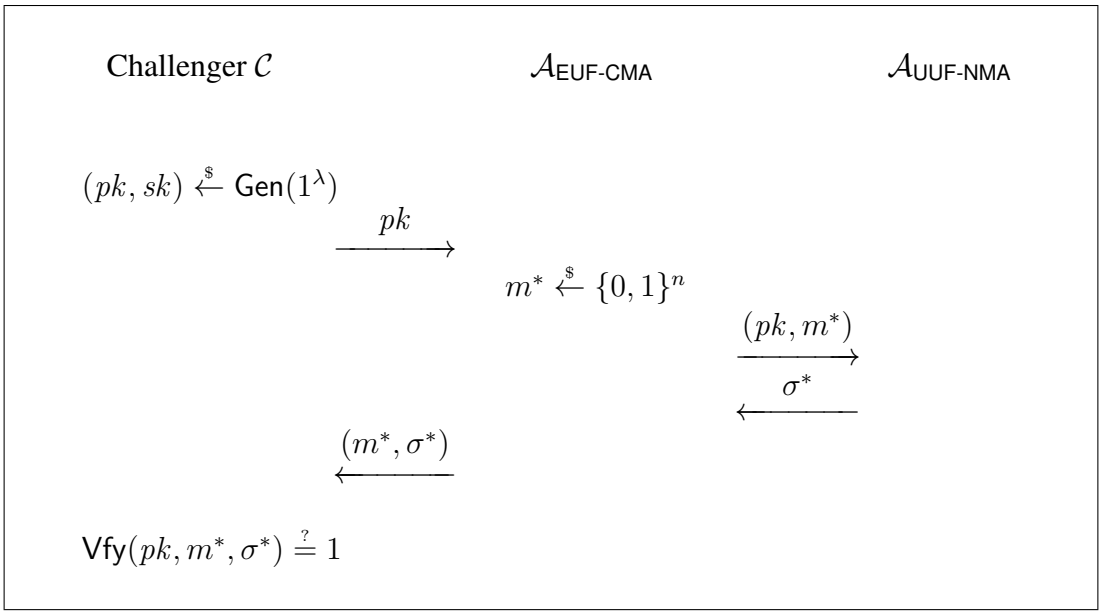

Figure 1.3: Construction of $\mathcal{A}_{\mathsf{EUF\text{-}CMA}}$ from $\mathcal{A}_{\mathsf{UUF\text{-}NMA}}$.

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

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

> *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 UUF-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 UUF-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 UUF-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 $\{0, \ldots, p-1\}$.

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 $(pk, sk) \xleftarrow{\$} \mathsf{Gen}'(1^\lambda)$ 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 $\sigma \xleftarrow{\$} \mathsf{Sign}'(sk, H(m))$.

$\mathsf{Vfy}(pk, m, \sigma)$**.** 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 consid-

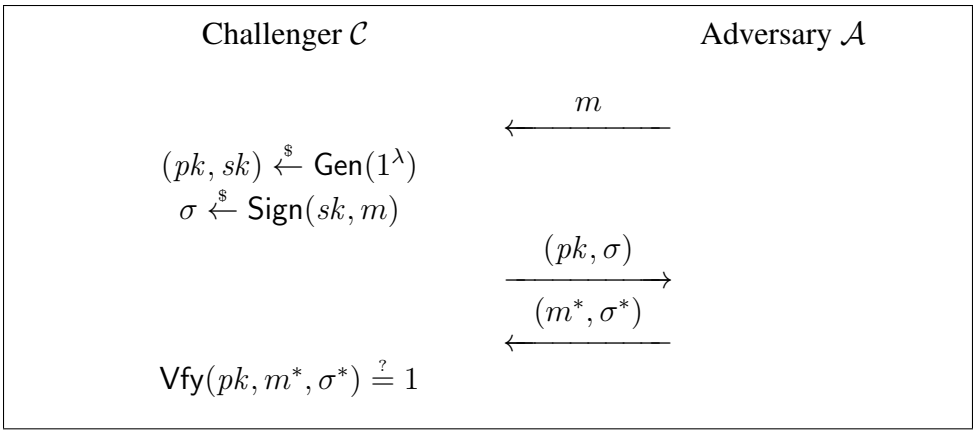

Figure 2.1: The EUF-1-naCMA security experiment.

eration 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 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.

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 it holds that

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

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 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.

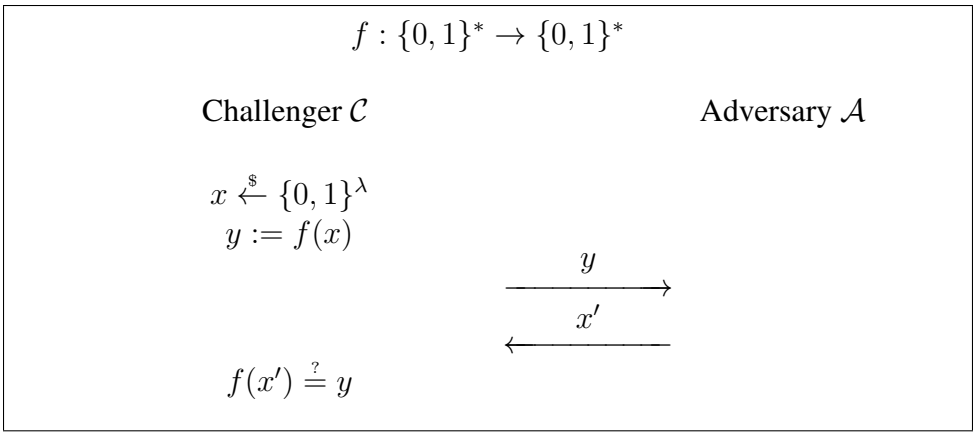

Figure 2.2: The security experiment for one-way functions.

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 $x \xleftarrow{\$} \{0,1\}^\lambda$ 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 it 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, for instance, adequately-chosen subgroups of the multiplicative group $\mathbb{Z}_p^*$ or certain elliptic curve groups.

The discrete exponential function is not immediately a function $f : \{0,1\}^* \to \{0,1\}^*$ 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 $\mathbb{Z}_N \to \mathbb{Z}_N$, $x \mapsto x^e \bmod N$ 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 $f : \{0,1\}^* \to \{0,1\}^*$, 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 $\{0,1\}^n$.

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

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

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 $m = (m_1, \ldots, m_n) \in \{0,1\}^n$ and a signature $\sigma = (x'_{1,m_1}, \ldots, x'_{n,m_n})$, it is checked whether

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

If this is true for all $i \in \{1, \ldots, n\}$, 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* EUF-1-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*

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

If $\mathcal{A}$ is a polynomial time attacker with a non-negligible success probability, $t_{\mathcal{A}}$ is bounded by $t_{\mathcal{A}} \leq \mathsf{poly}(\lambda)$, and $\epsilon_{\mathcal{A}}$ is bounded by $\epsilon_{\mathcal{A}} \geq 1/\mathsf{poly}(\lambda)$ 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 the 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 EUF-1-naCMA challenger for $\mathcal{A}$. Therefore, $\mathcal{B}$ first receives a message $m = (m_1, \ldots, m_n) \in \{0,1\}^\lambda$ 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 $\nu \xleftarrow{\$} \{1, \ldots, n\}$ 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 $(i,j) \in \{1, \ldots, n\} \times \{0,1\}$ with $(i,j) \neq (\nu, 1-m_\nu)$. The pk generated by $\mathcal{B}$ is

$$pk = (y_{1,0}, y_{1,1}, \ldots, y_{n,0}, y_{n,1}).$$

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 $m' = (m'_1, \ldots, m'_n)$, *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

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

$\mathcal{A}$ 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 $m^* = (m^*_1, \ldots, m^*_n) \in \{0,1\}^n$, 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. 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 part is the extraction of 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 desired 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 at the cost of more expensive calculations and an additional checksum. See [Mer88] for details.

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 $x, \omega \xleftarrow{\$} \mathbb{Z}_p$ 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

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

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 $\sigma \xleftarrow{\$} \mathbb{Z}_p$ 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, with probability $\epsilon_\mathcal{A}$, output (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 $\mathbb{Z}_N^* := \{a \in \mathbb{Z}_N : \gcd(a, N) = 1\}$. $\mathbb{Z}_N^*$ 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 distinct prime numbers. Let $e \in \mathbb{N}$ such that $e > 1$ and $\gcd(e, \phi(N)) = 1$, and let $y \xleftarrow{\$} \mathbb{Z}_N$ 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 [HW09a]. 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 $J, c \xleftarrow{\$} \mathbb{Z}_N$ are randomly selected. The public key is $pk := (N, e, J, c)$, and 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 EUF-1-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 $\sigma \xleftarrow{\$} \mathbb{Z}_N$. Elements σ and x are both distributed randomly over $\mathbb{Z}_N$, so 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 $\epsilon_{\mathcal{A}}$ adversary $\mathcal{A}$ 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 $m - m^* \in [1, 2^n - 1]$ (when $m > m^*$), or $m^* - m \in [1, 2^n - 1]$ (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 $S := \sigma^*/\sigma \in \mathbb{Z}_N$,[1] 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 (Lemma 31) 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 $J = y$.

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 an EUF-1-naCMA-secure one-time signature scheme to transform an 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: 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.

[1] 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 therefore non-invertible, then we can factorize N by calculating $\gcd(N, \sigma)$ and thus solve the given RSA problem instance.

To obtain an EUF-CMA-secure scheme in the end, we thus can first construct an 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 $\Sigma' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Vfy}')$ be a signature scheme. We describe a new scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ which uses $\Sigma^{(1)}$ and Σ' as building blocks.

$\mathsf{Gen}(1^\lambda)$**.** For key generation a key pair $(pk', sk') \xleftarrow{\$} \mathsf{Gen}'(1^\lambda)$ 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 $(pk^{(1)}, sk^{(1)}) \xleftarrow{\$} \mathsf{Gen}^{(1)}(1^\lambda)$ 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.

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

$\mathcal{B}$ generates a key pair $(pk, sk) \xleftarrow{\$} \mathsf{Gen}'(1^\lambda)$ 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 $(pk_i^{(1)}, sk_i^{(1)}) \xleftarrow{\$} \mathsf{Gen}^{(1)}(1^\lambda)$ 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 its EUF-1-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 EUF-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 EUF-1-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 EUF-1-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 EUF-naCMA security of Σ'. Attacker $\mathcal{B}$ attempts to break the EUF-naCMA security of Σ' by again simulating the EUF-CMA challenger for $\mathcal{A}$. This time $\mathcal{B}$ generates q key pairs

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

Then $\mathcal{B}$ outputs the list $(pk_1^{(1)}, \ldots, pk_q^{(1)})$ to its EUF-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 EUF-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 EUF-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 EUF-naCMA security with a probability

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

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 EUF-1-CMA-secure one-time signature scheme 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 EUF-1-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-Time 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-time 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.

$\mathsf{Gen}(1^\lambda)$**.** The public key of the q-time 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:[2]

$$pk = (pk_1, \ldots, pk_q) \quad \text{and} \quad sk = (sk_1, \ldots, sk_q, st)$$

[2]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.

where $(pk_i, sk_i) \xleftarrow{\$} \mathsf{Gen}^{(1)}(1^\lambda)$ for all $i \in \{1, \ldots, q\}$.

$\mathsf{Sign}(sk, m)$. 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 integer $i \in \{1, \ldots, q\}$, which tells the verifier to use the key pk_i for verification. Finally the state is incremented, via $st := st + 1$.

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

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

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-time 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.

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

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 $pk = (pk_1, \ldots, pk_q)$ of all q public keys. So

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

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

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 $\sigma = (\sigma_i, i, (pk_1, \ldots, pk_q))$. The verifier now checks whether

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

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

Exercise 35. Prove that this scheme is an EUF-naCMA-secure (EUF-CMA-secure) q-time signature scheme, if $\Sigma^{(1)}$ is EUF-1-naCMA secure (EUF-1-CMA secure) and the hash function is collision resistant.

This "hashed" variant of the original q-time 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 cleverly than sending the entire list* $(pk_1, \ldots, pk_q)$ *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 compressing a list $(pk_1, \ldots, pk_{2^t})$ 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 compute the hash value $h_{t,i} := H(pk_i)$, $i \in \{1, \ldots, 2^t\}$ for all keys in the list.
2. Then proceed recursively:

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

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 following it will be useful to define some terms about trees.

- 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 lie 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.

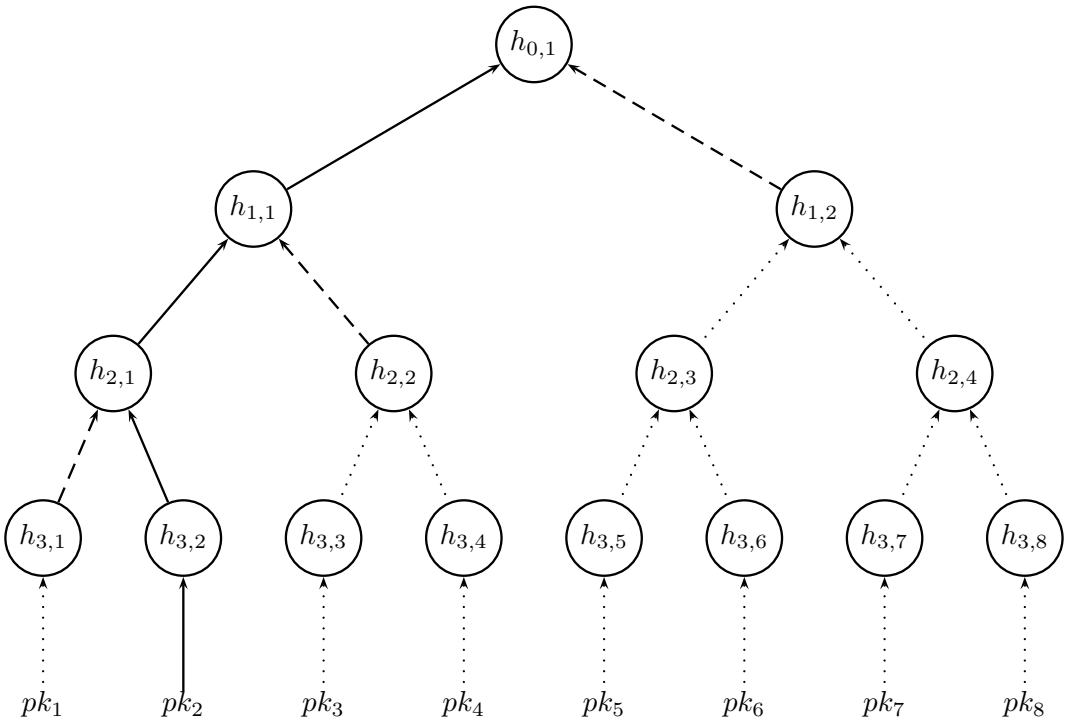

Figure 2.3: Merkle tree with depth 3.

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-time signature scheme, if the underlying one-time signature scheme is EUF-1-naCMA secure (or EUF-1-CMA 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 Pseudorandom Functions

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!

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

$$\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^\ell$$

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

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

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

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. For 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

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

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

2.5.5 Secret Key Compression

Our goal is to "compress" a list of secret keys $(sk_1,\ldots,sk_q)$, 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

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

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

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

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

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

where $s \xleftarrow{\$} \{0,1\}^\lambda$ 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 $s \xleftarrow{\$} \{0,1\}^\lambda$ is randomly chosen, the keys generated are indistinguishable from

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

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 $sk=(sk_1,\ldots,sk_q)$ of keys explicitly. Instead, it is sufficient to select a random seed $s \xleftarrow{\$} \{0,1\}^\lambda$, 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.

2.5.6 More Efficient Variants

One major disadvantage of all the methods presented here must still be pointed out. In each example *all* key pairs $(pk_1, sk_1), \ldots, (pk_q, sk_q)$ 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-time 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, and perhaps it is reassuring to mention right here that their purpose is not to hash any real chameleons. But they have a very interesting property, which is reminiscent of the versatility of a chameleon. They were introduced in Krawczyk and Rabin [KR00] and are an important building block for many cryptographic constructions — in particular, but not only, for digital signature schemes.

We will first motivate chameleon hash functions with a concrete application example. Then we will give an abstract definition and 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 so-called *deniable signatures*. Afterwards we show that one can always construct a one-time signature scheme from a chameleon hash fuction. Finally we show that one can construct signature schemes that are *strongly existentially 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 [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 (of the same quality). The two 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 know 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. H_1 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 (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 allows H_2 to undercut H_1 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 $(\mathsf{Gen}_{\mathsf{ch}}, \mathsf{TrapColl}_{\mathsf{ch}})$.

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

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

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

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

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.

$\mathsf{Gen}_{\mathsf{ch}}(1^\lambda)$**.** The generation algorithm chooses $x \xleftarrow{\$} \mathbb{Z}_p^*$ and computes $h := g^x$. The description of the chameleon 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 scheme 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

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

$\mathsf{Gen}_{\mathsf{ch}}(1^\lambda)$. 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 $(m, r) \in [0, 2^n - 1] \times \mathbb{Z}_N \setminus \{0\}$ 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.

Exercise 43. It happens "almost never" that $\mathsf{TrapColl}$ returns $\perp$. Explain why.

Exercise 44. 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 45. *For each PPT adversary $\mathcal{A}$ that receives (N, e, J) as input, runs in time $t_{\mathcal{A}}$, and outputs four values*

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

such 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 46. Prove Theorem 45. The proof is very similar to the security proof of the one-time signature scheme from Section 2.3.2.

3.4 Chameleon Signatures

With chameleon signatures, 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 of the signature* 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 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 $(\mathsf{ch}, \tau) \xleftarrow{\$} \mathsf{Gen}_{\mathsf{ch}}(1^\lambda)$.[1] Let $\Sigma' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Vfy}')$ be a signature scheme. We define a chameleon signature scheme as follows.

