CRYPTOGRAPHY IN THE PRESENCE OF KEY-DEPENDENT MESSAGES

by

Madeline Gonzalez

A Dissertation Submitted to the Faculty of

The Charles E. Schmidt College of Science

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

Florida Atlantic University

Boca Raton, Florida

December 2009

CRYPTOGRAPHY IN THE PRESENCE OF KEY-DEPENDENT MESSAGES

by

Madeline Gonzalez

This dissertation was prepared under the direction of the candidate's dissertation advisor, Dr. Rainer Steinwandt, Department of Mathematical Sciences, and it has been approved by the members of her supervisory committee. It was submitted to the faculty of the Charles E. Schmidt College of Science and was accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

SUPERVISORY COMITTEE:

[illegible]

Rainer Steinwandt, Ph.D.
Dissertation Advisor

[illegible]

Spyros Magliveras, Ph.D.

[illegible]

Frederick Hoffman, Ph.D.

RC Mullin

Ronald Mullin, Ph.D.

Lee Klingler

Lee Klingler, Ph.D.
Chair, Department of Mathematical Sciences

[illegible]

Gary W. Perry, Ph.D.
Dean, The Charles E. Schmidt College of Science

[illegible]

Barry T. Rosson, Ph.D.
Dean, Graduate College

November 12, 2009

Date

ACKNOWLEDGMENTS

First of all, I would like to thank my family for their love and emotional support throughout all of these years as a starving graduate student, especially my mother Marilyn. I want to thank my husband Luis who has been a constant source of encouragement. It goes without saying that my work would not have been possible without the wisdom, guidance, inspiration, and unending patience of my personal oracle and advisor Rainer. As for the rest of my committee, to Dr. Hoffman, I am grateful for the endless hours of abstract algebra tutoring despite my recalcitrant attitude. One of the reasons that I am a better person is that I had the pleasure of meeting the gardenia-loving Mullins; their smiles warm my heart. Thanks to my favorite Greek, Dr. Magliveras, who reminds me to not just compute, but dream.

Also, I want to thank Dr. Goldwyn for helping me believe in the power of my ideas. Thanks to Dr. Green and Janet for their kindness, avocadoes, and tofu. Many thanks to Bob for his spontaneous affection. A big warm hug with my appreciation to everyone in the math department, my second family. Hugs and kisses to the many pets in my life: Yoda, Casper, Lino, Xena, Blackie, Mocha, Mia and Roger (RIP). I would like to dedicate my dissertation to my grandfather in heaven, Marcelino Muñiz, for loving me just the way that I am, and for believing in me when I didn't believe in myself.

ABSTRACT

Author:	Madeline Gonzalez
Title:	Cryptography in the Presence of Key Dependent Messages
Institution:	Florida Atlantic University
Dissertation Advisor:	Dr. Rainer Steinwandt
Degree:	Doctor of Philosophy
Year:	2009

The aim of this work is to investigate a security model in which we allow an adversary to have access to functions of the secret key. In recent years, significant progress has been made in understanding the security of encryption schemes in the presence of *key-dependent plaintexts or messages* (known as KDM). Here, we motivate and explore the security of a setting, where an adversary against a message authentication code (MAC) or signature scheme can access signatures on key-dependent messages. We propose a way to formalize the security of message authentication schemes in the presence of *key-dependent MACs* (KD-EUF) and of signature schemes in the presence of *key-dependent signatures* (KDS). An attack on a message recognition protocol involving a MAC is presented.

It turns out that the situation is quite different from key-dependent encryption: To achieve KD-EUF-security or KDS-security under non-adaptive chosen message attacks, the use of a stateful signing algorithm is inevitable—even in the random oracle model. After discussing the connection between key-dependent signing and forward security, we describe a compiler which lifts any EUF-CMA secure one-time signature scheme to a forward secure signature scheme offering KDS-CMA security. Then, we discuss how aggregate signatures can be used to combine the signatures in the certificate chain used in the compiler.

A natural question arises about how to combine the security definitions of KDM and KDS to come up with a signcryption scheme that is secure. We also offer a connection with *Leakage-Resilient Signatures*, which take into account side-channel attacks. Lastly, we present some open problems for future research.

CRYPTOGRAPHY IN THE PRESENCE OF KEY-DEPENDENT MESSAGES

Chapter 1

Introduction

Until Eve arrived, this was a man's world. Richard Armour [4]

In 2002, Black, Rogaway, and Shrimpton formalized the notion of encryption security in the presence of key-dependent messages, known as KDM [13]. The paper was motivated by encryption cycles (a cycle is an encryption of the decryption key). Since then, significant progress has been made in understanding the KDM model in the standard setting (without random oracles). The KDM model considers only encryptions of functions of the decrypting key. In recent papers ([5], [22]), an adversary is also given access to decryptions of functions under certain conditions. After a brief introduction to KDM security, we motivate and explore the security of a setting where an adversary against a message authentication code (MAC) or signature scheme can access MACs, or respectively, signatures on key-dependent messages.

1.1 Motivation

Established security notions for encryption schemes like IND-CCA refer to scenarios where encrypted plaintexts do not depend on the secret key. For some scenarios—like encrypting a hard disk storing the secret decryption key—such a security model is inadequate. Here the question of secure encryption in the presence of key-dependent messages naturally arises, and in recent years marked progress in understanding such scenarios has been made (see [13, 32, 6, 33, 34] for instance). For MACs and signature schemes, scenarios with key-dependent messages seem much less understood.

Although perhaps being less obvious than for key-dependent encryption, a scenario where an adversary may have access to MACs on key-dependent messages is not that far-fetched: if we grant an adversary access to the MAC of a (possibly encrypted) backup of a hard disk containing the secret key, then this is a scenario not covered by EUF-CMA security. Both MACs and signature schemes provide data integrity, so a signature of a backup of a hard disk can also be used. Using a signature scheme allows a user to prove to another party whether or not the hard disk has been modified without revealing the secret key. On the other hand, MACs are symmetric, so the secret key must be revealed in order to prove that the hard disk has been tampered with.

Key-dependent signing seems also interesting in connection with *combined public key schemes* as discussed by Haber and Pinkas [31] or González Vasco et al. [56], for instance: Here keys used for decrypting and for signing are not necessarily independent, and signing a key-dependent ciphertext may actually imply signing a key-dependent message.

1.2 Contribution of this Dissertation

Following the notion of key-dependent message (KDM) security proposed by Black et al. [13], as presented in Chapter 2, we propose a formalization of security in the presence of key-dependent MACs and signatures (KD-EUF in Section 3.2.2 and KDS in Section 5.1.1, respectively). For stateless users, a natural definition—where an adversary can obtain MACs and signatures on chosen key-dependent messages—allows no secure realization, even in the random oracle model as proved in Section 3.2.3 and Section 5.1.2. We provide a MAC construction (MAC-ver) secure in the KD-EUF sense in the random oracle model in Section 3.3.

For signatures, a compiler is presented in Section 5.1.4 which transforms any EUF-CMA-secure one-time signature scheme into a (necessarily stateful) KDS-CMA-secure signature scheme, also offering forward security, which is discussed in Chapter 4. In Section 4.2 we show that KDS-security and forward security are related, but independent security goals. In Chapter 6, we show how aggregate signatures can be used to reduce the size of the signature in the KDS-CMA-secure compiler from Section 5.1.4. We also discuss leakage-resilient signatures in Section 1.3 and their connection to KDS-CMA-secure signatures.

In Chapter 7, we show how signatures and encryptions can be composed to form a signcryption that is secure against an adversary who can query functions of the secret key. On the negative side, we show that if the primitive applied first in the composition is not secure against key-dependent messages (or signatures), then the composition is not secure. We conclude with some open problems.

1.3 Leakage-Resilient Signatures

During implementations of cryptosystems, information about the state of the user may be leaked due to side-channel attacks during protocol executions [26]. Micali and Reyzin refer to this as physically observable cryptography [47]. To counter these types of attacks, the hardware may be changed to minimize the leakage of secret data. Another option is to change the algorithms in order to mask the computations that take place. The goal of leakage-resilient cryptography is to design cryptographic schemes that are provably secure even if they are implemented on hardware that may be subject to side-channel attacks. Examples of side-channel attacks include timing information, power consumption, and electromagnetic radiation to name a few. A general assumption is that only computation leaks information.

In modeling leakage resilience, an adversary can adaptively specify a series of *leakage functions* as defined in [35]. Whereas, in the KDS-CMA security definition to be presented the function can be arbitrary; in the leakage resilience model, the functions $\{f_i\}$ used are usually restricted. In addition to obtaining the information specified by the original security definition, the adversary gets the result of applying f_i to the secret key or state of the signer. In the stateless case, the secret key does not change over time, so security cannot be achieved once the total length of the leakage is the length of the secret key. Since we do not restrict the functions, Proposition 5.2 shows why we must avoid the stateless case altogether in the KDS-CMA setting. On the downside, the known leakage-resilient signature constructions bound the number of signatures, whereas in the KDS-CMA setting the number of signatures is not restricted.

When constructing leakage-resilient signatures, the total amount of leakage can be bounded as Katz does in [35], focusing on a stateless algorithm, or the amount of leakage in each invocation can be bounded (but overall can be arbitrarily large) as Faust et al. do in [27]. Faust et al. achieve leakage-resilience by evolving the key, which is reminiscent of FWD-CMA-secure signatures as defined in Chapter 4. However, their construction is not FWD-CMA-secure. Furthermore, they consider stateful signature schemes and define their notion of security as UF-CMLA (unforgeability under chosen message/leakage attacks). Their compiler lifts a 3-time signature scheme to an UF-CMLA-secure signature scheme that can sign a prespecified number of messages.

1.4 Preliminaries and Security Definitions

Cryptography uses protocols that have been created to deal with information security issues such as privacy, data integrity, message authentication, signatures, and non-repudiation. In this section, we define some protocols and state the security goals as in the *Handbook of Applied Cryptography* [44] and *Introduction to Modern Cryptography* [42].

1.4.1 Basic Terminology

Information Security Objectives

- *Privacy* means keeping information secret from all but those who are authorized to see it.
- *Data integrity* ensures information has not been altered by unauthorized or unknown means.
- *Message authentication* corroborates the source of information.
- *Signatures* bind information to an identity.
- *Non-repudiation* prevents the denial of previous commitments or actions.

Communication Participants

- An *entity* or *party* is someone or something which sends, receives, or manipulates information.
- A *sender* is an entity in a two-party communication which is the legitimate transmitter of information.

- A *receiver* is an entity in a two-party communication which is the intended recipient of information.

- An *adversary* is an entity in a two-party communication which is neither the sender nor receiver, and which tries to defeat the information security service being provided between the sender and receiver.

1.4.2 Symmetric Key Encryption

With the security goal of privacy (as with all encryption schemes), the symmetric setting considers two parties who share a key and will use the same key to encrypt and decrypt. An advantage to using symmetric key ciphers is that they can be designed to encrypt large amounts of data efficiently.

Definition 1.1 (Symmetric Key Encryption Scheme)**.** *A symmetric key encryption scheme $S = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ consists of three algorithms, as follows:*

- *The randomized key generation algorithm $\mathcal{K}$ on input of the security parameter 1^k returns a string K which we denote by $K \overset{\$}{\leftarrow} \mathcal{K}(1^k)$.*

- *The randomized encryption algorithm $\mathcal{E}$ takes a key K and a (polynomial length) plaintext $M \in \{0,1\}^*$ to return a ciphertext $C \in \{0,1\}^*$. We write $C \overset{\$}{\leftarrow} \mathcal{E}_K(M)$ to denote this operation.*

- *The deterministic decryption algorithm $\mathcal{D}$ takes a key K and ciphertext C to return a message M or an error symbol $\perp$. We write $M \leftarrow \mathcal{D}_K(C)$ to denote this operation.*

1.4.3 Public Key Encryption

Public key encryption involves the creation of a public key, which a party publishes to the world and a corresponding private key which is kept secret. In order for such a scheme to be secure, it must be computationally infeasible for an adversary to deduce the private key from the public key. Given a ciphertext and the public encryption key, an adversary computationally bounded by polynomial time cannot derive significant information about the plaintext.

Definition 1.2 (Public Key Encryption Scheme). *A public key encryption scheme* $E = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *consists of three algorithms, as follows:*

- $\mathcal{K}$ *is a probabilistic* key generation algorithm *which on input of the security parameter* 1^k *returns a pair* (sk, pk) *of keys—a public encryption key* pk *with matching secret signing key* sk *which is used to decrypt.*

- *The randomized encryption algorithm* $\mathcal{E}$ *takes the public key* pk *and a (polynomial length) plaintext* $M \in \{0,1\}^*$ *to return a ciphertext* $C \in \{0,1\}^*$*.*

- *The deterministic decryption algorithm* $\mathcal{D}$ *takes the secret key* sk *and a ciphertext* C *to return a message* M *or an error symbol* $\perp$*.*

We denote encrypting M with public key pk to get ciphertext C as an output by $C \overset{\$}{\leftarrow} \mathcal{E}_{pk}(M)$. Similarly, decrypting C with secret key sk to get message M is denoted by $M \leftarrow \mathcal{D}_{sk}(C)$.

1.4.4 Digital Signatures

Much like a handwritten signature, a digital signature aims to give a receiver reason to believe that the message was created by a known sender. A digital signature also preserves data integrity and provides non-repudiation. The sender signs message M with secret key sk and sends the receiver both the message and the signature. The receiver uses the verification algorithm to check the validity of the signature on the message with public key pk.

Definition 1.3 (Signature Scheme). *A signature scheme S is a triple of polynomial time algorithms $(\mathcal{K}, \mathcal{S}, \mathcal{V})$:*

- $\mathcal{K}$ *is a probabilistic* key generation algorithm *which on input of the security parameter 1^k returns a pair (sk, pk) of keys—a public verification key pk with matching secret signing key sk. In case of a stateful signer, we interpret sk as the initial state of the signer; i. e., all secret information of the signer is part of its state.*

- $\mathcal{S}$ *is a probabilistic* signing algorithm *which on input a (polynomial length) message $M \in \{0,1\}^*$ and state sk—which in case of a stateless signer is just the secret key—returns a signature $\sigma \in \{0,1\}^*$ on M or an error symbol $\perp$. Moreover, the state value sk is updated.*

- $\mathcal{V}$ *is a deterministic* verification algorithm *which on input a public key pk, a message M, and a candidate signature σ for M returns 1 or 0, indicating whether σ is a valid signature for M under the public key pk.*

For pairs (sk, pk) output by $\mathcal{K}$ we require that with overwhelming probability the obvious correctness condition holds: For all messages M we have $\mathcal{V}_{pk}(M, \mathcal{S}_{sk}(M)) = 1$.

We denote signing message M with signing key sk and getting the signature σ as output by $\sigma \overset{\$}{\leftarrow} \mathcal{S}_{sk}(M)$.

Chapter 2

Key-Dependent Message Encryption

No one can drive us crazy unless we give them the keys. Douglas Horton [25]

2.1 Security Against Chosen-Plaintext Attack

An established notion of security, indistinguishabililty under chosen plaintext attack (IND-CPA) allows the adversary to get encryptions of adaptively chosen messages [29]. The adversary $\mathcal{A}^{\text{ind}}$ wins if it can distinguish between an encryption of its chosen message and an encryption of a random message. $\mathcal{A}^{\text{ind}}$ should output a 1 if it believes that it has participated in experiment *Real*, otherwise it should output 0. We define IND-CPA security in both the symmetric and asymmetric settings.

Definition 2.1 (Symmetric IND-CPA). *Let the triple $E = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric key encryption scheme and $\mathcal{A}^{\text{ind}}$ a probabilistic polynomial time algorithm. Consider the following attack scenario:*

- *Compute a key $K \overset{\$}{\leftarrow} \mathcal{K}(1^k)$.*

- *Let $Real_K$ be the oracle that on input M returns $C \xleftarrow{\$} \mathcal{E}_K(M)$.*
- *Let $Fake_K$ be the oracle that on input M returns $C \xleftarrow{\$} \mathcal{E}_K(0^{|M|})$, where $|M|$ denotes the length of the string M.*

The IND-CPA *advantage of $\mathcal{A}^{\text{ind}}$ is defined as*

$$\text{Adv}_{\mathcal{A}^{\text{ind}}} := \left| \Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{Real_K} = 1] - \Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{Fake_K} = 1] \right|$$

and we call the symmetric key encryption scheme E secure in the sense of IND-CPA *if $\text{Adv}_{\mathcal{A}^{\text{ind}}}$ is negligible in the security parameter k (i.e., it vanishes faster than the inverse of any polynomial [7]).*

Definition 2.2 (Negligible Function). *$\epsilon(k)$ is* negligible *if for every $c \in \mathbb{N}$, there exists some N such that $\epsilon(k) < \frac{1}{k^c}$ for all $k > N$ where $\epsilon : \mathbb{N} \to \mathbb{R}$.*

Definition 2.3 (Asymmetric IND-CPA). *Let the triple $E = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a public key encryption scheme and $\mathcal{A}^{\text{ind}}$ a probabilistic polynomial time algorithm. Consider the following attack scenario:*

- *Compute a key pair $(sk, pk) \xleftarrow{\$} \mathcal{K}(1^k)$, and hand pk as input to $\mathcal{A}^{\text{ind}}$.*
- *Let $Real_{sk}$ be the oracle that on input M returns $C \xleftarrow{\$} \mathcal{E}_{pk}(M)$.*
- *Let $Fake_{sk}$ be the oracle that on input M returns $C \xleftarrow{\$} \mathcal{E}_{pk}(0^{|M|})$, where $|M|$ denotes the length of the string M.*

The IND-CPA *advantage of $\mathcal{A}^{\text{ind}}$ is defined as*

$$\text{Adv}_{\mathcal{A}^{\text{ind}}} := \left| \Pr[(sk, pk) \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{Real_{sk}} = 1] - \Pr[(sk, pk) \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{Fake_{sk}} = 1] \right|$$

and we call the public key encryption scheme E secure in the sense of IND-CPA *if* $\mathrm{Adv}_{\mathcal{A}^{\mathrm{ind}}}$ *is negligible in the security parameter k.*

A public key encryption scheme that is secure in the IND-CPA sense can be trivially modified and still remain IND-CPA secure yet completely insecure, when an adversary is somehow handed an encryption of the secret key, as pointed out by Black, Rogaway, and Shrimpton [13]. For example, modify $\mathcal{E}_{pk}(M)$ to $0 \ \| \ \mathcal{E}_{pk}(M)$ if $M \neq sk$, and $1 \ \| \ sk$ otherwise. An encryption of the decryption key is called an *encryption cycle of length one.*

Concurrently and independently, Camenisch and Lysyanskaya proposed encryption schemes for key-dependent messages, which they call circular encryption in [23]. In their setting, for key pairs (sk_i, pk_i) and (sk_j, pk_j), an adversary can get encryptions such as $\mathcal{E}_{pk_i}(sk_j)$ and $\mathcal{E}_{pk_j}(sk_i)$. As Boneh et al. [19] point out, while using BitLocker (a disk encryption utility used in Windows Vista), the disk encryption key can end up on the disk and be encrypted along with the other contents, so encryption cycles are a real concern.

2.2 KDM Symmetric Key Encryption Security

In encryption security, KDM stands for *key-dependent messages.* Informally, an encryption scheme is KDM-CPA secure in the sense of indistinguishability if it is secure despite an adversary's ability to obtain encryptions of functions of the secret key, in particular the identity function, which would be an encryption of the secret key. The adversary $\mathcal{A}^{\mathrm{kdm}}$ queries an efficiently computable function g on secret key K. In the case that $\mathcal{A}^{\mathrm{kdm}}$ wants to query a particular message M, then g is a constant function.

Definition 2.4 (Symmetric KDM-CPA)**.** *Let the triple $E = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric* *key encryption scheme and $\mathcal{A}^{\mathrm{kdm}}$ a probabilistic polynomial time algorithm. Consider the following attack scenario:*

- *Compute a key pair $K \xleftarrow{\$} \mathcal{K}(1^k)$.*
- *Let $Real_K$ be the oracle that on input g returns $C \xleftarrow{\$} \mathcal{E}_K(g(K))$.*
- *Let $Fake_K$ be the oracle that on input g returns $C \xleftarrow{\$} \mathcal{E}_K(0^{|g(K)|})$.*

The KDM-CPA *advantage of $\mathcal{A}^{\mathrm{kdm}}$ is defined as*

$$\mathrm{Adv}_{\mathcal{A}^{\mathrm{kdm}}} := \left| \Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{Real_K} = 1] - \Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{Fake_K} = 1] \right|$$

and we call the symmetric key encryption scheme E secure in the sense of KDM-CPA *if $\mathrm{Adv}_{\mathcal{A}^{\mathrm{kdm}}}$ is negligible in k.*

In [13], Black et al. prove that their proposed symmetric encryption scheme **ver** is secure in the random oracle model. In the scheme **ver**, an oracle $H \in \Omega$ is given where Ω is the set of all functions from $\{0,1\}^*$ to $\{0,1\}^\infty$.

Definition 2.5 (The scheme **ver**)**.** *Define the symmetric encryption scheme **ver** $= (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with security parameter k, message space $\{0,1\}^*$ and key space $\{0,1\}^k$ through*

- *$\mathcal{K}(1^k)$ outputs a uniform random key $K \in \{0,1\}^k$.*
- *$\mathcal{E}_K(M)$ samples $R \xleftarrow{\$} \{0,1\}^k$ and outputs the ciphertext $(R, H(K \parallel R) \oplus M)$.*
- *$\mathcal{D}_K(R, D)$ outputs the message $H(K \parallel R) \oplus D$.*

We will use a similar construction in the next chapter to create a message authentication code that is secure despite key-dependent messages.

