Bauhaus-Universität Weimar

Analysis, Design & Applications of Cryptographic Building Blocks

Inauguraldissertation
zur Erlangung des akademischen Grades
doctor rerum naturalium (Dr. rer. nat.)

der Bauhaus-Universität Weimar
an der Fakultät Medien

vorgelegt von
Christian Forler

September 2014

Gutachter: Professor Dr. Stefan Lucks
Bauhaus-Universität Weimar

Prof. Dr. Frederik Armknecht, Juniorprofessor
Universität Mannheim

Tag der mündlichen Prüfung: TBA

Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle anderen aus Quellen (direkt oder indirekt) übernommenen Inhalte, Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Weitere Personen waren an der inhaltlich-materiellen Erstellung der vorliegenden Arbeit nicht beteiligt. Insbesondere habe ich hierfür nicht die entgeltliche Hilfe von Vermittlungs- bzw. Beratungsdiensten (Promotionsberater oder anderer Personen) in Anspruch genommen. Niemand hat von mir unmittelbar oder mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen. Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt.
Ich versichere, dass ich nach bestem Wissen die reine Wahrheit gesagt und nichts verschwiegen habe.

Ort, Datum Unterschrift

Abstract

This thesis deals with the basic design and rigorous analysis of cryptographic schemes and primitives, especially of authenticated encryption schemes, hash functions, and password-hashing schemes.

In the last decade, security issues such as the PS3 jailbreak demonstrate that common security notions are rather restrictive, and it seems that they do not model the real world adequately. As a result, in the first part of this work, we introduce a less restrictive security model that is closer to reality. In this model it turned out that existing (on-line) authenticated encryption schemes cannot longer be considered secure, *i.e.,* they can guarantee neither data privacy nor data integrity. Therefore, we present two novel authenticated encryption schemes, namely COFFE and McOE, which are not only secure in the standard model but also reasonably secure in our generalized security model, *i.e.,* both preserve full data inegrity. In addition, McOE preserves a resonable level of data privacy.

The second part of this thesis starts with proposing the hash function Twister$_\pi$, a revised version of the accepted SHA-3 candidate Twister. We not only fixed all known security issues of Twister, but also increased the overall soundness of our hash-function design.

Furthermore, we present some fundamental groundwork in the area of password-hashing schemes. This research was mainly inspired by the medial omnipresence of password-leakage incidences. We show that the password-hashing scheme `scrypt` is vulnerable against cache-timing attacks due to the existence of a password-dependent memory-access pattern. Finally, we introduce Catena the first password-hashing scheme that is both memory-consuming and resistant against cache-timing attacks.

Zusammenfassung

Diese Dissertation widmet sich dem Design und der Analyse von kryptographischen Primitiven und deren korrekte Anwendung. Im Mittelpunkt dieser Arbeit stehen daher das Design von beweisbar sichere Verfahren zur authentisierten Verschlüsselung, kryptographische Hashfunktionen sowie Passwort-Hashing-Algorithmen.

In der Vergangenheit haben Sicherheitslücken wie beispielsweise der *PS3-Jailbreak* gezeigt, dass die gängigen Sicherheitsmodelle zu restriktiv sind und daher die Praxis eher unangemessen widerspiegeln. Der erste Hauptteil dieser Arbeit beschäftigt sich daher mit der Einführung eines realitätsnäheren Sicherheitsmodelles unter dem praktisch alle bestehenden On-line-Verfahren zur authentisierten Verschlüsselung als unsicher anzusehen sind. Weder Vertraulichkeit noch Integrität der zu schützenden Daten können noch sichergestellt werden. Aus diesem Grund stellen wir in dieser Arbeit zwei neue On-line-Verfahren zur authentisierten Verschlüsselung vor. Bei dem ersten handelt es sich um COFFE. Dieser Betriebsmodus für Hashfunktionen schützt selbst in unserem realitätsnäheren Sicherheitsmodelles noch die Integrität der verarbeiteten Daten. Das zweite ist McOE. Es ist ein äusserst robustes Verfahren – basierend auf einer Blockchiffre – welches auch in dem neuen Modell vollständige Integrität und *angemessene* Vertraulichkeit von den verarbeiteten Daten sicherstellt.

Im zweiten Hauptteil dieser Arbeit wird als Erstes die kryptographische Hashfunktion TWISTER$_\pi$ vorgestellt. Hierbei handelt es sich um eine überarbeitete Version von TWISTER (akzeptierter SHA-3-Kandidat). Zum einem behebt TWISTER$_\pi$ alle bekannten Sicherheitsprobleme der ursprünglichen Version, zum anderen bietet es im Vergleich noch signifikant höhere Performance.

Weiterhin werden neue Anforderungen für Sicherheit und Funktionalität an Passwort-Hashing-Verfahren vorgestellt. Im Zuge dessen wurde ein akademischer Cache-Timing-Angriff auf das derzeit führende Passwort-Hashing-Verfahren `scrypt` entwickelt, mit dessen Hilfe sich ein sehr effizienten Filter für Passwortkandidaten konstruieren lässt.

Letzlich stellen wir in dieser Arbeit noch Catena vor. Hierbei handelt es sich um das erste beweisbar sichere Passwort-Hashing-Verfahren welches sowohl speicherintensiv als auch sicher gegen Cache-Timing-Angriffe ist.

Contents

1. Introduction **1**

I. Foundations **5**

2. Hash Functions **7**
2.1. Security Notions 8
2.2. Iterated Hash Functions 12
2.3. Generic Attacks 13
2.4. Keyed Hash Functions 15

3. Block Ciphers **17**
3.1. Security Notions 18
3.2. Tweakable Block Ciphers 20

4. Authenticated Encryption Schemes **23**
4.1. Authenticated Encryption with Associated Data Schemes 24
4.2. Generic Composition 25
4.3. Security Notions 26
4.4. Game-Based Proofs 30

5. Misusing Authenticated Encryption Schemes **31**
5.1. Nonce Misuse 31

5.2. Decryption Misuse 35
5.3. Robustness 38

II. Authenticated Encryption **41**

6. Robustness of Authenticated Encryption Schemes **43**
6.1. Nonce-Misuse Resistance 43
6.2. Decryption-Misuse Resistance 48
6.3. Results Summary 50

7. COFFE: Ciphertext Output Feedback Faithful Encryption **53**
7.1. Specification 55
7.2. COFFE-SHA-224 – A Practical Instantiation 59
7.3. Security 60
7.4. Results Summary 69

8. McOE **71**
8.1. Generic Specification 73
8.2. Security Analysis 75
8.3. Practical Instances 85
8.4. Benchmarks 89
8.5. Results Summary 90

III. Cryptographic Hash Functions **91**

9. Twister$_\pi$ – A Framework for Fast and Secure Hash Functions **93**
9.1. Design Principles of TWISTER$_\pi$ 95
9.2. Specification of the TWISTER$_\pi$ Hash-Function Family 96
9.3. Security against Generic Attacks 105
9.4. Implementation Details 106
9.5. Benchmarks 109
9.6. From TWISTER to TWISTER$_\pi$ 111
9.7. Untwisting the Myth – Cryptanalysis of TWISTER 112
9.8. Results Summary 121

10. Catena: A Memory-Consuming Password Scrambler **123**
10.1. Background 125

10.2. Related Work 126
10.3. Memory-Related Properties 128
10.4. The `scrypt` Password Scrambler 131
10.5. Specification of Catena 136
10.6. Security Analysis of Catena 139
10.7. The Catena-KG Key-Derivation Function 141
10.8. Catena-DBG 142
10.9. Analysis of Catena-DBG 146
10.10Results Summary 149