$\mathsf{Gen}(1^\lambda)$**:** The key generation algorithm receives the security parameter as input. It computes the key pair by $(pk, sk) \xleftarrow{\$} \mathsf{Gen}'(1^\lambda)$ 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 $(pk, sk) \xleftarrow{\$} \mathsf{Gen}(1^\lambda)$ and a chameleon hash function of the receiver $(\mathsf{ch}, \tau) \xleftarrow{\$} \mathsf{Gen}_{\mathsf{ch}}(1^\lambda)$. 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 $q = q(\lambda)$ is a polynomial in the security parameter.

[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.

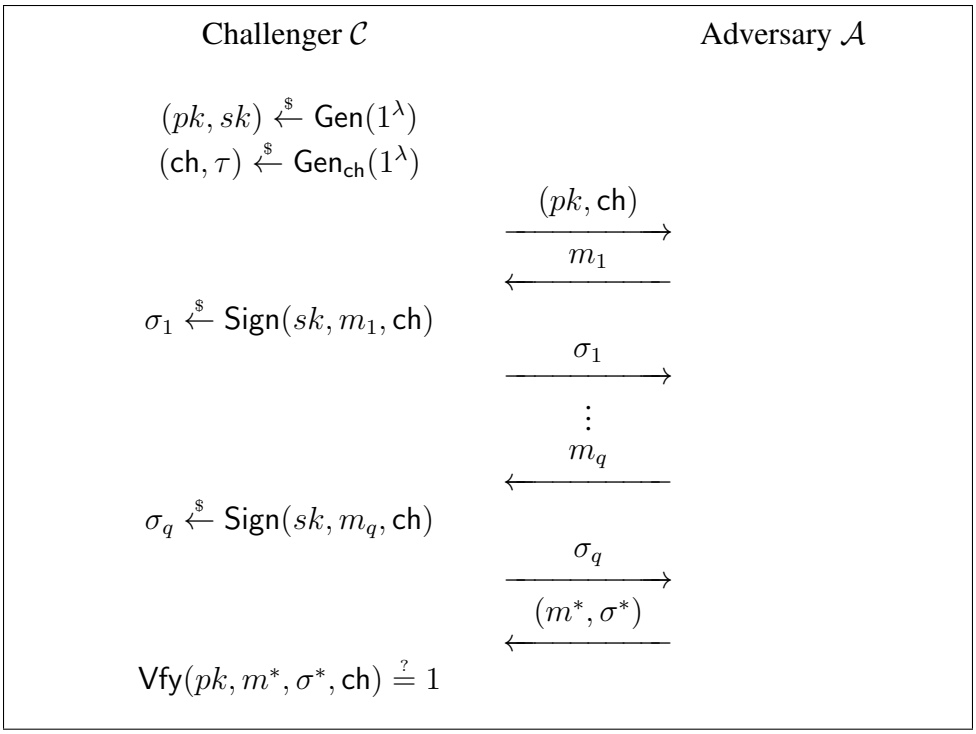

Figure 3.1: The EUF-CMA Security Experiment for Chameleon Signatures.

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 47. 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.

Theorem 48. *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 49. 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 to occur. 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 $(pk, sk) \xleftarrow{\$} \mathsf{Gen}'(1^\lambda)$. Then it starts $\mathcal{A}$ with input pk. $\mathcal{B}$ can simulate the EUF-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 $i \in \{1, \ldots, q\}$, such 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 EUF-naCMA security of Σ' as follows. It first generates a chameleon hash function $(\mathsf{ch}, \tau) \xleftarrow{\$} \mathsf{Gen}_{\mathsf{ch}}(1^\lambda)$. Then $\mathcal{B}$ chooses q random pairs $(\tilde{m}_i, \tilde{r}_i) \xleftarrow{\$} \mathcal{M} \times \mathcal{R}$ 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 EUF-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 $\mathcal{B}$ as follows:

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 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 $i \in \{1, \ldots, q\}$. 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 its challenger. So we get

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

In summary, by inserting the inequalities (3.2) 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 functions. 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 $(\mathsf{ch}, \tau) \xleftarrow{\$} \mathsf{Gen}_{\mathsf{ch}}(1^\lambda)$.

However, we can *"force"* the recipient to always know the trapdoor τ. 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 Signatures

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 these 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.

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

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

with trapdoor τ by running $(\mathsf{ch}, \tau) \xleftarrow{\$} \mathsf{Gen}_{\mathsf{ch}}(1^\lambda)$. Then it selects $(\tilde{m}, \tilde{r}) \xleftarrow{\$} \mathcal{M} \times \mathcal{R}$, 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 value 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

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

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

Theorem 50. *For every PPT attacker $\mathcal{A}$ that breaks the EUF-1-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 51. Prove theorem 50. Take advantage of the fact that in the EUF-1-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 52. Construct an EUF-1-CMA-secure one-time signature scheme from chameleon hash functions. Here you can apply the transformation from Section 2.4.

Remark 53. The one-time signature schemes described in Chapter 2 built from the discrete logarithm problem (DLP) and the RSA problem 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 Unforgeability from Chameleon Hashing

The strongest security definition for digital signatures presented so far is EUF-CMA. For some applications, however, even stronger security is required.

In the case of EUF-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 $m^* \in (m_1, \ldots, m_q)$ 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 of 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 $(pk, sk) \xleftarrow{\$} \mathsf{Gen}(1^\lambda)$. The adversary gets pk.

2. Now the attacker $\mathcal{A}$ may have messages $m_1, \ldots, m_q$ 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 $q = q(\lambda)$ is a polynomial in the security parameter.

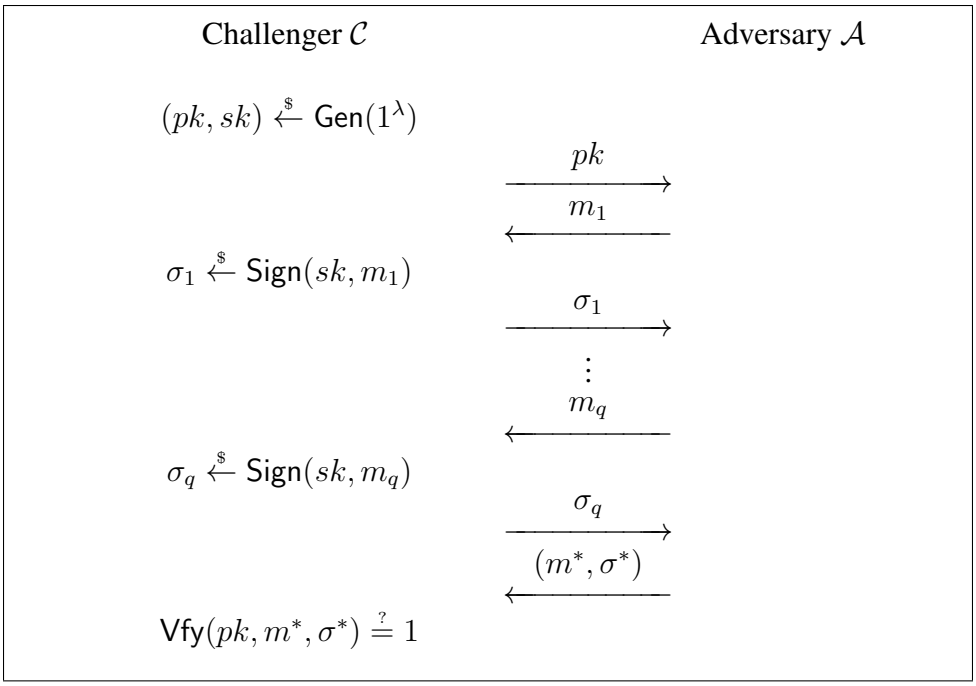

Figure 3.2: The sEUF-CMA security experiment is identical to the EUF-CMA security experiment.

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

Definition 54. 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

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

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

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

the stronger requirement for sEUF-CMA is that

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

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

Exercise 55. Specify the definitions of sEUF-1-naCMA and sEUF-1-CMA security by describing the security experiments and the winning condition.

Exercise 56. 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 57. 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 52, 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 an 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 57 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 an 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 Section 2.4.

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

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

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

1. First a key pair $(pk^{(1)}, sk^{(1)}) \xleftarrow{\$} \mathsf{Gen}^{(1)}(1^\lambda)$ 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 58. *For every PPT adversary $\mathcal{A}$ that breaks the sEUF-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* sEUF-1-CMA *security of* $\Sigma^{(1)}$ *with a success probability of at least*

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

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

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

The proof of this theorem is very similar to the proofs of Theorem 32 and Theorem 48. Therefore we do not give it here, but leave it as an exercise.

Exercise 59. Prove Theorem 58.

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 sEUF-1-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].

Chapter 4

RSA-based Signature Schemes

So far in this lecture we have learned about one-time signature schemes and q-time signature schemes based on Merkle trees, where q was polynomially bounded. In this chapter we start with the consideration of signature schemes that can sign an exponential number of messages. An important class of such procedures is based on variants of the RSA assumption.

In this chapter, we first consider the "textbook RSA" signature scheme, which has some disadvantages (such as in particular its homomorphic nature, which is often advantageous, but not necessarily in the context of signature schemes). Then we describe and analyze the following RSA-based signature schemes:

- The RSA Full-Domain Hash scheme [BR96]. The security analysis of this scheme is performed in an idealized model, the so-called *Random Oracle Model*.
- The signature scheme of Gennaro, Halevi and Rabin [GHR99]. It is the first "hash-and-sign" signature scheme with an EUF-CMA security proof without the need for an idealized model (i.e., proven secure in the so-called *standard model*), but it is based on a stronger complexity assumption, the so-called *Strong-RSA assumption*.
- The RSA-based signature scheme by Hohenberger and Waters [HW09b]. This scheme is the first one that can be proven EUF-CMA secure under the RSA assumption. Unfortunately, it is relatively inefficient. The construction of an efficient RSA-based signature scheme with security proof in the standard model (i.e. without random oracles or similar idealizations) is an important open problem.

4.1 Textbook RSA Signatures

We use the notation and definitions of $\mathbb{Z}_N$, $\mathbb{Z}_N^*$ and $\phi(N)$ introduced in Section 2.3.2 and the formulation of the RSA problem from Definition 29.

The textbook RSA signature scheme works like this:

$\mathsf{Gen}(1^k)$**.** The key generation algorithm generates an RSA modulus $N = PQ$, where P and Q are two randomly chosen prime numbers. Then a number $e \in \mathbb{N}$ is chosen with $e \neq 1$ and

$\gcd(e, \phi(N)) = 1$, and the value $d := e^{-1} \bmod \phi(N)$ is computed.

The public key is $pk := (N, e)$, and the secret key is $sk := d$.

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

$$\sigma \equiv m^d \bmod N.$$

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

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

holds, and otherwise 0.

Correctness. The correctness of the scheme results from the fact that $d \equiv e^{-1} \bmod \phi(N)$ and therefore

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

holds.

Security of Textbook RSA Signatures. The textbook RSA signature scheme is very simple and therefore well suited for explaining the principle of digital signatures in textbooks. However, it is too weak for many practical applications.

EUF security. It is quite easy to see that the scheme is not EUF-NMA secure, and thus not EUF-CMA secure.

An attacker could simply choose any signature $\sigma^* \in \mathbb{Z}_N$ and compute the matching message m^* as $m^* \equiv (\sigma^*)^e \bmod N$. Obviously (m^*, σ^*) is a valid existential forgery. The attacker does not even have to make a chosen-message request to the challenger, so this is a no-message attack.

UUF-CMA security Textbook RSA signatures are not UUF-CMA-secure either.

An attacker receives a message m^* as input from the challenger. His goal is to compute a signature σ^* with $(\sigma^*)^e \equiv m^* \bmod N$. It proceeds in this way:

1. The attacker chooses a random value $x \xleftarrow{\$} \mathbb{Z}_N^* \setminus \{1\}$ and computes $y \equiv x^e \bmod N$.
2. Then the attacker computes $m_1 := m^* \cdot y \bmod N$, and asks the challenger for a signature for m_1. Because $x \neq 1 \bmod N$ was chosen, $y \neq 1 \bmod N$ and therefore $m_1 \neq m^*$. The attacker obtains a value σ_1 with $\sigma_1^e \equiv m_1 \bmod N$.
3. Finally the attacker computes $\sigma^* \equiv \sigma_1 \cdot x^{-1} \bmod N$, and outputs σ^*. This is possible because $x \in \mathbb{Z}_N^*$ is invertible. Furthermore, this is a valid signature for m^*, because

$$(\sigma^*)^e \equiv (\sigma_1 \cdot x^{-1})^e \equiv \sigma_1^e \cdot (x^e)^{-1} \equiv m_1 \cdot y^{-1} \equiv m^* \cdot y \cdot y^{-1} \equiv m^* \bmod N.$$

The fact that textbook RSA signatures are *multiplicatively homomorphic* allows this UUF-CMA attack.

UUF-NMA Security It can be shown by a simple reduction that textbook RSA signatures are UUF-NMA secure under the assumption that the RSA problem is hard. However, this is a very weak security goal, and usually not sufficient for practical applications.

Exercise 60. Show that the textbook RSA signature scheme is UUF-NMA secure, assuming that the RSA problem is hard.

4.2 RSA Full-Domain Hash and the Random Oracle Model

In this chapter we describe the RSA-based *Full Domain Hash* signature scheme (RSA-FDH), a very important scheme. It is an extension of the textbook RSA scheme, in which a hash function is first applied to the message to be signed, and then the hash of the message is signed. By demanding a suitable (but very strong) security requirement for the hash function, it can be shown that this eliminates the weaknesses incurred by the homomorphism of the textbook RSA scheme.

$\mathsf{Gen}(1^k)$**.** The key generation algorithm generates an RSA modulus $N = PQ$, where P and Q are two randomly chosen prime numbers. Furthermore, a number $e \in \mathbb{N}$ is chosen with $e \neq 1$ and $\gcd(e, \phi(N)) = 1$, and the value $d := e^{-1} \bmod \phi(N)$ is computed. Finally a hash function

$$H : \{0,1\}^* \to \mathbb{Z}_N$$

is selected. The public key is $pk := (N, e, H)$, and the secret key is $sk := d$.

$\mathsf{Sign}(sk, m)$**.** To sign a message $m \in \{0,1\}^*$, the signature $\sigma \in \mathbb{Z}_N$ is computed as

$$\sigma := H(m)^d \bmod N.$$

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

$$H(m) \equiv \sigma^e \bmod N$$

holds, and otherwise 0.

Correctness. Again the *correctness* of the scheme can be shown by insertion:

$$\sigma^e \equiv (H(m)^d)^e \equiv H(m)^{ed \bmod \phi(N)} \equiv H(m) \bmod N,$$

since $d \cdot e \equiv 1 \bmod \phi(N)$.

Soundness. To prove the EUF-CMA security of the RSA-FDH scheme, we have to make a suitable assumption about the security of the hash function H in addition to the RSA assumption. Obviously we have to assume at least that H is collision resistant, otherwise an attacker could first find a collision m, m^* with $H(m) = H(m^*)$ for H and then use it to win the EUF-CMA experiment.

Furthermore, H must not be the identity map (which is obviously collision resistant), because otherwise the resulting full-domain hash method would be identical to the textbook RSA scheme, and thus cannot be provably secure in the considered sense due to the homomorphy. So the hash function must somehow "destroy" this homomorphy, which is a security feature that is not easily definable.

4.2.1 The Random Oracle Model

Let $H : \mathcal{D} \to \mathcal{R}$ be a hash function.[1] The Random Oracle Model [BR93] assumes that this hash function is *ideal*. This means intuitively:

- An attacker cannot do anything with the hash function *except for evaluating it*, in particular it cannot take advantage of the code of the hash function to help in an attack.
- For any input $m \in \mathcal{D}$ the hash function returns a unique (that is, always the same on the same input) but random element $H(m) \in \mathcal{R}$.

To formalize this intuition, we model the function H as an "oracle", the *random oracle*, which works as follows.

From the adversary's perspective, the random oracle is a *black box*. To evaluate the hash function, i.e. to compute the hash value $H(m) \in \mathcal{R}$ for a value $m \in \mathcal{D}$, the attacker makes a request to the random oracle. The query consists of a value $m \in \mathcal{D}$. Each query is answered with an uniformly distributed random element $y = H(m) \in \mathcal{R}$. However, the answers of the random oracle are *consistent*: If the attacker requests the same value m more than once, then it receives the same hash value $y = H(m)$ as the answer every time.

To give a concrete example, one can imagine here that the random oracle internally generates a list

$$\mathcal{L} \subseteq \mathcal{D} \times \mathcal{R},$$

where each entry $(m, y) \in \mathcal{L}$ assigns its hash value $y \in \mathcal{R}$ to the value $m \in \mathcal{D}$.

- Before the attacker's first request, the list is empty.
- Each time the attacker requests a value m, the random oracle checks if there is an entry $(m, y) \in \mathcal{L}$ whose first element corresponds to the requested value m.
  - If yes, the random oracle returns the value y.
  - If not, then

[1] In the case of the RSA-FDH scheme we have $\mathcal{D} = \{0,1\}^*$ and $\mathcal{R} = \mathbb{Z}_N$.

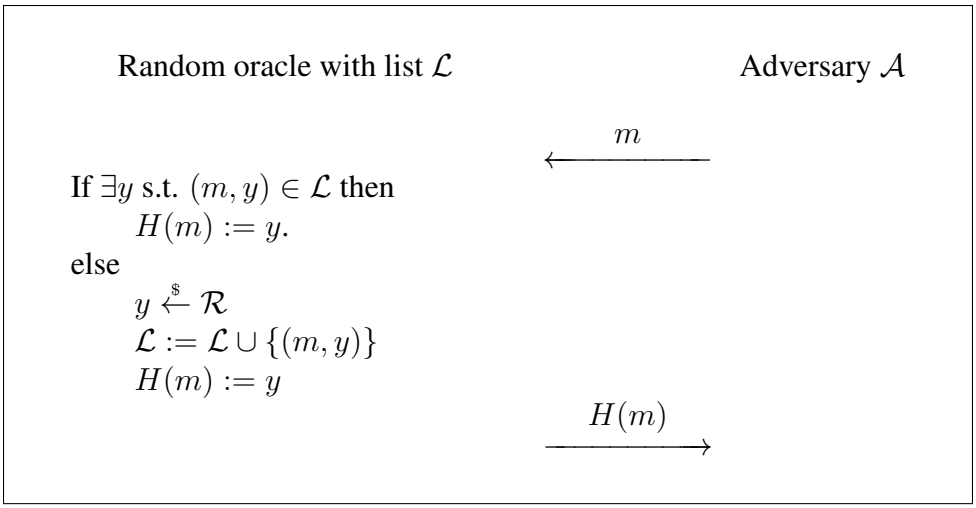

Figure 4.1: Interaction between adversary and random oracle. The attacker may send any number of requests m to the random oracle. At the beginning, i.e. before the attacker makes the first request, the list $\mathcal{L}$ is empty.