2.3 KDM Public Key Encryption Security

In this section, the adversary $\mathcal{A}^{\mathrm{kdm}}$ queries an efficiently computable function g on secret key sk.

Definition 2.6 (Asymmetric KDM-CPA). *Let the triple $E = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a public key encryption scheme and $\mathcal{A}^{\mathrm{kdm}}$ a probabilistic polynomial time algorithm. Consider the following attack scenario:*

- *Compute a key pair $(sk, pk) \xleftarrow{\$} \mathcal{K}(1^k)$, and hand pk as input to $\mathcal{A}^{\mathrm{kdm}}$.*
- *Let $Real_{sk}$ be the oracle that on input g returns $C \xleftarrow{\$} \mathcal{E}_{pk}(g(sk))$.*
- *Let $Fake_{sk}$ be the oracle that on input g returns $C \xleftarrow{\$} \mathcal{E}_{pk}(0^{|g(sk)|})$.*

The KDM-CPA *advantage of $\mathcal{A}^{\mathrm{kdm}}$ is defined as*

$$\mathrm{Adv}_{\mathcal{A}^{\mathrm{kdm}}} := \left| \Pr[(sk, pk) \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{Real_{sk}}(pk) = 1] - \Pr[(sk, pk) \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{Fake_{sk}}(pk) = 1] \right|$$

and we call the public key encryption scheme E secure in the sense of KDM-CPA *if* $\mathrm{Adv}_{\mathcal{A}^{\mathrm{kdm}}}$ *is negligible in k.*

In [13], Black et al. propose the asymmetric encryption scheme **VER** in the random oracle model as follows. As defined in [9], a family of trapdoor permutations is a family of permutations such that

- It is easy, given an integer k, to randomly select a permutation f which has security parameter k, together with some trapdoor information associated with f.
- f is easy to compute, and, given the trapdoor information, so is f^{-1}; but without the trapdoor information, f is hard to invert at a random input.

A function f is easy to compute if there exists a polynomial-time algorithm such that on input x outputs $f(x)$.

Definition 2.7 (The scheme **VER**). *Define the asymmetric encryption scheme* $\boldsymbol{VER} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *with security parameter* k*, message space* $\{0,1\}^*$ *and key space* $\{0,1\}^k$ *through*

- $\mathcal{F}(1^k)$ *is a trapdoor permutation generator that returns functions* (f, f^{-1}) *where* $f : \{0,1\}^k \to \{0,1\}^k$ *and* $f^{-1} : \{0,1\}^k \to \{0,1\}^k$ *is its inverse.*
- $\mathcal{K}(1^k)$ *is the same as* $\mathcal{F}(1^k)$*.*
- $\mathcal{E}_f(M)$ *samples* $R \overset{\$}{\leftarrow} \{0,1\}^k$ *and outputs the ciphertext* $(f(R), H(R) \oplus M)$*.*
- *Let* $D = Y \parallel C$ *where* $|Y| = k$ *then* $\mathcal{D}_{f^{-1}}(D)$ *outputs the message* $H(f^{-1}(Y)) \oplus C$ *or* $\perp$ *if* $|D| < k$*.*

In [34], Hofheinz and Unruh explore the KDM security notion in the standard model (without random oracles). As in [24], one first designs an ideal system in which all parties (including the adversary) have oracle access to a truly random function, and proves the security of this ideal system. This is known as the Random Oracle Model (RO). In [22], Camenisch et al. explore the key-dependent setting in which an adversary has access to decryptions of chosen ciphertexts, as do Backes et al. in [5].

2.4 Key Privacy

We now consider a security requirement known as key privacy or anonymity. An eavesdropper in possession of a ciphertext should not be able to tell which specific key, out of a set of known public keys, is the one under which the ciphertext was created.

To formalize the property of key privacy, we use the notion of indistinguishability of keys (IK-CPA) as in [8].

Definition 2.8 (IK-CPA). *Let $E = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an encryption scheme and $\mathcal{A}^{\mathrm{ik}}$ a probabilistic polynomial time algorithm. Consider the following experiment, which we denote by $\mathrm{Exp}^{\mathrm{ik-b}}_{E,\ \mathcal{A}^{\mathrm{ik}}}$:*

1. *Compute key pairs $(sk_0, pk_0) \overset{\$}{\leftarrow} \mathcal{K}(1^k)$ and $(sk_1, pk_1) \overset{\$}{\leftarrow} \mathcal{K}(1^k)$, and hand pk_0 and pk_1 as input to $\mathcal{A}^{\mathrm{ik}}$.*
2. *$\mathcal{A}^{\mathrm{ik}}$ outputs a message M of its choice and some state information s.*
3. *For $b \overset{\$}{\leftarrow} \{0, 1\}$ compute $y \overset{\$}{\leftarrow} \mathcal{E}_{pk_b}(M)$.*
4. *Hand y and s to $\mathcal{A}^{\mathrm{ik}}$.*
5. *$\mathcal{A}^{\mathrm{ik}}$ returns d.*

The IK-CPA advantage of $\mathcal{A}^{\mathrm{ik}}$ is defined as

$$\mathrm{Adv}_{\mathcal{A}^{\mathrm{ik}}} := \left| \Pr[\mathrm{Exp}^{\mathrm{ik-1}}_{E,\ \mathcal{A}^{\mathrm{ik}}} = 1] - \Pr[\mathrm{Exp}^{\mathrm{ik-0}}_{E,\ \mathcal{A}^{\mathrm{ik}}} = 1] \right|$$

and we call the public key encryption scheme E secure in the sense of IK-CPA*, if $\mathrm{Adv}_{\mathcal{A}^{\mathrm{ik}}}$ is negligible in the security parameter k.*

Since the scheme **VER** is KDM-CPA-secure, in order to have key-privacy, we need a family of trapdoor permutations that is anonymous. We now present an RSA-based scheme from Bellare et al. in [8] that will do the job. First we note that the standard RSA encryption does not provide key privacy. The ciphertext is an element $y = x^e \bmod N$ where x is a random member of $\mathbb{Z}_N^*$. The goal of the adversary is to distinguish between two keys N_0, e_0, and N_1, e_1. Now suppose that $N_0 \leq N_1$. If $y \geq N_0$ then the adversary bets that the encryption was created under N_1, e_1; otherwise, it bets that it was created under N_0, e_0.

A family of functions $\mathcal{F} = (K, S, E)$ is specified by three algorithms: randomized key-generation K, sampling algorithm S, and evaluation algorithm E. The following construction from [8] is a simple RSA-derived anonymous family.

Construction 2.1 (Anonymous Trapdoor Permutation). *We define a family $\mathcal{F} = (K, S, E)$ as follows. The key generation algorithm K takes as input a security parameter k and picks random, distinct primes p, q in the range $2^{k/2-1} < p, q < 2^{k/2}$. If k is odd, it is incremented by 1 before the primes are picked. It sets $N = pq$ and picks $e, d \in \mathbb{Z}^*_{\varphi(N)}$ such that $ed \equiv 1 \mod \varphi(N)$ where $\varphi(N) = (p-1)(q-1)$. The public key is (N, e) and the secret key is (N, d). We set the domain and range of $\mathcal{F}$ to $\{0,1\}^k$, and viewing $\mathbb{Z}^*_N$ as a subset, we define*

$$E_{N,e}(x) = \begin{cases} x^e \mod N & \textit{if } x \in \mathbb{Z}^*_N; \\ x & \textit{otherwise}; \end{cases}$$

for any $x \in \{0,1\}^k$. This is a permutation on $\{0,1\}^k$. The sampling algorithm S on input N, e simply returns a random k-bit string.

Using the sampling algorithm S, we can select a random k-bit string in the range of $\mathcal{F}$. Without the trapdoor information (the secret key), this random element in the range should be hard to invert. The inversion algorithm is

$$I_{N,d}(x) = \begin{cases} x^d \mod N & \text{if } x \in \mathbb{Z}^*_N; \\ x & \text{otherwise.} \end{cases}$$

If $x \in \mathbb{Z}^*_N$, then we are in the standard RSA family of trapdoor permutations. If $x \notin \mathbb{Z}^*_N$, the inversion is trivial and occurs with negligible probability in the security parameter, as shown by inequality that follows.

$\Pr[x \notin \mathbb{Z}_N^*] \leq 1 - \frac{(p-1)(q-1)}{pq} = \frac{p+q-1}{pq} < \frac{2^{k/2+1}-1}{2^k} < \frac{2^{k/2+1}}{2^k} = \frac{1}{2^{k/2-1}}$

As an open problem, the security of key-privacy can be studied where key-dependent messages are allowed in Step 2 of Definition 2.8.

Chapter 3

Key Dependence in Message Authentication Codes

The most authentic thing about us is our capacity to create, to overcome, to endure, to transform, to love and to be greater than our suffering. Ben Okri [50]

3.1 MACs

Informally, a *hash function* takes a message of arbitrary finite length as input and produces a string of fixed length as output (known as a message digest or hash). A distinct class of hash functions, message authentication codes (MACs) allow message authentication by symmetric techniques; by symmetric, we mean that there is only one key which is secret and is used both for the creation of the MAC and verification. A MAC is an example of a *keyed hash function* since producing it dictates the use of a message and a secret key as input. The purpose of a MAC is to provide assurances regarding both the source of a message and its integrity.

Definition 3.1 (MAC-Generation Algorithm)**.** *A message authentication code algorithm is a family of functions $\mathcal{T}_K$ parametrized by a secret key K, with the following properties:*

1. *ease of computation—for a known function $\mathcal{T}_K$, given a value K and an input M, $\mathcal{T}_K(M)$ is easy to compute.*
2. *compression—$\mathcal{T}_K$ maps an input M of arbitrary finite bit length to an output $\mathcal{T}_K(M)$ of fixed bit length n.*

   Furthermore, given a description of the function family $\mathcal{T}$, for every fixed allowable value of K (unknown to an adversary), the following property holds:
3. *computation-resistance—given zero or more message/tag pairs $(M_i, \mathcal{T}_K(M_i))$, it is computationally infeasible to compute any message/tag pair $(M, \mathcal{T}_K(M))$ for any new input $M \neq M_i$ (including possibly for $\mathcal{T}_K(M) = \mathcal{T}_K(M_i)$ for some i).*

The objective of the MAC-generation algorithm adversary is to compute a new message/tag pair $(M, \mathcal{T}_K(M))$ for some message $M \neq M_i$, given one or more pairs $(M_i, \mathcal{T}_K(M_i))$. We allow an *adaptive chosen-message attack* in which the M_i may be chosen by the adversary successively and can be based on the results of prior queries. The adversary is successful if it is able to produce an existential forgery; that is, a new message/tag pair, without necessarily controlling the value of the message.

Definition 3.2 (MAC)**.** *A message authentication code Π is a triple of polynomial time algorithms $(\mathcal{K}, \mathcal{T}, \mathcal{V})$:*

- *The randomized key generation algorithm $\mathcal{K}$ returns a string K on input of the security parameter 1^k, and we denote it by $K \xleftarrow{\$} \mathcal{K}(1^k)$.*

- *The MAC-generation algorithm $\mathcal{T}$, which might be randomized or stateful, takes a key K and a (polynomial length) message $M \in \{0,1\}^*$ to return a tag $T \in \{0,1\}^* \cup \{\perp\}$, and we denote it by $T \xleftarrow{\$} \mathcal{T}_K(M)$.*

- *The deterministic MAC-verification algorithm $\mathcal{V}$ takes a key K, a message $M \in \{0,1\}^*$ and a candidate tag $T \in \{0,1\}^*$ to return either 1 (Accept) or 0 (Reject). We write $d \leftarrow \mathcal{V}_K(M,T)$ with d denoting the decision bit returned.*

With overwhelming probability, we require that for any key K, any message $M \in \{0,1\}^*$, and tag $T \xleftarrow{\$} \mathcal{T}_K(M)$, $\mathcal{V}_K(M,T) = 1$.

When the MAC-generation algorithm is deterministic and stateless, the receiver computes $T' \leftarrow \mathcal{T}_K(M)$ for received message M and checks if $T' = T$. An adversary may repeat a transmission of a valid pair (M,T) and get the receiver to accept it once again; this is known as a *replay attack*. In the definition of security that we present, we do not consider this a valid forgery.

3.2 Security Definitions

3.2.1 MAC Security

EUF-CMA stands for existential unforgeability against chosen message attacks and we will use it again in the context of signatures. In the sections that follow, we explore the consequences of allowing key-dependent queries of a secure MAC. We also propose a MAC that is secure in the RO Model despite key-dependent queries.

Definition 3.3 (EUF-CMA)**.** *Let $\Pi = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ be a message authentication code, and let $\mathcal{A}^{\text{mac}}$ be a probabilistic polynomial time algorithm. Consider the following attack scenario:*

1. *Compute a key* $K \overset{\$}{\leftarrow} \mathcal{K}(1^k)$.

2. *The adversary* $\mathcal{A}^{\mathrm{mac}}$ *is given unrestricted access to a MAC-generation oracle* $\mathcal{O}_{\mathcal{T}}$ *and verification oracle* $\mathcal{O}_{\mathcal{V}}$ *to run* $\mathcal{T}_K$ *and* $\mathcal{V}_K$.

3. *Eventually,* $\mathcal{A}^{\mathrm{mac}}$ *outputs a message/tag pair* (M, T).

Let QueriedEarlier *be the event that* $\mathcal{A}^{\mathrm{mac}}$ *outputs a message* M *that has been queried to the MAC-generation oracle* $\mathcal{O}_{\mathcal{T}}$ *already. The* success probability $\mathrm{Succ}_{\mathcal{A}}^{\mathrm{mac}} = \mathrm{Succ}_{\mathcal{A}^{\mathrm{mac}}}(k)$ *of* $\mathcal{A}^{\mathrm{mac}}$ *is defined as*

$$\mathrm{Succ}_{\mathcal{A}}^{\mathrm{mac}} := \Pr[\mathcal{V}_K(M, T) = 1 \textit{ and } \neg\mathsf{QueriedEarlier}],$$

and we call the MAC EUF-CMA *secure if* $\mathrm{Succ}_{\mathcal{A}}^{\mathrm{mac}}$ *is negligible for all probabilistic polynomial time adversaries* $\mathcal{A}^{\mathrm{mac}}$.

3.2.2 KD-EUF Security

Informally, a MAC Π is KD-EUF *(key-dependent MAC)* secure if it is secure despite a forger's ability to obtain tags on arbitrary (efficiently computable) functions g of the state K. Unlike a digital signature, the verification of a MAC requires knowledge of the secret key, so we must provide our adversary $\mathcal{A}^{\mathrm{kdeuf}}$ with a verification oracle in addition to the key-dependent MAC-generation (tag) oracle.

Definition 3.4 (KD-EUF)**.** *Let* $\Pi = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ *be a message authentication code, and let* $\mathcal{A}^{\mathrm{kdeuf}}$ *be a probabilistic polynomial time algorithm. Consider the following attack scenario:*

1. *Compute a key* $K \overset{\$}{\leftarrow} \mathcal{K}(1^k)$.

2. *The adversary $\mathcal{A}^{\mathrm{kdeuf}}$ is given unrestricted access to a MAC-generation oracle $\widehat{\mathcal{O}}_{\mathcal{T}}$ and verification oracle $\mathcal{O}_{\mathcal{V}}$ to run $\mathcal{T}_K$ and $\mathcal{V}_K$. The oracle $\widehat{\mathcal{O}}_{\mathcal{T}}$ accepts as input a function g, represented as a boolean circuit of polynomial size, and executes the MAC-generation algorithm $\mathcal{T}$ with the current state K and the message $g(K)$ as input.*[1]

3. *Eventually, $\mathcal{A}^{\mathrm{kdeuf}}$ outputs a message $M \in \{0,1\}^*$ and a tag T.*

Let QueriedEarlier *be the event that $\mathcal{A}^{\mathrm{kdeuf}}$ outputs a message M such that one of $\mathcal{A}^{\mathrm{kdeuf}}$'s queries g to the signing oracle $\widehat{\mathcal{O}}_{\mathcal{T}}$ evaluated to $g(K) = M$. Then the* success probability $\mathrm{Succ}_{\mathcal{A}^{\mathrm{kdeuf}}} = \mathrm{Succ}_{\mathcal{A}^{\mathrm{kdeuf}}}(k)$ *of $\mathcal{A}^{\mathrm{kdeuf}}$ is defined as*

$$\mathrm{Succ}_{\mathcal{A}^{\mathrm{kdeuf}}} := \Pr[\mathcal{V}_K(M,T) = 1 \text{ and } \neg\mathsf{QueriedEarlier}],$$

and we call the MAC Π secure in the sense of KD-EUF *if* $\mathrm{Succ}_{\mathcal{A}^{\mathrm{kdeuf}}}$ *is negligible for all probabilistic polynomial time adversaries $\mathcal{A}^{\mathrm{kdeuf}}$.*

3.2.3 Impossibility of Stateless KD-EUF Secure MAC

As a negative result, we note that no MAC with a stateless MAC-generation algorithm can meet the security goal of KD-EUF with a simple attack.

Remark 3.1. *Let $\Pi = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ be a MAC with a stateless MAC-generation algorithm $\mathcal{T}$; i.e., the secret state K is not changed by executing $\mathcal{T}$. Then the MAC Π is not secure in the sense of* KD-EUF.

Proof. Let $K = b_0, \ldots, b_{\ell-1} \in \{0,1\}^\ell$ be the bit representation of the secret key and fix $i \in \{0, \ldots, \ell-1\}$ arbitrary. Then the adversary $\mathcal{A}^{\mathrm{kdeuf}}$ may query $\widehat{\mathcal{O}}_{\mathcal{T}}$ for a tag on b_i and use the verification oracle $\mathcal{O}_{\mathcal{V}}$ to determine if the returned tag T satisfies

[1]In the random oracle model, g may invoke the random oracle.

$\mathcal{V}_K(0,T) = 1$ or $V_K(1,T) = 1$. Thus, ℓ queries to $\widehat{\mathcal{O}}_{\mathcal{T}}$ are sufficient to extract the complete secret signing key K, and hereafter, creating a forgery is trivial.

$\square$

Note that only ℓ queries are necessary, because $\mathcal{V}_K(0,T) \neq 1$ implies that $\mathcal{V}_K(1,T) = 1$. Unlike encryption schemes, MACs and digital signatures have verification algorithms. As one can see from Remark 3.1, the verification algorithm is a very powerful tool for the adversaries in MACs and signatures, when functions of the secret key can be summoned.

3.3 The MAC-ver Construction

In this section, we define a stateful MAC that we prove is KD-EUF-secure in the random oracle model. Although proving MACs secure in the random oracle model may seem trivial, the added element of key-dependent messages makes the proof more involved.

Definition 3.5 (The scheme **MAC-ver**). *Define the stateful MAC scheme* ***MAC-ver*** $= (\mathcal{K}, \mathcal{T}, \mathcal{V})$ *with security parameter k, message space $\{0,1\}^*$, key space $\{0,1\}^k$, and oracle $H : \{0,1\}^* \to \{0,1\}^k$ through*

- $\mathcal{K}(1^k)$ *outputs a uniform random key $K \in \{0,1\}^k$.*
- *The sender runs $\mathcal{T}_K(M)$ which samples $R \xleftarrow{\$} \{0,1\}^k$, outputs the tag $T := (R, H(0 \parallel M \parallel R \parallel K))$, and sets $K := H(K \parallel R)$.*
- *If the receiver runs $\mathcal{V}_K(M,T)$ and verifies that $D = H(0 \parallel M \parallel R \parallel K)$ on input $T = (R, D)$, it sets $K := H(K \parallel R)$ and outputs 1. Otherwise $\mathcal{V}_K(M,T)$ outputs 0.*

We make the assumption that the receiver verifies the tag and updates the secret key before a new tag is created.

Proposition 3.1. *The scheme* $\boldsymbol{MAC\text{-}ver} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ *as in Definition 3.5 is secure in the sense of* $\mathsf{KD\text{-}EUF}$ *in the random oracle model.*

Proof. We will create a series of games in which we alter the environment of the adversary. During each transition, the adversary may only gain a negligible advantage; hence, the probability of creating a forgery differs negligibly. Suppose that the adversary $\mathcal{A}^{\text{kdeuf}}$ can forge with non-negligible probability p, and let q be a polynomial upper bound on the number of queries that $\mathcal{A}^{\text{kdeuf}}$ makes to the oracle (tag + direct RO queries).

Game 0. This is a trivial simulation of the original scheme. All needed oracles for $\mathcal{A}^{\text{kdeuf}}$ can be simulated faithfully.

Random oracle: To simulate $\mathcal{A}^{\text{kdeuf}}$'s random oracle, we create an empty list L^{RO}. Then, whenever $\mathcal{A}^{\text{kdeuf}}$ queries its random oracle with a message X such that L^{RO} contains no entry of the form $(X, \cdot)$, we choose a value $H(X) \in \{0,1\}^k$ uniformly at random, append the pair $(X, H(X))$ to L^{RO} and send $H(X)$ to $\mathcal{A}^{\text{kdeuf}}$. In case $\mathcal{A}^{\text{kdeuf}}$ queries L^{RO} a second time with the same value X, we return the stored random value $H(X)$. We assume that $\mathcal{A}^{\text{kdeuf}}$ does not repeat a direct RO query. We define Domain(H) to be the set of points X where an entry of the form $(X, \cdot)$ is in L^{RO}.

Tagging and verification oracle: Knowing the secret key, we can faithfully answer all tag queries $\widehat{\mathcal{O}}_{\mathcal{T}}$ and verification queries $\mathcal{O}_{\mathcal{V}}$, by executing $\mathcal{T}$ and $\mathcal{V}$ respectively with the appropriate input and using the above simulation of the random oracle H.