IV. Epilog **151**

11. Conclusion **153**
11.1. Summary 153
11.2. Further Research 154

12. List of Publications **157**

Bibliography **161**

A. Twister$_\pi$: Test Vectors **185**
A.1. Twister$_\pi$-256 185
A.2. Twister$_\pi$-512 186

Index **186**

List of Algorithms

1. Random Permutation 18

2. INT-CTXT Game 28

3. Random On-Line Permutation 33

4. Repeated-Keystream Adversary 44
5. Linear-Tag Adversary 44

6. COFFE 57
7. ProcessMessage/ProcessCiphertext 58
8. LLCP' 66

9. McOE 74
10. ProcessHeader 74

11. Twister$_\pi$ 96
12. `Mini-Round` 99
13. `Twister-Round` 103
14. `State-Finalization` 104
15. `Output-Round` 104

16. `scrypt/ROMix` 131
17. `ROMixMC` 132

18. Catena . . . 137
19. Catena-KG . . . 141
20. Double Butterfly Hashing (DBH) . . . 146

List of Figures

7.1. The generic COFFE construction. 56
7.2. Game G_1 for the proof of Lemma 7.3. 65
7.3. Games G_2 and G_3 for the proof of Lemma 7.3. 67

8.1. The generic McOE construction. 73
8.2. Games G_2 and G_3 for the proof of Lemma 8.2. 78
8.3. Game G_1 for the proof of Lemma 8.3. 80
8.4. Games G_2 and G_3 for the proof of Lemma 8.3. 82
8.5. Constructed tweakable block cipher TX. 86
8.6. Constructed tweakable block cipher TG. 88

9.1. Illustration of a `Mini-Round`. 100
9.2. Illustration of a `Twister-Round`. 102
9.3. The compression function `Twister-Round`-256. 111
9.4. The compression function `Twister-Round`-512. 112
9.5. Difference Distribution Table for a $2^2 \times 2^2$ S-box. 114
9.6. Illustration of the semi-free-start collision attack on Twister. 115
9.7. Inversion of the first 64-bit word of a Twister-512 hash value. 119

10.1. A Cooley-Tukey FFT graph with eight input and output vertices. . . . 143
10.2. Types of edges of an $(3, 1)$-Double Butterfly Graph. 144
10.3. An $(3, 1)$-Double Butterfly Graph. 145

List of Tables

6.1. Misuse resistance of on-line authenticated encryption schemes. 52

7.1. Comparison of authenticated encryption schemes. 55
7.2. Case analysis for the proof of Lemma 7.2 63

8.1. Classification of block cipher based authenticated encryption schemes. 72
8.2. Results of the performance benchmarks of McOE-X and McOE-G. . 89

9.1. Branch numbers of Twister_π's MDS matrix. 102
9.2. Results of the performance benchmarks for Twister_π. 110
9.3. Results of the 32-bit performance benchmarks for Twister_π. 110
9.4. Cryptanalytic results for Twister-512. 112

10.1. Comparison of contemporary password scramblers. 127

List of Abbreviations

AEAD Authenticated Encryption with Associated Data

AES Advanced Encryption Standard

BTM Bivariate Tag Mixing

CAESAR Competition for Authenticated Encryption: Security, Applicability, and Robustness

CBC Cipher Block Chaining

CCA3 Chosen-Ciphertext Attack 3

CCFB Counter-CipherFeedback

CCM Counter with CBC-MAC

CFB Cipher Feedback

CHM CENC with Hash-based MAC

COFFE Ciphertext Output Feedback Faithful Encryption

COTS Commercial Off-The-Shelf

CTR Counter

CTS Ciphertext Stealing

CWC	Carter–Wegman Counter
cpb	Cycles per Byte
DAE	Deterministic Authenticated Encryption
DAG	Directed Acyclic Graph
DBG	Double Butterfly Graph
DBH	Double Butterfly Hashing
DDT	Difference-Distribution Table
DES	Data Encryption Standard
ECB	Electronic Codebook
EME	Encrypt-Mix-Encrypt
EtM	Encrypt-then-Mac
FFT	Fast Fourier Transform
GCM	Galois/Counter Mode
GPU	Graphical Processing Unit
HBS	Hash Block Stealing
HMAC	Hash-Based Message Authentication Code
IACBC	Integrity Aware Cipher Block Chaining Mode
IAPM	Integrity Aware Parallelizable Mode
IND-CCA	Indistinguishability under Chosen-Ciphertext Attack
IND-CPA	Indistinguishability under Chosen-Plaintext Attack
IND-OCCA	Indistinguishability under On-Line Chosen-Ciphertext Attack
IND-OCCA2	Indistinguishability under On-Line Chosen-Ciphertext Attack 2
IND-OPRP	Indistinguishability from an On-line Pseudorandom Permutation

IND-PRP	Indistinguishability from a Pseudorandom Permutation
INT-CTXT	Integrity of Ciphertext
IoT	Internet of Things
KDF	Key Derivation Function
LCP	Longest Common Prefix
LSB	Least Significant Bit
MAC	Message Authentication Code
MDS	Maximum Distance Separable
MMO	Matyas-Meyer-Oseas
MSB	Most Significant Bit
NDMA	Nonce- and Decryption-Misuse Attack
NIST	National Institute of Standards and Technology
OAE	On-line Authenticated Encryption
OCB	Offset Codebook
OCCA3	On-line Chosen-Ciphertext Attack 3
OFB	Output Feedback
ONDMA	On-line Nonce- and Decryption-Misuse Attack
OPerm	On-line Permutation
OPRP	On-line Pseudorandom Permutation
PBKDF2	Password-Based Key Derivation Function 2
PHC	Password Hashing Competition
PIN	Personal Identification Number
PRF	Pseudorandom Function

PRF-RKA Pseudorandom Function under Related-Key Attacks

PRP Pseudorandom Permutation

PRP-RKA Pseudorandom Permutation under Related-Key Attacks

PRNG Pseudorandom Number Generator

RPC Related Plaintext Chaining

SIV Synthetic Initialization Vector

TAE Tweakable Authenticated Encryption

TDOWF Trap-Door One-Way Function

TLS Transport Layer Security

TS Tag Splitting

TMTO Time-Memory Tradeoff

WPA Wi-Fi Protected Access

XCBC-XOR eXtended Ciphertext Block Chaining with XOR

XEX XOR-Encrypt-XOR

List of Mathematical Symbols

$X \in \{0,1\}^n$	X is a bit string of n bits
$X \in \{0,1\}^*$	X is a bit string of abitrary length
$X \in \{0,1\}^+$	X is a bit string of at least one bit
$X \in (\{0,1\}^n)^m$	X is a bit string of $n \cdot m$ bits
$X \in (\{0,1\}^n)^+$	X is a bit string of $k \cdot n$ bits with $k \geq 1$
$X \xleftarrow{\$} \{0,1\}^k$	X is selected uniformly at random from the set $\{0,1\}^k$
$A \mid\mid B$	Concatenation of two bit strings A and B
$\lvert A \rvert$	Bit length of variable A
$\bar{A}$	Bitwise complement of the bit string A
$A \boxplus B$	Modular addition of A and B
$A \oplus B$	Exclusive-or (XOR) of the first α MSBs of A and B with $\alpha = \min\{\lvert A \rvert, \lvert B \rvert\}$
0^n	String of n zero bits
0^*	String of abitrary number of Zero bits (zero padding)

1
Introduction

> No amount of experimentation can ever prove me right; a single experiment can prove me wrong.
>
> Albert Einstein

This thesis is dedicated to provable security wich is an essential part of modern cryptography. It deals with the formalization and the rigorous analysis of cryptographic schemes with the goal to turn an ancient art into a science.

Foundations of Provable Security. Provable security arised in the late 1970s and was shaped in the 1980s. In 1976, Diffie and Hellman pioneered the public-key cryptography when they presented a key-exchange protocol based on the foundations of abstract mathematics [79]. This enabled them to justify their security claims by mathematical arguments. Furthermore, they introduced the concept of a Trap-Door One-Way Function (TDOWF), a function which can only be inverted when knowing auxiliary (trap-door) information, *i.e.,* the secret key. However, without giving an example. Two years later, in 1978, Rivest *et al.* introduced the first instance of a TDOWF: the famous RSA encryption scheme [202].