- the random oracle chooses a new value $y \xleftarrow{\$} \mathcal{R}$,
- adds the tuple (m, y) to the list $\mathcal{L}$,
- and then returns y.

Discussion of the random oracle model. There is a lot to say about the random oracle model, so let us discuss it with a sequence of remarks.

Remark 61. The random oracle model is an *idealization* of a concrete hash function that is often considered in security proofs. In practice, the cryptosystem would always be instantiated with a concrete hash function, such as SHA-3, for instance. The idealization is assumed *only for the security proof*.

Remark 62. If a scheme with a security proof in the random oracle model is used in practice, and the random oracle is instantiated with a concrete hash function, then one has to be aware that the security proof is not valid for the scheme anymore – because it could be that the concrete hash function does not approximate the random oracle sufficiently well. The heuristic that a hash function like SHA-3 is "as good as a random oracle" from the adversary's point of view is called the *random oracle heuristic*.

Remark 63. It is known [CGH98] that one can construct (somewhat "unrealistic") cryptosystems that are provably secure in the random oracle model, but completely insecure once the random oracle is instantiated *with any concrete hash function*. This of course raises strong concerns about the random oracle heuristic. However, these schemes are purposefully designed to be secure *only if* the hash function is a random oracle. There is no known realistic scheme with a similar property – a construction of such a scheme (if one exists) would be a very interesting research result.

Remark 64. The random oracle model can be seen as a tool to provide security proofs for cryptosystems for which we cannot otherwise find formal security arguments. It is generally much easier to find a security proof in the random oracle model than a proof in the standard model (i.e. without the random oracle heuristic or similar idealizations).

In most cases, cryptographic schemes with security proof in the random oracle model are much more efficient than schemes that have a standard model proof. The construction of such practical schemes is exactly the idea that Bellare and Rogaway seem to have had in mind when they described the random oracle approach. This is suggested by the title of their article [BR93] that introduced the random oracle model: *Random oracles are practical: a paradigm for designing efficient protocols*.

Having a security proof for a cryptosystem in the random oracle model is of course much better than having no security proof at all.

Remark 65. The random oracle model often also provides a first step for the construction of new cryptographic building blocks. For example, the first identity-based encryption scheme [BF01] or the first signatures over bilinear groups [BLS01, BLS04] (which we will consider in a later chapter) only had proofs in the random oracle model.

Such a first construction of a new building block, even if the security proof is initially only in the random oracle model, can help to learn how to construct this new building block *in principle*. A second step is then to look for a more sophisticated construction, possibly a variant of the first one, with a security proof in the standard model.

Remark 66. There also exist cryptographic building blocks that are known to exist *only* in the random oracle model. One example are encryption schemes that are provably secure against adaptive corruption of recipients, so-called *non-commiting encryption schemes* [Nie02].

4.2.2 Security Proof of RSA-FDH in the Random Oracle Model

Theorem 67. *Let* $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ *be the RSA-FDH signature scheme with hash function* H*. If* H *is modeled as a random oracle, then for each PPT adversary* $\mathcal{A}$ *that breaks the* EUF-CMA *security of* Σ *in time* $t_{\mathcal{A}}$ *with success probability* $\epsilon_{\mathcal{A}}$*, and thereby makes* q_H *requests to random oracle, there exists a PPT adversary* $\mathcal{B}$ *solving the RSA problem in time* $t_{\mathcal{B}} \approx t_{\mathcal{A}}$ *with success probability at least*

$$\epsilon_B \geq \frac{\epsilon_{\mathcal{A}} - 1/N}{q_H}.$$

Proof. The basis of the security proof is the observation that the adversary must ask the random oracle to compute a hash value. The random oracle is implemented by the challenger. The idea of proof is to embed a given RSA challenge in a clever way into the answers of the random oracle, such that the signature adversary can be used to solve the RSA problem.

A successful EUF-CMA adversary receives a *public key* (N, e) as input, is allowed to submit signature queries, and ends up outputting (m^*, σ^*) so that $\sigma^* = H(m^*)^{1/e}$. Again, we consider two events that may occur during the course of the experiment:

- We say that event E_0 occurs if the attacker $\mathcal{A}$ successfully outputs a valid signature $(m^*, \sigma^*) = (m^*, H(m^*)^{1/e})$ and $\mathcal{A}$ *has never asked* the random oracle for the hash value $H(m^*)$ during the experiment.
- We say that event E_1 occurs if the attacker outputs $\mathcal{A}$ $(m^*, \sigma^*) = (m^*, H(m^*)^{1/e})$ and $\mathcal{A}$ *has asked* the random oracle for the hash value $H(m^*)$ in the course of the experiment.

Every successful attacker causes either event E_0 or event E_1, so

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

This inequality can be transformed into

$$\Pr[E_1] \geq \epsilon_{\mathcal{A}} - \Pr[E_0]. \tag{4.1}$$

Adversaries that never request $H(m^*)$ from the random oracle. It is quite easy to see that such an adversary has only a very small chance of success. The random oracle selects every hash value randomly and uniformly distributed from $\mathbb{Z}_N$. In particular, the hash value $H(m^*)$ is uniformly distributed over $\mathbb{Z}_N$, and the attacker does not get any information about $H(m^*)$.

Since the map $h \mapsto h^{1/e} \bmod N$ is a bijection over $\mathbb{Z}_N$, the value $H(m^*)^{1/e}$ is equally distributed over $\mathbb{Z}_N$, too. The attacker can only guess this value, and thus his probability of success in this case (and thus the probability of event E_0) is at most

$$\Pr[E_0] \leq \frac{1}{N}. \tag{4.2}$$

Adversaries that request $H(m^*)$ from the random oracle. This is the more interesting part of the proof. We show that from an adversary $\mathcal{A}$ that causes event E_1 we can construct an adversary $\mathcal{B}$ that solves the RSA problem. So we consider an algorithm $\mathcal{B}$, which receives as input an RSA challenge (N, e, y). Algorithm $\mathcal{B}$ runs $\mathcal{A}$ as a subroutine by implementing the role of the challenger and the random oracle for $\mathcal{A}$.

$\mathcal{B}$ defines the public key as $pk := (N, e)$. It guesses an index $\nu \xleftarrow{\$} \{1, \ldots, q_H\}$ uniformly randomly, and starts $\mathcal{A}$ on input pk.

In the following, we assume that before every signature query for a message m_i, the attacker first queries the hash value $H(m_i)$ from the random oracle. (Alternatively, one can imagine that the challenger makes the corresponding random oracle query "on behalf of" $\mathcal{A}$ before it answers the signature query).

When $\mathcal{A}$ submits its j-th random oracle query with message $\tilde{m}_j$, $\mathcal{B}$ answers as follows:

- If $j \neq \nu$, then $\mathcal{B}$ chooses $x_j \xleftarrow{\$} \mathbb{Z}_N$ at random, computes $y_j := x_j^e \bmod N$, and returns y_j as the hash value $H(\tilde{m}_j) := y_j$ of $\tilde{m}_j$. The value x_j is stored by $\mathcal{B}$, in case $\mathcal{A}$ later requests a signature for message $\tilde{m}_j$. Since y_j is uniformly distributed, x_j is also uniformly distributed. The answer of the random oracle is therefore distributed exactly as in the real experiment.

- If $j = \nu$, then $\mathcal{B}$ defines $H(\tilde{m}_\nu) := y$, where y is from the RSA challenge (N, e, y). Since it is also random and uniformly distributed, the response of the random oracle is correctly distributed in this case as well.

Note that for each message $\tilde{m}_j$, $j \in \{1, \ldots, q\} \setminus \{\nu\}$, $\mathcal{B}$ is able to compute a signature by outputting x_j, because it holds that

$$x_j^e \equiv y_j \equiv H(\tilde{m}_j) \bmod N \quad \iff \quad x_j \equiv H(\tilde{m}_j)^{1/e} \bmod N.$$

$\mathcal{B}$ is only unable to compute a signature for the ν-th message $\tilde{m}_\nu$.

If event E_1 occurs, then $\mathcal{A}$ will eventually obtain a valid message-signature pair (m^*, σ^*) with $\sigma^* \equiv H(m^*)^{1/e} \bmod N$, such that $\mathcal{A}$ has requested the hash value $H(m^*)$, but no signature for m^*. $\mathcal{B}$ "hopes" now that m^* will be exactly the message asked in the ν-th query to H, so

$$m^* = \tilde{m}_\nu.$$

The probability that this happens is $1/q_H$, since the index ν was chosen uniformly randomly. If this is the case, then $\mathcal{B}$ can answer all signature queries for $\mathcal{A}$ correctly, and thus $\mathcal{A}$ will compute the e-th root of y for $\mathcal{B}$:

$$H(m^*)^{1/e} \equiv H(\tilde{m}_\nu)^{1/e} \equiv y^{1/e} \bmod N.$$

Algorithm $\mathcal{B}$ can thus solve the RSA problem with a probability of success of at least

$$\epsilon_{\mathcal{B}} \geq \Pr[E_1]/q_h. \tag{4.3}$$

By inserting the inequalities 4.2 and 4.3 in inequality 4.1 we obtain

$$\epsilon_B \cdot q_H \geq \Pr[E_1] \geq \epsilon_{\mathcal{A}} - \Pr[E_0] \geq \epsilon_{\mathcal{A}} - 1/N \iff \epsilon_B \geq \frac{\epsilon_{\mathcal{A}} - 1/N}{q_H}.$$

□

Remark 68. In the proof of Theorem 67 we have exploited the property of the random oracle that hash values are distributed randomly. We have "programmed" the random oracle so that with a good probability of $1/q_H$ we can answer all signature requests of the attacker ("simulation" of signatures) and at the same time "extract" a solution to the RSA problem from the forged signature. This "programming" proof technique is often used in security proofs in the random oracle model.

Remark 69. In the proof we made the assumption that the hash function H is a random oracle. This is a very strong assumption. An obvious question is now: *Is the random oracle model absolutely necessary to prove the security of RSA-FDH? Or is it possible to find a security proof for RSA-FDH in the standard model?* There are some results that say that it is not possible under certain conditions [DOP05, DHT12]. However, it is not impossible that there is a proof that cleverly circumvents these impossibility results. This is an important open problem.

Remark 70. In the proof the probability of success of the RSA adversary $\mathcal{B}$ was bounded by

$$\epsilon_{\mathcal{B}} \geq \frac{\epsilon_{\mathcal{A}} - 1/N}{q_H}.$$

Hence, the success probability of $\mathcal{B}$ decreases when q_H increases. The more hash queries the signature adversary $\mathcal{A}$ makes, the smaller the success probability of $\mathcal{B}$. It is possible that q_H becomes very large, since hash functions can usually be evaluated very efficiently.

Kakvi and Kiltz [KK12] presented an improved security proof, which is independent of q_H. However, it requires a stronger complexity assumption, the so-called *Phi-Hiding* assumption. The proof is very similar to the one presented here. It is strongly recommended to read this paper.

4.3 Gennaro-Halevi-Rabin Signatures

The RSA-FDH signature scheme presented in the previous chapter is very efficient. Unfortunately, however, it can only be proven to be EUF-CMA secure in the Random Oracle Model. A truly practical signature scheme whose provable security is based on the RSA assumption in the standard model is an important open problem.

One alternative option to build more practical signature schemes is to rely on the so-called *strong* RSA assumption, which is closely related to the classical RSA assumption. There are numerous signature schemes that can be proven secure based on strong RSA, such as for instance [GHR99, CS99, Fis03, Sch11]. In this chapter we describe the signature scheme of Gennaro, Halevi and Rabin (GHR) [GHR99]. This scheme is particularly suitable for this course due to its simplicity and the fact that it can be seen as a building block for the Hohenberger-Waters scheme that we will consider in Section 4.4.

4.3.1 The Strong RSA Assumption

Definition 71. Let $N := PQ$ be the product of two prime numbers and let $y \xleftarrow{\$} \mathbb{Z}_N$ be a random number modulo N. The *strong RSA problem* is: Given (N, y), compute $x \in \mathbb{Z}_N$ and $e \in \mathbb{N}$, $e > 1$, so that

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

The *strong RSA assumption* asserts that the strong RSA problem is hard.

The strong RSA problem is thus almost identical to the standard RSA problem (Definition 29), except that for the standard RSA problem the exponent e is prescribed, whereas in the strong RSA problem the exponent e may be chosen by the adversary.

Remark 72. The strong RSA assumption is a stronger assumption than the RSA assumption, so the name makes sense. In contrast, the strong RSA problem is potentially *easier* than the RSA problem, so the term is not really appropriate in this case. Nevertheless, it is widely used and commonly accepted, since it fits well to the corresponding assumption.

Exercise 73. Give a reduction showing that the strong RSA problem is easier to solve than the standard RSA problem.

4.3.2 GHR Signatures

In the following we take a hash function $h : \{0,1\}^* \to \mathbb{P}$, which maps bit strings (messages to be signed) to the set of prime numbers. For technical reasons we will further assume that these prime numbers are always larger than the RSA modulus N. We will sketch different ways to construct such a function h later.

Consider the following signature scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$:

$\mathsf{Gen}(1^k)$**.** The key generation algorithm generates an RSA modulus $N = PQ$, where P and Q are randomly chosen primes. Furthermore, a random number $s \xleftarrow{\$} \mathbb{Z}_N$ is selected.

The keys are $pk := (N, s, h)$ and $sk := \phi(N) = (P-1)(Q-1)$.

$\mathsf{Sign}(sk, m)$**.** To sign a message $m \in \{0,1\}^n$, first a prime $e_m = h(m)$ is derived from m, and then $d := 1/e_m = 1/h(m) \bmod \phi(N)$ is computed. The signature is

$$\sigma := s^d = s^{1/h(m)} \bmod N,$$

i.e., an $h(m)$-th root of s. Note that $1/h(m)$ only exists if $\gcd(h(m), \phi(N)) = 1$. However, this is guaranteed by the fact that h maps to prime numbers larger than N.

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

$$\sigma^{h(m)} \equiv s \bmod N.$$

holds. Otherwise 0 is output.

Theorem 74. *For each PPT adversary $\mathcal{A}$ that breaks the EUF-naCMA security of Σ in time $t_{\mathcal{A}}$ with success probability $\epsilon_{\mathcal{A}}$, there exists a PPT attacker $\mathcal{B}$ running in time $t_{\mathcal{B}} \approx t_{\mathcal{A}}$ and*

- *either breaks the collision resistance of h with success probability*

$$\epsilon_{\mathsf{coll}} \geq \epsilon_{\mathcal{A}}/2,$$

- *or solves the strong RSA problem with success probability*

$$\epsilon_{\mathsf{sRSA}} \geq \epsilon_{\mathcal{A}}/2.$$

Proof. Let $m_1, \ldots, m_q$ be the list of messages for which $\mathcal{A}$ has requested signatures in the EUF-naCMA experiment. Again we consider two events.

- Event E_0 occurs when $\mathcal{A}$ outputs a valid forgery (m^*, σ^*) and there exists an index i with $h(m^*) = h(m_i)$.
- Event E_1 occurs when $\mathcal{A}$ outputs a valid forgery (m^*, σ^*) and it holds that $h(m^*) \neq h(m_i)$ for all $i \in \{1, \ldots, q\}$.

Every successful adversary causes either event E_0 or event E_1, so again we have

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

and therefore either $\Pr[E_0] \geq \epsilon_A/2$ or $\Pr[E_1] \geq \epsilon_A/2$ or both must hold.

Event E_0. If event E_0 occurs, then we can break the collision resistance of the function h. This part of the proof is quite simple, and very similar to the proofs presented earlier, which argue with the collision resistance of a function. Therefore we omit it.

Event E_1. If event E_1 occurs, $\mathcal{B}$ can solve the strong RSA problem as follows. $\mathcal{B}$ receives as input an instance (N, y) and must output (x, e) so that $e > 1$ and $x^e \equiv y \bmod N$.

At the beginning of the experiment $\mathcal{B}$ receives a list of messages $m_1, \ldots, m_q$ from adversary $\mathcal{A}$. Then $\mathcal{B}$ computes the value $s \in \mathbb{Z}_N$ as

$$s := y^{\prod_{i \in \{1,\ldots,q\}} h(m_i)} \bmod N.$$

The public key is $pk = (N, s)$. Note that it is correctly distributed.

$\mathcal{B}$ simulates the j-th signature for message m_j by computing

$$\sigma_j := y^{\prod_{i \in \{1,\ldots,q\} \setminus \{j\}} h(m_i)} \bmod N.$$

Note that the value σ_j is a valid signature for message m_j, because the verification equation is fulfilled:

$$\sigma_j^{h(m_j)} \equiv \left(y^{\prod_{i \in \{1,\ldots,q\} \setminus \{j\}} h(m_i)}\right)^{h(m_j)} \equiv y^{\prod_{i \in \{1,\ldots,q\}} h(m_i)} \equiv s \bmod N.$$

With probability $\Pr[E_1]$, $\mathcal{A}$ outputs a forgery (m^*, σ^*) that satisfies

- $h(m^*) \neq h(m_i)$ for all $i \in \{1, \ldots, q\}$, and
- $(\sigma^*)^{h(m^*)} \equiv s \bmod N$.

Therefore $\mathcal{B}$ obtains a solution σ^* of the equation

$$(\sigma^*)^{h(m^*)} \equiv s \equiv y^{\prod_{i \in \{1,\ldots,q\}} h(m_i)} \bmod N, \tag{4.4}$$

where $\gcd(h(m^*), \prod_{i \in \{1,\ldots,q\}} h(m_i)) = 1$, because the function h maps to prime numbers and we have $h(m^*) \neq h(m_i)$ for all $i \in \{1, \ldots, q\}$.