Game 1. Let Collision be the event that during the simulation, the pairs $(X, H(X))$ and $(X', H(X'))$ in L^{RO} are stored, where $X \neq X'$ and $H(X) = H(X')$. Whenever the event Collision occurs, the simulation is restarted. As $\mathcal{A}^{\mathrm{kdeuf}}$ is polynomially bounded, Collision occurs with negligible probability only, and subsequently, we may assume that the event Collision does not occur.

Game 2. In this game, we pick a value $j \in \{0, \ldots, q\}$ uniformly at random, where $\mathcal{A}^{\mathrm{kdeuf}}$ may forge. If $\mathcal{A}^{\mathrm{kdeuf}}$ does not forge during the j^{th} query, we abort. Since the number of queries is polynomially bounded, $\mathcal{A}^{\mathrm{kdeuf}}$ can still forge with non-negligible probability, provided it was non-negligible before.

In simulating the tag oracle $\widehat{\mathcal{O}}_{\mathcal{T}}$, we claim that providing the adversary with $H(R \parallel K)$ instead of $H(0 \parallel M \parallel R \parallel K)$ during the j^{th} query does not significantly change $\mathcal{A}^{\mathrm{kdeuf}}$'s ability to forge. There are two cases to consider during the j^{th} query: $\mathcal{A}^{\mathrm{kdeuf}}$ can (Case 1) or cannot (Case 2) predict $g(K_j)$ with non-negligible probability.

Case 1: Suppose that $\mathcal{A}^{\mathrm{kdeuf}}$ can predict the value of $g(K_j)$ with non-negligible probability. If the query is key-dependent, we modify $\mathcal{A}^{\mathrm{kdeuf}}$ and force it to replace $g(K_j)$ with a key-independent query M. Note that the adversary wins if the verification algorithm accepts a tag for a message not previously summoned from the tagging oracle. Therefore, $\mathcal{A}^{\mathrm{kdeuf}}$ will not be able to forge if it verifies a message/tag pair returned from the tagging oracle, since this automatically updates the secret key of the scheme. Thus, without loss of generality, we can assume that $\mathcal{A}^{\mathrm{kdeuf}}$ does not verify the tag for message M.

Suppose that $\mathcal{A}^{\mathrm{kdeuf}}$ can distinguish between $H(0 \parallel M \parallel R \parallel K_j)$ and $H(R \parallel K_j)$ without using the verification oracle. Since the key K_j has not been used in a previous tag, then $\mathcal{A}^{\mathrm{kdeuf}}$ could only distinguish between the two values by using direct RO queries. Although $\mathcal{A}^{\mathrm{kdeuf}}$ knows M (with non-negligible probability) and R, this would also imply that $\mathcal{A}^{\text{kdeuf}}$ knows K_j. Since K_j is chosen fresh for each tag, $\mathcal{A}^{\text{kdeuf}}$ can guess K_j with probability of at most $1/2^k$, which is negligible. Since 0 is not prepended in $H(R \parallel K_j)$, the hash value will never be a valid tag for any M, so we are not creating a collision with any possible tag by replacing $H(0 \parallel M \parallel R \parallel K_j)$ with $H(R \parallel K_j)$. Hence, the substitution will not be noticed by $\mathcal{A}^{\text{kdeuf}}$.

Case 2: Suppose that $\mathcal{A}^{\text{kdeuf}}$ has a negligible probability of predicting the value $M = g(K_j)$. Verifying the tag for message M would contradict $\mathcal{A}^{\text{kdeuf}}$ being able to forge during the j^{th} query. Since $\mathcal{A}^{\text{kdeuf}}$ has a negligible probability of predicting the value M, $\mathcal{A}^{\text{kdeuf}}$'s probability of verifying the tag for M is also negligible. Therefore, without loss of generality, we may assume that $\mathcal{A}^{\text{kdeuf}}$ does not verify the tag for M. Similar to Case 1, $\mathcal{A}^{\text{kdeuf}}$ can only distinguish between $H(0 \parallel M \parallel R \parallel K_j)$ and $H(R \parallel K_j)$ using direct oracle queries with negligible probability. Hence, substituting $H(0 \parallel M \parallel R \parallel K_j)$ with $H(R \parallel K_j)$ will not be noticed by $\mathcal{A}^{\text{kdeuf}}$.

Game 3. In this game, we claim that there is no need to faithfully simulate the key update in the scheme. Given a tag $T = (R, D)$, the new key $H(K \parallel R)$ should be indistinguishable from a random k-bit string. Given $(R, H(R \parallel K))$ (instead of $H(0 \parallel M \parallel R \parallel K)$, due to *Game 2*), can $\mathcal{A}^{\text{kdeuf}}$ distinguish between $H(K \parallel R)$ and a random k-bit string where R is given and $|K| = k$? Since $K = R$ with probability at most $1/2^k$, which is negligible, we can assume that $K \neq R$ (otherwise distinguishing becomes trivial). Since we assumed from *Game 1* that the event $\mathsf{Collision}$ does not occur, we have that $H(K \parallel R)$ is not equal to an element previously output by H. As a result, $\mathcal{A}^{\text{kdeuf}}$ cannot distinguish between $H(K \parallel R)$ and a random k-bit string, so there is no need to faithfully simulate the key update in $\mathcal{T}$ or $\mathcal{V}$.

Game 4. In Figure 3.1, we have the final simulation for the scheme **MAC-ver**. The verification algorithm stays the same for $i \neq j$ and we assume from *Game 2* that $\mathcal{A}^{\mathrm{kdeuf}}$ does not verify the tags received from the tagging oracle. The flag *bad* gets set to true when the event $\mathsf{Collision}$ occurs either through a direct RO query or a tag query of which there are at most q altogether. There are $\binom{q}{2} = \frac{q(q-1)}{2}$ pairs where a collision can occur for q queries from a set of size 2^k. Clearly, $\frac{q(q-1)}{2} \leq q^2$ for $q \geq 0$. Therefore, $\mathcal{A}^{\mathrm{kdeuf}}$ can only succeed in distinguishing between a faithful simulation of the original scheme and the simulation in this game if the flag *bad* gets set to true, which occurs with probability less than or equal to $q^2/2^k$ (which is negligible).

Suppose that $\mathcal{A}^{\mathrm{kdeuf}}$ creates a forgery $(M_F, (R_F, D_F))$. If $0 \parallel M_F \parallel R_F \parallel K_j \notin$ Domain(H), then $H(0 \parallel M_F \parallel R_F \parallel K_j) \xleftarrow{\$} \{0,1\}^k$, and the probability that $D_F = H(0 \parallel M_F \parallel R_F \parallel K_j)$ is $1/2^k$, which is negligible. If $0 \parallel M_F \parallel R_F \parallel K_j \in$ Domain(H), then we need to consider two cases: either $0 \parallel M_F \parallel R_F \parallel K_j \in \mathcal{X}$ or $0 \parallel M_F \parallel R_F \parallel K_j \notin \mathcal{X}$. If $0 \parallel M_F \parallel R_F \parallel K_j \in \mathcal{X}$; then the tagging oracle has already signed this message, which contradicts a forgery.

Therefore, $0 \parallel M_F \parallel R_F \parallel K_j \notin \mathcal{X}$ which means that the hash value was assigned through a direct RO query by $\mathcal{A}^{\mathrm{kdeuf}}$. In turn, this implies that $\mathcal{A}^{\mathrm{kdeuf}}$ knows the full key K_j given $(R, H(R \parallel K_j))$. Since we assumed that the event $\mathsf{Collision}$ does not occur, then $\mathcal{A}^{\mathrm{kdeuf}}$ gets K_j by computing the preimage of $H(R \parallel K_j)$. Since $H(R \parallel K_j)$ is a random element and $|K_j| = k$, then the probability of $\mathcal{A}^{\mathrm{kdeuf}}$ computing the preimage of $H(R \parallel K_j)$ is negligible in k. This is a contradiction to $\mathcal{A}^{\mathrm{kdeuf}}$ forging with non-negligible probability p. Hence, the scheme **MAC-ver** is secure in the random oracle.

□

```
Initialization:
    for $i \leftarrow 1$ to $q$ do $K_i \overset{\$}{\leftarrow} \{0,1\}^k$
    $bad \leftarrow$ false; $\mathcal{X} \leftarrow \emptyset$
On RO Query $Q$:
    if $Q \in \mathcal{X}$ then $bad \leftarrow$true
    if $Q \notin$ Domain($H$) then $H(Q) \overset{\$}{\leftarrow} \{0,1\}^k$
    return $H(Q)$
On Tag Query $(M, i)$ for $i \neq j$:
    $R \overset{\$}{\leftarrow} \{0,1\}^k$
    $D \overset{\$}{\leftarrow} \{0,1\}^k$
    if $0 \parallel M \parallel R \parallel K_i \in$ Domain($H$) then $bad \leftarrow$ true
      $D \leftarrow H(0 \parallel M \parallel R \parallel K_i)$ (from list $L^{\mathrm{RO}}$)
    else
      $H(0 \parallel M \parallel R \parallel K_i) \leftarrow D$
      $\mathcal{X} \leftarrow \mathcal{X} \cup \{0 \parallel M \parallel R \parallel K_i\}$
    return $(R, D)$
On Tag Query $(M, j)$:
    $R \overset{\$}{\leftarrow} \{0,1\}^k$
    $D \overset{\$}{\leftarrow} \{0,1\}^k$
    if $R \parallel K_j \in$Domain($H$) then $bad \leftarrow$ true
      $D \leftarrow H(R \parallel K_j)$ (from list $L^{\mathrm{RO}}$)
    else
      $H(R \parallel K_j) \leftarrow D$
      $\mathcal{X} \leftarrow \mathcal{X} \cup \{R \parallel K_j\}$
    return $(R, D)$
```

Figure 3.1: Simulation of scheme from *Game 4*.

3.4 MACs in Mashatan and Stinson's Message Recognition Protocol

This section details an attack on a message recognition protocol that has withstood multiple independent reviews. The execution of the protocol uses a MAC to bind a message to a password. Since the hash appends the password after the message, the MAC used is known as the secret suffix method [51].

In [43], Mashatan and Stinson propose *a new message recognition protocol for ad hoc pervasive networks*, aiming at scenarios with resource restricted devices. Their protocol relies on the use of a cryptographic hash function providing suitable guarantees, and the protocol avoids the use of asymmetric cryptography. In informal terms, the scenario in [43] can be summarized as follows: during an initialization phase, two parties A and B are connected through an authentic channel of low bandwidth. While this narrow-band channel can be eavesdropped, the adversary is confined to be *passive*; i. e., no messages can be altered, deleted or inserted. Later on, A and B are connected via a public broadband channel that is completely controlled by the, now *active*, adversary. The protocol in [43] tries to make sure that messages sent over this public insecure channel by A are only accepted by B if they indeed originate from the party A with which the initialization phase was performed. Further, according to [43], the proposed protocol *provides a practical procedure for resynchronization in case of any adversarial disruption or communication failure.*

Mashatan and Stinson's proposal can be be seen in the same line of research as, for instance, Anderson et al.'s *Guy Fawkes protocol* [3], Stajano and Anderson's *resurrecting duckling* [55], Mitchell's scheme for remote user authentication [48], Weimerskirch and Westhoff's *zero common-knowledge authentication* [57], and Lucks et al.'s *Jane Doe protocol* [40].

Our contribution. Below, we show that the resynchronization mechanism suggested by Mashatan and Stinson unfortunately does not work as intended, but actually enables an attack: an adversary can abuse the resynchronization process to send forged messages that are accepted as legitimate. The computational effort for the attack is negligible, and there is no restriction on the contents of the messages that can be inserted.

3.4.1 The Proposal from CANS 2008

This section recalls Mashatan and Stinson's proposal from CANS 2008 to the extent necessary for describing our attack. The protocol splits into three components, which we discuss in the following three sections. For more details, we refer to the original paper [43], which elaborates on the underlying assumptions on the hash function H (*pre-image resistance*, *paired second pre-image resistance*, *paired collision resistance*, *binding pre-image resistance*, for instance). We denote *passwords*[2] for party A by x_i and for party B by y_i. Writing H for the underlying hash function, we set $X_i := H(x_i)$, $Y_i := H(y_i)$ and refer to the X_i and Y_i as *committing hash values* of the passwords. Finally, the *binding hash values* are denoted by $\mathcal{X}_{i(i+1)}$ and $\mathcal{Y}_{i(i+1)}$ for A and B respectively, where $\mathcal{X}_{i(i+1)} := H(x_i, X_{i+1})$ and $\mathcal{Y}_{i(i+1)} := H(y_i, Y_{i+1})$.

At any given time, the internal state of A is given by an 8-tuple $(x_i, x_{i+1}, X_i, X_{i+1}, \mathcal{X}_{i(i+1)}, y^*_{i-1}, Y^*_i, \mathcal{Y}^*_{i(i+1)})$ with y^*_{i-1}, Y^*_i, $\mathcal{Y}^*_{i(i+1)}$ being B's most recent password, committing hash value, and binding hash value accepted by A. Likewise, the internal state of B is given by an 8-tuple $(y_i, y_{i+1}, Y_i, Y_{i+1}, \mathcal{Y}_{i(i+1)}, x^*_{i-1}, X^*_i, \mathcal{X}^*_{i(i+1)})$ with x^*_{i-1}, X^*_i, $\mathcal{X}^*_{i(i+1)}$ being A's most recent password, committing hash value, and binding hash value accepted by B.

[2]Here we follow the terminology in [43] and stress that exhausting all possible passwords is assumed to be infeasible. In particular, this use of the term *password* differs from the common use in the context of password authenticated key establishment.

Adversarial model During the initialization phase of the protocol, the involved parties A and B exchange information through an authenticated channel which we will denote by $\Longrightarrow$. The adversary is restricted to passive eavesdropping of this channel; no delaying, deleting, inserting, or altering of messages is allowed. During the execution of the protocol and in the resynchronization process, A and B communicate over an insecure channel which we denote by $\longrightarrow$. The adversary has full control over the insecure channel, and in particular can delete and insert messages. The goal of the adversary is to create a forgery; i. e., to provoke a situation where B accepts a message-recipient pair (A, M) where the message M has never been sent by A.

Initialization phase

Figure 3.2 shows the steps performed by A and B in the initialization phase.

A		***B***
Choose random x_0 and x_1 and form $X_0 := H(x_0)$, $X_1 := H(x_1)$, and $\mathcal{X}_{01} := H(x_0, X_1)$.	$\xRightarrow{X_0, \mathcal{X}_{01}}$	Receive $X_0, \mathcal{X}_{01}$.
Receive $Y_0, \mathcal{Y}_{01}$.	$\xLeftarrow{Y_0, \mathcal{Y}_{01}}$	Choose random y_0 and y_1 and form $Y_0 := H(y_0)$, $Y_1 := H(y_1)$, and $\mathcal{Y}_{01} := H(y_0, Y_1)$.
Let $y^*_{-1} :=\perp$, so A's initial state is $(x_0, x_1, X_0, X_1, X_{01}, \perp, Y_0, \mathcal{Y}_{01})$.		Let $x^*_{-1} :=\perp$ so B's initial state is $(y_0, y_1, Y_0, Y_1, Y_{01}, \perp, X_0, \mathcal{X}_{01})$

Figure 3.2: Initialization phase of [43]

In summary, during the initialization phase A, does the following:

- Choose random x_0 and x_1.

- Compute $X_0 := H(x_0)$, $X_1 := H(x_1)$, and $\mathcal{X}_{01} := H(x_0, X_1)$.
- Send X_0, $\mathcal{X}_{01}$ to B over the authenticated channel.
- Receive Y_0, and $\mathcal{Y}_{01}$ from B over the authenticated channel.
- Set $y^*_{-1} :=\perp$, $Y^*_0 := Y_0$, $\mathcal{Y}^*_0 := \mathcal{Y}_0$.

Similarly, B performs the following steps:

- Choose random y_0 and y_1.
- Compute $Y_0 := H(y_0)$, $Y_1 := H(y_1)$, and $\mathcal{Y}_{01} := H(y_0, Y_1)$.
- Send Y_0, $\mathcal{Y}_{01}$ to A over the authenticated channel.
- Receive X_0, and $\mathcal{X}_{01}$ from A over the authenticated channel.
- Set $x^*_{-1} :=\perp$, $X^*_0 := X_0$, $\mathcal{X}^*_0 := \mathcal{X}_0$.

The values X_0, $\mathcal{X}_{01}$, Y_0, $\mathcal{Y}_{01}$ which are interchanged by A and B over the authenticated channel can be eavesdropped—but not altered—by the adversary.

Execution of the protocol

Once the initialization phase has been completed, the actual protocol execution can take place as described in Figure 3.3.
Summarizing, on input a message-recipient pair (M, B), A does the following during a protocol execution:

- Choose a random x_2 and form $X_2 := H(x_2)$, $\mathcal{X}_{12} := H(x_1, X_2)$.
- Compute $h := H(M, x_0)$.

A's internal state: $(x_0, x_1, X_0, X_1, \mathcal{X}_{01}, y^*_{-1}, Y^*_0, \mathcal{Y}^*_{01})$		B's internal state: $(y_0, y_1, Y_0, Y_1, \mathcal{Y}_{01}, x^*_{-1}, X^*_0, \mathcal{X}^*_{01})$
A		***B***
Receive input (M, B). Choose a random x_2 and form $X_2 := H(x_2)$, $\mathcal{X}_{12} := H(x_1, X_2)$. Compute $h := H(M, x_0)$.	$\xrightarrow{M,h}$	Receive M', h'.
Receive y'_0, Y'_1, $\mathcal{Y}'_{12}$.	$\xleftarrow{y_0, Y_1, \mathcal{Y}_{12}}$	Choose a random y_2 and form $Y_2 := H(y_2)$, $\mathcal{Y}_{12} := H(y_1, Y_2)$.
If $H(y'_0) = Y^*_0$ and $H(y'_0, Y'_1) = \mathcal{Y}^*_{01}$, then send x_0, X_1, $\mathcal{X}_{12}$ and update your internal state as follows: $(x_1, x_2, X_1, X_2, \mathcal{X}_{12}, y'_0, Y'_1, \mathcal{Y}'_{12})$ else initiate resynchronization.	$\xrightarrow{x_0, X_1, \mathcal{X}_{12}}$	Receive x'_0, X'_1, $\mathcal{X}'_{12}$. If $H(x'_0) = X^*_0$, $H(x'_0, X'_1) = \mathcal{X}^*_{01}$, and $h' = H(m', x')$, then update your internal state as follows: $(y_1, y_2, Y_1, Y_2, \mathcal{Y}_{12}, x'_0, X'_1, \mathcal{X}'_{12})$ and output (A, M') else initiate resynchronization.

Figure 3.3: Protocol execution of [43]

- Send (M, h) and wait to receive y'_0, Y'_1, $\mathcal{Y}'_{01}$ from B. Resend if B does not respond.

- If $H(y'_0) = Y^*_0$ and $H(y'_0, Y'_1) = \mathcal{Y}_{01}$, send x_0, X_1, $\mathcal{X}_{01}$ to B and update the internal state to $(x_1, x_2, X_1, X_2, \mathcal{X}_{12}, y'_0, Y'_1, \mathcal{Y}'_{12})$; else initiate resynchronization.

After receiving (M', h'), B does the following:

- Choose a random y_2 and compute $Y_2 := H(y_2)$, $\mathcal{Y}_{12} := H(y_1, Y_2)$.

- Send y_0, Y_1, $\mathcal{Y}_{12}$ to A and wait to receive x'_0, X'_1, $\mathcal{X}_{12}$. Resend if A does not respond.

- If $H(x'_0) = X^*_0$, $H(x'_0, X'_1) = \mathcal{X}_{01}$, and $h' = H(M', x'_0)$ then update the internal state to $(y_1, y_2, Y_1, Y_2, \mathcal{Y}_{12}, x'_0, X'_1, \mathcal{X}'_{12})$ and output (A, M'); else initiate resynchronization.

Note that all messages are sent over an insecure channel, where the adversary can delete, modify, and insert messages at will. Further, it is possible for A to update its internal state after sending x_0, X_1, $\mathcal{X}_{12}$ without B updating its state. Therefore, the resynchronization process that follows is not symmetric.

Resynchronization process

In the case of adversarial intrusion or communication failure, either A or B can initiate the resynchronization process in Figure 3.4. As shown in this figure, B has two sets of conditions that can update its internal state, whereas A has only one.
We can summarize the resynchronization process as follows:

- A and B respectively choose random x_2, y_2 and form $X_2 := H(x_2)$, $Y_2 := H(y_2)$, $\mathcal{X}_{12} := H(x_1, X_2)$, and $\mathcal{Y}_{12} := H(y_1, Y_2)$.

- B sends y_0, Y_1, $\mathcal{Y}_{01}$ to A.

- A sends x_0, X_1, $\mathcal{X}_{01}$ to B.

- If $y^*_{-1} = y'_0$ and $Y^*_0 = Y'_1$, then A sets $\mathcal{Y}^*_{01} := \mathcal{Y}'_{12}$; else A initiates resynchronization.

- If $x^*_{-1} = x'_0$ and $X^*_0 = X'_1$, then B sets $\mathcal{X}^*_{01} := \mathcal{X}'_{12}$; else if $H(x'_0) = X^*_0$ and $H(x'_0, X'_1) = \mathcal{X}^*_{01}$, then B sets $x^*_{-1} := x'_0$, $X^*_0 := X'_1$, $\mathcal{X}^*_{01} := \mathcal{X}'_{12}$; else B initiates resynchronization.

During resynchronization, A can only refresh the value $\mathcal{Y}^*_{01}$, whereas B can either refresh the value $\mathcal{X}^*_{01}$ or update x^*_{-1}, X^*_0, $\mathcal{X}^*_{01}$.