In 1982, Goldwasser and Micali showed that the concept of a TDOWF is not sound since it allows to leak certain information about the encrypted plaintext such as its parity [116]. Therefore, they introduced a superior security notion, *probabilistic encryption*[1]. This revolutionary work paved the way for the theory of provable security.

[1] Ciphertext indistinguishability where a secure encryption scheme must not leak any information about the plaintext, except its length.

In 2012, the importance of their work was dignified by *The Association for Computing Machinery Advancing Computing as a Science & Profession* with the Turing Award.

Polynomial Security. The security notions introduced by Goldwasser and Micali in [116] have their roots in complexity theory and the intractability of well-known and hard mathematical problems, *e.g.,* the discrete logarithm problem or the integer factorization problem. This allows cryptographers to apply polynomial-time reductions to proof the security of an encryption scheme. However, such a reduction does not make any statements about the choice of the security parameter, *e.g.,* the key length. Practitioner have to guess reasonable values. Therefore, polynomial security results are rather unsatisfactory for practical applications, but very crucial for the field of information-theoretic cryptography.

Concrete Security. In 1993, Bellare and Rogaway introduced the *random oracle model* [24] where a cryptographic primitive is modeled as a random oracel, *i.e.,* a black box that always returns a random value that does not depend on its input. An adversary, allowing to ask queries to such an oracle, is modeled as a *computationally unbounded* algorithm which is only limited by the number of oracle queries. For the first time, this approach allowed to observe, in some very limited way, the behavior of an adversary during the attack by recording the queries to the oracle. Moreover, it allowed to upper bound the concrete success probability of adversaries in breaking encryption schemes.

Security proofs in the random oracle model are controversial. In 1998, Canetti *et al.* presented a separation result [59]: "There exist signature and encryption schemes which are secure in the Random Oracle Model, but for which ANY implementation of the random oracle results in insecure schemes". Nevertheless, concrete security bounds derived from security proofs in this model are meaningful since they enable practitioners to apply reasonable security parameters.

A much less controversial approach is the *standard model* where a cryptographic primitive, *e.g.,* a hash function or a block cipher, is replaced by a (keyed) *pseudorandom* counterpart ,*e.g.,* Pseudorandom Function (PRF) [114] or Pseudorandom Permutation (PRP) [20], instead of an ideal one. An adversary $\mathcal{A}$ is modeled as a (time- and) computationally bounded algorithm since an unbounded algorithm can apply an exhaustive search on the key space to reveal the secret key. Like in the random oracle model, $\mathcal{A}$ has black-box access to the cryptographic scheme Π. In the standard model, the concrete security of a scheme against an adversary $\mathcal{A}$ is determined by the success probability of $\mathcal{A}$ in breaking Π. A scheme is considered to be secure if the maximum success probability over all adversaries that ask at most q oracle queries is negligible. In 1994, Bellare *et al.* introduced concrete security in the standard model by presenting a security notion for Message Authentication Codes (MACs) [20]. In the following years, Bellare *et al.* introduced several standard-model security notions for all common cryptographic schemes such as digital signatures [26], symmetric encryption [18], and authenticated encryption schemes [21]. Starting from the last decade, it is costume to introduce a novel cryptographic scheme along with a concrete security claim supported by a security proof given either in the standard or in the stronger random oracle model.

Misuse Resistance. A provably secure cryptographic scheme provides rigorous security properties, *e.g.,* integrity and confidentiality, only under well-defined assumptions against well-defined adversaries. Hence, the term *secure* is a placeholder for *to protect something against a well-defined class of adversaries.* In contrast to cryptographers, who exactly know what is meant by referring a cryptographic algorithm to be secure, regular users implicitly assume that a secure scheme matches any of their security requirements; without further investigation. This common misconception causes serious security issues and is typical for the human nature. Thomas Gray pointed this out by the felicitous idiom: *"Where ignorance is bliss, 'tis folly to be wise"*[2]. Hence, re-education of all users is an enormous hard task for the cryptographic community. This virtuality leads to the conclusion that cryptographers should take responsiblility and should design their algorithms in a way that the naive usage of their algorithms should not end up into big security disasters.

But, this is easier said than done. On the one hand, it is not the task of a digital signature scheme to provide any kind of data privacy. On the other hand, a digital signature scheme should not reveal the secret key when one of its security assumption is violated once such as the ECDSA signature scheme [128]. It is a good starting point to design robust algorithms that still offer some decent level of security, even when a security assumption is occasionally violated. In this thesis we introduce the first authenticated encryption scheme that provides full security under standard assumptions, and still a *reasonable* level of security under much weaker assumptions. Therefore, such an algorithm provides a second line of defense in a misuse scenario, *e.g.,* faulty random number generator.

[2] Ode on a Distant Prospect of Eton College

Outline

In the first part of this work we introduce the concepts, security notions, and definitions needed to grasp the latter parts. Nevertheless, the experienced reader can directly start with the second part of this thesis.

The essential elements of this work, namely Part II and Part III, are partitioned as follows:

Part II: First, in Section 5 we introduce the concept of robust authenticated encryption schemes. Then we show in Section 6 that published authenticated encryption schemes are not robust, so far. In Section 7 we present COFFE, a partially robust On-line Authenticated Encryption (OAE) scheme. Finally, in Section 8 we introduce McOE, the first robust OAE scheme. Note that a preliminary version of McOE was published before in [98, 100], and has been thoroughly revised.

Part III: First, in Section 9 we present Twister$_\pi$, a family of cryptographic hash functions. It is a rigorously improved revision of the accepted SHA-3 candidate Twister [93]. Furthermore, in Section 10 we introduce Catena, a novel memory-consuming password scrambling framework that is based on a cryptographic hash function. Note that a preliminary version of Catena was published before in [103] and the extended abstract will be appear at ASIACRYPT'14 [104].

Further notable results of my studies that are not mentioned in this work can be found in [1–4, 92, 94, 95, 97, 99, 101, 102]. A complete list of my publications, so far, is given in Section 12.

Part I

Foundations

2

Hash Functions

> Nothing in life is to be feared, it is only to be understood. Now is the time to understand more, so that we may fear less.
>
> Marie Curie

The concept of hash functions was introduced in the early 1950s [142], and became vital in the field of modern cryptography. Informally, a hash function *compresses* an input of arbitrary length to a fixed-length output which is usually referred to as a *hash value* or *message digest*. Today, hash functions have many applications and are virtually used everywhere in, *e.g.,* encryption schemes [25], digital signature schemes [26, 108], key derivation schemes [131, 191], key exchange protocols [109], and MACs [17, 240]. Due to the wide range of use-cases, a good hash function should be both memory- and time-efficient to be applicable on restricted devices, *e.g.,* wireless sensor nodes [244], trusted computing modules [228], and smart meters [172]. In this thesis we borrow the notion of unkeyed hash function that was presented by Rogaway in [207].

Definition 2.1 (Hash Function). *An n-bit* hash function $\mathcal{H}$ *is a function*

$$\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^n, \qquad n \in \mathbb{N}^+.$$

In practice, the notion *arbitrary-length input* is usually interpreted as messages up to 2^r bits for a reasonable large r with $r \ll n/2$. Cryptographic hash functions have this *limitation* since their expected security properties cannot longer be guaranteed if the length of an input exceeds $2^{n/2}$ bits [238]. Considering the fact that contemporary hash functions have an output length of 256 or 512 bits, this observation is only of academical interest. Anyway, common hash functions follow an iterative approach to process a long message M; they divide M into m blocks of n bits, *i.e.,* $M = M_1, \ldots, M_m$, where $|M_i| = n$ for $i = 1, \ldots, m$. Individual message blocks are processed iteratively by a *compression function*.