To extract a solution for the strong RSA problem, $\mathcal{B}$ applies Shamir's trick (Lemma 31) to compute the value x that satisfies $x^{h(m^*)} \equiv y \bmod N$ from Equation (4.4), and outputs $(x, h(m^*))$. □

Exercise 75. Explain why the simulated public key $pk = (N, s)$ is indeed correctly distributed.

EUF-CMA-secure signatures based on the strong RSA assumption. We have proven that the GHR scheme is EUF-naCMA secure. However, our ultimate goal is to achieve EUF-CMA security. We can apply the transformation from Section 2.4 to the GHR scheme and the RSA-based one-time signature from Section 2.3.2 to get an EUF-CMA-secure scheme. Alternatively, we can use the RSA-based chameleon hash function from Section 3.3.2 and the transformation from Section 3.6 to even get an sEUF-CMA-secure scheme. This shows how useful these generic transformations are. Since the RSA assumption is implied by the strong RSA assumption, we get EUF-CMA- and sEUF-CMA-secure schemes that are based on the strong RSA assumption.

4.3.3 Hash Functions Mapping to Primes

There are many different ways of constructing such functions. We will show two examples, and we will consider another example in Section 4.4.

Heuristic construction. Let $H : \{0,1\}^* \to \{0,1\}^\ell$ be a collision resistant hash function. One possible approach to construct the function h required above from this hash function H is to define the function h as

$$h(m) := H(m||\gamma),$$

where $\gamma \in \mathbb{N}$ is the smallest natural number so that the value $H(m||\gamma)$, interpreted as a natural number in the interval $[0, 2^\ell - 1]$ in the canonical way, is prime. During evaluation of h, γ is simply incremented until $H(m||\gamma)$ is prime.

If the hash values $H(m||\gamma)$ are (fairly) evenly distributed over the interval $[0, 2^\ell - 1]$, then according to the Prime Number Theorem one can expect to find a γ so that $H(m||\gamma)$ is prime after about $\log 2^\ell = \ell$ attempts. Thus, the evaluation of h is reasonably efficient (in the asymptotic sense), if the evaluation of H is efficient. This construction is of course heuristic and depends strongly on the properties of the function H.

Exercise 76. Construct a hash function $H : \{0,1\}^* \to [0, 2^\ell - 1]$ which is collision resistant, but for which the value $H(m||\gamma) \in [0, 2^\ell - 1]$ is never prime. Use a collision resistant hash function $H' : \{0,1\}^* \to B$ with suitable range B as a building block.

Construction based on a chameleon hash function. Let $N' \in \mathbb{N}$ be a natural number and let

$$\mathsf{ch} : \{0.1\}^n \times \mathbb{Z}_{N'} \to [0, N' - 1]$$

be the RSA-based chameleon hash function from Section 3.3.2. This chameleon hash function has two helpful properties:

- it maps to *natural numbers* in the interval $[0, N' - 1]$, and
- for any message m and an uniformly distributed random random value $r \xleftarrow{\$} \mathbb{Z}_{N'}$ the hash value $\mathsf{ch}(m, r)$ is *uniformly distributed* over $[0, N' - 1]$.

Based on this chameleon hash function, a (randomized) hash function h, which maps to primes, can be constructed. To compute the hash value $h(m)$ of a message m, proceed as follows:

1. Choose random values $(m', r') \xleftarrow{\$} \{0,1\}^n \times \mathbb{Z}_{N'}$ until $p = \mathsf{ch}(m', r') \in [0, N' - 1]$ is prime. Again, this can be accomplished relatively quickly, in about $\log N'$ expected attempts according to the Prime Number Theorem.
2. Compute $r = \mathsf{TrapColl}(\tau, m', r', m)$ to obtain a randomness r so that $\mathsf{ch}(m, r) = p$.

The randomness r is then sent along with the signature. The collision resistance of this function h follows from the collision resistance of the chameleon hash function. Another advantage is that the chameleon property of h can now be used to directly prove sEUF-CMA security of the resulting scheme.

4.4 Hohenberger-Waters Signatures

A provably EUF-CMA-secure signature scheme based on the hardness of the RSA problem in the standard model, that is, without idealizations such as the random oracle model, had been an open problem for a very long time. In 2009 Hohenberger and Waters [HW09b] presented the first such scheme.

The construction of Hohenberger and Waters can also be seen as a clever extension of the GHR scheme from Section 4.3. It is based on three observations, which we describe in the following sections:

- It can be shown that GHR signatures are secure under the standard RSA assumption when considering a weaker security target, namely *selective security* (SUF-naCMA).
- There is a *generic transformation* with which can be used to construct a EUF-naCMA-secure scheme from a SUF-naCMA-secure scheme.
- This transformation can be applied particularly efficiently to the GHR scheme.

4.4.1 Selective Security of GHR Signatures

Let us first have another look at GHR signatures. Why exactly is the *strong* RSA assumption necessary? More precisely, which particular part of the proof requires the freedom to choose an arbitrary exponent e, such that the standard RSA assumption is not sufficient?

As always when we want to prove the security of a signature scheme, we have to pay attention to two things:

Simulation. We (resp. the algorithm $\mathcal{B}$, which we construct in the proof of the Theorem 74) must be able to *simulate* a valid pk and valid signatures for the messages in the proof.

Here we could take advantage of the fact that the adversary in the EUF-naCMA experiment must output the messages for which it wants to see signatures at the beginning of the experiment – *before* it sees the public key pk. Therefore, we could set up the value $s \in \mathbb{Z}_N$ contained in pk in such a way that we could compute the required $h(m_i)^{\text{th}}$ roots of s:

$$s := y^{\prod_i h(m_i)} \bmod N.$$

Note that this step would work the same way if we used the standard RSA assumption instead of strong RSA. So this does not seem to be the problem.

Extraction. If the attacker gives us a forgery (m^*, σ^*) with

$$s \equiv (\sigma^*)^{h(m^*)} \bmod N$$

then (unless $\mathcal{A}$ found a collision for h) we could use Shamir's trick to compute a *new* $h(m^*)^{\text{th}}$ root of y, and thus solve the Strong RSA problem.

Unfortunately we (i.e. the simulator in the proof) had no influence on the value $h(m^*)$, because m^* is chosen by the attacker in the EUF-naCMA experiment. However, to solve the strong RSA problem it is sufficient to compute *any* non-trivial root of y – we do not need to know anything about $h(m^*)$ except that $h(m^*) \neq h(m_i)$ must hold for all $i \in \{1, \ldots, q\}$. With the RSA problem it is different, because we have to compute a *very specific* e-th root. Namely for exactly the value e from the given instance (N, e, y) of the RSA problem.

So this is the problem. The adversary in the EUF-naCMA experiment wins if it outputs a valid forgery (m^*, σ^*) for a *any new message* m^*. Therefore, it is difficult for us to get the adversary to compute the "right" e^{th} root for us.

Selective Security. Can we give a security proof for GHR signatures under the standard RSA assumption if we force the attacker in the experiment to output a forgery for a *specific* message m^*?

We weaken the EUF-naCMA experiment a bit and consider *selective unforgeability under non-adaptive chosen-message attack* (SUF-naCMA), which is defined as follows (see also Figure 4.2)

1. The adversary initially commits to the message m^* for which it will forge a signature.
2. Then the adversary outputs q messages $(m_1, \ldots, m_q)$ so that $m_i \neq m^*$ for all $i \in \{1, \ldots, q\}$. These are the messages for which $\mathcal{A}$ requests a signature.
3. The challenger $\mathcal{C}$ generates a key pair $(pk, sk) \xleftarrow{\$} \mathsf{Gen}(1^k)$ and computes a signature $\sigma_i := \mathsf{Sign}(sk, m_i)$ for each $i \in \{1, \ldots, q\}$. The attacker gets pk and the signatures $(\sigma_1, \ldots, \sigma_q)$.
4. 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 77. We say that $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ is SUF-naCMA secure, if for all PPT adversaries $\mathcal{A}$ in the SUF-naCMA experiment it holds that

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

for a negligible function negl in the security parameter.

In the following we show that the SUF-naCMA security of GHR signatures can be proven under the RSA assumption. The trick is to define the function h appropriately.

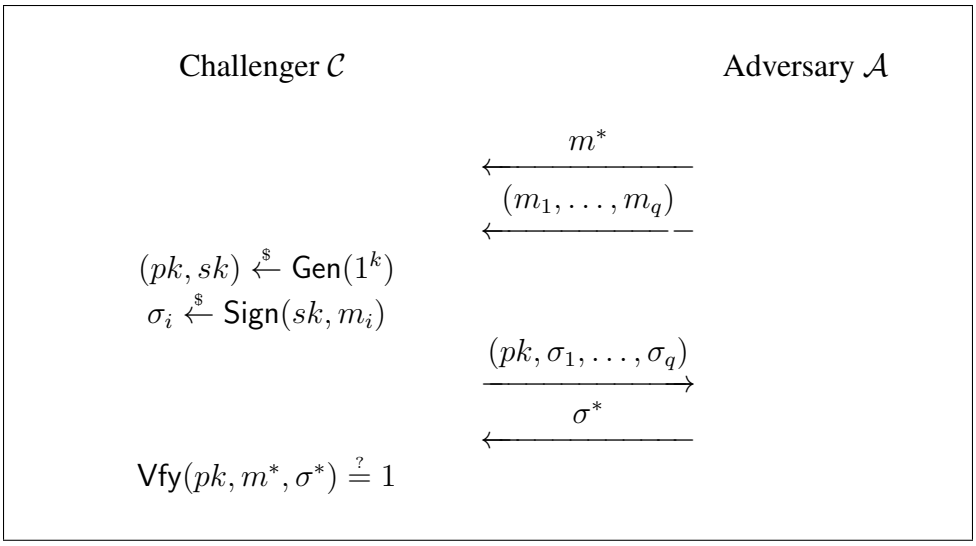

Figure 4.2: The SUF-naCMA Security Experiment.

Clever definition of the function h. Let

$$\mathsf{PRF} : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^\ell$$

be a pseudorandom function (cf. Definition 37) with key space $\{0,1\}^k$, which maps bit strings of arbitrary length to ℓ-bit strings. We define the function h, which maps bit strings to prime numbers, as follows.

- To generate the function description, a key $\kappa \xleftarrow{\$} \{0,1\}^k$ (the seed) is chosen for the PRF, as well as a random bit string $\alpha \xleftarrow{\$} \{0,1\}^\ell$.
- The function h is defined by the function PRF and the values (κ, α). Concretely, we define

  $$h : \{0,1\}^n \to \mathbb{P}_\ell,$$

  which maps n-bit strings to ℓ-bit primes, as

  $$h(m) := \mathsf{PRF}(\kappa, m||\gamma) \oplus \alpha,$$

  where $\gamma \in \mathbb{N}$ is the smallest non-negative integer such that the value $\mathsf{PRF}(\kappa, m||\gamma) \oplus \alpha \in \{0,1\}^\ell$, interpreted as a natural number in the interval $[0, 2^\ell - 1]$ in the canonical way, is a prime number. Note the similarity to the heuristic construction from Section 4.3.3.

During the evaluation of h, γ is incremented until $\mathsf{PRF}(\kappa, m||\gamma) \oplus \alpha$ is prime. According to the Prime Number Theorem this is expected to happen quite fast: one expects about $\log 2^\ell = \ell$ attempts.

GHR signatures with this special function h. If this function h is used in the GHR signature scheme, then we obtain the following signature scheme.

$\mathsf{Gen}(1^k)$**.** The key generation algorithm generates an RSA modulus $N = PQ$, where P and Q are two randomly chosen primes. Furthermore, $s \xleftarrow{\$} \mathbb{Z}_N$ is selected.

To describe the function h, $\kappa \xleftarrow{\$} \{0,1\}^k$ and $\alpha \xleftarrow{\$} \{0,1\}^\ell$ are additionally chosen.

The keys are $pk := (N, s, (\mathsf{PRF}, \kappa, \alpha))$ and $sk := \phi(N) = (P-1)(Q-1)$.

$\mathsf{Sign}(sk, m)$**.** To sign a message $m \in \{0,1\}^n$, $d := 1/h(m) \bmod \phi(N)$ is computed. The signature is

$$\sigma := s^d = s^{1/h(m)} \bmod N,$$

an $h(m)^{\text{th}}$ root of s.

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

$$\sigma^{h(m)} \equiv s \bmod N$$

holds. Otherwise 0 is output.

This scheme can be proven to be SUF-naCMA secure under the RSA assumption. However, one has to restrict to instances of the RSA problem (N, e, y), where e is a suitably distributed prime number.

Proof idea. Let $\mathcal{B}$ be an algorithm that uses an SUF-naCMA adversary $\mathcal{A}$ on the above GHR variant to solve the RSA problem. $\mathcal{B}$ receives as input an instance (N, e, y) of the RSA problem, where e is a *random* ℓ-bit prime number.

The basic idea of the security proof is that $\mathcal{B}$ "programs" the function h cleverly so that $h(m^*) = e$. If this succeeds, then $\mathcal{B}$ can use the same strategy for simulation and extraction as in the strong RSA-based security proof (Theorem 74). Since it is necessary for the attacker $\mathcal{A}$ to forge a signature for message m^* in the SUF-naCMA experiment, and $h(m^*) = e$ holds, $\mathcal{A}$ will compute the "right" e^{th} root of y for $\mathcal{B}$.

To this end, $\mathcal{B}$ defines the function h by first choosing a random seed $\kappa \xleftarrow{\$} \{0,1\}^k$ and a random value $\gamma \xleftarrow{\$} \{1, \ldots, \ell\}$. The value $\alpha \in \{0,1\}^\ell$ is defined as

$$\alpha := \mathsf{PRF}(k, m^* || \gamma) \oplus e.$$

Now, assuming that γ is the smallest non-zero integer so that $\mathsf{PRF}(k, m^*) \oplus \alpha$ is prime, we have

$$h(m^*) = \mathsf{PRF}(k, m^* || \gamma) \oplus \alpha = e.$$

Thus, by choosing α appropriately, $\mathcal{B}$ is able to "program" the function h so that $h(m^*) = e$ holds. Note the similarity of this "programming" to the security proof of RSA Full-Domain Hash signatures (Section 4.2).

4.4.2 From SUF-naCMA-Security to EUF-naCMA-Security

Our goal is the construction of an EUF-naCMA secure signature scheme based on the RSA assumption. So far we have an RSA-based scheme, which unfortunately is only SUF-naCMA-secure. In this chapter we describe a generic transformation, which allows to construct a EUF-naCMA-secure scheme from a SUF-naCMA-secure scheme. This transformation was explicitly described in [BK10], however the idea is implicitly contained in [HW09b].

Prefixes. Before we specify the transformation, we need to define *prefixes* of bit strings and give a technical lemma which we will use in the proof.

Definition 78. Let $m \in \{0,1\}^n$ be a bit string. We write $m_{|w}$, and say that $m_{|w}$ is the w-th prefix of m, if

- $|m_{|w}| = w$ holds, so the length of $m_{|w}$ is exactly w, and
- $m_{|w}$ is identical to the first w bits of m.

Example 79. Let $n = 4$ and $m = 1011 \in \{0,1\}^4$. Then $m_{|1} = 1$, $m_{|2} = 10$, $m_{|3} = 101$, and $m_{|4} = 1011$.

The proof of Theorem 82 will use the following lemma.

Lemma 80. *Let $m_1, \ldots, m_q$ be a list of q arbitrary messages, where $m_i \in \{0,1\}^n$ for all $i \in \{1,\ldots,q\}$. Let $m^* \in \{0,1\}^n$ such that $m^* \neq m_i$ for all $i \in \{1,\ldots,q\}$.*

1. *There exists a* shortest *prefix $m^*_{|w}$ of m^* which is* no *prefix of any message m_i from the list $m_1, \ldots, m_q$.*
2. *Even if we are only given $(m_1, \ldots, m_q)$, but not m^*, we can guess this prefix correctly with a probability of success of at least $1/(qn)$.*

Exercise 81. Prove Lemma 80.

The Transformation. Let $\widetilde{\Sigma} = (\widetilde{\mathsf{Gen}}, \widetilde{\mathsf{Sign}}, \widetilde{\mathsf{Vfy}})$ be a signature scheme. We construct a new signature scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$. We will prove below that SUF-naCMA security of $\widetilde{\Sigma}$ implies EUF-naCMA security of Σ.

$\mathsf{Gen}(1^k)$**.** The key generation algorithm generates a key pair $(pk, sk) \xleftarrow{\$} \widetilde{\mathsf{Gen}}(1^k)$.

$\mathsf{Sign}(sk, m)$**.** To sign a message $m \in \{0,1\}^n$, all prefixes of m are signed with $\widetilde{\mathsf{Sign}}$:

$$\tilde{\sigma}^{(w)} = \widetilde{\mathsf{Sign}}(sk, m_{|w}) \qquad \forall w \in \{1,\ldots,n\}.$$

The signature for m consists of $\sigma := (\tilde{\sigma}^{(1)}, \ldots, \tilde{\sigma}^{(n)})$.

$\mathsf{Vfy}(pk, m, \sigma)$**.** The verification algorithm, on input $\sigma := (\tilde{\sigma}^{(1)}, \ldots, \tilde{\sigma}^{(n)})$, checks if $\tilde{\sigma}^{(w)}$ is a valid signature for $m_{|w}$ for all $w \in \{1, \ldots, n\}$:

$$\widetilde{\mathsf{Vfy}}(pk, m_{|w}, \tilde{\sigma}^{(w)}) \stackrel{?}{=} 1 \qquad \forall w \in \{1, \ldots, n\}.$$

So the basic idea of the transformation is very simple: Not only the message $m = m_{|n}$ is signed, but also all prefixes of m.

We will now show that scheme Σ is EUF-naCMA-secure if the underlying scheme $\widetilde{\Sigma}$ is SUF-naCMA-secure.