A's internal state: $(x_0, x_1, X_0, X_1, \mathcal{X}_{01}, y^*_{-1}, Y^*_0, \mathcal{Y}^*_{01})$		B's internal state: $(y_0, y_1, Y_0, Y_1, \mathcal{Y}_{01}, x^*_{-1}, X^*_0, \mathcal{X}^*_{01})$
A		***B***
Choose a random x_2 and form $X_2 := H(x_2)$, $\mathcal{X}_{12} := H(x_1, X_2)$.		Choose a random y_2 and form $Y_2 := H(y_2)$, $\mathcal{Y}_{12} := H(y_1, Y_2)$.
Receive y'_0, Y'_1, $\mathcal{Y}'_{12}$.	$\xleftarrow{y_0, Y_1, \mathcal{Y}_{12}}$	Send y_0, Y_1, $\mathcal{Y}_{12}$.
Send x_0, X_1, $\mathcal{X}_{12}$.	$\xrightarrow{x_0, X_1, \mathcal{X}_{12}}$	Receive x'_0, X'_1, $\mathcal{X}'_{12}$.
If $y^*_{-1} = y'_0$ and $Y^*_0 = Y'_1$, then $\mathcal{Y}^*_{01} := \mathcal{Y}'_{12}$; else initiate resynchronization.		If $x^*_{-1} = x'_0$ and $X^*_0 = X'_1$, then $\mathcal{X}^*_{01} := \mathcal{X}'_{12}$; otherwise, if $H(x'_0) = X^*_0$ and $H(x'_0, X'_1) = \mathcal{X}^*_{01}$, then $x^*_{-1} := x'_0$, $X^*_0 := X'_1$, $\mathcal{X}^*_{01} := \mathcal{X}'_{12}$; else initiate resynchronization.

Figure 3.4: Resynchronization process of [43]

3.4.2 Provoking an Unrecoverable Situation

If A or B suspects a communication failure or a possible adversarial intrusion, it can initiate the resynchronization process. Here we show that

- an adversary can create a situation where A keeps on initiating the resynchronization process, but the protocol does not recover, and
- an adversary can create a situation where B keeps on initiating the resynchronization process, but the protocol does not recover.

It is worth noting that in both cases, modification of a single message on the public channel is sufficient; i. e., the adversary does not have to stay "online" for achieving this type of denial of service: these attacks are qualitatively different from simply blocking communication between A and B. Section 3.4.3 builds on these observations to create a successful forgery.

Unrecoverability with resynchronization initiated by A

As depicted in Figure 3.5, assume that after a successful initialization phase, A has internal state $(x_0, x_1, X_0, X_1, X_{01}, \perp, Y_0, \mathcal{Y}_{01})$, and B has internal state $(y_0, y_1, Y_0, Y_1, Y_{01}, \perp, X_0, \mathcal{X}_{01})$. Now A starts executing a protocol as specified in Section 3.4.1, sending a message M along with matching h-value to B. In response to this, B sends y_0, Y_1 and $\mathcal{Y}_{12}$.

The adversary can replace y_0 with a (random) value such that A's validity check $H(y_0') = Y_0$ and $H(y_0', Y_1') = \mathcal{Y}_{01}^*$ fails. Following the protocol specification, now A initiates the resynchronization process (see upper part of Figure 3.5). Note that so far A never updated its internal state and still has stored the values $y_{-1}^* =\perp$, $Y_0* = Y_0$, and $\mathcal{Y}_{01}^* = \mathcal{Y}_{01}$.

Now, in the resynchronization phase, B sends to A the values y_0', Y_1', and $\mathcal{Y}_{12}'$. These values do not match the values stored by A, however. Consequently, A initiates resynchronization again. Re-running the resynchronization will not help the situation, so the protocol becomes unrecoverable. Figure 3.5 summarizes the sequence of events.

Unrecoverability with resynchronization initiated by B

Consider a second scenario as in Figure 3.6. Assume that after a successful initialization phase A has internal state $(x_0, x_1, X_0, X_1, \mathcal{X}_{01}, \perp, Y_0, \mathcal{Y}_{01})$, and B has internal state $(y_0, y_1, Y_0, Y_1, \mathcal{Y}_{01}, \perp, X_0, \mathcal{X}_{01})$ as before. As before, A initiates an execution of the protocol in [43] by sending a message M along with matching h-value to B. In response, A receives y_0', Y_1', and $\mathcal{Y}_{12}'$ from B. Our adversary faithfully transmits these messages, so that A's validity check succeeds, and A updates its internal state to

A's internal state: $(x_0, x_1, X_0, X_1, \mathcal{X}_{01}, \perp, Y_0, \mathcal{Y}_{01})$		B's internal state: $(y_0, y_1, Y_0, Y_1, \mathcal{Y}_{01}, \perp, X_0, \mathcal{X}_{01})$

<u>execution of the protocol</u>

A		***B***
Receive input (M, B). Choose random x_2 and form $X_2 := H(x_2)$, $\mathcal{X}_{12} := H(x_1, X_2)$. Compute $h := H(M, x_0)$.	$\xrightarrow{M,h}$	Receive M', h'.
Receive y_0', Y_1', $\mathcal{Y}_{12}'$.	$\xleftarrow{y_0,Y_1,\mathcal{Y}_{12}}$	Choose a random y_2 and form $Y_2 := H(y_2)$, $\mathcal{Y}_{12} := H(y_1, Y_2)$.
Suppose that $H(y_0') \neq Y_0^*$ or $H(y_0', Y_1') \neq \mathcal{Y}_{01}^*$; hence, initiate resynchronization.		

A's internal state: $(x_0, x_1, X_0, X_1, \mathcal{X}_{01}, \perp, Y_0, \mathcal{Y}_{01})$		B's internal state: $(y_0, y_1, Y_0, Y_1, \mathcal{Y}_{01}, \perp, X_0, \mathcal{X}_{01})$

<u>resynchronization process</u>

A		***B***
Choose a random x_2 and form $X_2 := H(x_2)$, $\mathcal{X}_{12} := H(x_1, X_2)$.		Choose a random y_2 and form $Y_2 := H(y_2)$, $\mathcal{Y}_{12} := H(y_1, Y_2)$
Receive y_0', Y_1', $\mathcal{Y}_{12}'$.	$\xleftarrow{y_0,Y_1,\mathcal{Y}_{12}}$	Send y_0, Y_1, $\mathcal{Y}_{12}$
Send x_0, X_1, $\mathcal{X}_{12}$.	$\xrightarrow{x_0,X_1,\mathcal{X}_{12}}$	Receive x_0', X_1', $\mathcal{X}_{12}'$.
Since $y_{-1}^* \neq y_0'$ and $Y_0* \neq Y_1'$, initiate resynchronization.		

Figure 3.5: Unrecoverability after a resynchronization initiated by A

$(x_1, x_2, X_1, X_2, X_{12}, y'_0, Y'_1, \mathcal{Y}'_{12})$. Further, A sends x_0, X_1 and $\mathcal{X}_{12}$ to B. Our adversary can replace x_0 with a (random) value so that the values x'_0, X'_1, and $\mathcal{X}'_{12}$ received by B from A do not verify. Consequently, following the protocol specification in Section 3.4.1, B will initiate the resynchronization process. Note that so far B never updated its internal state and has stored $x^*_{-1} = \perp$, $X^*_0 = X_0$, and $\mathcal{X}^*_{01} = \mathcal{X}_{01}$.

In the resynchronization process, A sends x_1, X_2, and $\mathcal{X}_{23}$ to B. Even if the values x'_1, X'_2, and $\mathcal{X}'_{23}$ received by B are identical to the values sent by A, $x'_1 \neq x^*_{-1}$ and $H(x'_1) \neq X^*_0$ cause B to initiate resynchronization again. Re-running the resynchronization will not resolve the situation, and analogously, as in the previous section, the protocol becomes unrecoverable. Figure 3.6 summarizes the sequence of events.

A's internal state: $(x_0, x_1, X_0, X_1, \mathcal{X}_{01}, \perp, Y_0, \mathcal{Y}_{01})$		B's internal state: $(y_0, y_1, Y_0, Y_1, \mathcal{Y}_{01}, \perp, X_0, \mathcal{X}_{01})$

<u>execution of the protocol</u>

A		***B***
Receive input (M, Bob). Choose random x_2 and form $X_2 := H(x_2)$, $\mathcal{X}_{12} := H(x_1, X_2)$. Compute $h := H(M, x_0)$.	$\xrightarrow{M,h}$	Receive M', h'.
Receive y'_0, Y'_1, $\mathcal{Y}'_{12}$.	$\xleftarrow{y_0, Y_1, \mathcal{Y}_{12}}$	Choose a random y_2 and form $Y_2 := H(y_2)$, $\mathcal{Y}_{12} := H(y_1, Y_2)$.
Suppose that $H(y'_0) = Y^*_0$ and $H(y'_0, Y'_1) = \mathcal{Y}^*_{01}$. Then send x_0, X_1, $\mathcal{X}_{12}$ and update the internal state to $(x_1, x_2, X_1, X_2, \mathcal{X}_{12}, y'_0, Y'_1, \mathcal{Y}'_{12})$.	$\xrightarrow{x_0, X_1, \mathcal{X}_{12}}$	Receive x'_0, X'_1, $\mathcal{X}'_{12}$. Suppose $H(x'_0) \neq X^*_0$, or $H(x'_0, X'_1) \neq \mathcal{X}^*_{01}$, or $h' \neq H(M', x')$. Then initiate resynchronization.

A's internal state: $(x_1, x_2, X_1, X_2, \mathcal{X}_{12}, y'_0, Y'_1, \mathcal{Y}'_{12})$		B's internal state: $(y_0, y_1, Y_0, Y_1, \mathcal{Y}_{01}, \perp, X_0, \mathcal{X}_{01})$

<u>resynchronization process</u>

A		***B***
Choose a random x_3 and form $X_3 := H(x_3)$, $\mathcal{X}_{23} := H(x_2, X_3)$.		Choose a random y_2 and form $Y_2 := H(y_2)$, $\mathcal{Y}_{12} := H(y_1, Y_2)$
Receive y'_0, Y'_1, $\mathcal{Y}'_{12}$.	$\xleftarrow{y_0, Y_1, \mathcal{Y}_{12}}$	Send y_0, Y_1, $\mathcal{Y}_{12}$
Send x_1, X_2, $\mathcal{X}_{23}$.	$\xrightarrow{x_1, X_2, \mathcal{X}_{23}}$	Receive x'_1, X'_2, $\mathcal{X}'_{23}$.
		Since $x^*_{-1} \neq x'_1$ and $H(x'_1) \neq X^*_0$, initiate resynchronization.

Figure 3.6: Unrecoverability after a resynchronization initiated by B

3.4.3 Creating a Forgery

To describe the attack, in subsequent figures we denote the adversary by F. Messages delivered faithfully by F are denoted by $\rightharpoonup$ and the messages created by F are denoted by $\rightharpoondown$. To begin our attack, we assume that A and B have successfully completed the initialization phase of the protocol. From here on, the attack unfolds in four steps:

1. executing the message recognition protocol
2. first resynchronization (unsuccessful)
3. second resynchronization (successful)
4. executing the message recognition protocol a second time

The subsequent four subsections elaborate on each of these steps.

Execution of the recognition protocol

In this first step, the goal of F is to learn the initial password x_0 from A. For this, F proceeds as shown in Figure 3.7.
Summarizing, in this first step of the attack F does the following:

- Forward M, h faithfully from A to B.
- Forward the values y_0, Y_1, $\mathcal{Y}_{12}$ sent from B faithfully to A.
- Choose a (random) $\widetilde{x} \neq x_0$ so that that $H(\widetilde{x}) \neq X_0$.
- Send $\widetilde{x}$, X_1, $\mathcal{X}_{12}$ to B; i. e., replace the value x_0 sent by A with $\widetilde{x}$.

Since $H(\widetilde{x}) \neq X_0$, B initiates resynchronization after A has already updated its internal state to $(x_1, x_2, X_1, X_2, \mathcal{X}_{12}, y_0, Y_1, \mathcal{Y}_{12})$, and we are in a similar situation to that discussed in Section 3.4.2.

A	F	B
A's internal state: $(x_0, x_1, X_0, X_1, \mathcal{X}_{01}, \perp, Y_0, \mathcal{Y}_{01})$		B's internal state: $(y_0, y_1, Y_0, Y_1, \mathcal{Y}_{01}, \perp, X_0, \mathcal{X}_{01})$
Receive (M, B) as input. Choose a random x_2 and form $X_2 := H(x_2)$, $\mathcal{X}_{12} := H(x_1, X_2)$. Compute $h := H(M, x_0)$.	$\xrightarrow{M,h}$	Receive M, h.
Receive y_0, Y_1, $\mathcal{Y}_{12}$.	$\xleftarrow{y_0,Y_1,\mathcal{Y}_{12}}$	Choose a random y_2 and form $Y_2 := H(x_2)$, $\mathcal{Y}_{12} := H(y_1, Y_2)$.
Since $H(y_0) = Y_0^*$ and $H(y_0, Y_1) = \mathcal{Y}_{01}$, send x_0, X_1, $\mathcal{X}_{12}$ and update the internal state to $(x_1, x_2, X_1, X_2, \mathcal{X}_{12}, y_0, Y_1, \mathcal{Y}_{12})$.	$\xrightarrow{\widetilde{x},X_1,\mathcal{X}_{12}}$	Since $H(\widetilde{x}) \neq X_0$, initiate resynchronization.

Figure 3.7: First step of the attack: execution of the protocol

First resynchronization (unsuccessful)

In this second step of the attack, F extracts the value x_1 from A, using the resynchronization process as shown in Figure 3.8.

Thus F's actions in this step of the attack can be summarized as follows:

- Forward the values y_0, Y_1, $\mathcal{Y}_{12}$ sent by B faithfully to A.
- Receive x_1, X_2, $\mathcal{X}_{23}$ from A.
- Send $\widetilde{x}$, X_1, $\mathcal{X}_{12}$ to B; i. e., the same values as above.

Since y_0 and Y_1 match what A has stored, A refreshes the value $\mathcal{Y}_{12}$ with the new one sent by B. Recall that B has two sets of conditions to check, as shown in Figure 3.4.

A's internal state: $(x_1, x_2, X_1, X_2, \mathcal{X}_{12}, y_0, Y_1, \mathcal{Y}_{12})$		B's internal state: $(y_0, y_1, Y_0, Y_1, \mathcal{Y}_{01}, \perp, X_0, \mathcal{X}_{01})$
A	***F***	***B***
Choose a random x_3, and form $X_3 := H(x_3)$, $\mathcal{X}_{23} := H(x_2, X_3)$.		Choose a random y_2 and form $Y_2 := H(y_2)$, $\mathcal{Y}_{12} := H(y_1, Y_2)$.
Receive y_0, Y_1, $\mathcal{Y}_{12}$.	$\xleftarrow{y_0, Y_1, \mathcal{Y}_{12}}$	Send y_0, Y_1, $\mathcal{Y}_{12}$.
Send x_1, X_2, $\mathcal{X}_{23}$	$\xrightarrow{\widetilde{x}, X_1, \mathcal{X}_{12}}$	Since $x^*_{-1} \neq \widetilde{x}$, and $H(\widetilde{x}) \neq X_0$, initiate resynchronization.

Figure 3.8: Second step of the attack: unsuccessful resynchronization

As B has not accepted a password from A yet, we clearly have $x_{-1} \neq \widetilde{x}$ and the first condition is not met. Further, we have $H(\widetilde{x}) \neq X_0$, so the second condition is not met either. Hence the resynchronization is unsuccessful and B initiates resynchronization a second time. Note that at this point, F knows both x_0 and x_1.

Second resynchronization (successful)

During this second resynchronization, B will update its internal state. In preparation of the subsequent forgery, F binds the x_1-value received from A to F's own value $\widetilde{x}$. Figure 3.9 delineates the sequence of events during this second (successful) resynchronization.

Summarizing, F does the following:

- Forward the values y_0, Y_1, $\mathcal{Y}_{12}$ sent by B faithfully to A.

- For the random $\widetilde{x}$ from the first step of the attack, form $\widetilde{X} := H(\widetilde{x})$ and $\widetilde{\mathcal{X}} := H(x_1, \widetilde{X})$.

A's internal state: $(x_1, x_2, X_1, X_2, \mathcal{X}_{12}, y_0, Y_1, \mathcal{Y}_{12})$		B's internal state: $(y_0, y_1, Y_0, Y_1, \mathcal{Y}_{01}, \perp, X_0, \mathcal{X}_{01})$
A	***F***	***B***
Receive y_0, Y_1, $\mathcal{Y}_{12}$.	$\xleftarrow{y_0, Y_1, \mathcal{Y}_{12}}$	Choose a random y_2 and form $Y_2 := H(y_2)$, $\mathcal{Y}_{12} := H(y_1, Y_2)$. Send y_0, Y_1, $\mathcal{Y}_{12}$.
Choose a random x_3 and form $X_3 := H(x_3)$, $\mathcal{X}_{23} := H(x_2, X_3)$.	$\xrightarrow{x_0, X_1, \widetilde{\mathcal{X}}}$	Verify that $H(x_0) = X_0$, $H(x_0, X_1) = \mathcal{X}_{01}$; then update internal state.

Figure 3.9: Third step of the attack: successful resynchronization

- Send x_0, X_1, $\widetilde{\mathcal{X}}$ to B.

Since y_0 and Y_1 match what A has stored, A refreshes the value $\mathcal{Y}_{12}$ with the new one sent by B once again. As $H(x_0) = X_0$ and $H(x_0, X_1) = \mathcal{X}_{01}$, the second set of B's conditions is met, and B updates its internal state to $(y_0, y_1, Y_0, Y_1, \mathcal{Y}_{01}, x_0, X_1, \widetilde{\mathcal{X}})$. Hence, the second resynchronization is successful, and F can initiate an execution of the message recognition protocol with B.

Executing the message recognition protocol a second time

In the final step of the attack, F uses x_1 and the committing hash value $\widetilde{X}$ with the $\widetilde{x}$ chosen earlier. As seen in Figure 3.10, only F is communicating with B at this stage, and the message $\widetilde{M}$ can chosen arbitrarily (with $M \neq \widetilde{M}$ to indeed achieve a forgery).

The actions of F in this last part of the attack can be summarized as follows:

- Choose a message $\widetilde{M} \neq m$ and compute $h := H(\widetilde{M}, x_1)$.
- Send $\widetilde{M}$, h to B.

		B's internal state: $(y_0, y_1, Y_0, Y_1, \mathcal{Y}_{01}, x_0, X_1, \widetilde{\mathcal{X}})$
F		***B***
Choose a message $\widetilde{M} \neq M$. Compute $h := H(\widetilde{M}, x_1)$.	$\xrightarrow{\widetilde{M},h}$	Receive $\widetilde{M}$, h.
Receive y_0, Y_1, $\mathcal{Y}_{12}$.	$\xleftarrow{y_0,Y_1,\mathcal{Y}_{12}}$	Choose a random y_2 and form $Y_2 := H(y_2)$, $\mathcal{Y}_{12} := H(y_1, Y_2)$. Send y_0, Y_1, $\mathcal{Y}_{12}$.
Choose a random $\widehat{x}$ and form $\widehat{X} := H(\widehat{x})$ and $\widehat{\mathcal{X}} := H(\widetilde{x}, \widehat{X})$.	$\xrightarrow{x_1,\widetilde{X},\widehat{\mathcal{X}}}$	Verify that $H(x_1) = X_1$ and $H(x_1, \widetilde{X}) = \widetilde{\mathcal{X}}$; then update internal state to $(y_1, y_2, Y_1, Y_2, \mathcal{Y}_{12}, x_1, \widetilde{X}, \widehat{\mathcal{X}})$ and output $(A, \widetilde{M})$.

Figure 3.10: Fourth step of the attack: inserting a forged message

- Receive y_0, Y_1, $\mathcal{Y}_{12}$ from B.
- Choose a random $\widehat{x}$ and form $\widehat{X} := H(\widehat{x})$ and $\widehat{\mathcal{X}} := H(\widetilde{x}, \widehat{X})$.
- Send x_1, $\widetilde{X}$, $\widehat{\mathcal{X}}$ to B.

Conclusion The above discussion shows that the message recognition protocol suggested by Mashatan and Stinson in [43] does not provide the intended security guarantees: the resynchronization procedure can be abused to provoke a situation where the protocol does not recover and enables a successful forgery attack. Consequently, the protocol from [43] should not be used in the present form.

Chapter 4

Forward Security

The longer you can look back, the farther you can look forward. Winston Churchill [38]

In forward security, so-called *key-evolving signature schemes* are considered, and compromise of the current secret key does not enable an adversary to forge signatures pertaining to the past. Signatures for messages signed in the past under a fixed public key are valid even if the current secret key is exposed. Furthermore, the adversary cannot forge signatures with a "date" prior to key exposure. In a forward secure signature, the secret key sk is updated during each time period (not for each signature). In this chapter, we discuss forward security, which we refer to as FWD-CMA.

4.1 Key-evolving Signature Schemes

We adopt some terminology from Bellare and Miner [10, 11], starting by defining a key-evolving signature scheme.

Definition 4.1 (Key-evolving signature scheme)**.** *A key-evolving signature scheme S^f is a quadruple of polynomial time algorithms $S = (\mathcal{K}^f, \mathcal{U}^f, \mathcal{S}^f, \mathcal{V}^f)$:*

1. $\mathcal{K}^f$ *is a probabilistic* key generation algorithm $\mathcal{K}$ *which on input of the security parameter* 1^k*, the total number of time periods* $T \in \mathbb{N}$ *(and possibly other parameters) returns a pair* (sk_0, pk) *of keys—a public verification key pk with matching (base) secret signing key* sk_0.

2. $\mathcal{U}^f$ *is a deterministic* secret key update algorithm *which takes as input of the secret signing key* sk_{j-1} *of the previous time period* $j-1$ *and returns the secret signing key* sk_j *for time period* j.