Definition 2.2 (Compression Function). *A* compression function $\mathcal{F}$ *is a function*

$$\mathcal{F} : \{0,1\}^h \times \{0,1\}^n \to \{0,1\}^h, \qquad h, n \in \mathbb{N}^+,$$

where h denotes the size of the chaining value and n the size of the message block.

In the area of cryptography, the majority of iterative hash functions are based on the *Merkle-Damgård* design [72, 171] where the message blocks are sequentially processed by a fixed-length compression function.

This approach obtained its popularity for being *property-preserving*, *i.e.,* certain properties of the hash function are inherit from the compression function. In the light of the SHA-3 competition, the recent research has put a focus on designing constructions that preserve as many properties of the compression function as possible [36, 41].

2.1. Security Notions

2.1.1. Random Oracle Model

Ideally, a hash function should be *indistinguishable* from a *random oracle* [24] with fixed output size. A random oracle is an abstract and ideal primitive that returns a random bit string for each fresh input. Thus, the output of a random oracle is independent of the input, except that repeated queries are always treated consistently, *i.e.,* the function property is always fulfilled. Furthermore, random oracles are atomic building blocks, *i.e.,* they cannot be decomposed. In the context of provable security, random oracles are used for hiding implementation details, *e.g.,* the insides of a specific hash function like MD5 [200] or SHA-256 [184]. Random oracles become handy when no known implementable function provides the mathematical properties required for the proof – or when it gets too tedious to formalize these. A security proof of a cryptographic scheme using a random oracle as a component function is said to be in the *random oracle model*. From a theoretical point of view, it is clear that such a security proof is only a heuristic indication of the security of the scheme when instantiated with a specific hash function.

In fact, many recent separation results [16, 60, 81, 115, 163, 179] illustrate that various cryptographic schemes are secure in the random oracle model, but completely insecure for *any* efficient instantiation. According to [144], all such counterexamples are *artificial* and do not seem to attack any practically relevant scheme directly. Nevertheless, a security proof in the random oracle model is at least an indication for the soundness of the analyzed scheme.

2.1.2. Standard Model

Beside the *random oracle model*, the security of a hash function can also be determined under the three much weaker *standard model* assumptions: (1) collision resistance, (2) preimage resistance, and (3) 2nd-preimage resistance. The insecurity of a cryptographic function is quantified by the success probability of an optimal and resource-bounded adversary $\mathcal{A}$. Depending on the setting, different notions of success and different limitations of the resources apply for the adversary. Actually, the standard model does only work for families of hash functions $\mathfrak{H}$ where $\mathfrak{H}$ is considered to be secure if there exists no *efficient* adversary $\mathcal{A}$ that violates at least one out of the three standard assumptions for $\mathcal{H} \xleftarrow{\$} \mathfrak{H}$. The standard model is not suitable to prove the security of a single n-bit hash function $\mathcal{H}$ such as SHA-256 [184] since here $\mathcal{A}$ is not restricted in the access to SHA-256. Suppose $\mathcal{A}_{X,Y}$ with some fixed $X, Y \in \{0,1\}^{2n}$ is an adversary that just outputs the two $2n$-bit values X and Y. By the pigionhole principle, there must be two values X' and Y' such that $\mathcal{H}(X') = \mathcal{H}(Y')$ and thus, there exists an efficient adversary, namely $\mathcal{A}_{X',Y'}$. In 2006, Rogaway introduced a way to analyze the security of a single hash function in the standard model by bringing human ignorance into equation [207] which means that $\mathcal{H}$ is secure if there is no efficient algorithm which is *known to man* that violates at least one out of the three standard assumptions. Thus, a hash function is considered to be secure if mankind is unable to find an efficient adversary.

Next, we introduce a common *hybrid model* where an adversary has only restricted access to a hash function.

2.1.3. Hybrid Standard Model

In this thesis, any analyzed cryptographic system is an algorithm that uses (at least one) other component function – the primitive – inside. As the adversary is assumed to have no knowledge about the inner workings of these primitives – in the past always formalized by assuming a secret key – these are accessed by the adversary via an oracle interface. Such an oracle interface essentially formalizes the black-box mode of operation of an adversary towards the scheme or primitive being attacked. It provides a clearly defined set of exposed functions an adversary is able to send queries to and can expect to get an answer from. We always assume that such an adversary is an efficient algorithm, *i.e.,* it has resource-bounded access to the compression or hash function. Next, we give formal definitions of the mentioned standard model assumptions.

Collision Resistance. A hash function $\mathcal{H}$ is collision resistant if it is hard to find two distinct inputs that are mapped to the same output. More formally, the advantage of an adversary $\mathcal{A}$ with oracle access to $\mathcal{H}$ is defined as follows:

Definition 2.3 (Collision Resistance). *Let $\mathcal{H}$ be a hash function and $\mathcal{A}$ be an adversary. Then, the collision advantage of $\mathcal{A}$ against $\mathcal{H}$ is given by*

$$\mathbf{Adv}_{\mathcal{H}}^{coll}(\mathcal{A}) = \Pr\left[(M, M') \leftarrow \mathcal{A}^{\mathcal{H}} : \mathcal{H}(M) = \mathcal{H}(M') \wedge M \neq M'\right].$$

Note that the adversary $\mathcal{A}$ is only limited by the number of queries to its oracles. Thus, we write

$$\mathbf{Adv}_{\mathcal{H}}^{coll}(q, t) = \max_{\mathcal{A}} \left\{\mathbf{Adv}_{\mathcal{H}}^{coll}(\mathcal{A})\right\},$$

where the maximum is taken over all adversaries that ask at most q oracle queries and run in time at most t.

For an n-bit hash function, the number of message pairs with q messages is $\binom{q}{2} = q(q-1)/2 \approx q^2/2$. An ideal n-bit hash function returns random n-bit strings. Since two of these are equal with probability 2^{-n}, one needs 2^n pairs before a collision can be expected. More precisely with $q = 2^{(n+1)/2}$ queries, the probability of a collision is greater than 0.5, *i.e.,* $1 - \frac{1}{e} \approx 0.63$. This generic attack works for any hash function and is commonly known as the *birthday attack*.

Preimage Resistance. A hash function $\mathcal{H}$ is preimage resistant if, given a hash value, it is hard to find a message that hashes to this value.

More formally, the advantage of an adversary $\mathcal{A}$ with oracle access to $\mathcal{H}$ is defined as follows:

Definition 2.4 (Preimage Resistance). *Let $\mathcal{H}$ be a hash function and $\mathcal{A}$ be an adversary. Then, we define the preimage advantage of $\mathcal{A}$ against $\mathcal{H}$ as*

$$\mathbf{Adv}_{\mathcal{H}}^{pre}(\mathcal{A}) = \Pr\left[Y \xleftarrow{\$} \{0,1\}^n, M \leftarrow \mathcal{A}^{\mathcal{H},Y} : \mathcal{H}(M) = Y\right],$$

and

$$\mathbf{Adv}_{\mathcal{H}}^{pre}(q,t) = \max_{\mathcal{A}} \left\{\mathbf{Adv}_{\mathcal{H}}^{pre}(\mathcal{A})\right\}$$

as the maximum advantage over all preimage adversaries that ask at most q oracle queries and run in time at most t.

A method for finding preimages that works for any hash function is the brute-force attack, *i.e.,* one hashes random messages until the hash value Y is reached. Assuming that the output of the hash function is uniformly balanced, an adversary is expected to try 2^n distinct messages in order to be successful.