Theorem 82. *For each PPT adversary $\mathcal{A}$ breaking the EUF-naCMA security of Σ in time $t_{\mathcal{A}}$ with success probability $\epsilon_{\mathcal{A}}$, there exists a PPT adversary $\mathcal{B}$ breaking the SUF-naCMA security of $\widetilde{\Sigma}$ in time $t_{\mathcal{B}}$ with success probability*

$$\epsilon_{\mathcal{B}} \geq \frac{\epsilon_{\mathcal{A}}}{qn},$$

where n is the message length and q is the number of chosen message signature queries of $\mathcal{A}$.

Proof of Theorem 82. $\mathcal{B}$ starts $\mathcal{A}$ and receives a list of messages $(m_1, \ldots, m_q)$. Based on this list, $\mathcal{B}$ guesses the shortest prefix $m^*_{|w}$ of m^*, so that $m^*_{|w}$ is not a prefix of any message m_i, $i \in \{1, \ldots, q\}$, and commits to forging a signature for the message $\tilde{m}^* := m^*_{|w}$.

Now $\mathcal{B}$ asks for signatures for all prefixes of the messages in $(m_1, \ldots, m_q)$. For this purpose $\mathcal{B}$ defines a new list of messages $(\tilde{m}_1, \ldots, \tilde{m}_{qn})$ as

$$\begin{aligned}
(\tilde{m}_1, \ldots, \tilde{m}_n) &:= (m_{1|1}, \ldots, m_{1|n}) \\
(\tilde{m}_{n+1}, \ldots, \tilde{m}_{2n}) &:= (m_{2|1}, \ldots, m_{2|n}) \\
&\vdots \\
(\tilde{m}_{(q-1)n+1}, \ldots, \tilde{m}_{qn}) &:= (m_{q|1}, \ldots, m_{q|n}).
\end{aligned}$$

The list $(\tilde{m}_1, \ldots, \tilde{m}_{qn})$ thus consists of all possible prefixes of all messages in the list $(m_1, \ldots, m_q)$. Then $\mathcal{B}$ asks the challenger for signatures for all messages in $(\tilde{m}_1, \ldots, \tilde{m}_{qn})$.

From the challenger $\mathcal{B}$ receives the public key pk and a list of signatures $(\tilde{\sigma}_1, \ldots, \tilde{\sigma}_{qn})$. Now $\mathcal{B}$ defines

$$\begin{aligned}
\sigma_1 &:= (\tilde{\sigma}_1, \ldots, \tilde{\sigma}_n) \\
\sigma_2 &:= (\tilde{\sigma}_{n+1}, \ldots, \tilde{\sigma}_{2n}) \\
&\vdots \\
\sigma_q &:= (\tilde{\sigma}_{(q-1)n+1}, \ldots, \tilde{\sigma}_{qn}).
\end{aligned}$$

Thus a signature σ_i consists of a list of signatures over all prefixes of the message m_i, and is therefore a valid signature for message m_i with respect to the signature scheme Σ. The pk and the list $(\sigma_1, \ldots, \sigma_q)$ are forwarded by $\mathcal{B}$ to $\mathcal{A}$. Note that this provides a proper simulation of signatures.

With probability $\epsilon_{\mathcal{A}}$ the adversary $\mathcal{A}$ will produce a signature forgery (m^*, σ^*), where $\sigma^* = (\tilde{\sigma}_1^*, \ldots, \tilde{\sigma}_n^*)$ contains signatures over each prefix of m^*. In particular σ^* contains a signature over the shortest prefix which is not a prefix of any message m_i.

Now if $\mathcal{B}$ guessed correctly, $m_{|w}^*$ is the shortest prefix of m^*, which is not a prefix of any message m_i, $i \in \{1, \ldots, q\}$. In this case, $\mathcal{B}$ can output the signature $\tilde{\sigma}_w^*$ as a valid forgery for $m_{|w}^*$. According to Lemma 80, $\mathcal{B}$ guesses correctly with probability at least $1/(qn)$. Thus the success probability of $\mathcal{B}$ is at least

$$\epsilon_{\mathcal{B}} \geq \frac{\epsilon_{\mathcal{A}}}{qn}.$$

□

The transformation presented in this chapter is very nice because it is *generic*. This means that it works with any SUF-naCMA-secure signature scheme. It opens up a new way to construct EUF-CMA-secure signature schemes:

$$\text{SUF-naCMA} \overset{\text{This section}}{\Longrightarrow} \text{EUF-naCMA} \overset{\text{Section 2.4}}{\Longrightarrow} \text{EUF-CMA}.$$

But there is still a major disadvantage, since the signatures become rather large. To sign an n-bit message $m \in \{0,1\}^n$, each prefix of the message m must be signed individually. Therefore the size of the signatures increases by the factor n. Fortunately (and very interestingly), in the case of GHR signatures, it is possible to avoid this factor n. This will be explained in the next section.

4.4.3 Clever "Compression" of GHR Signatures

If one naively applies the transformation from the previous chapter to GHR signatures, then the signatures become quite large. Let $pk = (N, s, h)$ be a GHR public key. A signature over an n-bit message $m \in \{0,1\}^n$ consists of n values:

$$\sigma' = (s^{1/h(m_{|1})}, \ldots, s^{1/h(m_{|n})}) \in \mathbb{Z}_N^n.$$

In the case of GHR signatures, however, the transformation can be applied much more efficiently. We simply compute the signature as

$$\sigma = s^{\prod_{j=1}^{n} 1/h(m_{|j})} \mathbb{Z}_N$$

So the n values of the signature σ' can be "compressed" into a single value $\sigma \in \mathbb{Z}_N$.

The Hohenberger-Waters Scheme. If we apply this optimized transformation to GHR signatures, then we obtain the signature scheme of Hohenberger and Waters [HW09b].

$\mathsf{Gen}(1^k)$**.** The key generation algorithm generates an RSA modulus $N = PQ$, where P and Q are two randomly chosen primes. In addition, $s \xleftarrow{\$} \mathbb{Z}_N$ is selected, as well as $\kappa \xleftarrow{\$} \{0,1\}^k$ and $\alpha \xleftarrow{\$} \{0,1\}^\ell$ to define the function h as above.

The keys are $pk := (N, s, (\mathsf{PRF}, \kappa, \alpha))$ and $sk := \phi(N) = (P-1)(Q-1)$.

$\mathsf{Sign}(sk, m)$**.** To sign a message $m \in \{0,1\}^n$

$$d := \frac{1}{\prod_{i=1}^{n} h(m_{|i})} \bmod \phi(N)$$

is computed. The signature is

$$\sigma := s^d \bmod N.$$

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

$$\sigma^{\prod_{i=1}^{n} h(m_{|i})} \equiv s \bmod N.$$

holds. Otherwise 0 is output.

It can be shown that this scheme is EUF-naCMA secure. In combination with an RSA-based one-time signature or a chameleon hash function, this yields the first RSA-based signature scheme whose EUF-CMA security is based solely on the difficulty of the RSA problem.

4.5 Open Problems

The Hohenberger-Waters scheme is the first RSA-based signature scheme with EUF-naCMA security proof in the standard model, and thus an important milestone. However, the search for RSA-based signature schemes is far from over. The Hohenberger-Waters scheme is unfortunately rather impractical, since it requires many prime numbers to be found, not only during key generation but also during signature generation and verification.[2] If you have ever generated an RSA key with PGP/GnuPG or searched for large prime numbers with a computer algebra program such as MAGMA, then you know how time-consuming this is. If not, you should try it yourself.

More efficient RSA-based signatures based on the techniques of [HW09b] can be found for example in [HJK11, BHJ+13]. However, even these variants are not yet efficient enough to be used in practice. A really *practical* RSA-based signature scheme with an EUF-CMA security proof in the standard model is a foundational open problem in cryptography.

[2] For comparison, the much more efficient RSA-FDH procedure from Section 4.2 requires two large prime numbers to be found only during key generation.

Chapter 5

Pairing-based Signature Schemes

Signatures based on so-called *pairings* are a second major class of signature schemes, besides RSA-based signatures. While pairings were initially used in cryptanalysis, especially to solve the discrete logarithm problem efficiently in certain groups [MVO91], it has turned out that they can also be very helpful for the construction of provably secure cryptosystems. In fact, pairings enable a variety of interesting cryptographic primitives beyond signatures, such as identity-based encryption [Sha84, BF01, Wat05] or the non-interactive proof systems of Groth and Sahai [GS08].

In this chapter we first give a short introduction to pairings and show the 3-party key exchange protocol of Joux [Jou04] as a first application, in order to get used to the concept. Then we describe the pairing-based signature scheme of Boneh, Lynn and Shacham [BLS01, BLS04]. This very simple scheme has many nice features, such as short signatures, aggregatability, batch verification and provable EUF-CMA security in the ROM. Finally we describe the signature scheme of Waters [Wat05]. We will introduce the concept of *programmable hash functions* (PHFs) [HK08], which are a very useful abstraction for the construction of provably secure signatures and identity-based encryption schemes, for example. Especially the EUF-CMA security proof of the Waters signature scheme is greatly simplified by this abstraction.

5.1 Pairings

Definition 83. Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be cyclic groups of order p. A *pairing* is a map

$$e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$$

with the following three properties.

1. **Bilinearity.** For all $g_1, g_1' \in \mathbb{G}_1$ and $g_2, g_2' \in \mathbb{G}_2$ holds that

$$e(g_1 g_1', g_2) = e(g_1, g_2) \cdot e(g_1', g_2) \qquad \text{and} \qquad e(g_1, g_2 g_2') = e(g_1, g_2) \cdot e(g_1, g_2').$$

2. **Non-degeneracy.** For generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, we have that $e(g_1, g_2)$ is a generator of $\mathbb{G}_T$. If the groups have prime order, then this is equivalent to

$$e(g_1, g_2) \neq 1.$$

3. **Efficient computability.** The function e can be evaluated efficiently.

Exercise 84. Ler $a \in \mathbb{Z}_p$. Show that the following equation follows from the bilinearity of e:

$$e(g_1^a, g_2) = e(g_1, g_2)^a = e(g_1, g_2^a)$$

The groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are usually elliptic curve groups. We also call these groups the *source groups* of the pairing. The group $\mathbb{G}_T$ is usually a subgroup of a finite field $\mathbb{F}_Q$, in the following also called the *target group*. We will not go further into the (highly non-trivial) concrete construction of pairings here, but will only use it as an abstract building block for the construction of digital signature schemes.

Remark 85. The original application of pairings in cryptography was in cryptanalysis. Menezes, Okamoto and Vanstone [MVO91] showed that a map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ can help solve the discrete logarithm problem in $\mathbb{G}$ – namely, when the discrete logarithm problem is simpler in $\mathbb{G}_T$ than in $\mathbb{G}$. Fortunately, there are also pairings for which it seems reasonable to assume that the discrete logarithm problem in $\mathbb{G}_T$ is at least as difficult as in $\mathbb{G}$. These pairings can be used for cryptographic applications. See for example [CF05] for details.

Types of Pairings. There are different types of pairings, Galbraith *et al.* [GPS08] define three categories.

- **Type-1 pairings.** These are pairings where $\mathbb{G}_1$ and $\mathbb{G}_2$ are the same group, i.e. $\mathbb{G}_1 = \mathbb{G}_2$. This class of pairings is also called *symmetric pairings*.

  Since it is not necessary to distinguish between $\mathbb{G}_1$ and $\mathbb{G}_2$ in this case, so we sometimes write $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$.

- **Type-2 pairings.** These are *asymmetric* pairings, so we have $\mathbb{G}_1 \neq \mathbb{G}_2$. However, there exists an efficiently computable, non-trivial homomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$.

- **Type-3 pairings.** These are asymmetric pairings, i.e. $\mathbb{G}_1 \neq \mathbb{G}_2$, but no efficiently computable, non-trivial homomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ is known.

The properties of these pairings sometimes differ considerably, e.g., with regard to the size of the representation of group elements, or whether hash functions are known whose image set corresponds to one of the groups. We refer to [GPS08] for an overview.

In the following, we will mainly consider type-1 pairings for the sake of simplicity. However, many of the constructions presented also work with type-2 and type-3 pairings.

Joux's 3-Party Diffie-Hellman Protocol. Due to its simplicity, the 3-party Diffie-Hellman protocol of Joux [Jou04] is a nice example to illustrate the usefulness of pairings. This protocol allows three parties A, B and C to compute a shared key in just one round of communication.

Let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be a symmetric pairing and g a generator of group $\mathbb{G}$ with order p.

- Party A chooses a secret, random value $a \xleftarrow{\$} \mathbb{Z}_p$. B chooses $b \xleftarrow{\$} \mathbb{Z}_p$, and C chooses $c \xleftarrow{\$} \mathbb{Z}_p$.

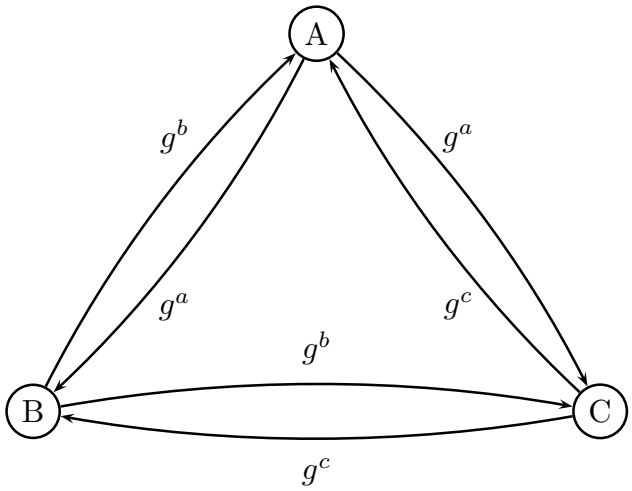

Figure 5.1: Message exchange in Joux's 3-party Diffie-Hellman protocol. The shared key k of parties A, B and C is computed as $k = e(g, g,)^{abc}$.

- Party A sends the value $g^a \in \mathbb{G}$ to B and C. In parallel, B sends the value g^b to A and C, and C sends the value g^c to A and B. Cf. Figure 5.1.
- Party A compute the common key as $k = e(g^b, g^c)^a$. B computes $k = e(g^a, g^c)^b$, and C computes $k = e(g^a, g^b)^c$. The *correctness* of this protocol follows from the bilinearity of e. Since
$$e(g^b, g^c)^a = e(g^a, g^c)^b = e(g^a, g^b)^c = e(g, g)^{abc},$$
all parties compute the same key k.

5.2 Boneh-Lynn-Shacham Signatures

The signature scheme of Boneh, Lynn and Shacham [BLS01, BLS04] ("BLS signatures") is the simplest pairing-based signature scheme. It offers very short signatures, is very efficient, and provable EUF-CMA-secure (in the Random Oracle Model, see Section 4.2.1). Due to its *aggregability* and *batch verification* features, it furthermore has additional functional properties that make it interesting for application scenarios where the bandwidth of the transmission channel or the computational resources of the verifier are limited.

Let $\mathbb{G}$ and $\mathbb{G}_T$ be two groups of prime order p and g a generator of $\mathbb{G}$. Let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be a type-1 pairing. Let $H : \{0,1\}^* \to \mathbb{G}$ be a hash function that maps bit strings to group elements.

Gen(1^k). The key generation algorithm selects $x \xleftarrow{\$} \mathbb{Z}_p$ and computes $g^x \in \mathbb{G}$. The public key is $pk := (g, g^x)$, the secret key is $sk := x$.

Sign(sk, m). To sign a message $m \in \{0,1\}^*$, the signature $\sigma \in \mathbb{G}$ is computed as

$$\sigma := H(m)^x \in \mathbb{G}.$$

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

$$e(H(m), g^x) = e(\sigma, g)$$

holds, and otherwise 0.

Correctness. The *Correctness* follows again from the bilinearity of pairing, because it holds that

$$e(H(m), g^x) = e(H(m)^x, g) = e(\sigma, g)$$

Remark 86. A remarkable feature of the BLS scheme is that a signature consists of only one group element. If the group is chosen appropriately, then very short signatures are possible. There exists also a standard-model signature scheme with such short signatures [JK18], however, based on a new security assumption about the cryptographic hash function and with significantly larger public keys. This scheme is based on a somewhat more complicated security proof, which relies on the techniques of Boneh and Boyen [BB04b, BB08] that we will discuss in Section 5.3.

Remark 87. The BLS signature scheme is very closely related to the identity-based encryption (IBE) scheme of Boneh and Franklin [BF01]. The generation of a key for an identity ID in [BF01] corresponds to the calculation of a BLS signature for the "message" ID. This similarity is not a coincidence. An IBE scheme always implies a signature scheme. The generic construction of a signature scheme from an IBE scheme is also known as *Naor transformation*, see [BF01].

Exercise 88. The BLS scheme can also be instantiated with asymmetrical pairings. Give a description based on type-2 or type-3 pairings. Try to optimize the size of public keys in both settings.

5.2.1 The Computational Diffie-Hellman Problem

The security of the BLS scheme is based on the so-called *computational Diffie-Hellman* problem in the group $\mathbb{G}$.

Definition 89. Let $\mathbb{G}$ be a group of order p, and let g be a random generator of $\mathbb{G}$. Let $x, y \xleftarrow{\$} \mathbb{Z}_p$. The *computational Diffie-Hellman* (CDH) problem in $\mathbb{G}$ is

- Given (g, g^x, g^y),
- compute g^{xy}.

5.2.2 Security of BLS Signatures

Theorem 90. *If the hash function H is modeled as random oracle, then for each PPT adversary $\mathcal{A}$ who breaks the* EUF-CMA *security of the BLS signature scheme in time $t_{\mathcal{A}}$ with success probability $\epsilon_{\mathcal{A}}$, one can construct a PPT adversary $\mathcal{B}$ who solves the CDH problem in $\mathbb{G}$ in time $t_{\mathcal{B}} \approx t_{\mathcal{A}}$ with a success probability of at least*

$$\epsilon_{\mathcal{B}} \geq \frac{\epsilon_{\mathcal{A}}}{q_H},$$

where q_H is the number of requests to the random oracle made by $\mathcal{A}$ in the EUF-CMA *experiment.*

Proof idea. The idea of the proof is very similar to the security proof of the RSA-FDH signature scheme from Section 4.2. Again, the "programmability" of the random oracle is used to simulate valid signatures and to carefully embed a given CDH challenge.