3. $\mathcal{S}^f$ *is a probabilistic* signing algorithm *that on input of a (polynomial length) message* $M \in \{0,1\}^*$ *and the secret signing key* sk_j *of the current time period* j *returns a signature* $\langle j, \zeta \rangle \overset{\$}{\leftarrow} \mathcal{S}^f_{sk_j}(M)$ *for* M *for time period* $j \in \mathbb{N}$ *or returns an error symbol* $\perp$.

4. $\mathcal{V}^f$ *is a deterministic* verification algorithm *which on input of a public key pk, a message* M*, and a signature* $\langle j, \zeta \rangle$ *returns 1 or 0, indicating whether the signature is accepted or rejected, respectively.*

We may assume that sk_j stores the value j itself for period $j \in \{1, \ldots, T\}$ as well as the total number T of time periods. Further on, we adopt the convention that sk_{T+1} is the empty string and that $\mathcal{U}^f(sk_T)$ returns sk_{T+1}. Both the current time period j and the total number of time periods T are publicly known and accessible to an adversary $\mathcal{A}^{\text{fwd}}$ along with the attacked public key *pk*. The actual attack game used to define forward security of a key-evolving signature scheme involves three stages: the chosen message attack phase (*cma*), the break-in phase (*breakin*), and the forgery phase (*forge*).

4.2 FWD-CMA Security

Definition 4.2 (FWD-CMA). *Let $S^f = (\mathcal{K}^f, \mathcal{U}^f, \mathcal{S}^f, \mathcal{V}^f)$ be a key-evolving signature scheme, and let $\mathcal{A}^{\text{fwd}}$ be a probabilistic polynomial time algorithm. Consider the following attack scenario:*

1. *cma phase.*

Set $j \leftarrow 0$, and compute a key pair $(sk_0, pk) \overset{\$}{\leftarrow} \mathcal{K}^f(1^k, \dots, T)$.[1]

```
repeat
  $j \leftarrow j + 1$; $sk_j \leftarrow \mathcal{U}^f(sk_{j-1})$
  $d \overset{\$}{\leftarrow} \mathcal{A}^{\text{fwd}\,\mathcal{O}^j_{\mathcal{S}^f}}(\mathsf{cma}, pk)$
until $(d = \mathsf{breakin})$ or $(j = T)$
if $d \neq \mathsf{breakin}$ and $j = T$
  then $j = T + 1$
end if
```

2. *breakin phase.*

The adversary $\mathcal{A}^{\text{fwd}}$ is handed the current secret key sk_j.

3. *forge phase.*

Eventually, $\mathcal{A}^{\text{fwd}}$ outputs a message M and a signature $\langle b, \zeta \rangle$ with $b < j$.

Let QueriedEarlier *be the event that $\mathcal{A}^{\text{fwd}}$ outputs a message M that has been queried to a signing oracle $\mathcal{O}^j_{\mathcal{S}^f}$ already. The* success probability $\text{Succ}_{\mathcal{A}^{\text{fwd}}} = \text{Succ}_{\mathcal{A}^{\text{fwd}}}(1^k, \dots, T)$ *of $\mathcal{A}^{\text{fwd}}$ is defined as*

$$\text{Succ}_{\mathcal{A}^{\text{fwd}}} := \Pr[\mathcal{V}^f_{pk}(M, \langle b, \zeta \rangle) = 1 \textit{ and } \neg\mathsf{QueriedEarlier}],$$

[1]Here,'. . . ' indicates that further auxiliary input parameters may be present.

and we call the signature scheme S^f forward-secure *if* $\mathrm{Succ}_{\mathcal{A}^{\mathrm{fwd}}}$ *is negligible (in k) for all probabilistic polynomial time adversaries $\mathcal{A}^{\mathrm{fwd}}$.*

The process in Definition 4.2 is strictly ordered, in that once an adversary gives up the signing oracle for sk_j, it cannot obtain access to that oracle again. At some point, the adversary $\mathcal{A}^{\mathrm{fwd}}$ decides to use its break-in privilege and is returned the current secret key sk_j. To be successful, $\mathcal{A}^{\mathrm{fwd}}$ must forge a signature under sk_b for some $b < j$ and new message M.

Remark 4.1. *By definition, a* FWD-CMA *secure scheme allows an adversary $\mathcal{A}^{\mathrm{fwd}}$ to submit a polynomial number of queries to its signing oracle within a single time period j. Thus, in the presence of key-dependent messages, a similar attack to the one presented in the proof of Remark 3.1 may reveal the complete secret key, before an update of the secret key occurs. In other words, security in the sense of* FWD-CMA *does not imply strong security guarantees in the presence of key-dependent messages.*

Contrasting the above negative statement, after applying some technical modifications to obtain a syntactically correct key-evolving signature scheme, the compiler in Chapter 5 can be used to lift an EUF-CMA secure one-time signature scheme S to a forward secure key-evolving signature scheme S^f.

4.3 Long Signatures

In [11], Bellare and Miner propose a long signature $S^l = (\mathcal{K}^l, \mathcal{U}^l, \mathcal{S}^l, \mathcal{V}^l)$ which is defined in Figure 4.1. We will prove that this scheme is not FWD-CMA. We also suggest a small modification of S^l that fixes the security gap.

Proposition 4.1. *Let $S^l = (\mathcal{K}^l, \mathcal{U}^l \mathcal{S}^l, \mathcal{V}^l)$ be the long signature scheme described in Figure 4.1. Then the signature scheme S^l is not secure in the sense of* FWD-CMA*.*

$\mathcal{K}^l$**:** Create a key pair $(sk_0, pk_0) \overset{\$}{\leftarrow} \mathcal{K}$, return $pk := pk_0$ as public verification key and use $sk := (sk_0, [])$ as initial state, where [] is an empty list.

$\mathcal{U}^l$**:** To update the secret state sk_{i-1} for $1 \le i \le T$ proceed as follows:

- Create a fresh key pair $(sk_i, pk_i) \overset{\$}{\leftarrow} K(1^k)$.
- Compute $Cert_i := (pk_i, \sigma_i)$, where $\sigma_i \overset{\$}{\leftarrow} \mathcal{S}_{sk_{i-1}}(i \parallel pk_i)$.
- Update the internal state $sk(= (sk_{i-1}, [Cert_\mu]_{1 \le \mu \le i-1}))$ to $sk := (sk_i, [Cert_\mu]_{1 \le \mu \le i})$.

$\mathcal{S}^l$**:** To sign during the i^{th} $(1 \le i \le T)$ time period a message $M \in \{0,1\}^*$, proceed as follows:

- Compute $s \overset{\$}{\leftarrow} \mathcal{S}_{sk_i}(M)$.
- Return the signature $(s, [Cert_\mu]_{1 \le \mu \le i})$.

$\mathcal{V}^l$**:** On input a message M and a candidate signature $(s, [Cert_\mu]_{1 \le \mu \le i})$, output 1 if all of the following conditions hold. Otherwise output 0:

- $\mathcal{V}_{pk_i}(M, s) = 1$
- $\mathcal{V}_{pk_{\mu-1}}(i \parallel pk_\mu, \sigma_\mu) = 1$ for all $1 \le \mu \le i$, where $Cert_\mu = (pk_\mu, \sigma_\mu)$.

Figure 4.1: Long signatures.

Proof. An adversary $\mathcal{A}^{\text{fwd}}$ creates a key pair (sk^*, pk^*) and requests the signature of $i \parallel p^*$ during period $i-1$. The signature is stored by $\mathcal{A}^{\text{fwd}}$ as σ_i where $Cert_i := (pk_i, \sigma_i)$. For an arbitrary message M, $\mathcal{A}^{\text{fwd}}$ uses $sk_i = sk^*$ to compute $s \overset{\$}{\leftarrow} \mathcal{S}_{sk_i}(M)$. Hence, $(s, [Cert_\mu]_{1 \le \mu \le i})$ is a valid forgery as verified by $\mathcal{V}^l$. $\square$

The adversary $\mathcal{A}^{\text{fwd}}$ succeeds because the signatures requested can be used later on as certificates in the forgery. Therefore, we can avoid this attack if we append a 0 in front of any public key pk_i to be used in a certificate, as well as append a 1 in front of any message M to be signed. So in Figure 4.1, we write $\sigma_i \overset{\$}{\leftarrow} \mathcal{S}_{sk_{i-1}}(0 \parallel i \parallel pk_i)$ in the *secret key update algorithm* $\mathcal{U}^l$, and $s \overset{\$}{\leftarrow} \mathcal{S}_{sk_i}(1 \parallel M)$ in the *signing algorithm* $\mathcal{S}^l$. With one-time signatures in Chapter 5, we will use public keys in the positions of 0 and 1 to create a different format for signatures of messages versus signatures of certificates.

Chapter 5

Key Dependence in One-Time Signatures

There are no better cosmetics than a severe temperance and purity, modesty and humility, a gracious temper and calmness of spirit; and there is no true beauty without the signatures of these graces in the very countenance. Sir Arthur Helps [21]

5.1 Security in the Presence of Key-Dependent Signatures

The standard security requirement for signature schemes is EUF-CMA, which stands for *existential unforgeability under adaptive chosen message attack* (cf. [30]):

Definition 5.1 (EUF-CMA). *Let $S = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ be a signature scheme, and let $\mathcal{A}^{\mathrm{euf}}$ be a probabilistic polynomial time algorithm. Consider the following attack scenario:*

1. *Compute a key pair $(sk, pk) \overset{\$}{\leftarrow} \mathcal{K}(1^k)$, and hand pk as input to $\mathcal{A}^{\mathrm{euf}}$.*

2. *The adversary $\mathcal{A}^{\text{euf}}$ is given unrestricted access to a signing oracle $\mathcal{O}_{\mathcal{S}}$ to run $\mathcal{S}_{sk}(\cdot)$.*

3. *Eventually, $\mathcal{A}^{\text{euf}}$ outputs a message M and a signature σ.*

Let QueriedEarlier *be the event that $\mathcal{A}^{\text{euf}}$ outputs a message M that has been queried already to the signing oracle $\mathcal{O}_{\mathcal{S}}$. The* success probability $\text{Succ}_{\mathcal{A}}^{\text{euf}} = \text{Succ}_{\mathcal{A}^{\text{euf}}}(k)$ *of $\mathcal{A}^{\text{euf}}$ is defined as*

$$\text{Succ}_{\mathcal{A}^{\text{euf}}} := \Pr[\mathcal{V}_{pk}(M, \sigma) = 1 \textit{ and } \neg\mathsf{QueriedEarlier}],$$

and we call the signature scheme S secure in the sense of EUF-CMA *if* $\text{Succ}_{\mathcal{A}^{\text{euf}}}$ *is negligible for all probabilistic polynomial time adversaries $\mathcal{A}^{\text{euf}}$.*

Remark 5.1. *The above definition of* EUF-CMA *security carries over to one-time signature schemes in the obvious way—the only modification being that $\mathcal{A}^{\text{euf}}$ can query the signing oracle $\mathcal{O}_{\mathcal{S}}$ only once.*

In particular, security in the sense of EUF-CMA does not allow an adversary to obtain signatures on key-dependent messages—like a signature on the complete secret key (state) sk. In fact, given an EUF-CMA secure signature scheme, it is easy to come up with a signature scheme that is still EUF-CMA secure, but where a single key-dependent message query breaks the security of the scheme.

5.1.1 KDS-CMA Security

Informally, a signature scheme $S = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ is referred to as KDS-CMA secure if it is secure despite a forger's ability to obtain signatures on arbitrary (efficiently computable) functions g of the signer's state sk. In particular, g has access to the secret key stored at the time of signing.

Definition 5.2 (KDS-CMA). *Let the triple $S = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ be a signature scheme, and let $\mathcal{A}^{\text{kds}}$ be a probabilistic polynomial time algorithm. Consider the following attack scenario:*

1. *Compute a key pair $(sk, pk) \overset{\$}{\leftarrow} \mathcal{K}(1^k)$, and hand pk as input to $\mathcal{A}^{\text{kds}}$.*
2. *The adversary $\mathcal{A}^{\text{kds}}$ is given unrestricted access to a signing oracle $\widehat{\mathcal{O}}_{\mathcal{S}}$. The oracle $\widehat{\mathcal{O}}_{\mathcal{S}}$ accepts as input a function g, represented as a boolean circuit of polynomial size, and executes the signing algorithm $\mathcal{S}$ with the current state sk and the message $g(sk)$ as input.*[1]
3. *Eventually, $\mathcal{A}^{\text{kds}}$ outputs a message $M \in \{0,1\}^*$ and a signature σ.*

Let QueriedEarlier *be the event that $\mathcal{A}^{\text{kds}}$ outputs a message M such that one of $\mathcal{A}^{\text{kds}}$'s queries g to the signing oracle $\widehat{\mathcal{O}}_{\mathcal{S}}$ evaluated to $g(sk) = M$. Then the* success probability $\text{Succ}_{\mathcal{A}^{\text{kds}}} = \text{Succ}_{\mathcal{A}^{\text{kds}}}(k)$ *of $\mathcal{A}^{\text{kds}}$ is defined as*

$$\text{Succ}_{\mathcal{A}^{\text{kds}}} := \Pr[\mathcal{V}_{pk}(M, \sigma) = 1 \textit{ and } \neg\mathsf{QueriedEarlier}],$$

and we call the signature scheme S secure in the sense of KDS-CMA *if* $\text{Succ}_{\mathcal{A}^{\text{kds}}}$ *is negligible for all probabilistic polynomial time adversaries $\mathcal{A}^{\text{kds}}$.*

By definition, security in the sense of KDS-CMA implies security in the sense of EUF-CMA, and the question arises whether/how security in the sense of Definition 5.2 can be achieved.

[1]In the random oracle model, g may invoke the random oracle.

5.1.2 Impossibility of KDS-CMA with a Stateless Signing Algorithm

As a first (negative) result, we note that no signature scheme with a stateless signing algorithm can meet the security goal of KDS-CMA security.

Remark 5.2. *Let $S = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ be a signature scheme with a stateless signing algorithm $\mathcal{S}$; i. e., the secret signing key sk is not changed by executing $\mathcal{S}$. Then the signature scheme S is not secure in the sense of* KDS-CMA.

Proof. Let $sk = b_0, \ldots, b_{\ell-1} \in \{0,1\}^\ell$ be the bit representation of the secret key and fix $i \in \{0, \ldots, \ell-1\}$ arbitrary. Then the adversary $\mathcal{A}$ may query $\widehat{\mathcal{O}}_\mathcal{S}$ for a signature on b_i and use the public verification algorithm $\mathcal{V}$ to determine if the returned signature σ satisfies $\mathcal{V}_{pk}(0, \sigma) = 1$ or $V_{pk}(1, \sigma) = 1$. Thus ℓ queries to $\widehat{\mathcal{O}}_\mathcal{S}$ are sufficient to extract the complete secret signing key sk, and hereafter creating a forgery is trivial. □

Even with a polynomial number of secret keys, the adversary can use the verification algorithm to its advantage. For example, ring signatures as proposed by Rivest et al. in [53] can be signed by one of many users. Suppose that we have a vector $\mathbf{sk} = (sk_1, sk_2, \ldots, sk_n)$ of secret keys and the adversary $\mathcal{A}$ does not know which of the n keys will be used. $\mathcal{A}$ would like to find one of the keys arbitrarily, say $sk_j = b_0, \ldots, b_{\ell-1}$, and use it to forge.

$\mathcal{A}$ requests a signature of the first two bits of the current signing key $sk^* = b_0^*, \ldots, b_{\ell-1}^*$ (note that sk^* can change with each signature request). After at most three verifications, $\mathcal{A}$ sets $b_0 := b_0^*$ and $b_1 := b_1^*$. Note that the number of bits requested at the beginning is arbitrary depending on the number of bits that $\mathcal{A}$ can computationally exhaust.

Next, $\mathcal{A}$ requests a signature of function g_2 of the secret key sk^* which does the following: XOR b_0 and b_1 with the first two bits of the current key sk^* and append the third bit of sk^*. Let σ be the signature on message $M = b_0 \oplus b_0^*, b_1 \oplus b_1^*, b_2^*$. If the first two bits of sk_j and sk^* coincide, the message M is either 000 or 001. If $\mathcal{V}_{pk}(000, \sigma) = 1$ or $\mathcal{V}_{pk}(001, \sigma) = 1$, then set $b_2 := b_2^*$; otherwise $\mathcal{A}$ requests a signature of function g_2 of the secret key sk^* again. Eventually $\mathcal{A}$ will find the value b_2.

In general, to find the m^{th} bit after finding $b_0, \ldots, b_{m-2}$, $\mathcal{A}$ defines g_m to be the following function: XOR the first $m-1$ bits of current key sk^* with $b_0, \ldots, b_{m-2}$ and append b_{m-1}^*. If the all zero message verifies, $\mathcal{A}$ sets $b_{m-1} := 0$. Else, if the message consisting of $m-1$ zeros and a 1 in the m^{th} position verifies, $\mathcal{A}$ sets $b_{m-1} := 1$. Otherwise $\mathcal{A}$ requests a signature of function g_m of the secret key sk^* again. Since the number of keys is polynomial in the security parameter, eventually $\mathcal{A}$ will find the value b_{m-1}. Using this method, $\mathcal{A}$ can recover the key sk_j with non-negligible probability.

Despite its simplicity, the attack in the proof of Remark 5.2 is quite devastating, and it might not be obvious if KDS-CMA security can be achieved at all. In the next section we show that, in the random oracle model, allowing the signing algorithm to be stateful enables the derivation of a KDS-CMA secure signature scheme from any EUF-CMA secure one.

5.1.3 One-Time Signatures

One-time digital signature schemes can be used at most once to sign, since otherwise signatures can possibly be forged. They were first presented by Lamport in [37] and Rabin in [52], and constructed from one-way functions. A *one-way function* (OWF) is a function f such that for each x in the domain of f, it is easy to compute $f(x)$; but for essentially all y in the range of f, it is computationally infeasible to find any

x such that $y = f(x)$. As proven in [54] and [36], secure signature schemes exist if and only if one-way functions exist.

As explained in [49], one-time signatures can be viewed as public commitments to a set of secrets chosen by the signer prior to signing a message. The commitments are elements of the range and the secrets are in the domain of the one-way functions. A well known scheme is the one proposed by Merkle in [45]. In [46], Merkle introduced the idea of using a hash tree to authenticate a large number of one-time signatures using a path. The one-time signature is EUF-CMA secure if an attacker, given a signature for a chosen message, can provide a valid signature for a different message with non-negligible probability. In Definition 5.1, $\mathcal{A}^{\text{euf}}$ can query $\mathcal{O}_{\mathcal{S}}$ only once.

5.1.4 From One-Time EUF-CMA to KDS-CMA: a Compiler

The compiler in Figure 5.1 uses a random oracle $H: \{0,1\}^* \longrightarrow \{0,1\}^k$ to transform any one-time EUF-CMA secure signature scheme into one that is KDS-CMA secure (in the random oracle model). While we do not expect this construction to be optimal from an efficiency point of view, it provides a tool to systematically construct KDS-CMA secure signature schemes.

Proposition 5.1. *Let $S = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ be a one-time signature scheme that is secure in the sense of* EUF-CMA. *Then the signature scheme $\widehat{S} = (\widehat{\mathcal{K}}, \widehat{\mathcal{S}}, \widehat{\mathcal{V}})$ obtained from the compiler in Figure 5.1 is secure in the sense of* KDS-CMA *in the random oracle model.*

Proof. Let $\mathcal{A}^{\text{kdm}}$ be an adversary in the sense of KDS-CMA, having a non-negligible success probability in creating a forgery for the signature scheme $\widehat{S}$. Then we can construct an adversary $\mathcal{A}^{\text{euf}}$ that violates EUF-CMA security of the underlying one-time signature scheme S. For doing so, we start with $\mathcal{A}^{\text{euf}}$ running a simulation

$\widehat{\mathcal{K}}$: Create a key pair $(sk_0^{\text{crt}}, pk_0^{\text{crt}}) \overset{\$}{\leftarrow} \mathcal{K}(1^k)$, return pk_0^{crt} as public verification key and use $sk := (sk_0^{\text{crt}}, \lambda, [])$ as initial state, where $[]$ is an empty list and λ the empty string.

$\widehat{\mathcal{S}}$: To sign the i^{th} $(1 \leq i)$ message $M \in \{0,1\}^*$, proceed as follows:

- Create two fresh key pairs $(sk_i^{\text{crt}}, pk_i^{\text{crt}}) \overset{\$}{\leftarrow} \mathcal{K}(1^k)$, $(sk_i^{\text{msg}}, pk_i^{\text{msg}}) \overset{\$}{\leftarrow} \mathcal{K}(1^k)$.
- Compute $Cert_i := pk_i^{\text{crt}} \parallel pk_i^{\text{msg}} \parallel \sigma_i$ with $\sigma_i \overset{\$}{\leftarrow} \mathcal{S}_{sk_{i-1}^{\text{crt}}}(pk_i^{\text{crt}} \parallel pk_i^{\text{msg}})$.
- Update the internal state $sk = (sk_{i-1}^{\text{crt}}, sk_{i-1}^{\text{msg}}, [Cert_\mu]_{1 \leq \mu \leq i-1}))$ to $sk \leftarrow (sk_i^{\text{crt}}, sk_i^{\text{msg}}, [Cert_\mu]_{1 \leq \mu \leq i})$.
- Compute $s \overset{\$}{\leftarrow} \mathcal{S}_{sk_i^{\text{msg}}}(r \parallel H(M \parallel r))$, where $r \overset{\$}{\leftarrow} \{0,1\}^k$ is chosen uniformly at random.
- Return the signature $(r, s, [Cert_\mu]_{1 \leq \mu \leq i})$.