2nd-Preimage Resistance. A hash function $\mathcal{H}$ is 2nd-preimage resistant if, given a hash value message pair (Y,M) where $Y = \mathcal{H}(M)$, it is hard to find a fresh message that also produces the same hash value. More formally, the advantage of an adversary $\mathcal{A}$ with oracle access to $\mathcal{H}$ is defined as follows:

Definition 2.5 (2nd-Preimage Resistance). *Let $\mathcal{H}$ be a hash function and $\mathcal{A}$ be an adversary. Then, the 2nd-preimage advantage of $\mathcal{A}$ against $\mathcal{H}$ for a random message $M \xleftarrow{\$} \{0,1\}^*$ is defined as*

$$\mathbf{Adv}_{\mathcal{H}}^{2nd\text{-}pre}(\mathcal{A}) = \Pr\left[Y \leftarrow \mathcal{H}(M), M' \leftarrow \mathcal{A}^{\mathcal{H},M,Y} : \mathcal{H}(M') = Y \wedge M' \neq M\right],$$

and

$$\mathbf{Adv}_{\mathcal{H}}^{2nd\text{-}pre}(q,t) = \max_{\mathcal{A}} \left\{\mathbf{Adv}_{\mathcal{H}}^{2nd\text{-}pre}(\mathcal{A})\right\}$$

as the maximum advantage over all 2nd-preimage adversaries that ask at most q oracle queries and run in time at most t.

2.2. Iterated Hash Functions

At CRYPTO'98, Damgård [72] and Merkle [171] proposed – independently from each other – an iterative approach to construct a collision resistant hash function based on a fixed-input length compression function. This idea has influenced the design of virtually all popular hash functions such as MD4 [201], MD5 [200], SHA-0/1 [183, 185], and the SHA-2 family [184].

Definition 2.6 (Iterated Hash Function). *Let $\mathcal{F} : \{0,1\}^h \times \{0,1\}^n \to \{0,1\}^h$ be a compression function and let $M = M_1, \ldots, M_m$ be a message with $M_i \in \{0,1\}^n$ for $i = 1, \ldots, m$. For a fixed inital value $V_0 \in \{0,1\}^h$, the* iterated hash function *$\mathcal{H} : (\{0,1\}^n)^* \to \{0,1\}^h$ is defined as*

$$V_i \leftarrow \mathcal{F}(V_{i-1}, M_i), \text{ where } Y = \mathcal{H}(M) = V_{m+1} \text{ with } i = 1, \ldots, m$$

Usually, the message length in bits, denoted by $|M|$, is not necessarily a multiple of n. Thus, a *padding procedure* is required. Note that it can also be applied if the message length is already a multiple of n bits since it can serve as a pre-processing function. This step is sometimes called *message expansion*. The most common padding procedure is the so called 10^*-padding specified in [200].

Definition 2.7 (10^*-Padding). *Suppose M is an ℓ-bit input message. Then,*

$$b = n - (\ell + 1) \pmod n$$

denotes the number of appended zero bits. And the padded message is computed by the following rule:

$$M' \leftarrow M \;||\; 1 \;||\; 0^b,$$

where '1' denotes a single one-bit and '0^b' denotes a sequence of b zero-bits.

There are numerous further padding rules known and the choice depends on the application. More examples are given in [123, 213, 221].

Next, we discuss why the length of the message might also be included into the padding as a security measure. Damgård and Merkle independently provided theorems in their papers that essentially show Theorem 2.8.

Theorem 2.8 (*Merkle-Damgård* Security [72, 171]). *Suppose $\mathcal{H}$ is an iterated hash function as in Definition 2.6 and $\mathcal{F}$ its underlying compression function. If the initial chaining value V_0 is fixed and if the padding procedure includes the message length into the padding bits, it holds that*

$$\mathcal{F} \text{ is collision resistant} \implies \mathcal{H} \text{ is collision resistant.}$$

Fixing the initial value and adding a representation of the message length, is called MD-*strengthening*. Unfortunately, this result does not extend to pre- and 2nd-preimage resistance. Recent results highlight some intrinsic limitations of the *Merkle-Damgård* approach. This includes being vulnerable to multi-collision attacks [129], long 2nd-preimage attacks [135], and herding attacks [134]. Even though the practical relevance of these attacks is unclear, they highlight some security issues which designers should take care of. Therefore, in recent years, research has put a focus on designing constructions that preserve as many properties of the compression function as possible, *e.g.,* [9, 10, 23, 36, 41, 64, 91].

2.3. Generic Attacks

On one hand, the iterative structure of cryptographic hash functions makes it possible to design time- and memory-efficient hash functions, and handling inputs of arbitrary length. On the other hand, iterative modes of operation for compression functions allow generic attacks; even for an *ideal* compression function, *i.e.,* a random oracle. Next, we give a brief introduction to generic attacks on hash functions.

Length-Extension Attacks. Given a *Merkle-Damgård*-based hash function $\mathcal{H}$. If one can find a collision for two messages M and M' with $M \neq M'$, such that $\mathcal{H}(M) = \mathcal{H}(M')$, then, one can apply a length-extension attack. For any message M'', one can easily produce a collision for $M \mid\mid M''$ and $M' || M''$.

Multi-Collision Attacks. Joux [129] found that when iterative hash functions are used, finding a set of 2^k message all colliding on the same hash value (a 2^k-multi-collision) is as easy as finding k collisions for the hash function. After finding a collision in the compression function, one can find k of such collisions each starting from the chaining value produced by the previous one-block collision. In other words, one has to find two distinct messages blocks M_i and M'_i with $\mathcal{F}(V_{i-1}, M_i) = \mathcal{F}(V_{i-1}, M'_i)$, where $\mathcal{F}(\cdot,\cdot)$ represents the compression function and V_i the chaining value. Then, it is possible to construct 2^k messages with the same hash value by choosing for block i either the message block M_i or M'_i. This attack can find 2^k-way internal multi-collisions with a complexity of $k \cdot 2^{n/2}$ compression function calls. Joux also showed that the concatenation of two different hash functions is not more secure against collision attacks than the strongest one.

Herding Attacks. The herding attack [134] works as follows: An adversary $\mathcal{A}$ takes 2^k chaining values which are fixed or randomly chosen. Then, $\mathcal{A}$ chooses $O(2^{n/2-k/2})$ message blocks. Next, $\mathcal{A}$ computes the output of the compression function for each chaining value and each block. It is expected that for each chaining value there exists another chaining value, such that both collide to the same value. Then, $\mathcal{A}$ stores the message block that leads to such a collision in a table and repeats this process again with the newly found chaining values. Once the adversary has only one chaining value, it is used to compute the hash value to be published. To find a message whose chaining value is among the 2^k original values, the attacker has to perform $O(2^{n-k})$ operations. For such a message, the attacker can retrieve from the stored messages the message blocks that would lead to the desired hash value. The time complexity of this attack is about $O(2^{n/2+k/2})$ operations for the first and $O(2^{n-k})$ operations for the second step.

Long 2nd-Preimage Attacks. Dean [74] found that fix points in the compression function $\mathcal{F}$, *i.e.,* a point $(Y_i, M_i) \in \{0,1\}^h \times \{0,1\}^n$ with $Y_i = \mathcal{F}(Y_i, M_i)$, can be used for a 2nd-preimage attack against long messages in time $O(n \cdot 2^{n/2})$ and memory $O(n \cdot 2^{n/2})$. Kelsey and Schneier [135] extended this result and provided an attack to find a 2nd-preimage on a *Merkle-Damgård* construction with MD-strengthening much faster than the expected workload of $O(2^n)$. The complexity of the attack is determined by the complexity of finding expandable messages. These are messages of varying sizes such that all these messages collide internally for a given initial value. Expandable messages can either be found using internal collisions or fix points between a single-block message and a multi-block message. The complexity of the generic attack to find a 2nd-preimage for a 2^k-block message is about $k \cdot 2^{n/2+1} + 2^{n-k+1}$ compression function calls.

Andreeva *et al.* [8] showed that a combination of the attacks from [74, 134, 135] can be mounted on dithered hash functions, *i.e.,* hash functions based on compression functions with an additional input, which gives an adversary $\mathcal{A}$ more control on the

2nd-preimage since $\mathcal{A}$ can choose about the first half of the message in an arbitrary way. This attack can be done in time $2^{n/2+k/2+2} + 2^{n-k}$. Although, it is more expensive than the attack of Kelsey and Schneier [135]. As a hash function designer, one has to make the dithering as huge as possible, such that there are no small cycles.