More precisely, algorithm $\mathcal{B}$ receives (g, g^x, g^y) as input. $\mathcal{B}$ guesses an index $\nu \xleftarrow{\$} \{1, \ldots, q_H\}$ and defines the public key as $pk = (g, g^x)$.

Responding to random oracle requests. The i-th request m_i to random oracle H is answered by $\mathcal{B}$ as follows:

If $i \neq \nu$, then $\mathcal{B}$ chooses a uniformly distributed random value $z_i \xleftarrow{\$} \mathbb{Z}_p$, defines $H(m_i) := g^{z_i}$, and answers with g^{z_i}. Since z_i is uniformly distributed over $\mathbb{Z}_p$, g^{z_i} is also uniformly distributed over $\mathbb{G}$. Thus the responses of the random oracle are correctly distributed.

If $i = \nu$, then $\mathcal{B}$ defines $H(m_\nu) := g^y$, where g^y is from the given CDH challenge, and answers with g^y. Since g^y is aslo uniformly distributed over $\mathbb{G}$, the response of the random oracle is correctly distributed. $\mathcal{B}$ "hopes" that $\mathcal{A}$ forges a signature for message $m^* = m_\nu$.

Responding to signature queries. We assume again (as in Section 4.2) that $\mathcal{A}$ always requests the hash value $H(m_i)$ *before* $\mathcal{A}$ requests a signature for m_i. (Otherwise $\mathcal{B}$ can simply make the corresponding random oracle query on behalf of $\mathcal{A}$).

Then $\mathcal{B}$ is able to sign all messages m_i with $i \neq \nu$ as

$$\sigma_i := (g^x)^{z_i}$$

This is a valid signature, because

$$e(g, \sigma_i) = e(g, g^{xz_i}) = e(g, g)^{xz_i} = e(g^x, g^{z_i}) = e(g^x, H(m_i)),$$

so the verification equation is fulfilled.

Extraction of the CDH solution. Recall that we assume that the adversary always outputs a signature for message m^* for which it has previously requested $H(m^*)$ from the random oracle (if he does not, then the adversary can output a valid signature only with negligible probability).

If the attacker outputs a valid signature σ^* for message $m^* = m_\nu$, then we have $H(m^*) = H(m_\nu) = g^y$ and the verification equation

$$e(g, \sigma^*) = e(g^x, g^y),$$

is fulfilled exactly when $\sigma^* = g^{xy}$. $\mathcal{B}$ can thus output the signature σ^* as a solution to the CDH challenge.

Exercise 91. Provide a complete formal proof of Theorem 90, based on the proof sketch outlined above. *Hint:* You can use the proof of Theorem 67 as a guildeline, it is very similar.

5.2.3 Aggregability of BLS Signatures

An interesting property of BLS signatures is that they are *aggregable* [BGLS03].

The problem. Let $U_1, \ldots, U_n$ a list of senders. These can be, for example, sensors in a sensor network which transmit measurement data to a receiving station V. Let $m_1, \ldots, m_n$ be the list of messages sent to V, where the message m_i comes from sender U_i. The data is then transmitted to the receiver V via a channel with possibly low bandwidth.

The messages are digitally signed, in order to prevent tampering. Each sender U_i has a public signature key pk_i with a corresponding secret key sk_i. If a classic signature scheme is used, then n message-signature pairs must be sent to V,

$$(m_1, \sigma_1), \ldots, (m_n, \sigma_n).$$

In total n signatures must be transferred in addition to the data, which may be a considerable overhead for many applications.

The solution: aggregability. If the senders $U_1, \ldots, U_n$ use the BLS signature scheme, then it is possible to reduce this overhead considerably. In this case each sender has a public BLS key $pk_i = (g, g^{x_i})$ with a secret key $sk_i = x_i$.[1]

Before the signed data is transferred to V, the signatures are *aggregated*. To do this, each sender U_i first computes a signature $\sigma_i := H(m_i)^{x_i}$ for message m_i, and then sends the pair (m_i, σ_i) to a distinguished sender, the *aggregator*.

The aggregator computes an *aggregated signature*

$$\sigma := \prod_{i=1}^{n} \sigma_i,$$

and then sends the list of messages $m_1, \ldots, m_n$ together with the aggregated signature σ to V.

Remark 92. The aggregated signature can be computed from the indidual signatures *without knowing a secret key*. So the aggregation is a "public" operation.

The verifier receiving the messages $m_1, \ldots, m_n$ along with the aggregated signature σ then checks whether

$$e(g, \sigma) \stackrel{?}{=} \prod_{i=1}^{n} e(H(m_i), g^{x_i})$$

holds.

The security of aggregated signatures can again be shown under the CDH assumption in the random oracle model. The proof is very similar to the proof of Theorem 90, see also [BGLS03].

Exercise 93. Show the *correctness* of aggregated BLS signatures.

Remark 94. So far, no aggregable signature scheme in the standard model (i.e. without random oracles) is known. We know only how to construct *sequentially* aggregable signatures, see for example [LOS+06].

[1] All senders use the same generator g. So the generator can also be seen as *system parameter* and does not have to be included in every public key.

5.2.4 Batch Verification of BLS Signatures

Another interesting feature of BLS signatures is that it is possible to verify many signatures "at once" ("batch verification").

The problem. Let V be a recipient who has received a list of n message-signature pairs

$$(m_1, \sigma_1), \ldots, (m_n, \sigma_n)$$

from a sender U with public key pk. When using a classic signature scheme, V would have to verify each signature individually, i.e. compute

$$\mathsf{Vfy}(pk, m_i, \sigma_i) \stackrel{?}{=} 1 \qquad \forall i \in \{1, \ldots, n\}$$

The Solution: Batch Verification. In case of BLS signatures, the verifier can verify all signatures at once. Let $pk = (g, g^x)$ be the public key of the sender U. The verifier first computes the product

$$h := \prod_{i=1}^{n} H(m_i) \qquad \text{and} \qquad \sigma := \prod_{i=1}^{n} \sigma_i$$

and then checks if

$$\mathsf{Vfy}(pk, h, \sigma) \stackrel{?}{=} 1 \qquad \Longleftrightarrow \qquad e(g, \sigma) \stackrel{?}{=} e(g^x, h)$$

holds.

This batch verification is much more efficient, because it is often significantly more expensive to evaluate a pairing than to perform a multiplication in $\mathbb{G}$. Batch verification is especially interesting in scenarios where a large number of signatures of a sender has to be verified in a very short time.

Exercise 95. Show the *correctness* of the batch verification procedure for BLS signatures.

Exercise 96. Does the batch verification also work with aggregated signatures?

5.3 Boneh Boyen Signatures

Given that BLS signatures could only be proven secure in the Random Oracle Model, it is natural to ask whether it is also possible to construct digital signatures in the pairing-based setting without random oracles. In order to explain a classical technique for constructing such signatures, we now describe a slightly simplified version of the scheme of Boneh and Boyen [BB04b, BB08].

Remark 97. We will describe a variant[2] of the simplified "weakly-secure" scheme from [BB04b], and prove it EUF-naCMA-secure. This weakly-secure scheme already contains the central trick of

[2]Essentially, the difference is only that we consider the symmetric type-1 bilinear group setting, while Boneh and Boyen considered asymmetric type-2 bilinear groups in their paper.

Boneh and Boyen that we want to consider here. The fully EUF-CMA-secure scheme of [BB04b] is essentially an optimized combination of this weakly-secure scheme with the discrete log-based chameleon hash function from Section 3.3.1, see the paper of Boneh and Boyen [BB04b, BB08] for details.

5.3.1 The Signature Scheme

Let $\mathbb{G}$ and $\mathbb{G}_T$ be prime-order groups, let h be a random generator of $\mathbb{G}$, and let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be a Type-1 bilinear map. The scheme has message space $\mathbb{Z}_p$.

$\mathsf{Gen}(1^k)$**.** The key generation algorithm chooses a random exponent $x \xleftarrow{\$} \mathbb{Z}_p$. The public key is $pk := (h, h^x) \in \mathbb{G}^2$, the corresponding secret key is $sk := x \in \mathbb{Z}_p$.

$\mathsf{Sign}(sk, m)$**.** A signature for message $m \in \mathbb{Z}_p$ is computed as follows. If $m + x = 0$, then set $\sigma := 1 \in \mathbb{G}$, i.e., the identity element in $\mathbb{G}$. Otherwise, compute

$$\sigma := h^{\frac{1}{x+m}}$$

Return σ.

$\mathsf{Vfy}(pk, m, \sigma)$**.** If $h^x \cdot h^m \neq 1$, then the verification algorithm outputs 1 if and only if

$$e(h^x \cdot h^m, \sigma) = e(h, h)$$

To make the scheme perfectly correct, we furthermore have to consider the case $h^x \cdot h^m = 1$, even though this should be very unlikely in practice. In this case, the verification algorithm outputs 1 if and only if $\sigma = 1$.

Remark 98. Note that the above signing algorithm essentially outputs a trivial signature when given as input a message m such that $m = -x$. We do this to obtain a perfectly-correct signature scheme, however, we will have to deal with this in the security proof.

Exercise 99. Prove that the above signature scheme is *correct*.

5.3.2 Security Analysis

This signature scheme can be proven secure under the following complexity assumption, which is called the *Q-strong Diffie-Hellman assumption*.

Definition 100. Let $\mathbb{G}$ be a group of order p and let $g \in \mathbb{G}$ be a random generator. Let $x \xleftarrow{\$} \mathbb{Z}_p$. The *$Q$-strong Diffie-Hellman problem* in $\mathbb{G}$ is:

- Given $(g, g^x, g^{x^2}, \ldots, g^{x^Q})$
- Compute $(y, s) \in \mathbb{G} \times \mathbb{Z}_p$ such that $y = g^{\frac{1}{x+s}}$

Theorem 101. *From each adversary $\mathcal{A}$ that breaks the EUF-naCMA-security of the Boneh-Boyen signature scheme by querying for q signatures in the experiment and running in time $t_{\mathcal{A}}$ and with success probability $\epsilon_{\mathcal{A}}$, we can construct an algorithm $\mathcal{B}$ that solves the Q-strong Diffie-Hellman problem with $Q = q + 1$ in time $t_{\mathcal{B}} \approx t_{\mathcal{A}}$ with success probability*

$$\epsilon_{\mathcal{B}} \geq \epsilon_{\mathcal{A}}$$

Proof. Algorithm $\mathcal{B}$ receives as input a challenge $(g, g^x, g^{x^2}, \ldots, g^{x^{q+1}})$ of the Q-strong Diffie-Hellman problem. It starts algorithm $\mathcal{A}$, which outputs a list of q messages $m_1, \ldots, m_q \in \mathbb{Z}_p$. First, $\mathcal{B}$ checks whether there exists an $i \in \{1, \ldots, q\}$ such that $g^x = g^{-m_i}$. If this holds for some i, then $\mathcal{B}$ is trivially able to solve the Q-strong Diffie-Hellman challenge. Thus, we may henceforth assume that $x \neq -m_i$ for all i.

Simulation, part one: set-up of the public key. Let P be the polynomial $P(X) := \prod_{i=1}^{q}(X + m_i)$. Algorithm $\mathcal{B}$ defines the public key as $pk := (h, u)$, where

$$h := g^{P(x)} \qquad \text{and} \qquad u := h^x = g^{xP(x)}$$

where x is the random value from the Q-strong Diffie-Hellman challenge. The difficulty here is that we have to describe a way how $\mathcal{B}$ is able to compute $h = g^{P(x)}$ and $h^x = g^{xP(x)}$ *without knowing* x. Here we will use the additional terms $g^x, g^{x^2}, \ldots, g^{x^{q+1}}$ from the Q-strong Diffie-Hellman challenge.

To this end, consider the polynomial $P(X)$, which has degree q. We can write this polynomial as

$$P(X) = \prod_{i=1}^{q}(X + m_i) = \sum_{i=0}^{q} a_i X^i \tag{5.1}$$

for values $a_1, \ldots, a_q \in \mathbb{Z}_p$ that are *efficiently computable*, given $m_1, \ldots, m_q$. Thus, algorithm $\mathcal{B}$ is able to compute h and h^x efficiently as follows:

1. First, it computes the values $a_1, \ldots, a_q$ as in Equation (5.1) from $m_1, \ldots, m_q$
2. Then it defines h and u as

$$h := \prod_{i=0}^{q} \left(g^{x^i}\right)^{a_i} \qquad \text{and} \qquad u := \prod_{i=0}^{q} \left(g^{x^{i+1}}\right)^{a_i}$$

   Note that $u = h^x$. Moreover, note that $\mathcal{B}$ can efficiently compute h and u, because the Q-strong Diffie-Hellman challenge contained exactly the required terms g^{x^i} for all $i \in \{0, \ldots, q + 1\}$.

Note that

$$h = \prod_{i=0}^{q} \left(g^{x^i}\right)^{a_i} = \prod_{i=0}^{q} g^{a_i x^i} = g^{\sum_{i=0}^{q} a_i x^i} = g^{P(x)}$$

and similarly $h^x = g^{xP(x)}$, as desired.

To see that this public key is correctly distributed, recall that we have $x \neq -m_i$ for all i, thus x is not a root of P modulo p. Therefore h is a random generator of $\mathbb{G}$, and it holds that $u = h^x$ for uniformly random $x \in \mathbb{Z}_p$, as required.

Simulation, part two: computing valid signatures. Next, $\mathcal{B}$ has to compute valid signatures for the messages $m_1, \ldots, m_q$ chosen by $\mathcal{A}$. For each $j \in \{1, \ldots, q\}$, the signature σ_j for message m_j is computed as follows.

1. First, $\mathcal{B}$ uses $m_1, \ldots, m_q$ to compute values $b_1, \ldots, b_q \in \mathbb{Z}_p$ that satisfy the equation

$$\prod_{i=1, i \neq j}^{q} (x + m_i) = \sum_{i=0}^{q-1} b_i x^i$$

   (Note that the index i with $i = j$ is removed from the product.)

2. Then it defines σ_j as

$$\sigma_j := \prod_{i=1, i \neq j}^{q} \left(g^{x^i}\right)^{b_i}$$

To verify that indeed σ_j is a valid signature for m_j, observe that it holds that

$$\sigma_j = \prod_{i=0, i \neq j}^{q} \left(g^{x^i}\right)^{b_i} = g^{\sum_{i=0}^{q-1} b_i x^i} = g^{\prod_{i=1, i \neq j}^{q} (x+m_i)} = g^{P(x)/(x+m_j)}$$

which implies

$$e(h^x \cdot h^{m_j}, \sigma_j) = e(h^{x+m_j}, g^{P(x)/(x+m_j)}) = e(h, g^{P(x)}) = e(h, h)$$

Therefore σ_j is a valid signature for m_j with respect to $pk = (h, h^x)$.

Finally, algorithm $\mathcal{B}$ outputs $(pk, \sigma_1, \ldots, \sigma_q)$ to $\mathcal{A}$.

Extraction. Assume that $\mathcal{A}$ outputs a pair (m^*, σ^*) that satisfies $m^* \notin \{m_1, \ldots, m_q\}$ and

$$e(h^x \cdot h^{m^*}, \sigma^*) = e(h, h)$$

By our assumption on $\mathcal{A}$, this happens with probability $\epsilon_{\mathcal{A}}$. Algorithm $\mathcal{B}$ now proceeds as follows to extract a solution to the Q-strong Diffie-Hellman problem.

Note first that $e(h^x \cdot h^{m^*}, \sigma^*) = e(h^x \cdot h^s, \sigma^*) = e(h, h)$ implies that

$$\sigma^* = h^{1/(x+s)} = g^{P(x)/(x+s)}$$

Using long division, we can write $P(x)$ as

$$P(x) = C(x) \cdot (x + s) + c'$$

for some polynomial $C(x) = \sum_{i=0}^{q-1} c_i x^i$ and some constant $c' \in \mathbb{Z}_p$, and the coefficients $c_0, \ldots, c_{q-2}, c' \in \mathbb{Z}_p$ are efficiently computable, given $m_1, \ldots, m_q, m^*$.

Exercise 102. Show that indeed the values $c_0, \ldots, c_{q-2}, c' \in \mathbb{Z}_p$ can efficiently be computed from $m_1, \ldots, m_q, m^*$.

Therefore we have

$$\sigma^* = g^{P(x)/(x+s)} = g^{(C(x)\cdot(x+s)+c')/(x+s)} = g^{C(x)+c'/(x+s)} = g^{\left(\sum_{i=0}^{q-1} c_i x^i\right)+c'/(x+s)}$$
$$= \prod_{i=0}^{q-1} g^{c_i x^i} \cdot g^{c'/(x+s)}$$

Now, using $c_0, \ldots, c_{q-2}, c'$ and again the g^{x^i}-terms from the Q-strong Diffie-Hellman challenge, adversary $\mathcal{B}$ computes

$$y := \left(\sigma^* / \prod_{i=0}^{q-1} g^{c_i x^i}\right)^{1/c'}$$

ans sets $s := m^*$.

Finally, to see that indeed (y, s) is a valid solution to the Q-strong Diffie-Hellman challenge, note that

$$y = \left(\sigma^* / \prod_{i=0}^{q-1} g^{c_i x^i}\right)^{1/c'}$$
$$= \left(\left(\prod_{i=0}^{q-1} g^{c_i x^i} \cdot g^{c'/(x+s)}\right) / \prod_{i=0}^{q-1} g^{c_i x^i}\right)^{1/c'}$$
$$= \left(g^{c'/(x+s)}\right)^{1/c'} = g^{1/(x+s)}$$

□

5.3.3 The Fully Secure Scheme of Boneh and Boyen

The fully EUF-CMA-secure scheme of Boneh and Boyen works as follows:

$\mathsf{Gen}(1^k)$**.** The key generation algorithm chooses random exponents $x, u \xleftarrow{\$} \mathbb{Z}_p$. The public key is $pk := (h, h^x, h^u) \in \mathbb{G}^2$, the corresponding secret key is $sk := (x, u) \in \mathbb{Z}_p$.