$\widehat{\mathcal{V}}$: On input a message M and a candidate signature $(r, s, [Cert_\mu]_{1 \leq \mu \leq i})$, output 1 if all of the following conditions hold. Otherwise output 0:

- $\mathcal{V}_{pk_i^{\text{msg}}}(r \parallel H(M \parallel r), s) = 1$.
- $\mathcal{V}_{pk_{\mu-1}^{\text{crt}}}(pk_\mu^{\text{crt}} \parallel pk_\mu^{\text{msg}}, \sigma_\mu) = 1$ for all $1 \leq \mu \leq i$, where $Cert_\mu = pk_\mu^{\text{crt}} \parallel pk_\mu^{\text{msg}} \parallel \sigma_\mu$.

Figure 5.1: Deriving a KDS-CMA secure signature scheme, where it is assumed that public keys pk_i^{crt}, pk_i^{msg} are represented with a fixed length encoding.

of $\mathcal{A}^{\text{kds}}$, including a simulation of all oracles. We modify $\mathcal{A}^{\text{euf}}$'s simulation strategy through a short sequence of games, the last one yielding an attack on the EUF-CMA security of the underlying one-time signature scheme S.

Game 0. This is a trivial simulation of the original attack game played by $\mathcal{A}^{\text{kds}}$: The public verification key and initial secret state of the challenge for $\mathcal{A}^{\text{kds}}$ are fixed by $\mathcal{A}^{\text{euf}}$ by running the key generation algorithm $\widehat{\mathcal{K}}$. From here on, all needed oracles for $\mathcal{A}^{\text{kds}}$ can be simulated faithfully:

Random oracle: For simulating $\mathcal{A}^{\text{kds}}$'s random oracle, $\mathcal{A}^{\text{euf}}$ creates an empty list

L^{RO}. Then, whenever $\mathcal{A}^{\mathrm{kds}}$ queries its random oracle with a message x such that L^{RO} contains no entry of the form $(x, \cdot)$, $\mathcal{A}^{\mathrm{euf}}$ chooses a value $r_x \in \{0,1\}^k$ uniformly at random, appends the pair (x, r_x) to L^{RO} and sends r_x to $\mathcal{A}^{\mathrm{kds}}$. In case $\mathcal{A}^{\mathrm{kds}}$ queries L^{RO} a second time with the same value x, $\mathcal{A}^{\mathrm{euf}}$ returns the stored random value r_x.

Signing oracle: Knowing the initial secret key, $\mathcal{A}^{\mathrm{euf}}$ can faithfully answer queries to $\widehat{\mathcal{O}}_{\mathcal{S}}$ by simply executing $\widehat{\mathcal{S}}$ with the appropriate input and using the above simulation of the random oracle H.

Game 1. Let Collision be the event that during the simulation, $\mathcal{A}^{\mathrm{euf}}$ stores pairs (x, r_x) and $(x', r_{x'})$ in L^{RO} where $x \neq x'$ and $r_x = r_{x'}$. Whenever the event Collision occurs, $\mathcal{A}^{\mathrm{euf}}$ gives up, without creating a successful forgery. As $\mathcal{A}^{\mathrm{kds}}$ is polynomially bounded, Collision occurs with negligible probability only, and subsequently we may assume that the event Collision does not occur.

Game 2. Let q_s be a polynomial upper bound for the number of signing queries made by $\mathcal{A}^{\mathrm{kds}}$, and let g_i by the i^{th} function/message submitted to $\widehat{\mathcal{O}}_{\mathcal{S}}$ by $\mathcal{A}^{\mathrm{kds}}$. By pk^*, we denote the public key to be attacked by $\mathcal{A}^{\mathrm{euf}}$ in the definition of EUF-CMA security. In this game, $\mathcal{A}^{\mathrm{euf}}$ chooses an index $i^* \in \{0, \ldots, q_s\}$ and then, if $i^* \neq 0$, a flag $\Gamma \in \{\mathrm{crt}, \mathrm{msg}\}$ uniformly at random—for $i^* = 0$, we always set $\Gamma := \mathrm{crt}$. Now, in the simulation $\mathcal{A}^{\mathrm{euf}}$ replaces the public key $pk_{i^*}^{\Gamma}$ with the challenge public key pk^*. In case that $\mathcal{A}^{\mathrm{kds}}$ does not submit any signature queries to $\widehat{\mathcal{O}}_{\mathcal{S}}$, this modification is not detectable for $\mathcal{A}^{\mathrm{kds}}$. Similarly, answering signature queries g_i with $i < i^*$ is still possible, as the secret key sk^* associated to the challenge key pk^* is not needed here. Moreover, $\mathcal{A}^{\mathrm{euf}}$ can compute $Cert_{i^*+1}$:

- If $\Gamma = \mathrm{crt}$, then $\mathcal{A}^{\mathrm{euf}}$ can use its signing oracle to compute $Cert_{i^*+1}$.

- If $\Gamma = \text{msg}$, then $\mathcal{A}^{\text{euf}}$ knows $sk_{i^*}^{\text{crt}}$.

Consequently, $\mathcal{A}^{\text{euf}}$ is able to correctly answer all signature queries g_i with $i > i^*$, too. For the only "critical" query g_{i^*}, we consider two cases:

- If the value

$$\begin{cases} g_{i^*}((sk^*, sk_{i^*}^{\text{msg}}, [Cert_\mu]_{1 \le \mu \le i^*})) & \text{, if } \Gamma = \text{crt} \\ g_{i^*}((sk_{i^*}^{\text{crt}}, sk^*, [Cert_\mu]_{1 \le \mu \le i^*})) & \text{, if } \Gamma = \text{msg} \end{cases} \tag{5.1.1}$$

  can be predicted by $\mathcal{A}^{\text{kds}}$ with non-negligible probability, we modify $\mathcal{A}^{\text{kds}}$ to make such a prediction, therewith replacing the potentially key-dependent query g_{i^*} with a key-independent query g_{i^*}. The success probability of $\mathcal{A}^{\text{kds}}$ remains non-negligible, provided it was non-negligible before.

- If the value (5.1.1) can be predicted with negligible probability only, $\mathcal{A}^{\text{euf}}$ creates a key pair $(sk', pk') \overset{\$}{\leftarrow} \mathcal{K}(1^k)$, a random $r' \overset{\$}{\leftarrow} \{0,1\}^k$ and queries

$$m' \parallel r' := \begin{cases} g_{i^*}((sk', sk_{i^*}^{\text{msg}}, [Cert_\mu]_{1 \le \mu \le i^*})) \parallel r' & \text{, if } \Gamma = \text{crt} \\ g_{i^*}((sk_{i^*}^{\text{crt}}, sk', [Cert_\mu]_{1 \le \mu \le i^*})) \parallel r' & \text{, if } \Gamma = \text{msg} \end{cases}$$

  to its simulation of the random oracle. The use of sk' instead of sk^* in the evaluation of g_{i^*} cannot be noticed by $\mathcal{A}^{\text{kds}}$ unless the event $\mathsf{Collision}$ occurs.

The value $\mathcal{S}_{sk^*}(r' \parallel H(m' \parallel r'))$ is handed to $\mathcal{A}^{\text{kds}}$ as s-component of the signature. If $\Gamma = \text{msg}$, this value can be obtained from $\mathcal{A}^{\text{euf}}$'s signing oracle; otherwise $\mathcal{A}^{\text{euf}}$ can compute this value itself.

Game 3 Let $(M, (r, s, [Cert_\mu]_{1 \le \mu \le i}))$ be a successful forgery returned by $\mathcal{A}^{\text{kds}}$. With probability $\ge 1/(2q_s + 1)$ this forgery includes a signature on a message that can be verified successfully with $pk_{i^*}^{\Gamma} = pk^*$ and one of the following holds—in all other cases the simulation of $\mathcal{A}^{\text{kds}}$ needs to be restarted.

- The list $[Cert_\mu]_{1 \le \mu \le i}$ contains a $Cert_\mu = pk_\mu^{\text{crt}} \parallel pk_\mu^{\text{msg}} \parallel \sigma_\mu$, where $\mathcal{V}_{pk^*}(pk_\mu^{\text{crt}} \parallel pk_\mu^{\text{msg}}, \sigma_\mu) = 1$ and $pk_\mu^{\text{crt}} \parallel pk_\mu^{\text{msg}}$ has not been submitted to $\mathcal{A}^{\text{euf}}$'s signing oracle. Consequently, $\mathcal{A}^{\text{euf}}$ has created a valid forgery.

- We have $\mathcal{V}_{pk^*}(r \parallel H(M \parallel r), s) = 1$ and $r \parallel H(M \parallel r)$ has, with overwhelming probability, not been submitted to $\mathcal{A}^{\text{euf}}$'s signing oracle. Consequently, $\mathcal{A}^{\text{euf}}$ has created a valid forgery.

Summarizing, we see that if $\mathcal{A}^{\text{kds}}$'s forgery is valid with non-negligible probability, the same holds for $\mathcal{A}^{\text{euf}}$'s forgery.

□

5.2 Relation to Forward Security

Figure 5.2 summarizes the necessary small changes to the compiler in Figure 5.1. The time periods are included in the state, the signature, and the certificates.

Using Definitions 4.1 and 4.2 we can show that the scheme derived in Figure 5.2 is FWD-CMA secure by adapting the proof of Proposition 5.1 accordingly:

Proposition 5.2. *Let $S = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ be a one-time signature scheme that is secure in the sense of* EUF-CMA*. Then the key-evolving signature scheme $S^f = (\mathcal{K}^f, \mathcal{U}^f, \mathcal{S}^f, \mathcal{V}^f)$ obtained from the compiler in Figure 5.2 is secure in the sense of* FWD-CMA *in the random oracle model.*

Proof. Let $\mathcal{A}^{\text{fwd}}$ be an adversary in the sense of FWD-CMA, having a non-negligible success probability in creating a forgery for the signature scheme S^f given in Figure 5.2.

Then we can construct an adversary $\mathcal{A}^{\text{euf}}$ that violates $\mathsf{EUF\text{-}CMA}$ security of the underlying one-time signature scheme S.

cma phase We start with $\mathcal{A}^{\text{euf}}$ running a simulation of $\mathcal{A}^{\text{fwd}}$, and adapt *Game 0* in the proof of Proposition 5.1 in the obvious way: replace adversary $\mathcal{A}^{\text{kds}}$ with $\mathcal{A}^{\text{fwd}}$, signature scheme $\widehat{S}$ with S^f, and signing oracle $\widehat{\mathcal{O}}_{\mathcal{S}}$ with $\mathcal{O}^j_{\mathcal{S}^f}$. The public verification key and initial secret key of the challenge for $\mathcal{A}^{\text{fwd}}$ are fixed by $\mathcal{A}^{\text{euf}}$ by running $\mathcal{K}^f$. Likewise, we replace adversary $\mathcal{A}^{\text{kds}}$ with $\mathcal{A}^{\text{fwd}}$ in *Game 1*, and denote by q_s a polynomial upper bound for the number of signing queries made by $\mathcal{A}^{\text{fwd}}$ and by $\Gamma \in \{\text{crt}, \text{msg}\}$ a randomly chosen flag.

Let M_i be the i^{th} message submitted to $\mathcal{O}^j_{\mathcal{S}^f}$ by $\mathcal{A}^{\text{fwd}}$. By pk^* we denote the public key to be attacked by $\mathcal{A}^{\text{euf}}$ in the definition of $\mathsf{EUF\text{-}CMA}$ security. Analogously as in *Game 2*, $\mathcal{A}^{\text{euf}}$ selects an index $i^* \in \{-1, \ldots, q_s\}$ uniformly at random, and in the simulation replaces the public key $pk^{\Gamma}_{i^*}$ with the challenge public key pk^*. Answering signature queries M_i with $i \neq i^*$ is possible, as the relevant secret keys $sk_i^{\text{crt}}, sk_i^{\text{msg}}$ are known to $\mathcal{A}^{\text{euf}}$. Moreover, $\mathcal{A}^{\text{euf}}$ can use its own signing oracle to compute a valid signature of M_{i^*}. This enables $\mathcal{A}^{\text{euf}}$ to correctly answer all signature queries M_i for all time periods j.

breakin phase At any time interval j, $\mathcal{A}^{\text{fwd}}$ can output a special value $\mathsf{breakin}$ and obtain the current secret key $(j, sk_i^{\text{crt}}, sk_i^{\text{msg}}, [Cert_\mu]_{0\le\mu\le i})$, but must create a forgery using an index $b < j$ to be successful. When $i \neq i^*$, $\mathcal{A}^{\text{euf}}$ can correctly reveal the secret key sk_i. In case $i = i^*$, however, $\mathcal{A}^{\text{euf}}$ gives up without creating a forgery. The probability that $\mathcal{A}^{\text{fwd}}$ chooses $pk^{\Gamma}_{i^*}$ at target of its forgery—and consequently is not handed $sk^{\Gamma}_{i^*}$—is $\ge 1/(2q_s + 3)$, and therefore non-negligible.

forge phase Since the probability that $i = i^*$ is at least $1/(2q_s + 3)$, if $\mathcal{A}^{\text{fwd}}$ can forge with non-negligible probability p, then $p/(2q_s + 3)$ is still non-negligible, since q_s is polynomial in the security parameter. If $\mathcal{A}^{\text{fwd}}$ does not use $pk_{i^*}^{\Gamma}$ in its forgery, the simulation needs to be restarted. Otherwise, $\mathcal{A}^{\text{fwd}}$ outputs message M with verifiable signature $\langle b, (r, s, [Cert_\mu]_{0 \le \mu \le i}) \rangle$ and one of the following cases holds.

- The list $[Cert_\mu]_{0 \le \mu \le i}$ contains a $Cert_\mu = b \parallel pk_\mu^{\text{crt}} \parallel pk_\mu^{\text{msg}} \parallel \sigma_\mu$, where $\mathcal{V}_{pk^*}(b \parallel pk_\mu^{\text{crt}} \parallel pk_\mu^{\text{msg}}, \sigma_\mu) = 1$ and $b \parallel pk_\mu^{\text{crt}} \parallel pk_\mu^{\text{msg}}$ has not been submitted to $\mathcal{A}^{\text{euf}}$'s signing oracle. Consequently, $\mathcal{A}^{\text{euf}}$ has created a valid forgery.
- We have $\mathcal{V}_{pk^*}(r \parallel H(M \parallel r), s) = 1)$ and $r \parallel H(M \parallel r)$ has, with overwhelming probability, not been submitted to $\mathcal{A}^{\text{euf}}$'s signing oracle. Consequently, $\mathcal{A}^{\text{euf}}$ has created a valid forgery.

Hence, with non-negligible probability, $\mathcal{A}^{\text{fwd}}$'s forgery is also valid for $\mathcal{A}^{\text{euf}}$.

□

$\mathcal{K}^f$: Create a key pair $(sk_{-1}^{\mathrm{crt}}, pk_{-1}^{\mathrm{crt}}) \xleftarrow{\$} \mathcal{K}(1^k)$, return pk_{-1}^{crt} as public verification key and use $sk := (0, sk_{-1}^{\mathrm{crt}}, \lambda, [])$ as initial state, where [] is an empty list and λ the empty string.

$\mathcal{U}^f$: On input of the state $sk = (j, sk_i^{\mathrm{crt}}, sk_i^{\mathrm{msg}}, [Cert_\mu]_{0 \le \mu \le i})$, if $j = -1$, then we create two fresh key pairs $(sk_0^{\mathrm{crt}}, pk_0^{\mathrm{crt}}) \xleftarrow{\$} \mathcal{K}(1^k)$, $(sk_0^{\mathrm{msg}}, pk_0^{\mathrm{msg}}) \xleftarrow{\$} \mathcal{K}(1^k)$ and update the internal state to $sk \leftarrow (0, sk_0^{\mathrm{crt}}, sk_0^{\mathrm{msg}}, [Cert_0])$. Else, the internal state becomes $sk \leftarrow (j+1, sk_i^{\mathrm{crt}}, sk_i^{\mathrm{msg}}, [Cert_\mu]_{0 \le \mu \le i})$ leaving the secret keys and certificates the same.

$\mathcal{S}^f$: To sign the i^{th} $(1 \le i)$ message $M \in \{0,1\}^*$ proceed as follows:

- Create two fresh key pairs $(sk_i^{\mathrm{crt}}, pk_i^{\mathrm{crt}}) \xleftarrow{\$} \mathcal{K}(1^k)$, $(sk_i^{\mathrm{msg}}, pk_i^{\mathrm{msg}}) \xleftarrow{\$} \mathcal{K}(1^k)$.
- Set $Cert_i := j \parallel pk_i^{\mathrm{crt}} \parallel pk_i^{\mathrm{msg}} \parallel \zeta_i$ with $\zeta_i \xleftarrow{\$} \mathcal{S}_{sk_{i-1}^{\mathrm{crt}}}(j \parallel pk_i^{\mathrm{crt}} \parallel pk_i^{\mathrm{msg}})$ where j is the current time period.
- Update the internal state $sk = (j, sk_{i-1}^{\mathrm{crt}}, sk_{i-1}^{\mathrm{msg}}, [Cert_\mu]_{0 \le \mu \le i-1})$ to $sk \leftarrow (j, sk_i^{\mathrm{crt}}, sk_i^{\mathrm{msg}}, [Cert_\mu]_{0 \le \mu \le i})$.
- Compute $s \xleftarrow{\$} \mathcal{S}_{sk_i^{\mathrm{msg}}}(r \parallel H(M \parallel r))$, where $r \xleftarrow{\$} \{0,1\}^k$ is chosen uniformly at random.
- Return the signature $\langle j, (r, s, [Cert_\mu]_{0 \le \mu \le i}) \rangle$.

$\mathcal{V}^f$: On input a message M and a candidate signature $\langle j, (r, s, [Cert_\mu]_{0 \le \mu \le i}) \rangle$, output 1 if all of the following conditions hold. Otherwise output 0:

- $\mathcal{V}_{pk_i^{\mathrm{msg}}}(r \parallel H(M \parallel r), s) = 1$.
- $\mathcal{V}_{pk_{\mu-1}^{\mathrm{crt}}}(j \parallel pk_\mu^{\mathrm{crt}} \parallel pk_\mu^{\mathrm{msg}}, \zeta_\mu) = 1$ for all $0 \le \mu \le i$, where $Cert_\mu = j \parallel pk_\mu^{\mathrm{crt}} \parallel pk_\mu^{\mathrm{msg}} \parallel \zeta_\mu$.

Figure 5.2: Forward-secure modification of the KDS-CMA compiler from Figure 5.1, with time periods $j \in \{1, \ldots, T\}$ being understood as being represented with a fixed length encoding.

Chapter 6

Aggregate Signatures

Science means simply the aggregate of all the recipes that are always successful. All the rest is literature. Paul Valéry [28]

In Chapter 5, each signature in the compiler from Figure 5.1 contains a certificate chain in which a pair of public keys is signed with the previous certificate private key. Each time that a signature is created by the signer, the certificate chain gets appended with two public keys and a new signature. The KDS-CMA-secure signature quickly becomes very large in size. In this chapter, we discuss the definition and security model of an aggregate signature, and how it can make our certificates significantly smaller by aggregating all signatures in the chain.

6.1 Sequential Aggregate Signatures

In multisignatures (see [12], [15]), a set of users all sign the same message and the result is a single signature. For certificate chains, each message is different, so we want to aggregate signatures on distinct messages. In an aggregate signature, a single short object—called an aggregate—takes the place of n signatures by n signers on n messages. Because the signing and verification is ordered, we are interested in *sequential aggregate signatures* [39]. Some of the methods by which signatures can be aggregated include the use of bilinear maps [17], trapdoor permutations [41] , and identity-based schemes [16].

Definition 6.1 (Sequential Aggregate Signature Scheme). *A sequential aggregate signature scheme S_A is a triple of polynomial time algorithms $(\mathcal{K}_A, \mathcal{S}_A, \mathcal{V}_A)$:*

- $\mathcal{K}_A$ *is a probabilistic* key generation algorithm *which on input of the security parameter 1^k returns a pair (sk, pk) of keys—a public verification key pk with matching secret signing key sk.*
- $\mathcal{S}_A$ *is a probabilistic* signing algorithm *which on input of a (polynomial length) message $M \in \{0,1\}^*$, secret key sk, and aggregate-so-far by a set of l signers on l corresponding messages folds the new signature into the aggregate, yielding a new aggregate signature $\sigma \in \{0,1\}^*$ by $l+1$ signers on $l+1$ messages or an error symbol $\perp$.*
- $\mathcal{V}_A$ *is a deterministic* verification algorithm *which on input of l public keys, l corresponding messages, and a candidate aggregate signature σ returns 1 or 0, indicating whether the aggregate σ is a valid signature for the l messages under the l public keys.*

Remark 6.1. *Not all authors define aggregate signatures as above. In fact, we will use the scheme in [17] which has five algorithms: key generation, signing, verification, aggregation, and aggregate verification.*

6.2 Security Model

We now define the aggregate chosen-key security model as in [18]. Our adversary $\mathcal{A}^{\mathrm{agg}}$ is given a challenge public key and its goal is the existential forgery of an aggregate signature. The adversary is also given access to a signing oracle on the challenge key and the power to choose all other public keys.