Slide Attacks. Slide attacks are common in block-cipher cryptanalysis, but also applicable to hash functions. Given a hash function $\mathcal{H}$ and two messages M and M', where M is a prefix of M', one can find a slide pair of messages (M, M') such that the last message block of the longer message M' performs only an additional blank round, *e.g.,* for sponge constructions. These two messages are then slide by one blank round. This attack allows to recover the internal state of a slide pair of messages and even to perform backward computation, as shown in [118].

Differential Attacks. The essential idea of differential attacks on hash functions [61], as used to break MD5 [200] and SHA-0/1 [183, 185], is to exploit a high probability input/output differential over some component of the hash function, *e.g.,* in the form of a 'perturb-and-correct' strategy for the latter functions, exploiting high probability linear/non-linear characteristics.

2.4. Keyed Hash Functions

In 1992, Tsudik introduced the concept of keyed hash functions which compress a k-bit key and an input of arbitrary length to an output of fixed length [230]. From start, they were used to generate MACs [17, 32, 223, 230].

Definition 2.9 (Keyed Hash Function). *A* keyed hash function $\mathcal{H}$ *is a function*

$$\mathcal{H} : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^n, \qquad n, k \in \mathbb{N}^+.$$

Any hash function $\mathcal{H}(\cdot)$ can be easily transformed into a keyed hash function $\mathcal{H}(\cdot,\cdot)$ by prepending the key to the message, *i.e.,* $\mathcal{H}(K, M) = \mathcal{H}(K \mid\mid M)$ with $K \in \{0,1\}^k$ and $M \in \{0,1\}^*$. In the following we use $\mathcal{H}_K(M)$ and $\mathcal{H}(K, M)$ as synonymes.

Pseudorandom Function (PRF) Model. Let $\$_n : \{0,1\}^* \to \{0,1\}^n$ be a random oracle (random function). An n-bit keyed hash function $\mathcal{H}_K$, under a secret key

$K \xleftarrow{\$} \{0,1\}^k$, can be considered as secure, namely PRF-secure, if it is *indistinguishable* from \$. This security notion can be formalized by giving an adversary $\mathcal{A}$ either black-box (oracle) access to $\mathcal{H}_K$ (under a random key K) or to \$. Suppose the choice is based on the result of a fair coin toss. Let denote *heads* ('1') the case where $\mathcal{A}$ gets oracle access to the keyed hash function, and let denote *tails* ('0') the case where $\mathcal{A}$ gets oracle access to the random function. The task of $\mathcal{A}$ is to guess the result of the coin toss after a certain amount of oracle queries. More formally, the advantage of $\mathcal{A}$ can be defined as follows:

Definition 2.10 (PRF Advantage). *Let $\$: \{0,1\}^* \to \{0,1\}^n$ be a random function and $\mathcal{H} : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^n$ be a keyed hash function. Then, we define the PRF advantage of an adversary $\mathcal{A}$ against $\mathcal{H}$ as*

$$\mathbf{Adv}_{\mathcal{H}}^{PRF}(\mathcal{A}) = \left| \Pr\left[K \xleftarrow{\$} \{0,1\}^k : \mathcal{A}^{\mathcal{H}(K,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot)} \Rightarrow 1\right] \right|,$$

and

$$\mathbf{Adv}_{\mathcal{H}}^{PRF}(q,t) = \max_{\mathcal{A}} \left\{ \mathbf{Adv}_{\mathcal{H}}^{PRF}(\mathcal{A}) \right\}$$

as the maximum advantage over all PRF adversaries that ask at most q oracle queries and run in time at most t.

3

Block Ciphers

> An expert is a person who has made all the mistakes that can be made in a very narrow field.
>
> Niels Bohr

A block cipher is a cryptographic primitive with a fixed input and output size, *e.g.,* 64 or 128 bit, to either encrypt or decrypt blocks of data. In modern cryptography, block ciphers are the most common building blocks for symmetric encryption schemes. They are omnipresent to provide confidentiality (*data privacy*) for both network traffic [58, 78, 138, 196] and data storage [49, 89, 160]. Furthermore, several MACs are based on block ciphers [19, 46, 125].

Definition 3.1 (Block Cipher). *Let $E : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$ be a family of functions for some $k, n \in \mathbb{N}^+$. We denote E as a (k,n)-block cipher* iff *for any $K \in \{0,1\}^k$ it holds that*

$$E(K, \cdot) \text{ is a permutation.}$$

We denote the first input as key, the second input as message or plaintext, and the output as ciphertext.

Mathematically, a (k,n)-block cipher E is a keyed family of permutations, *i.e.,* a set of 2^k n-bit permutations. Since for a *fixed* key $K \in \{0,1\}^k$, $E(K, \cdot)$ is a

Algorithm 1 Random Permutation $\mathcal{P}$ Implemented via Lazy Sampling

Init()	$\mathcal{P}(M)$	$\mathcal{P}^{-1}(C)$
$X \leftarrow \perp$	**if** $X[M] = \perp$ **then**	**if** $Y[C] = \perp$ **then**
$Y \leftarrow \perp$	$X[M] \xleftarrow{\$} \{0,1\}^n \setminus \mathfrak{R}$	$Y[C] \xleftarrow{\$} \{0,1\}^n \setminus \mathfrak{D}$
$\mathfrak{D} \leftarrow \emptyset$	$Y[X[M]] \leftarrow M$	$X[Y[C]] \leftarrow C$
$\mathfrak{R} \leftarrow \emptyset$	$\mathfrak{D} \leftarrow \mathfrak{D} \cup \{X[M]\}$	$\mathfrak{D} \leftarrow \mathfrak{D} \cup \{C\}$
	$\mathfrak{R} \leftarrow \mathfrak{R} \cup \{M\}$	$\mathfrak{R} \leftarrow \mathfrak{R} \cup \{Y[C]\}$
	end if	**end if**
	return $C \leftarrow X[M]$	**return** $M \leftarrow Y[C]$

bijection, it has an inverse, namely $E^{-1}(K, \cdot)$, *i.e.,* for all $M \in \{0,1\}^n$ it holds that $M = E^{-1}(K, E(K, M))$. In this thesis we use $E(K, \cdot)$ and $E_K(\cdot)$ as synonyms. Furthermore, we denote $\texttt{Block}(k, n)$ as the set of all (k, n)-block ciphers. Note that for each key, there exists $2^n!$ n-bit permutations, and any permutation can be assigned to a given key. Thus, we have a huge set of $(2^n!)^{2^k}$ possible block ciphers.

3.1. Security Notions

Ideal Cipher Model. Let $\mathbf{Perm}_n$ denote the family of all possible n-bit permutations. We denote by $\mathcal{P} \xleftarrow{\$} \mathbf{Perm}_n$ a random permutation. In the *ideal cipher model* [48, 87, 139], the block cipher is modeled as a family of 2^k random permutations $\mathfrak{P}$. Let denote $\mathcal{P}_i$ the i-th element of $\mathfrak{P}$. Ideally, under a secret key a (k, n)-block cipher should be *computationally indistinguishable* from $\mathfrak{P}$, *i.e.,* it should not be possible to distinguish E_K from $\mathcal{P}_K$. Similar to the random oracle model, there are also separation results published for the ideal cipher model [44]. Hence, a cryptographic scheme proven to be secure in the ideal cipher model does not preserve its security properties – such as collision resistance – when instantiated with a real block cipher like the Advanced Encryption Standard (AES) [176] or the Data Encryption Standard (DES) [186].

Pseudorandom Permutation (PRP) Model. Beside the artificial strong *ideal cipher model*, the security of a block cipher can also be determined by the common notion of a PRP. A family E_K of n-bit permutations is called PRP when the input-output behaviour of E_K is *computationally indistinguishable* from that of an n-bit random permutation. Next, we introduce the security notion of (strong) Indistinguishability from a Pseudorandom Permutation (IND-PRP).