$\mathsf{Sign}(sk, m)$**.** A signature for message $m \in \mathbb{Z}_p$ is computed as follows. Choose $r \xleftarrow{\$} \mathbb{Z}_p$. If $m + x + ur = 0$, then set $\sigma := 1 \in \mathbb{G}$. Otherwise compute

$$\sigma' := h^{\frac{1}{x+ur+m}}$$

and return $\sigma = (\sigma', r)$.

$\mathsf{Vfy}(pk, m, \sigma)$**.** The verification algorithm parses $(\sigma', r) := \sigma$. If $h^x h^m (h^u)^r \neq 1$, then it outputs 1 if and only if

$$e(h^x \cdot h^m \cdot (h^u)^r, \sigma') = e(h, h)$$

Otherwise, $h^x h^m (h^u)^r = 1$, it outputs 1 if and only if $\sigma = 1 \in \mathbb{G}$.

Exercise 103. Prove that for each algorithm $\mathcal{A}$ that breaks the EUF-CMA-security of this scheme by querying the signing-oracle at most q times in time $t_{\mathcal{A}}$ with success probability $\epsilon_{\mathcal{A}}$, there exists an algorithm $\mathcal{B}$ that solves the Q-strong Diffie-Hellman problem with $Q = q + 1$ in time $t_{\mathcal{B}} \approx t_{\mathcal{A}}$ with success probability at least $\epsilon_{\mathcal{A}}$. (You may want to have a look at the papers of Boneh and Boyen [BB04b, BB08] to receive some hints.)

5.3.4 Similarity to GHR Signatures

In a sense, the Boneh and Boyen [BB04b, BB08] can be seen as a pairing-based variant of the strong-RSA-based signature scheme of Gennaro, Halevi and Rabin that were introduced and analyzed in Section 4.3. Note that both the construction and the security analysis share many similarities:

Signatures. Recall that a GHR-signature is defined as

$$\sigma = s^{1/h(m)}$$

where s is some value contained in the public key, and h is a hash function that maps messages to primes.

Similarly, a Boneh-Boyen signature is defined as

$$\sigma = h^{1/f(m)}$$

where h is some value from the public key, and function f is defined by the value h^x contained in the public key as $f(m) = x + m$.

Simulation in the security proof. Recall that in the security proof of GHR-signatures, we have defined the value s contained in the public key as

$$s := y^{\prod_i h(m_i)}$$

Here we used the fact that in the EUF-naCMA security experiment we receive all messages that the attacker wishes to have signed at the very beginning of the experiment, before we have to provide the attacker with the public key. We have used this setup in the proof to simulate signatures, by computing a signature for message m_j as

$$\sigma_i := y^{\prod_{i \neq j} h(m_i)} = s^{1/h(m_j)}$$

Similarly, for Boneh-Boyen signatures we have defined the value h in the public key as

$$h := g^{\prod_i (x+m_i)}$$

and simulated signatures by computing

$$\sigma_i := g^{\prod_{i \neq j} (x+m_i)} = h^{1/h(m_j)}$$

Extraction in the security proof. In the case of GHR-signatures, a successful attacker provided us with a signature

$$\sigma^* = s^{1/h(m^*)} = y^{\left(\prod_i h(m_i)\right)/h(m^*)}$$

and we used Shamir's trick (Lemma 31) to extract $y^{1/h(m^*)}$ from σ^*.

In comparison, in the case of Boneh-Boyen signatures the attacker gives us

$$\sigma^* = h^{1/(x+m^*)} = g^{\left(\prod_i (x+m_i)\right)/(x+m^*)}$$

and we used a different trick to extract $g^{1/(x+m^*)}$ from σ^*.

"Flexible" complexity assumptions. Finally, note that we were able to prove the EUF-naCMA-security of GHR-signatures under the strong-RSA assumption, which states that given (N, y) it is hard to compute a pair (x, e) with $e > 1$ such that $x = y^{1/e} \bmod N$. Note that this is a "flexible" assumption, which asserts that it is hard to compute e-th roots for *any* e, which may be chosen by the adversary (of course e must be non-trivial, hence $e > 1$).

Similarly, we proved the EUF-naCMA-security of Boneh-Boyen signatures under the assumption that, given $g, g^x, \ldots, g^{x^{q+1}}$, it is hard to compute a pair $(g^{1/(x+m^*)}, m^*)$ for any m^*. Again, this is a "flexible" assumption, where the attacker is able to compute an $(x + m^*)$-th root of g for *any* m^*, which again may be chosen by the adversary.

Therefore the Boneh-Boyen signature scheme is sometimes considered as the "pairing-based dual" of GHR-signatures, and the q-strong Diffie-Hellman assumption can be seen as the "pairing-based dual" of the strong-RSA assumption.

5.4 Waters Signatures

Another important signature scheme based on pairings is the scheme of Waters [Wat05]. This scheme is based on the CDH problem in groups with pairing, like the BLS scheme, but it can be proven EUF-CMA-secure in the standard model (without random oracles).

The Waters signature scheme uses (implicitly) a *programmable hash function* (PHF). PHFs are an abstraction of a frequently used proof technique, which was explicitly described by Hofheinz and Kiltz [HK08]. This abstraction is well suited to make the proof of Waters signatures more accessible. Therefore, we will first introduce the concept of programmable hash functions, and then explain the Waters signature scheme and its security proof.

5.4.1 Programmable Hash Functions

In the security proofs of RSA-FDH (Section 4.2) and Boneh-Lynn-Shacham signatures (Section 5.2), we took advantage of the fact that we could "program" the random oracle appropriately, in such a way that we could embed a given RSA/CDH challenge in exactly the right place (with sufficiently high success probability) in the simulation of the challenger by the algorithm $\mathcal{B}$. *Is it possible to construct a concrete hash function that can be programmed in a similar way?*

In general, the answer is unfortunately *no*. However, it is possible to achieve a somewhat weaker form of programmability, which is sufficient to construct provably secure cryptosystems.

Group hash functions. Programmable hash functions are hash functions which map a set to group elements. Therefore, we first define group hash functions. In the following, we consider hash functions with preimage set $\{0,1\}^\ell$, where $\ell = \ell(k)$ is some polynomial in the security parameter.

Definition 104. A *group hash function* for a group $\mathbb{G}$ consists of two polynomial time algorithms $(\mathsf{Gen}, \mathsf{Eval})$.

- The *generation algorithm* $\kappa \xleftarrow{\$} \mathsf{Gen}(g)$ receives as input a generator g of the group $\mathbb{G}$, and outputs a function description κ. This algorithm can be probabilistic.
- The *evaluation algorithm* $\mathsf{Eval}(\kappa, m)$ is a deterministic algorithm that implements a hash function
$$H_\kappa : \{0,1\}^\ell \to \mathbb{G},$$
This hash function maps bit strings to group elements and is parameterized by κ. Given a value $m \in \{0,1\}^\ell$, $\mathsf{Eval}(\kappa, m)$ outputs the hash value $H_\kappa(m)$.

Programmable hash functions. Based on Definition 104 we can now define *programmable hash functions* (PHFs), because PHFs are group hash functions with an additional "trapdoor property".

Definition 105. A group hash function $(\mathsf{Gen}, \mathsf{Eval})$ is a (v, w, γ)-*programmable hash function*, if additionally two polynomial time algorithms $(\mathsf{TrapGen}, \mathsf{TrapEval})$ exist with the following properties.

- The *trapdoor generation algorithm* $\mathsf{TrapGen}(g, h)$ receives as input two generators $g, h \in \mathbb{G}$ of a group of order p, and outputs a function description κ and a trapdoor τ. $\mathsf{TrapGen}$ may be probabilistic.

  We require that function descriptions generated by Gen or $\mathsf{TrapGen}$ are perfectly identical distribuited.[3] More precisely, let $\kappa_0 \xleftarrow{\$} \mathsf{Gen}(g)$ and $(\kappa_1, \tau) \xleftarrow{\$} \mathsf{TrapGen}(g, h)$. Then it holds for all generators g, h that κ_0 and κ_1 are identically distributed.
- The *trapdoor evaluation algorithm* $\mathsf{TrapEval}(\tau, m)$ is a deterministic algorithm which receives the trapdoor τ as input and a value $m \in \{0,1\}^\ell$. This algorithm outputs $(a, b) \in \mathbb{Z}_p^2$ so that for $(\kappa, \tau) \xleftarrow{\$} \mathsf{TrapGen}(g, h)$ the following holds.
$$H_\kappa(m) = h^a g^b$$

[3] This requirement can be relaxed. In general, a *statistically* indistinguishable distribution is sufficient. For many applications *computational* indistinguishability is sufficient, which demands that functions generated by Gen or $\mathsf{TrapGen}$ cannot be distinguished *efficiently*. See [HK08] for details.

- The "a-component" of the output of the $\mathsf{TrapEval}$ algorithm is *well-distributed*. More precisely, let π_a be the projection $\pi_a(a,b) = a$. Then
  - for all g, h,
  - for all κ generated by $(\kappa, \tau) \xleftarrow{\$} \mathsf{TrapGen}(g,h)$,
  - for all $m_1^*, \ldots, m_v^* \in \{0,1\}^\ell$ and all $m_1, \ldots, m_w \in \{0,1\}^\ell$,

  must hold that

$$\Pr\left[\begin{array}{ll} \pi_a(\mathsf{TrapEval}(\tau, m_i^*)) \equiv 0 \bmod p & \forall i \in \{1,\ldots,v\} \wedge \\ \pi_a(\mathsf{TrapEval}(\tau, m_i)) \not\equiv 0 \bmod p & \forall i \in \{1,\ldots,w\} \end{array}\right] \geq \gamma,$$

  where the probability is over the choice of the trapdoor τ (from all possible trapdoors for κ).

Remark 106. This definition is unfortunately more unwieldy than its intuition. But the idea is actually very simple:

- The $\mathsf{TrapGen}(g,h)$ algorithm makes it possible to generate the group hash function along with a trapdoor τ so that hash functions generated with trapdoor are *indistinguishable* from honestly generated hash functions.
- The trapdoor τ makes it possible to compute the *"discrete logarithm"* (a,b) of hash values with respect to the generators (g,h).
- The *well-distribution* of the a-component provides the actual "programmability" (which is, of course, very limited compared to a random oracle). A hash function is (v,w,γ)-programmable, if there is a probability of at least γ that for *any* values $m_1^*, \ldots, m_v^*$ and $m_1, \ldots, m_w$ chosen by the attacker, it holds that
  - the h-component *is not contained* in hash values $H(m_i^*) = h^{a_i^*} g^{b_i^*}$, since we have $a_i^* \equiv 0 \bmod p$ is valid for all $m_1^*, \ldots, m_v^*$,
  - The h-component *is contained* in all hash values $H(m_i) = h^{a_i} g^{b_i}$, i.e. $a_i \not\equiv 0 \bmod p$ holds for all $m_1, \ldots, m_w$.

Remark 107. We will later be interested in $(1,q,\gamma)$-programmable PHFs with γ not negligible. Let $m_1, \ldots, m_q$ be the list of signature queries the attacker makes in the $\mathsf{EUF\text{-}CMA}$ experiment, and let m^* be the message for which the attacker issues a forgery. We will then hope in the proof that

$$\pi_a(\mathsf{TrapEval}(\tau, m^*)) \equiv 0 \bmod p \quad \wedge \pi_a(\mathsf{TrapEval}(\tau, m_i)) \not\equiv 0 \bmod p \quad \forall i \in \{1,\ldots,q\}$$

olds. If the PHF $(1,q,\gamma)$ is programmable, then this will happen with probability at least γ.

Remark 108. In other contexts $(v,1,\gamma)$-programmable hash functions can also be very helpful, with small v like $v = 4$ or $v = 8$. For example, in [HJK11] such PHFs were combined with the "prefix trick" by Hohenberger and Waters (Section 4.4) to obtain more efficient and shorter RSA-based signatures. Efficient $(v,1,\gamma)$-PHFs for large values of v unfortunately do not exist, because otherwise one could specify a standard model proof for RSA-FDH (and BLS signatures) that contradicts the impossibility result in [DHT12].

Exercise 109. Let H be a $(1, 1, \gamma)$-programmable hash function for $\mathbb{G}$. Show that H is collision resistant (Definition 15) if γ is not negligible and the discrete logarithm problem in $\mathbb{G}$ is hard. *Hint:* Assume there is an algorithm $\mathcal{A}_H$ that computes a collision for H with a non-negligible probability. Show that you can then use $\mathcal{A}_H$ to construct an algorithm $\mathcal{A}_{\mathbb{G}}$ that solves the discrete logarithm problem in $\mathbb{G}$ so that $\mathcal{A}_{\mathbb{G}}$ has about the same runtime as $\mathcal{A}_H$. Finally, estimate the probability of success of $\mathcal{A}_{\mathbb{G}}$.

5.4.2 The Signature Scheme

Let $\mathbb{G}$ and $\mathbb{G}_T$ be groups of prime order p and let g be a generator of $\mathbb{G}$. Let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be a pairing of type 1. Let $(\mathsf{Gen}, \mathsf{Eval})$ be a group hash function mapping to $\mathbb{G}$.

$\mathsf{Gen}(1^k)$**.** The key generation algorithm selects a random group element $g^\alpha \xleftarrow{\$} \mathbb{G}$ and computes $e(g, g)^\alpha = e(g, g^\alpha)$. Then $\kappa \xleftarrow{\$} \mathsf{Gen}(g)$ is used to sample the description of a group hash function $H_\kappa : \{0, 1\}^\ell \to \mathbb{G}$. In the following we will write H as short form for H_κ.

The public key is $pk := (g, \kappa, e(g, g)^\alpha)$, the secret key is $sk := g^\alpha$.

$\mathsf{Sign}(sk, m)$**.** To sign a message $m \in \{0, 1\}^\ell$ a random value $r \xleftarrow{\$} \mathbb{Z}_p$ is chosen and the values

$$\sigma_1 := g^r \in \mathbb{G} \qquad \text{and} \qquad \sigma_2 := g^\alpha \cdot H(m)^r \in \mathbb{G}$$

are computed. A signature for message m consists of $\sigma := (\sigma_1, \sigma_2) \in \mathbb{G}^2$.

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

$$e(g, g)^\alpha \cdot e(\sigma_1, H(m)) = e(g, \sigma_2)$$

holds, and otherwise 0.

Exercise 110. Show the *correctness* of the Waters signature scheme.

Remark 111. Just as the BLS signature scheme is derived from the identity key generation algorithm of the Boneh-Franklin IBE scheme (see Remark 87), the Waters signature scheme is derived from the identity key generation algorithm of Waters' IBE scheme [Wat05].

5.4.3 Security of Waters Signatures

Theorem 112. *Let $\mathcal{A}$ be a PPT adversary that breaks the* EUF-CMA *security of the Waters signature scheme in time $t_{\mathcal{A}}$ with probability of success $\epsilon_{\mathcal{A}}$, with at most q signature queries. If the hash function used in the Waters signature scheme is a $(1, q, \gamma)$-PHF, then there exists a PPT adversary $\mathcal{B}$ breaking the CDH-problem (Definition 89) in $\mathbb{G}$ in time $t_{\mathcal{B}} \approx t_{\mathcal{A}}$ with success probability*

$$\epsilon_{\mathcal{B}} \geq \gamma \cdot \epsilon_{\mathcal{A}}$$

Proof. Algorithm $\mathcal{B}$ receives as input a CDH challenge $(g, g^x, g^y) \in \mathbb{G}^3$, where $|\mathbb{G}| = p$. $\mathcal{B}$ generates the hash function H by computing

$$(\kappa, \tau) \xleftarrow{\$} \mathsf{TrapGen}(g, g^x).$$

The public key is defined as $pk := (g, \kappa, e(g^x, g^y))$. Note that this is a correctly distributed public key. Note also that the third value $e(g^x, g^y) = e(g, g)^{xy}$ in the public key implicitly defines the secret key g^α as $g^\alpha := g^{xy}$. The value g^{xy} is the solution of the CDH problem $\mathcal{B}$ is looking for.

$\mathcal{B}$ outputs the pk to $\mathcal{A}$, who is now allowed to make signature queries for messages $m_1, \ldots, m_q$. Here $\mathcal{B}$ "hopes" that the following two conditions hold:

- For each message m_i for which $\mathcal{A}$ requests a signature, it holds that $\pi_a(\mathsf{TrapEval}(\tau, m_i)) \not\equiv 0 \bmod p$.

  Exactly in this case $\mathcal{B}$ can *simulate* a valid signature without knowing the secret key, as we will explain below.

- If $\mathcal{A}$ outputs a forgery (m^*, σ^*), then $\pi_a(\mathsf{TrapEval}(\tau, m^*)) \equiv 0 \bmod p$.

  We will also explain that exactly in this case $\mathcal{B}$ is able to *extract* the solution g^{xy} of the CDH problem from the forgery (m^*, σ^*).

If these two conditions are met at the end of the game, then we say that event E has occurred. Since we assume that H is a $(1, q, \gamma)$-programmable hash function, we have

$$\Pr[E] = \Pr\left[\begin{array}{c} \pi_a(\mathsf{TrapEval}(\tau, m_i)) \not\equiv 0 \bmod p \quad \forall i \in \{1, \ldots, q\} \\ \wedge \\ \pi_a(\mathsf{TrapEval}(\tau, m^*)) \equiv 0 \bmod p \end{array}\right] \geq \gamma.$$

In the following we assume that event E occurs. In this case $\mathcal{B}$ can answer all signature requests from $\mathcal{A}$, and at the same time extract a solution to the CDH problem from the forgery.

Simulation of valid signatures. If $\mathcal{A}$ requests the i-th signature with message m_i, then $\mathcal{B}$ uses the trapdoor evaluation algorithm $(a_i, b_i) = \mathsf{TrapEval}(\tau, m_i)$ from H to compute (a_i, b_i) with

$$H(m_i) = (g^x)^{a_i} \cdot g^{b_i}.$$

Because event E occurs, and thus $a_i \not\equiv 0 \bmod p$ holds, $\mathcal{B}$ can compute the signature $\sigma_i = (\sigma_{i,1}, \sigma_{i,2})$ as follows.