Definition 6.2 (AGG-CMA)**.** *Let the triple $S_A = (\mathcal{K}_A, \mathcal{S}_A, \mathcal{V}_A)$ be a sequential aggregate signature scheme and $\mathcal{A}^{\mathrm{agg}}$ a probabilistic polynomial time algorithm. Consider the following attack scenario:*

1. *The aggregate forger $\mathcal{A}^{\mathrm{agg}}$ is provided with a public key pk_1, generated at random.*
2. *Proceeding adaptively, $\mathcal{A}^{\mathrm{agg}}$ requests signatures with pk_1 on messages of its choice.*
3. *Finally, $\mathcal{A}^{\mathrm{agg}}$ outputs $l-1$ additional public keys $pk_2, \ldots, pk_l$. Here l is at most polynomial in the security parameter. These keys, along with the initial key pk_1 will be included in $\mathcal{A}^{\mathrm{agg}}$'s forged aggregate. $\mathcal{A}^{\mathrm{agg}}$ also outputs messages $M_1, \ldots, M_l$ and an aggregate signature σ by the l users, each on his corresponding message.*

Let QueriedEarlier *be the event that $\mathcal{A}^{\mathrm{agg}}$ outputs a message M_1 that has been queried to the signing oracle $\mathcal{O}_{\mathcal{S}_A}$ under pk_1 already. The* success probability $\mathrm{Succ}_{\mathcal{A}}^{\mathrm{agg}} = \mathrm{Succ}_{\mathcal{A}^{\mathrm{agg}}}(k)$ *of $\mathcal{A}^{\mathrm{agg}}$ is defined as*

$$\mathrm{Succ}_{\mathcal{A}^{\mathrm{agg}}} := \Pr[\mathcal{V}_A(pk_1, \ldots, pk_l, M_1, \ldots, M_l, \sigma) = 1 \ \textit{and} \ \neg\mathsf{QueriedEarlier}],$$

and we call the aggregate signature scheme S_A secure in the sense of AGG-CMA *if* $\mathrm{Succ}_{\mathcal{A}^{\mathrm{agg}}}$ *is negligible for all probabilistic polynomial time adversaries $\mathcal{A}^{\mathrm{agg}}$.*

6.3 Aggregate Signatures from Bilinear Maps

Bilinear maps are used in Pairing-Based Cryptography (for example, Weil pairings as used to create short signatures in [20]). The scheme presented in this section from [18] would serve our purpose of compressing the certificate chain as presented in Figure 5.1 and has been proven secure under the Co-Gap Diffie-Hellman assumption. As a matter of fact, the signature will consist of a single group element. Although the creation of our certificate chain is done sequentially, the scheme presented in this section is more general, so verification does not have to occur in order.

6.3.1 Co-Gap Diffie-Hellman

To describe the bilinear aggregate signature, we need to introduce the following notation:

- G_1 and G_2 are two multiplicative cyclic groups of prime order p
- g_1 is a generator of G_1 and g_2 is a generator of G_2
- ψ is a computable isomorphism from G_2 to G_1 with $\psi(g_2) = g_1$
- e is a computable bilinear map $e : G_1 \times G_2 \to G_T$

We assume that ψ exists and is efficiently computable.

Definition 6.3 (Computational Co-Diffie Hellman). *Given $g_2, g_2^a \in G_2$ and $h \in G_1$ compute $h^a \in G_1$.*

Definition 6.4 (Decision Co-Diffie Hellman). *Given $g_2, g_2^a \in G_2$ and $h, h^b \in G_1$ output* yes *if $a = b$ and* no *otherwise. When the answer is* yes *we say that (g_2, g_2^a, h, h^a) is a co-Diffie-Hellman tuple.*

Definition 6.5 (Decision Group Pair)**.** *Two groups (G_1, G_2) are a decision group pair for co-Diffie-Hellman if the group action on G_1, the group action on G_2, and the map ψ from G_2 to G_1 can be computed in one time unit, and the Decision co-Diffie-Hellman on (G_1, G_2) can be solved in one time unit.*

Gap groups have the property that the Decision Co-Diffie-Hellman problem is easy, but the Computational Co-Diffie-Hellman is hard.

Definition 6.6. *The advantage of an algorithm $\mathcal{A}^{\text{cdh}}$ in solving the Computational co-Diffie-Hellman problem in groups G_1 and G_2 is*

$$\mathsf{Adv\ co\text{-}CDH}_{\mathcal{A}^{\text{cdh}}} = \Pr\left[\mathcal{A}^{\text{cdh}}(g_2, g_2^a, h) = h^a : a \overset{\$}{\leftarrow} \mathbb{Z}_p, h \overset{\$}{\leftarrow} G_1\right].$$

Definition 6.7 (Gap Group Pair)**.** *Two groups (G_1, G_2) are a co-GDH group pair if they are a decision group pair for co-Diffie-Hellman and no algorithm breaks Computational co-Diffie-Hellman in at most time t with $\mathsf{Adv\ co\text{-}CDH}_{\mathcal{A}^{\text{cdh}}}$ at least ϵ.*

6.3.2 Bilinear Maps

Let G_1 and G_2 be two groups as described in Section 6.3.1 with an additional target group G_T such that $|G_1| = |G_2| = |G_T|$. A bilinear map is a map $e : G_1 \times G_2 \to G_T$ with the following properties:

- Bilinear: for all $u \in G_1$, $v \in G_2$, and $a, b \in \mathbb{Z}$, $e(u^a, v^b) = e(u, v)^{ab}$.
- Non-degenerate: $e(g_1, g_2) \neq 1$.

These properties imply that for any $u_1, u_2 \in G_1$, $v \in G_2$, $e(u_1 u_2, v) = e(u_1, v) \cdot e(u_2, v)$ and for any $u, v \in G_2$, $e(\psi(u), v) = e(\psi(v), u)$.

6.3.3 Bilinear Aggregate Signatures

In this scheme, all messages $M_i \in \{0,1\}^*$ must be distinct. We will use groups G_1 and G_2 as described above, along with isomorphism ψ, and bilinear map e. The scheme has five algorithms and employs a random oracle H.

- **Key Generation.** For a particular user, pick random $x \overset{\$}{\leftarrow} \mathbb{Z}_p$, and compute $v \leftarrow g_2^x$. The user's public key is $v \in G_2$. The user's secret key is $x \in \mathbb{Z}_p$.

- **Signing.** For a particular user, given the secret key x and a message $M \in \{0,1\}^*$, compute $h \leftarrow H(M)$, where $h \in G_1$, and $\sigma \leftarrow h^x$. The signature is $\sigma \in G_1$.

- **Verification.** Given a user's public key v, a message M, and a signature σ, compute $h \leftarrow H(M)$; accept if $e(\sigma, g_2) = e(h, v)$ holds.

- **Aggregation.** For the aggregating subset of users $U \subseteq \mathbb{U}$, assign to each user an index i, ranging from 1 to $k = |U|$. Each user $u_i \in U$ provides a signature $\sigma_i \in G_1$ on a message $M_i \in \{0,1\}^*$ of its choice. The messages M_i must all be distinct. Compute $\sigma \leftarrow \prod_{i=1}^{k} \sigma_i$. The aggregate signature is $\sigma \in G_1$.

- **Aggregate Verification.** We are given aggregate signature $\sigma \in G_1$ for an aggregating subset of users U, indexed as before, and are given the original messages $M_i \in \{0,1\}^*$ and public keys $v_i \in G_2$ for all users $u_i \in U$. To verify the aggregate signature σ,

  1. ensure that the messages M_i are all distinct, and reject otherwise;
  2. compute $h_i \leftarrow H(M_i)$ for $1 \leq i \leq k = |U|$, and accept if $e(\sigma, g_2) = \prod_i e(h_i, v_i)$ holds.

Using the properties of a bilinear map, one can show that $e(\sigma, g_2) = \prod_i e(h_i, v_i)$ as required.

6.4 Application to $\mathsf{KDS\text{-}CMA}$ Compiler

For the compiler in Figure 5.1, the signing and verification algorithm in Section 6.3.3 can be used for signing key-dependent messages. The aggregation and aggregate verification can be used to create the certificate chain. However, the adversary $\mathcal{A}^{\text{kds}}$ cannot request any signatures with pk^* when $\Gamma = crt$ on a message of its choice during the simulation. Therefore $\mathcal{A}^{\text{kds}}$ is a passive adversary in the certificates; i.e., the aggregate setting. Since the bilinear aggregate signature is $\mathsf{AGG\text{-}CMA}$ secure, then the scheme must also be secure against a passive adversary, since it is a weaker attack. Below, we outline the modifications that can be made to the $\mathsf{KDS\text{-}CMA}$ compiler using any secure aggregate signature.

6.4.1 Signature Modification

The creation of the certificates occurs in the signing algorithm of the compiler. The certificates are modified in the following way after the creation of the two fresh key pairs:

- Compute aggregate signature $\sigma_i \overset{\$}{\leftarrow} \mathcal{S}_A(pk_i^{\text{crt}} \parallel pk_i^{\text{msg}}, sk_{i-1}^{\text{crt}}, \sigma_{i-1})$. Note that by Definition 6.1, the input to the aggregate signature consists of a message, secret key, and aggregate-so-far.
- Set $Cert_i := pk_i^{\text{crt}} \parallel pk_i^{\text{msg}} \parallel \sigma_i$.
- Set $Cert_{i-1} := pk_{i-1}^{\text{crt}} \parallel pk_{i-1}^{\text{msg}}$.

From here, the signer updates the internal state, signs the message, and returns the signature. Since we no longer need σ_{i-1} thanks to our aggregate, we can erase σ_{i-1} from our internal state. Our certificate chain may contain many public keys but only one signature.

We can now apply the bilinear aggregate signature. Let x^{crt} and x^{msg} be the secret keys with corresponding public keys v^{crt} and v^{msg} in G_2. To compute the i^{th} certificate, the signer does the following:

- Compute $h_i := H(v_i^{\text{crt}} \parallel v_i^{\text{msg}}) \in G_1$.
- Set $\sigma_i := h_i^{x_{i-1}^{\text{crt}}} \cdot \sigma_{i-1}$.
- Update the internal state by setting $Cert_i := v_i^{\text{crt}} \parallel v_i^{\text{msg}} \parallel \sigma_i$ and $Cert_{i-1} := v_{i-1}^{\text{crt}} \parallel v_{i-1}^{\text{msg}}$.

6.4.2 Verification Modification

The compiler verification algorithm will consist of only two verifications (as opposed to $i+1$). Let $pk^{\text{crt}} = pk_0^{\text{crt}} \parallel \cdots \parallel pk_i^{\text{crt}}$ and $M = (pk_1^{\text{crt}} \parallel pk_1^{\text{msg}}) \parallel \cdots \parallel (pk_i^{\text{crt}} \parallel pk_i^{\text{msg}})$. If both of the following conditions hold output 1, otherwise output 0:

- $\mathcal{V}_{pk_i{}^{\text{msg}}}(r \parallel H(M \parallel r), s) = 1$
- $\mathcal{V}_A(pk^{\text{crt}} \parallel M \parallel \sigma_i) = 1$.

Recall that the aggregate verification algorithm takes as input the public keys, messages, and aggregate. For the bilinear aggregate signature, the verifier does the following:

- Check that the messages $M_j = v_j^{\text{crt}} \parallel v_j^{\text{msg}}$ for $j = 1 \ldots i$ are distinct.

- Compute $h_j := H(M_j)$ for $j = 1 \ldots i$.
- If $e(\sigma_i, g_2) = \prod_{j=1}^{i} e(h_j, v_j^{\mathrm{crt}})$, then output 1. Otherwise, output 0.

Using an aggregate signature, the growth of the KDS-CMA-secure compiled signature is linear in the security parameter k.

Chapter 7

Key Dependence in Signcryption

Cypher: I know what you're thinking, 'cause right now I'm thinking the same thing. Actually, I've been thinking it ever since I got here: Why oh why didn't I take the BLUE pill? [1]

In 1996, Zheng [58] introduced the concept of signcryption which achieves privacy and authenticity with one cryptographic primitive rather than two. Traditionally, one signs a message by attaching to it a digital signature and then encrypts the message by using an encryption algorithm. Zheng noted that the computational costs for signing and then encrypting in the public key setting can be quite high and proposed a more economical approach. Since then, many signcryption schemes have been proposed.

In [2], An et al. explore the generic sequential composition methods of building signcryption from a signature and an encryption scheme. In this chapter, we will use their security definitions and state some of their results. Then, we explore the setting in which we compose KDM-secure encryption schemes with KDS-secure signatures and provide some partial results.

7.1 Signcryption in the Two-User Setting

A signcryption scheme $\mathcal{SC}$ is a triple of algorithms (G, SE, VD). The algorithm $G(1^k)$, where k is the security parameter, outputs a pair of keys (SDK, VEK). SDK is the key used to sign and decrypt; VEK is the key used to verify and encrypt. Each participant must invoke G. We denote the sender's secret and public key by SDK_S and VEK_S; likewise, the receiver's secret and public key by SDK_R and VEK_R. The randomized signcryption algorithm SE takes as input SDK_S, VEK_R, and a message m in message space $\mathcal{M}$ to output signcryption u. We write $U \overset{\$}{\leftarrow} SE(M)$. The deterministic de-signcryption algorithm VD takes as input the signcryption U, SDK_R, and VEK_S to output a message $M \in \mathcal{M} \cup \{\perp\}$. We write $M \leftarrow VD(U)$. We require that $VD(SE(M)) = M$ for any $M \in \mathcal{M}$.

7.1.1 Outsider Security

Outsider security assumes that the signcryption adversary $\mathcal{A}_{SC}$ is neither the sender nor receiver. That is, $\mathcal{A}_{SC}$ only knows the public information $pub = [VEK_R, VEK_S]$. We give $\mathcal{A}_{SC}$ access to a signcryption oracle.

To break the EUF-CMA security of the signcryption scheme $\mathcal{A}_{SC}$ has to come up with a valid signcryption U of a message M that was not previously queried to the signcryption oracle (but $\mathcal{A}_{SC}$ is not required to know M). The scheme is *outsider-secure* in the EUF-CMA sense if any probabilistic polynomial time algorithm $\mathcal{A}_{SC}$ has a negligible probability of creating a valid signcryption.

To break the IND-CPA security of the signcryption scheme, $\mathcal{A}_{SC}$ has to come up with two messages M_0 and M_1, one of which is signcrypted at random. The corresponding signcryption U is given to $\mathcal{A}_{SC}$, who must guess which message was signcrypted. If $\Pr[\mathcal{A}_{SC} \text{ succeeds}] \leq \frac{1}{2} + negl(k)$, then we say that the scheme is *outsider-secure* in the IND-CPA sense.

7.1.2 Insider Security

Stronger than outsider security, insider security assumes that the adversary $\mathcal{A}_{SC}$ is a legal user of the system (either the sender or receiver). Given any signcryption scheme $\mathcal{SC} = (G, SE, VD)$, we can define an induced signature scheme $S = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ and encryption scheme $E = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. The signing/verification and encryption/decryption keys are (SK, VK) and (EK, DK) respectively.

- **Signature** S**.** The generation algorithm $\mathcal{K}$ runs $G(1^k)$ twice to produce two key pairs (SDK_S, VEK_S) and (SDK_R, VEK_R). Let $pub = [VEK_S, VEK_R]$. We set the signing key to $SK = [SDK_S, pub]$ and the verification key to $VK = [SDK_R, pub]$. The signing algorithm $\mathcal{S}$ outputs $U = SE(M)$ as a signature for message M. The verification algorithm $\mathcal{V}$ outputs a 1 if $VD(U) \neq \perp$; otherwise $\mathcal{V}$ outputs 0.

- **Encryption** E**.** The generation algorithm $\mathcal{K}$ runs $G(1^k)$ twice to produce two key pairs (SDK_S, VEK_S) and (SDK_R, VEK_R). Let $pub = [VEK_S, VEK_R]$. We set the encryption key to $EK = [SDK_S, pub]$ and the decryption key to $DK = [SKD_R, pub]$. The encryption algorithm $\mathcal{E}$ takes a message M as input and outputs $U = SE(M)$. The decryption algorithm $\mathcal{D}$ outputs $VD(U)$.

We say that the signcryption scheme is *insider-secure* against the coresponding attack (CPA/CMA) on the privacy/authenticity property, if the corresponding induced encryption/signature is secure against the same attack.

7.2 Sequential Compositions of Encryption and Signature

The two methods of constructing signcryption schemes using compositions that we will discuss are encrypt-then-sign ($\mathcal{E}t\mathcal{S}$), and sign-then-encrypt ($\mathcal{S}t\mathcal{E}$), where the encryption scheme is $E = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ and the signature scheme is $S = (\mathcal{K}, \mathcal{S}, \mathcal{V})$. Both $\mathcal{E}t\mathcal{S}$ and $\mathcal{S}t\mathcal{E}$ have the same generation algorithm $G(1^k)$. It runs the encryption key generation algorithm which outputs (EK, DK) and the signing key generation algorithm which outputs (SK, VK). Again, we use S to denote the sender and R to denote the receiver.

We define $\mathcal{E}t\mathcal{S}$ by $U \leftarrow SE(M; (SK_S, EK_R)) = \mathcal{S}_S(\mathcal{E}_R(M))$. We assume that the message can be determined by the signature. So let $\widetilde{U}$ be the message in signature U. If $\mathcal{V}_S(U) = 1$, then $\widetilde{M} = \mathcal{D}_R(\widetilde{U})$; otherwise $\widetilde{M} = \perp$. Similarly, we define $\mathcal{S}t\mathcal{E}$ by $U \leftarrow SE(M; (SK_S, EK_R)) = \mathcal{E}_R(\mathcal{S}_S(M))$. To de-signcrypt U, let $s = \mathcal{D}_R(U)$. If $\mathcal{V}_S(s) = 1$, then $\widetilde{M}$ is the message corresponding to signature s; otherwise, $\widetilde{M} = \perp$. We define $VD(U; (DK_R, VK_S)) = \widetilde{M}$ in both cases.

Both $\mathcal{E}t\mathcal{S}$ and $\mathcal{S}t\mathcal{E}$ preserve the security properties of E and S in terms of insider-security as seen in Theorem 7.1. Theorems 7.2 and 7.3 show that they improve the security properties of $\mathcal{E}$ and $\mathcal{S}$ in terms of outsider-security. IND-CCA security means indistinguishability against adaptively chosen ciphertext attacks and is stronger than IND-CPA. EUF-NMA security means existential unforgeability against no message attacks and is weaker than EUF-CMA. Here are some results from [2].

Theorem 7.1. *If E is* IND-CCA*-secure and S is* EUF-CMA*-secure, then $\mathcal{E}t\mathcal{S}$ and $\mathcal{S}t\mathcal{E}$ are both* IND-CCA *and* EUF-CMA *secure in the insider-security model.*

Theorem 7.2. *If E is* IND-CPA*-secure and S is* EUF-CMA*-secure, then $\mathcal{E}t\mathcal{S}$ is*

IND-CCA*-secure in the outsider-security model and* EUF-CMA*-secure in the insider-security model.*

Theorem 7.3. *If E is* IND-CCA*-secure, and S is* EUF-NMA*-secure, then $\mathcal{S}t\mathcal{E}$ is* IND-CCA*-secure in the insider-security model and* EUF-CMA*-secure in the outsider-security model.*

7.3 Insider Security in the Presence of Key Dependent Messages

Consider the case where $E = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is KDM-CPA-secure and $S = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ is KDS-CMA-secure. We now prove that $\mathcal{E}t\mathcal{S}$ and $\mathcal{S}t\mathcal{E}$ preserve the security properties of $\mathcal{E}$ and $\mathcal{S}$ in the insider-security model.

Theorem 7.4. *If E is* KDM-CPA*-secure, and S is* KDS-CMA*-secure, then $\mathcal{E}t\mathcal{S}$ and $\mathcal{S}t\mathcal{E}$ are both* KDM-CPA*-secure and* KDS-CMA*-secure in the insider-security model.*

Proof. This is similar to the proof of Theorem 1 given in [2]. We assume that each signature has the corresponding message appended, except when a function of the secret key is queried to the signing oracle.

(1) **KDS-CMA-Security of $\mathcal{E}t\mathcal{S}$.** Given any forger $\mathcal{A}'$ for $\mathcal{E}t\mathcal{S}$, we can construct a forger $\mathcal{A}$ for S. $\mathcal{A}$ views the keys of S as (SK_S, VK_S), and by itself picks a pair of encryption keys $(EK_R, DK_R) \xleftarrow{\$} \mathcal{K}(1^k)$. $\mathcal{A}$ then gives $\mathcal{A}'$ the public keys (EK_R, DK_R, VK_S) of the induced signature scheme. $\mathcal{A}$ simulates the signcryption query of $\mathcal{A}'$ for message M' by first creating $e' \leftarrow \mathcal{E}_R(M')$ and then asking the signing oracle for $\mathcal{S}$ to sign e'. Note that M' can depend on SK_S and the oracle for $\mathcal{S}$ can handle such queries. At some point, $\mathcal{A}'$ produces

a forgery U and message M for $\mathcal{E}t\mathcal{S}$. $\mathcal{A}$ outputs signature U and message e (which is an encryption of M) as a forgery.

(2) KDS-CMA-**Security of $\mathcal{S}t\mathcal{E}$.** Given any forger $\mathcal{A}'$ for $\mathcal{S}t\mathcal{E}$, we can construct a forger $\mathcal{A}$ for S. $\mathcal{A}$ views the keys of S as (SK_S, VK_S), and by itself picks a pair of encryption keys $(EK_R, DK_R) \overset{\$}{\leftarrow} \mathcal{K}(1^k)$. $\mathcal{A}$ then gives $\mathcal{A}'$ the public key (EK_R, DK_R, VK_S) of the induced signature scheme. $\mathcal{A}$ simulates the signcryption query of $\mathcal{A}'$ for message M' by first asking the oracle for $\mathcal{S}$ to produce the signature s' for M', and then returning $U' \leftarrow \mathcal{E}_R(s')$. Again, the message M' can depend on the secret key SK_S, since $\mathcal{S}$ has a signing oracle that can handle key-dependent queries. When $\mathcal{A}'$ produces a forged signcryption U of a new message M (w.r.t. $\mathcal{S}t\mathcal{E}$), $\mathcal{A}$ outputs $\mathcal{D}_R(u)$ (since $\mathcal{A}$ picked the encryption keys) and M as a forgery.