Suppose, depending on the result of a coin toss, an adversary $\mathcal{A}$ has either black-box access to a block cipher $E \in \texttt{Block}(k, n)$ under a secret key $K \xleftarrow{\$} \{0,1\}^k$, or to an n-bit random permutation $\mathfrak{P}$ which is independent from K. We denote E_K as IND-PRP-secure when $\mathcal{A}$ cannot distinguish between these two scenarios. Let '1' denote the real scenario where $\mathcal{A}$ has access to the block cipher and '0' the random scenario, where $\mathcal{A}$ has access to a random permutation, which is usually implemented in an efficient way using the lazy-sampling technique (cf. Algorithm 1). Then, $\Pr[K \xleftarrow{\$} \{0,1\}^k : \mathcal{A}^{E_K(\cdot),E_K^{-1}(\cdot)} \Rightarrow 1]$ denotes the success probability that $\mathcal{A}$ guesses '1' when in the real scenario. Then, the formal definition of the IND-PRP-advantage is defined as follows.

Definition 3.2 (IND-PRP-Advantage). *Let $E \in \texttt{Block}(k, n)$ be a block cipher and $\mathcal{P} \xleftarrow{\$} \mathbf{Perm}_n$ a random permutation. Then, we define the IND-PRP advantage of an adversary $\mathcal{A}$ as*

$$\mathbf{Adv}^{PRP}_{E,E^{-1}}(\mathcal{A}) = \left| \Pr\left[K \xleftarrow{\$} \{0,1\}^k : \mathcal{A}^{E_K(\cdot),E_K^{-1}(\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathcal{P}(\cdot),\mathcal{P}^{-1}(\cdot)} \Rightarrow 1\right] \right|,$$

and

$$\mathbf{Adv}^{PRP}_{E,E^{-1}}(q,t) = \max_{\mathcal{A}} \left\{ \mathbf{Adv}^{PRP}_{E,E^{-1}}(\mathcal{A}) \right\}$$

as the maximum advantage over all IND-PRP adversaries that run in time at most t and ask a total maximum of q queries to the encryption and decryption oracles.

PRP under Related-Key Attacks. In a related-key scenario we assume that an adversary $\mathcal{A}$ has partial control over the secret key $K \xleftarrow{\$} \{0,1\}^k$ of a block cipher $E \in \texttt{Block}(k, n)$. Following the security notions of Lucks [154], the partial control over the key is modeled as a key-transformation function $\varphi : \{0,1\}^k \times \{0,1\}^k \to \{0,1\}^k$. In this thesis we assume that $\varphi(K, \cdot)$ is the XOR-operation. In contrast to the IND-PRP security model, in the Pseudorandom Permutation under Related-Key Attacks (PRP-RKA) model the adversary $\mathcal{A}$ has either access to the *related-key encryption oracle* $E_{\varphi(K,\cdot)}(\cdot)$ or to a set of 2^k random permutations $\mathfrak{P} \in \mathbf{Perm}_n^k$, where $\mathbf{Perm}_n^k = \underbrace{\mathbf{Perm}_n \times \ldots \times \mathbf{Perm}_n}_{2^k \text{ times}}$. Depending on the setting, the first input is either the key relation or an index that determines a specific random permutation, and the second input is the plaintext. The PRP-RKA advantage is defined as follows:

Definition 3.3 (PRP-RKA Advantage). *Let $E \in \texttt{Block}(k, n)$ be a block cipher and let $\mathfrak{P} \xleftarrow{\$} \mathbf{Perm}_n^k$ be a family of random permutations. Let $\varphi : \{0,1\}^k \times \{0,1\}^k \to \{0,1\}^k$ is a key transformation function, and $K \xleftarrow{\$} \{0,1\}^k$. Then, we define the PRP-RKA-advantage of an adversary $\mathcal{A}$ as*

$$\mathbf{Adv}_{E,E^{-1}}^{\textit{PRP-RKA}}(\mathcal{A}) = \left|\Pr\left[\mathcal{A}^{E_{\varphi(K,\cdot)}(\cdot),E^{-1}_{\varphi(K,\cdot)}(\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathfrak{P}(\cdot,\cdot),\mathfrak{P}^{-1}(\cdot,\cdot)} \Rightarrow 1\right]\right|,$$

and

$$\mathbf{Adv}_{E,E^{-1}}^{\textit{PRP-RKA}}(q,t) = \max_{\mathcal{A}} \left\{\mathbf{Adv}_{E}^{\textit{PRP-RKA}}(\mathcal{A})\right\}$$

as the maximum advantage over all PRP-RKA adversaries that run in time at most T and ask a total maximum of q queries to the encryption and decryption oracles.

3.2. Tweakable Block Ciphers

The concept of tweakable block ciphers was introduced by Liskov *et al.* in [153]. The design is based on a common block cipher, which is extended by a so called tweak. A tweakable block cipher $\widetilde{E} : \{0,1\}^k \times \{0,1\}^u \times \{0,1\}^n \to \{0,1\}^n$ is defined as follows:

Definition 3.4 (Tweakable Block Cipher). *Let $\widetilde{E} : \{0,1\}^k \times \{0,1\}^u \times \{0,1\}^n \to \{0,1\}^n$ be a family of functions for some $k, u, n \in \mathbb{N}^+$. We denote $\widetilde{E}$ as a tweakable (k, u, n)-block cipher* iff *for any key tweak tuple $(K, U) \in \{0,1\}^k \times \{0,1\}^u$ it holds that*

$$\widetilde{E}(K, U, \cdot) \textit{ is a permutation.}$$

We denote the first input as key, the second input as tweak and the third input as message or plaintext, and the output as ciphertext.

We denote $\widetilde{E}_K^{-1}(U, \cdot)$ as the inverse of $\widetilde{E}_K(U, \cdot)$, *i.e.,* for all $M \in \{0,1\}^n$ it holds that $M = \widetilde{E}_K^{-1}(U, \widetilde{E}_K(U, M))$. Furthermore, we denote $\texttt{Block}(k, u, n)$ as the set of all tweakable (k, u, n)-block ciphers.

A tweakable block cipher $\widetilde{E} \in \texttt{Block}(k, u, n)$ is considered to be secure if it is *computationally indistinguishable* from a family of 2^u random n-bit permutations.

The formal definition of the IND-PRP advantage for tweakable block ciphers is similar to Definition 3.3.

Definition 3.5 (IND-PRP Advantage). *Let* $\widetilde{E} \in \texttt{Block}(k, u, n)$ *be a tweakable block cipher and let* $\mathfrak{P} \xleftarrow{\$} \mathbf{Perm}_n^k$ *be a family of random permutations. Suppose that* $K \xleftarrow{\$} \{0,1\}^k$*. Then, we define the IND-PRP advantage of an adversary* $\mathcal{A}$ *as*

$$\mathbf{Adv}_{\widetilde{E},\widetilde{E}^{-1}}^{\textit{IND-PRP}}(\mathcal{A}) = \left| \Pr\left[\mathcal{A}^{\widetilde{E}_K(\cdot,\cdot),\widetilde{E}_K^{-1}(\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathfrak{P}(\cdot,\cdot),\mathfrak{P}^{-1}(\cdot,\cdot)} \Rightarrow 1\right] \right|,$$

and

$$\mathbf{Adv}_{\widetilde{E},\widetilde{E}^{-1}}^{\textit{IND-PRP}}(q,t) = \max_{\mathcal{A}} \left\{ \mathbf{Adv}_{\widetilde{E},\widetilde{E}^{-1}}^{\textit{IND-PRP}}(\mathcal{A}) \right\}$$

as the maximum advantage over all IND-PRP adversaries that run in time at most t *and ask a total maximum of* q *queries to the encryption and decryption oracles.*

4 Authenticated Encryption Schemes

> Any sufficiently advanced technology
> is indistinguishable from magic.
>
> Arthur C. Clarke

A common requirement for cryptographic applications is to establish a secure channel between a sender and a receiver – usually referred to as *Alice* and *Bob* – who share a secret key. It may well be the case that sender and receiver represent the same entity, *e.g.,* the same person can first write sensitive data to an insecure storage, and later read these data. Usually, a secure channel should provide data privacy to prevent an eavesdropper from revealing any information about a message sent from Alice to Bob, except its length. The cryptographic technique to ensure this requierement is "encryption". Sometimes, data authenticity/integrity is required, *i.e.,* an adversary should not be able to manipulate messages without being noticed. This is cryptographically ensured by "authentication". Nevertheless, most of the time, users need both encryption and authentication: *authenticated encryption* (AE). An authenticated encryption scheme is a special kind of an encryption scheme that encrypts plaintext to authenticated ciphertexts.