1. $\mathcal{B}$ chooses a random value $s_i \xleftarrow{\$} \mathbb{Z}_p$.

2. The first part of the signature is computed as $\sigma_{i,1} := (g^y)^{-1/a_i} \cdot g^{s_i}$. Because $s_i \xleftarrow{\$} \mathbb{Z}_p$ is uniformly distributed over $\mathbb{Z}_p$, $\sigma_{i,1}$ is also uniformly distributed over $\mathbb{G}$.

   Note that, when writing $\sigma_{i,1}$ in the form $\sigma_{i,1} = g^{r_i}$, then this implicitly defines the value r_i as $r_i = -y/a_i + s_i$.

3. The second part of the signature is computed as $\sigma_{i,2} := (g^x)^{a_i s_i} \cdot (g^y)^{-b_i/a_i} \cdot g^{b_i s_i}$.

   To show that this is a valid signature we have to show that $\sigma_{i,2} = g^{xy} H(m_i)^{r_i}$ is valid, where $r_i = -y/a_i + s_i$. In fact this is true:

$$\begin{aligned} g^{xy} H(m_i)^{r_i} &= g^{xy} \left((g^x)^{a_i} g^{b_i}\right)^{r_i} = g^{xy} \left((g^x)^{a_i} g^{b_i}\right)^{-y/a_i + s_i} \\ &= g^{xy} \left(g^{x a_i (-y/a_i + s_i)} g^{b_i (-y/a_i + s_i)}\right) \\ &= g^{xy} \left(g^{-xy} g^{x a_i s_i} g^{b_i (-y/a_i + s_i)}\right) \\ &= g^{x a_i s_i} g^{b_i (-y/a_i + s_i)} = (g^x)^{a_i s_i} (g^y)^{-b_i/a_i} g^{b_i s_i}. \end{aligned}$$

   The simulated signatures are therefore valid and correctly distributed.

   The "trick" of the simulation is that all values are set up in a clever way, so that the g^{xy}-term cancels out. This trick goes back to Boneh and Boyen [BB04a].

Extraction of the solution of the CDH problem. If $\mathcal{A}$ outputs a valid forgery (m^*, σ^*), $\sigma^* = (\sigma_1^*, \sigma_2^*)$, then $\mathcal{B}$ uses the trapdoor evaluation algorithm $(a^*, b^*) = \mathsf{TrapEval}(\tau, m^*)$ from H to compute (a^*, b^*) with

$$H(m^*) = (g^x)^{a^*} g^{b^*} = (g^x)^0 g^{b^*} = g^{b^*},$$

where $a^* \equiv 0 \bmod p$, because event E occurs.

So $\mathcal{B}$ can compute the value g^{xy} as $g^{xy} = \sigma_2^* \cdot (\sigma_1^*)^{-b^*}$. To see that this is indeed the value g^{xy} we are looking for, we write

$$\sigma_1^* = g^{r^*} \qquad \text{and} \qquad \sigma_2^* = g^{xy} \cdot H(m^*)^{r^*}.$$

This is possible because σ^* is a valid signature. Then

$$\sigma_2^* \cdot (\sigma_1^*)^{-b^*} = g^{xy} \cdot H(m^*)^{r^*} \cdot g^{-r^* b^*} = g^{xy} \cdot g^{b^* r^*} \cdot g^{-r^* b^*} = g^{xy}.$$

Analysis. So if event E occurs (which will happen with probability at least γ for *any* messages $m^*, m_1, \ldots, m_q$ chosen by the attacker $\mathcal{A}$, due to the properties of the PHF H), then $\mathcal{B}$ perfectly simulates the real $\mathsf{EUF\text{-}CMA}$ experiment. In this case $\mathcal{B}$ can solve the CDH problem if $\mathcal{A}$ issues a valid forgery and we obtain

$$\epsilon_{\mathcal{B}} \geq \Pr[E] \cdot \epsilon_{\mathcal{A}} \geq \gamma \cdot \epsilon_{\mathcal{A}}.$$

□

5.4.4 The Programmable Hash Function of Waters

Waters [Wat05] has given the following concrete construction of a hash function, which is often referred to as the "Waters hash function".

- The generation algorithm $\mathsf{Gen}(g)$ receives as input a generator $g \in \mathbb{G}$. It selects $\ell + 1$ random group elements $u_0, \ldots, u_\ell \xleftarrow{\$} \mathbb{G}$ and outputs $\kappa = (u_0, \ldots, u_\ell)$.

- The evaluation algorithm Eval receives as input κ and $m = (m_1, \ldots, m_\ell) \in \{0,1\}^\ell$ and outputs

$$H_\kappa(m) = u_0 \prod_{i=1}^{\ell} u_i^{m_i} \in \mathbb{G}$$

Theorem 113 ([HK08, Theorem 4]). *Let $q = q(k)$ be a polynomial in the security parameter. The Waters hash function is a $(1, q, \gamma)$ programmable hash function with*

$$\gamma \geq \frac{1}{8(\ell+1)q}.$$

Proof idea. In order to prove the theorem, we have to describe and analyze algorithms TrapGen and TrapEval.

- The TrapGen algorithm receives as input two generators (g, h) and selects uniformly random

$$\hat{a}_0, \ldots, \hat{a}_\ell \xleftarrow{\$} \{-1, 0, 1\} \qquad \text{and} \qquad \hat{b}_0, \ldots, \hat{b}_\ell \xleftarrow{\$} \mathbb{Z}_p$$

  The vector $\kappa = (u_0, \ldots, u_\ell)$ is defined as $u_i := g^{\hat{b}_i} h^{\hat{a}_i}$ for all $i \in \{0, \ldots, \ell\}$. Since the $\hat{b}_i$ values are uniformly distributed over $\mathbb{Z}_p$, all u_i values are uniformly distributed over $\mathbb{G}$. Thus, κ is perfectly indistinguishable from a description of a hash function generated with Gen.

  The trapdoor $\tau = (\hat{a}_0, \ldots, \hat{a}_\ell, \hat{b}_0, \ldots, \hat{b}_\ell)$ consists of the discrete logarithms of the u_i elements to the base (g, h).

- The TrapEval algorithm receives τ and $m = (m_1, \ldots, m_\ell) \in \{0,1\}^\ell$ as input. It computes $(a, b) \in \mathbb{Z}_p$ as

$$a := \hat{a}_0 + \sum_{i=1}^{\ell} \hat{a}_i \cdot m_i \qquad \text{and} \qquad b := \hat{b}_0 + \sum_{i=1}^{\ell} \hat{b}_i \cdot m_i$$

  and outputs (a, b). It is easy to verify that

$$H_\kappa(m) = u_0 \prod_{i=1}^{\ell} u_i^{m_i} = g^{\hat{b}_0 + \sum_{i=1}^{\ell} \hat{b}_i \cdot m_i} h^{\hat{a}_0 + \sum_{i=1}^{\ell} \hat{a}_i \cdot m_i} = g^b h^a$$

  holds.

- The analysis of the well-distribution of the a-component is very technical. Therefore we only give intuition here, and refer to [Wat05, HK08, HJK12] for details.

  The analysis exploits the fact that the $\hat{a}_i$ values are randomly chosen from $\{-1, 0, 1\}$. The sum

$$a := \hat{a}_0 + \sum_{i=1}^{\ell} \hat{a}_i \cdot m_i$$

thus corresponds to a *random walk* of length ℓ over $\{-1, 0, 1\}$. This yields that the value a is (discretely) Gaussian-distributed around zero, which can then be exploited to prove the well-distribution of the a-component.[4]

5.5 Open Problems

The programmable hash function by Waters is the only known example of a $(1, q, \gamma)$-PHF, where $q = q(k)$ can be any polynomial in the security parameter and γ is still not negligible. The Waters hash function is described by $\ell + 1$ group elements, where ℓ is the length of the messages to be signed. Even if one uses a collision resistant hash function to sign larger messages, ℓ will still be in the order of $\ell \approx 256$ for reasonable practical deployments, which is rather large. A more efficient construction of a $(1, q, \gamma)$-PHF (or an impossibility result showing that it does not exist) is an interesting open problem.

The construction of a more efficient signature scheme with *constant* size of the public key *and* signatures based on the difficulty of the CDH problem is also an interesting open problem. So far we know only a *stateful* CDH-based scheme with these properties [HW09a]. Furthermore, asymptotically more efficient schemes have been described in [BHJ+13, BHJ+15].

[4] In fact, the $\hat{a}_i$ values in the proof from [HK08] are not chosen from $\{-1, 0, 1\}$, but discretely normally distributed (with an expected value 0 and a variance depending on q). However, to understand the intuition of the proof it is sufficient to imagine $\{-1, 0, 1\}$, we refer to [HK08] for details.

Bibliography

[BB04a] Dan Boneh and Xavier Boyen. Efficient selective-ID secure identity based encryption without random oracles. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 223–238, Interlaken, Switzerland, May 2–6, 2004. Springer, Heidelberg, Germany.

[BB04b] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 56–73, Interlaken, Switzerland, May 2–6, 2004. Springer, Heidelberg, Germany.

[BB08] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *Journal of Cryptology*, 21(2):149–177, April 2008.

[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.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 416–432, Warsaw, Poland, May 4–8, 2003. Springer, Heidelberg, Germany.

[BHJ+13] Florian Böhl, Dennis Hofheinz, Tibor Jager, Jessica Koch, Jae Hong Seo, and Christoph Striecks. Practical signatures from standard assumptions. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 461–485, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany.

[BHJ+15] Florian Böhl, Dennis Hofheinz, Tibor Jager, Jessica Koch, and Christoph Striecks. Confined guessing: New signatures from standard assumptions. *Journal of Cryptology*, 28(1):176–208, January 2015.

[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.

[BK10] Zvika Brakerski and Yael Tauman Kalai. A framework for efficient signatures, ring signatures and identity based encryption in the standard model. Cryptology ePrint Archive, Report 2010/086, 2010. http://eprint.iacr.org/2010/086.

[Ble98] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In Hugo Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 1–12, Santa Barbara, CA, USA, August 23–27, 1998. Springer, Heidelberg, Germany.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532, Gold Coast, Australia, December 9–13, 2001. Springer, Heidelberg, Germany.

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, September 2004.

[BPVY00] Ernest F. Brickell, David Pointcheval, Serge Vaudenay, and Moti Yung. Design validations for discrete logarithm based signature schemes. In Hideki Imai and Yuliang Zheng, editors, *PKC 2000: 3rd International Workshop on Theory and Practice in Public Key Cryptography*, volume 1751 of *Lecture Notes in Computer Science*, pages 276–292, Melbourne, Victoria, Australia, January 18–20, 2000. Springer, Heidelberg, Germany.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press.

[BR95] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In Alfredo De Santis, editor, *Advances in Cryptology – EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111, Perugia, Italy, May 9–12, 1995. Springer, Heidelberg, Germany.

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 399–416, Saragossa, Spain, May 12–16, 1996. Springer, Heidelberg, Germany.

[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.

[Bro02] Daniel R. L. Brown. Generic groups, collision resistance, and ecdsa. Cryptology ePrint Archive, Report 2002/026, 2002. http://eprint.iacr.org/.

[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.

[CF05] Henri Cohen and Gerhard Frey, editors. *Handbook of elliptic and hyperelliptic curve cryptography*. CRC Press, 2005.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th Annual ACM Symposium on Theory of Computing*, pages 209–218, Dallas, TX, USA, May 23–26, 1998. ACM Press.

[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.

[Cra96] Ronald Cramer. *Modular Design of Secure yet Practical Cryptographic Protocols*. PhD thesis, CWI and Uni.of Amsterdam, November 1996.

[CS99] Ronald Cramer and Victor Shoup. Signature schemes based on the strong RSA assumption. In Juzar Motiwalla and Gene Tsudik, editors, *ACM CCS 99: 6th Conference on Computer and Communications Security*, pages 46–51, Singapore, November 1–4, 1999. ACM Press.

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

[Deb08] Debian Sicherheitsankündigung. DSA-1571-1 openssl – Voraussagbarer Zufallszahlengenerator, 2008. http://www.debian.org/security/2008/dsa-1571.

[DHT12] Yevgeniy Dodis, Iftach Haitner, and Aris Tentes. On the instantiability of hash-and-sign RSA signatures. In Ronald Cramer, editor, *TCC 2012: 9th Theory of Cryptography Conference*, volume 7194 of *Lecture Notes in Computer Science*, pages 112–132, Taormina, Sicily, Italy, March 19–21, 2012. Springer, Heidelberg, Germany.

[DN10] Ivan Damgård and Jesper Buus Nielsen. Cryptologic Protocol Theory. Course Website, 2010. http://www.daimi.au.dk/~ivan/CPT.html.

[DOP05] Yevgeniy Dodis, Roberto Oliveira, and Krzysztof Pietrzak. On the generic insecurity of the full domain hash. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 449–466, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Heidelberg, Germany.

[DSS09] Digital signature standard (DSS). National Institute of Standards and Technology (NIST), FIPS PUB 186-3, U.S. Department of Commerce, 2009. http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf.

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

[Fin06] Hal Finney. Bleichenbacher's RSA signature forgery based on implementation error. Posting at IETF-OpenPGP Mailing List, 2006. https://www.ietf.org/mail-archive/web/openpgp/current/msg00999.html.

[Fis03] Marc Fischlin. The Cramer-Shoup strong-RSA signature scheme revisited. In Yvo Desmedt, editor, *PKC 2003: 6th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 116–129, Miami, FL, USA, January 6–8, 2003. Springer, Heidelberg, Germany.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany.

[GHR99] Rosario Gennaro, Shai Halevi, and Tal Rabin. Secure hash-and-sign signatures without the random oracle. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 123–139, Prague, Czech Republic, May 2–6, 1999. Springer, Heidelberg, Germany.

[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.

[GPS08] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[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.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 415–432, Istanbul, Turkey, April 13–17, 2008. Springer, Heidelberg, Germany.

[HJK11] Dennis Hofheinz, Tibor Jager, and Eike Kiltz. Short signatures from weaker assumptions. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 647–666, Seoul, South Korea, December 4–8, 2011. Springer, Heidelberg, Germany.

[HJK12] Dennis Hofheinz, Tibor Jager, and Edward Knapp. Waters signatures with optimal security reduction. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012: 15th International Conference on Theory and Practice of Public Key Cryptography*, volume 7293 of *Lecture Notes in Computer Science*, pages 66–83, Darmstadt, Germany, May 21–23, 2012. Springer, Heidelberg, Germany.

[HK08] Dennis Hofheinz and Eike Kiltz. Programmable hash functions and their applications. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 21–38, Santa Barbara, CA, USA, August 17–21, 2008. Springer, Heidelberg, Germany.

[HW09a] 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.

[HW09b] Susan Hohenberger and Brent Waters. Short and stateless signatures from the RSA assumption. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume

5677 of *Lecture Notes in Computer Science*, pages 654–670, Santa Barbara, CA, USA, August 16–20, 2009. Springer, Heidelberg, Germany.

[JK03] J. Jonsson and B. Kaliski. Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1. RFC 3447 (Informational), February 2003.

[JK18] Tibor Jager and Rafael Kurek. Short digital signatures and ID-KEMs via truncation collision resistance. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 221–250, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany.

[Jou04] Antoine Joux. A one round protocol for tripartite Diffie-Hellman. *Journal of Cryptology*, 17(4):263–276, September 2004.

[Kal98] B. Kaliski. PKCS #1: RSA Encryption Version 1.5. RFC 2313 (Informational), March 1998. Obsoleted by RFC 2437.

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

[KK12] Saqib A. Kakvi and Eike Kiltz. Optimal security proofs for full domain hash, revisited. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 537–553, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

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

[KPR03] Vlastimil Klíma, Ondrej Pokorný, and Tomás Rosa. Attacking RSA-based sessions in SSL/TLS. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 426–440, Cologne, Germany, September 8–10, 2003. Springer, Heidelberg, Germany.

[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.

[KR02] Vlastimil Klíma and Tomás Rosa. Attack on private signature keys of the OpenPGP format, PGP(TM) programs and other applications compatible with OpenPGP. Cryptology ePrint Archive, Report 2002/076, 2002. http://eprint.iacr.org/.

[KS98] B. Kaliski and J. Staddon. PKCS #1: RSA Cryptography Specifications Version 2.0. RFC 2437 (Informational), October 1998. Obsoleted by RFC 3447.

[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.

[LOS+06] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 465–485, St. Petersburg, Russia, May 28 – June 1, 2006. 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.

[MKJR16] Kathleen Moriarty, Burt Kaliski, Jakob Jonsson, and Andreas Rusch. PKCS #1: RSA Cryptography Specifications Version 2.2. RFC 8017, November 2016.

[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.

[MVO91] Alfred Menezes, Scott A. Vanstone, and Tatsuaki Okamoto. Reducing elliptic curve logarithms to logarithms in a finite field. In Cris Koutsougeras and Jeffrey Scott Vitter, editors, *STOC*, pages 80–89. ACM, 1991.

[Nie02] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 111–126, Santa Barbara, CA, USA, August 18–22, 2002. 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.

[PS96] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 387–398, Saragossa, Spain, May 12–16, 1996. 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.

[Sch90a] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Heidelberg, Germany.

[Sch90b] Claus-Peter Schnorr. Efficient identification and signatures for smart cards (abstract) (rump session). In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology – EUROCRYPT'89*, volume 434 of *Lecture Notes in Computer Science*, pages 688–689, Houthalen, Belgium, April 10–13, 1990. Springer, Heidelberg, Germany.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, January 1991.

[Sch11] Sven Schäge. Tight proofs for signature schemes without random oracles. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 189–206, Tallinn, Estonia, May 15–19, 2011. Springer, Heidelberg, Germany.

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

[Sha84] Adi Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology – CRYPTO'84*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53, Santa Barbara, CA, USA, August 19–23, 1984. Springer, Heidelberg, Germany.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266, Konstanz, Germany, May 11–15, 1997. Springer, Heidelberg, Germany.

[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.

[Wat05] Brent R. Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 114–127, Aarhus, Denmark, May 22–26, 2005. 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.