(3) KDM-CPA-**Security of $\mathcal{E}t\mathcal{S}$.** Assume that $\mathcal{E}t\mathcal{S}$ is not KDM-CPA-secure and $\mathcal{A}'$ is a distinguisher. We will use $\mathcal{A}'$ to construct a distinguisher $\mathcal{A}$ for E as follows. $\mathcal{A}$ views the keys of E as (EK_R, DK_R), and creates a pair of signing keys $(SK_S, VK_S) \overset{\$}{\leftarrow} \mathcal{K}(1^k)$. $\mathcal{A}$ then gives $\mathcal{A}'$ the public key (EK_R, SK_S, VK_S) of the induced encryption scheme. To simulate the signcryption query M' made by $\mathcal{A}'$, $\mathcal{A}$ computes $e' \leftarrow \mathcal{E}_R(M')$, and then computes the signature $s' \leftarrow \mathcal{S}_S(e')$ when M' is not key dependent.

For key-dependent queries, $\mathcal{A}$ queries $g'(DK_R)$ to its encryption oracle, which can handle key-dependent messages, and receives signature s'. Then $\mathcal{A}$ computes the signature $s' \leftarrow \mathcal{S}_S(e')$ and hands it over to $\mathcal{A}'$. When $\mathcal{A}'$ outputs a message M to be signcrypted in either the real or random oracle, $\mathcal{A}$ outputs the same message. When $\mathcal{A}$ receives the challenge ciphertext e, $\mathcal{A}$ hands $U \leftarrow \mathcal{S}_S(e)$ to $\mathcal{A}'$. Recall that the adversary outputs a bit b' which stands for real when it is 1 and for random when 0. $\mathcal{A}$ outputs the same guess b', which gives $\mathcal{A}$ the same probability of being correct as $\mathcal{A}'$.

(4) $\mathsf{KDM\text{-}CPA}$-**Security of $\mathcal{S}t\mathcal{E}$.** Assume that $\mathcal{S}t\mathcal{E}$ is not $\mathsf{KDM\text{-}CPA}$-secure and $\mathcal{A}'$ is a distinguisher. We will use $\mathcal{A}'$ to construct a distinguisher $\mathcal{A}$ for E as follows. $\mathcal{A}$ views the keys of E as (EK_R, DK_R), and creates a pair of signing keys $(SK_S, VK_S) \overset{\$}{\leftarrow} \mathcal{K}(1^k)$. $\mathcal{A}$ then gives $\mathcal{A}'$ the public key (EK_R, SK_S, VK_S) of the induced encryption scheme. To simulate the signcryption query M' made by $\mathcal{A}'$, $\mathcal{A}$ computes $s' \leftarrow \mathcal{S}_S(M')$, and then computes the encryption $e' \leftarrow \mathcal{E}_R(s')$ when M' is not key dependent.

For key-dependent queries, $\mathcal{A}$ defines signing as a function g and then requests an encryption of the composition $g'(g(DK_R))$ to the encryption oracle and hands the output e to $\mathcal{A}'$. Since signing may be probabilistic, by fixing the secret key and selecting the random coins, we can define the signing algorithm as a function g that depends only on the message to be signed. When $\mathcal{A}'$ outputs a message M to be signcrypted in either the real or random oracle, $\mathcal{A}$ outputs $s \leftarrow \mathcal{S}_S(M)$. Then $\mathcal{A}$ gives $\mathcal{A}'$ the same challenge that it receives, U. $\mathcal{A}$ outputs the same guess b' as $\mathcal{A}$, which gives both adversaries the same probability of being correct.

□

7.4 Outsider Security in the Presence of Key Dependent Messages

Notice that when it comes to outsider-secure compositions, Theorems 7.2 and 7.3 show that the encryption and signing security can be relaxed while keeping the compositions secure. The question that naturally arises is whether one can compose a scheme that is secure against key-dependent messages with one that is not, while keeping the composition secure against key dependence. Here we offer a counterexample in the composition $\mathcal{E}t\mathcal{S}$.

Remark 7.1. *Let E be an encryption scheme secure in the sense of* IND-CPA *and S a signature scheme secure in the sense of* KDS-CMA*. Then $\mathcal{E}t\mathcal{S}$ is not necessary insider-secure in the sense of* KDS-CMA *and outsider-secure in the sense of* KDM-CPA*.*

Proof. Suppose that $\mathcal{E}$ is an IND-CPA-secure encryption scheme. We modify $\mathcal{E}$ as follows:

- If $M \neq DK_R$, then output $\mathcal{E}_R(M)$.
- If $M = DK_R$, then output DK_R.

The modified encryption algorithm is still IND-CPA-secure.

Now suppose that $\mathcal{S}$ is an KDS-CMA-secure signature scheme. We modify $\mathcal{S}$ as follows:

- If $M \neq DK_R$, then output $\mathcal{S}_S(M)$.
- If $M = DK_R$, then output DK_R.

The modified signature algorithm is still KDS-CMA-secure, because the only key that S protects is SK_S, which is independent of DK_R. Therefore, a signcryption of the secret encryption key $\mathcal{E}t\mathcal{S}(DK_R)$ reveals the entire key.

□

Likewise, we offer a similar counterexample in the composition $\mathcal{S}t\mathcal{E}$.

Remark 7.2. *Let E be an encryption scheme secure in the sense of* KDM-CPA *and S a signature scheme secure in the sense of* EUF-CMA*. Then $\mathcal{S}t\mathcal{E}$ is not necessarily insider-secure in the sense of* KDM-CPA *and outsider-secure in the sense of* KDS-CMA*.*

Proof. Now suppose that $\mathcal{S}$ is an EUF-CMA-secure signature scheme. We modify $\mathcal{S}$ as follows:

- If $M \neq SK_S$, then output $\mathcal{S}_S(M)$.
- If $M = SK_S$, then output SK_S.

The modified signature algorithm is still EUF-CMA-secure.

Now suppose that $\mathcal{E}$ is an KDM-CPA-secure encryption scheme. We modify $\mathcal{E}$ as follows:

- If $M \neq SK_S$, then output $\mathcal{E}_R(M)$.
- If $M = SK_S$, then output SK_S.

The modified encryption algorithm is still KDM-CPA-secure because the only key that E protects is DK_R which is independent of SK_S. Therefore, a signcryption of the secret signing key $\mathcal{S}t\mathcal{E}(SK_S)$ reveals the entire key.

$\square$

Chapter 8

Conclusion and Open Problems

A conclusion is the place where you got tired of thinking. Arthur Bloch [14]

8.1 Summary

In this dissertation, we began by discussing the concept of key dependence in the encryption setting. Then, we explored key dependence in message authentication codes and signature schemes. Given an existentially unforgeable one-time signature scheme, the construction we presented yields a signature scheme offering strong guarantees in the presence of key-dependent messages. This construction used the concept of evolving the secret key as done in forward secure signature schemes. Since the compiler created a large signature, we discussed how aggregate signatures can alleviate the size problem in the certificate chain. In the signcryption setting, we showed that KDS-CMA security and KDM-CPA security can be achieved in the insider-security model when encryption and signing are composed.

8.2 Open Problems

An open problem is whether one can construct a (stateful) constant length signature scheme with a fixed public key. The challenge is to tie the signer to the public key without using an increasing sequence of certification chains or expanding trees. One possible solution could be to create a group signature scheme without a group manager, where current members can add new members. The adversary will not be helped by the current state, because it will not include the new member's secret key which will be used in the next signature. As we have seen, the verification algorithm is a very powerful tool for the adversary in MACs and signatures which are key-dependent. Also, what would a KDS-CMA-secure signature look like in the standard model? Perhaps some of the tools in leakage-resilience can be used here.

Another open problem is to define the security of key-privacy in the presence of key-dependent messages and construct a scheme that meets this new definition. In addition, one would like to construct a signcryption scheme that is KDS-CMA and KDM-CPA insider-secure that is smaller than the composition of signing then encrypting or vice versa. As for the message recognition protocol from [43], an open problem is to provide an alternative construction that avoids the attack in Section 3.4 along with a new security proof.

BIBLIOGRAPHY

[1] The Matrix. Distributed by Warner Bros. Pictures, 1999.

[2] J. H. An, Y. Dodis, and T. Rabin. On the Security of Joint Signature and Encryption. In L. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002: International Conference on the Theory and Applications of Cryptographic Techniques*, Lecture Notes in Computer Science, pages 83–107, London, UK, 2002. Springer-Verlag.

[3] R. Anderson, F. Bergadano, B. Crispo, J. H. Lee, C. Manifavas, and R. Needham. A New Family of Authentication Protocols. *ACM SIGOPS Operating Systems Review*, 32(4):9–20, 1998.

[4] R. Armour. *It All Started With Eve.* Mosby, 1970.

[5] M. Backes, M. Dürmuth, and D. Unruh. OAEP is Secure Under Key-dependent Messages. In J. Pieprzyk, editor, *Advances in Cryptology – ASIACRYPT 2008: 14th International Conference on the Theory and Application of Cryptology and Information Security*, volume 5350 of *Lecture Notes in Computer Science*, pages 506–523, Berlin / Heidelberg, 2008. Springer.

[6] M. Backes, B. Pfitzmann, and A. Scedrov. Key-Dependent Message Security under Active Attacks–BRSIM/UC-Soundness of Symbolic Encryption with Key Cycles. In *CSF 2007: Proceedings of the 20th IEEE Computer Security Foundations Symposium*, pages 112–124, Washington, DC, USA, 2007. IEEE Computer Society.

[7] M. Bellare. A Note on Negligible Functions. *Journal of Cryptology*, 15(4):271–284, 2002.

[8] M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval. Key-Privacy in Public-Key Encryption. In C. Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001: 7th International Conference on the Theory and Application of Cryptology and Information Security*, volume 2248 of *Lecture Notes in Computer Science*, pages 566–582, London, UK, 2001. Springer-Verlag.

[9] M. Bellare and S. Micali. How to Sign Given Any Trapdoor Permutation. *Journal of the ACM*, 39(1):214–233, 1992.

[10] M. Bellare and S. Miner. A Forward-Secure Digital Signature Scheme. In M. Wiener, editor, *Advances in Cryptology – CRYPTO 2009: 29th Annual International Cryptology Conference*, volume 1666 of *Lecture Notes in Computer Science*, pages 431–448, Berlin / Heidelberg, 1999. Springer.

[11] M. Bellare and S. Miner. A Forward-Secure Digital Signature Scheme. Available at `http://www-cse.ucsd.edu/users/sminer/abstracts/ForwSecSigs.html`, July 1999. Full version of [10].

[12] M. Bellare and G. Neven. Multi-Signatures in the Plain Public-Key Model and a General Forking Lemma. In *CCS 2006: Proceedings of the 13th ACM Conference on Computer and Communications Security*, pages 390–399, New York, NY, USA, 2006. ACM.

[13] J. Black, P. Rogaway, and T. Shrimpton. Encryption-Scheme Security in the Presence of Key-Dependent Messages. In K. Nyberg and H. M. Heys, editors, *Selected Areas in Cryptography – SAC 2003: 10th Annual International*

Workshop, volume 2595 of *Lecture Notes in Computer Science*, pages 62–75, London, UK, 2003. Springer-Verlag.

[14] A. Bloch. *Murphy's Law: The 26th Anniversary Edition*. Perigee Trade, 2003.

[15] A. Boldyreva. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In Y. Desmedt, editor, *Public Key Cryptography – PKC 2003: 6th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 31–46, Berlin / Heidelberg, 2003. Springer.

[16] A. Boldyreva, C. Gentry, A. O'Neill, and D. H. Yum. Ordered Multisignatures and Identity-Based Sequential Aggregate Signatures, with Applications to Secure Routing. In *CCS 2007: Proceedings of the 14th ACM Conference on Computer and Communications Security*, pages 276–285, New York, NY, USA, 2007. ACM.

[17] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In E. Biham, editor, *Advances in Cryptology – EUROCRYPT 2003: International Conference on the Theory and Applications of Cryptographic Techniques*, volume 2656 of *Lecture Notes in Computer Science*, pages 416–432, Berlin / Heidelberg, 2003. Springer.

[18] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In E. Biham, editor, *Advances in Cryptology – EUROCRYPT 2003: International Conference on the Theory and Applications of Cryptographic Techniques*, volume 2656 of *Lecture Notes in Computer Science*, pages 416–432, Berlin / Heidelberg, 2003. Springer.

[19] D. Boneh, S. Halevi, M. Hamburg, and R. Ostrovsky. Circular-Secure Encryption from Decision Diffie-Hellman. In D. Wagner, editor, *Advances in Cryptology –*

CRYPTO 2008: 28th Annual International Cryptology Conference, volume 5157 of *Lecture Notes in Computer Science*, pages 108–125, Berlin / Heidelberg, 2008. Springer-Verlag.

[20] D. Boneh, B. Lynn, and H. Shacham. Short Signatures from the Weil Pairing. *Journal of Cryptology*, 17(4):297–319, 2004.

[21] B. Borden. *Living Like Benjamin: Making Dreams Come True.* AuthorHouse, 2007.

[22] J. Camenisch, N. Chandran, and V. Shoup. A Public Key Encryption Scheme Secure against Key Dependent Chosen Plaintext and Adaptive Chosen Ciphertext Attacks. In A. Joux, editor, *Advances in Cryptology – EUROCRYPT 2009: International Conference on the Theory and Applications of Cryptographic Techniques*, volume 5479 of *Lecture Notes in Computer Science*, pages 351–368, Berlin / Heidelberg, 2009. Springer-Verlag.

[23] J. Camenisch and A. Lysyanskaya. An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation. In B. Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001: International Conference on the Theory and Applications of Cryptographic Techniques*, volume 2045 of *Lecture Notes in Computer Science*, pages 93–118, Berlin / Heidelberg, 2001. Springer.

[24] R. Canetti, O. Goldreich, and S. Halevi. The Random Oracle Methodology, revisited. *Journal of the ACM*, 51(4):557–594, 2004.

[25] K. Covy. *When Happily Ever After Ends.* Sphinx Publishing, 2006.

[26] S. Dziembowski and K. Pietrzak. Leakage-Resilient Cryptography. In *FOCS 2008: Proceedings of the 2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 293–302, Washington, DC, USA, 2008. IEEE Computer Society.

[27] S. Faust, E. Kiltz, K. Pietrzak, and G. Rothblum. Leakage-Resilient Signatures. Cryptology ePrint Archive, Report 2009/282, 2009. Available at `http://eprint.iacr.org/`.

[28] S. Gilbert. *The Collected Works of Paul Valéry: Analects*. Princeton University Press, 1971.

[29] S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[30] S. Goldwasser, S. Micali, and R. L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[31] S. Haber and B. Pinkas. Securely Combining Public-Key Cryptosystems. In M. Reiter and P. Samarati, editors, *CCS 2001: Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 215–224, New York, NY, USA, 2001. ACM.

[32] I. Haitner and T. Holenstein. On the (Im)Possibility of Key Dependent Encryption. In O. Reingold, editor, *Theory of Cryptography – TCC 2009: Sixth Theory of Cryptography Conference*, Lecture Notes in Computer Science, pages 202–219, Berlin / Heidelberg, 2009. Springer-Verlag.

[33] S. Halevi and H. Krawczyk. Security Under Key-Dependent Inputs. In *CCS 2007: Proceedings of the 14th ACM Conference on Computer and Communications Security*, pages 466–475, New York, NY, USA, 2007. ACM.

[34] D. Hofheinz and D. Unruh. Towards Key-Dependent Message Security in the Standard Model. In N. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008: International Conference on the Theory and Applications of Cryptographic Techniques*, volume 4965 of *Lecture Notes in Computer Science*, pages 108–126, Berlin / Heidelberg, 2008. Springer.

[35] J. Katz. Signature Schemes with Bounded Leakage Resilience. Cryptology ePrint Archive, Report 2009/220, 2009. Available at `http://eprint.iacr.org/`.

[36] J. Katz and C. Y. Koo. On Constructing Universal One-Way Hash Functions from Arbitrary One-Way Functions. Available at `http://www.cs.umd.edu/~jkatz/papers/rompel.pdf`.

[37] L. Lamport. Constructing Digital Signatures from a One-Way Function. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory, October 1979.

[38] R. Langworth, editor. *Churchill by Himself: The Definitive Collection of Quotations*. PublicAffairs, 2008.

[39] S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters. Sequential Aggregate Signatures and Multisignatures Without Random Oracles. In S. Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006: International Conference on the Theory and Applications of Cryptographic Techniques*, volume 4004 of *Lecture Notes in Computer Science*, pages 465–485, Berlin / Heidelberg, 2006.

[40] S. Lucks, E. Zenner, A. Weimerskirch, and D. Westhoff. Concrete Security for Entity Recognition: The Jane Doe Protocol. In D. Roy Chowdhury, V. Rijmen, and A. Das, editors, *Progress in Cryptology – INDOCRYPT 2008: 9th International Conference on Cryptology in India*, volume 5365 of *Lecture Notes in Computer Science*, pages 158–171, Kharagpur, India, 2008. Springer-Verlag.

[41] A. Lysyanskaya, S. Micali, L. Reyzin, and H. Shacham. Sequential Aggregate Signatures from Trapdoor Permutations. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004: International Conference on the Theory and Applications of Cryptographic Techniques*, volume 3027 of *Lecture Notes in Computer Science*, pages 74–90, Berlin / Heidelberg, 2004. Springer.

[42] P. Rogaway M. Bellare. Introduction to Modern Cryptography. Available at `http://www.cs.ucdavis.edu/~rogaway/classes/227/spring05/book/main.pdf`.

[43] A. Mashatan and D. Stinson. A New Message Recognition Protocol for Ad Hoc Pervasive Networks. In M. K. Franklin, L. Chi Kwong Hui, and D. S. Wong, editors, *Cryptology and Network Security – CANS 2008: 7th International Conference*, volume 5339 of *Lecture Notes in Computer Science*, pages 378–394, Hong-Kong, China, 2008. Springer-Verlag.

[44] A. Menezes, S. Vanstone, and P. Van Oorschot. *Handbook of Applied Cryptography.* CRC Press, Boca Raton, FL, USA, 1996.

[45] R. C. Merkle. A Digital Signature Based on a Conventional Encryption Function. In C. Pomerance, editor, *Advances in Cryptology – CRYPTO 1987: A Conference on the Theory and Applications of Cryptographic Techniques*, volume

293 of *Lecture Notes in Computer Science*, pages 369–378, London, UK, 1987. Springer-Verlag.

[46] R. C. Merkle. A Certified Digital Signature. In G. Brassard, editor, *Advances in Cryptology – CRYPTO 1989: A Conference on the Theory and Applications of Cryptographic Techniques*, Lecture Notes in Computer Science, pages 218–238, New York, NY, USA, 1989. Springer-Verlag.

[47] S. Micali and L. Reyzin. Physically Observable Cryptography (Extended Abstract). In M. Naor, editor, *Theory of Cryptography – TCC 2004: First Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 278–296, Berlin / Heidelberg, 2004. Springer.

[48] C. Mitchell. Remote User Authentication Using Public Information. In K. G. Paterson, editor, *Cryptography and Coding: 9th IMA International Conference*, volume 2398 of *Lecture Notes in Computer Science*, pages 360–369, London, UK, 2003. Springer-Verlag.

[49] D. Naor, A. Shenhav, and A. Wool. One-Time Signatures Revisited: Have They Become Practical? Cryptology ePrint Archive, Report 2005/442, 2005. Available at `http://eprint.iacr.org/`.

[50] B. Okri. *Mental Fight: An Anti-Spell for the 21st Century*. Phoenix House, 1999.

[51] B. Preneel and P.C. van Oorschot. On the Security of Iterated Message Authentication Codes. *IEEE Transactions on Information Theory*, 45(1):188–199, 1999.

[52] M. O. Rabin. Digitalized Signatures. *Foundations of Secure Computation*, pages 155–168, 1978.

[53] R. Rivest, A. Shamir, and Y. Tauman. How to Leak a Secret. In C. Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001: 7th International Conference on the Theory and Application of Cryptology and Information Security*, volume 2248 of *Lecture Notes in Computer Science*, pages 552–565, London, UK, 2001. Springer.

[54] J. Rompel. One-Way Functions are Necessary and Sufficient for Secure Signatures. In H. Ortiz, editor, *STOC 1990: Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, pages 387–394, New York, NY, USA, 1990. ACM.

[55] F. Stajano and R. Anderson. The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks. In B. Christianson, B. Crispo, J. A. Malcolm, and M. Roe, editors, *Security Protocols: 7th International Workshop*, volume 1796 of *Lecture Notes in Computer Science*, pages 172–182, London, UK, 2000. Springer-Verlag.

[56] M. I. González Vasco, F. Hess, and R. Steinwandt. Combined (Identity-Based) Public Key Schemes. Cryptology ePrint Archive: Report 2008/466, February 2009. Available at `http://eprint.iacr.org/2008/466`.

[57] A. Weimerskirch and D. Westhoff. Zero Common-Knowledge Authentication for Pervasive Networks. In M. Matsui and R. Zuccherato, editors, *Selected Areas in Cryptography – SAC 2003: 10th Annual International Workshop*, volume 3006 of *Lecture Notes in Computer Science*, pages 73–87, Berlin, 2004. Springer-Verlag.

[58] Y. Zheng. Digital Signcryption or How to Achieve Cost(Signature & Encryption << Cost(Signature) + Cost(Encryption). In B. S. Jr. Kaliski, editor, *Advances in Cryptology – CRYPTO 1997: Proceedings of the 17th Annual International Cryptology Conference*, Lecture Notes in Computer Science, pages 165–179, London, UK, 1997. Springer-Verlag.