Nonce. Goldwasser and Micali [117] formalized encryption schemes as stateful or probabilistic: otherwise, the data privacy is lost. Rogaway [204, 206, 208] proposed a unified point of view, by defining a cryptographic scheme as an always-deterministic algorithm that takes a user-supplied state called *nonce* (a *number used once*). We assume that an adversary is *nonce-respecting* when not stated otherwise. This type of adversary has full control over a nonce with the restriction to never choose the same value twice. This limitation only holds for encryption queries. Thus, an adversary is allowed to use a nonce multiple times when query the decryption oracle.

Deterministic Authenticated Encryption (DAE). In [209], Rogaway and Shrimpton addressed authenticated encryption schemes which provide security against repeated nonces . Furthermore, the authors shaped the notion of *misuse-resistance* and they proposed Synthetic Initialization Vector (SIV) as a solution. SIV and related schemes (Bivariate Tag Mixing (BTM) [126] and Hash Block Stealing (HBS) [127]) actually provide excellent security against nonce-reusing adversaries. Though, they are inherently off-line, *i.e.,* for encryption, one must either keep the entire plaintext in memory, or read the plaintext twice. This renders such deterministic approaches only practical for small messages.

On-line Authenticated Encryption (OAE). It is folklore that application programmers are used to process messages in an on-line manner. Hence, to seamlessly integrate authenticated encryption schemes into a typical software architecture, they should be on-line, *i.e.,* plaintexts and ciphertexts are split into conveniently-sized blocks, and the i-th ciphertext block can be written before the $(i+1)$-th plaintext block has to be read. An AE scheme fulfilling the on-line requirement is referred to as an OAE scheme.

4.1. Authenticated Encryption with Associated Data Schemes

An authenticated encryption scheme is a triple $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ of three algorithms.

1. The key-generation algorithm $\mathcal{K}$ takes no input and returns a randomly chosen key K from the key space $\{0,1\}^k$.
2. The deterministic encryption algorithm

$$\mathcal{E} : \{0,1\}^k \times (\{0,1\}^n)^* \times (\{0,1\}^n)^* \to (\{0,1\}^n)^* \times \{0,1\}^r$$

   maps a key-header-message tuple (K, H, M) to a ciphertext-tag-tuple (C, T)
3. The deterministic decryption algorithm

$$\mathcal{D} : \{0,1\}^k \times (\{0,1\}^n)^* \times (\{0,1\}^n)^* \to \{(\{0,1\}^n)^*, \perp\}$$

maps a header-ciphertext-tag-tuple (H, C, T) either to the authentic plaintext, if the input is valid, or returns $\perp$.

Usually, AE schemes operate on n-bit blocks, where n is the block length of the underlying primitive, *e.g.,* a block cipher. This is reflected by the notion $(\{0,1\}^n)^*$ where $M \in (\{0,1\}^n)^m$ implies that the message M consists of m message blocks, $m-1$ blocks of n-bit and a final message block M_m that can contain less then n-bits, *i.e.,* $M = M_1, \cdots, M_m$ with $|M_i| = n$ and $|M_m| \leq n$ with $i = 1, \ldots, m-1$.

It always holds that $|M| \leq |C| + |T|$ where C denotes the ciphertext and T the (authentication) tag. Note that OAE schemes require a nonce $N \in \{0,1\}^v$. In our notation N is a mandatory part of the header, whereas the optional part consists of associated data or meta data of the plaintext, *e.g.,* the TCP/IP header. Usually, for the sake of simplification, the nonce size v matches the block size n of the underlying primitive, *e.g.,* a (k, n)-block cipher.

An Authenticated Encryption with Associated Data (AEAD) scheme ensures privacy and integrity for the plaintext; in addition, it ensures the integrity of the header. This renders those schemes useful in settings where the associated data of messages is predictable.

4.2. Generic Composition

An AE scheme can be generated by combining a secure encryption scheme with a secure MAC. Given two independent keys K and L, the common literature lists three construction approaches for such a *generic composition* [22].

Encrypt-and-Mac. Encrypt the plaintext M and append a MAC of the plaintext to the ciphertext: $E_K(M) \mid\mid \mathrm{MAC}_L(M)$. Variants of this method are used in the transport layer of the SSH protocol [242].

Mac-then-Encrypt. Append a MAC of the plaintext to the plaintext and then encrypt them together. Here, the output is $E_K(M \mid\mid \mathrm{MAC}_L(M))$. Variants of this method are used in the TLS protocol version 1.0 and 1.1 [76, 77].

Encrypt-then-Mac. Encrypt the plaintext to get a ciphertext and append a MAC of this ciphertext: $E_K(M) \mid\mid \mathrm{MAC}_L(E_K(M))$. Variants of this method are used in the IPSec protocol [137].

Out of these, only the *Encrypt-then-Mac* scheme is free of weaknesses [22]. Note that this approach can fail trivially by key management errors: suppose the receiver side only updates the authentication key, but not the encryption key. Then, *Encrypt-then-Mac* will decrypt a ciphertext into "authentic" random garbage. Therefore, it is less error-prone to use a dedicated authenticated encryption scheme and not a generic composition.

4.3. Security Notions

Authenticated encryption schemes require security notions for both privacy and integrity. Notions and their relations were introduced for deterministic schemes in [210] and for nonce-based schemes in [22, 27, 133, 204, 208]. In this thesis we adopt the notion of Chosen-Ciphertext Attack 3 (CCA3) security suggested in [210]. Similar to the security definition of IND-PRP (cf. Definition 3.2), a CCA3 adversary $\mathcal{A}$ has to distinguish between the real world, where it has oracle access to $\mathcal{E}_K(\cdot,\cdot)$ and $\mathcal{D}_K(\cdot,\cdot,\cdot)$ of an AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, and the random world, where $\mathcal{A}$ has access to the oracles $\$(\cdot,\cdot)$ and $\perp(\cdot,\cdot,\cdot)$. The random oracle $\$(\cdot,\cdot)$ returns a string of random bits, whereas $\perp(\cdot,\cdot,\cdot)$ always returns $\perp$. Note that the equation $|\$(H,M)| = |\mathcal{E}_K(H,M)|$ holds for all header-plaintext tuple (H,M). For the sake of simplification, we assume that an adversary never asks a query to which the corresponding answer is already known.

Definition 4.1 (CCA3 Advantage). *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an authenticated encryption scheme as described in Section 4.1. The advantage of an adversary $\mathcal{A}$ in breaking Π is defined as*

$$\mathbf{Adv}_{\Pi}^{\mathit{CCA3}}(\mathcal{A}) = \left|\Pr\left[K \leftarrow \mathcal{K} : \mathcal{A}^{\mathcal{E}_K(\cdot,\cdot),\mathcal{D}_K(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot,\cdot),\perp(\cdot,\cdot,\cdot)} \Rightarrow 1\right]\right|,$$

and

$$\mathbf{Adv}_{\Pi}^{\mathit{CCA3}}(q,\ell,t) = \max_{\mathcal{A}} \left\{\mathbf{Adv}_{\Pi}^{\mathit{CCA3}}(\mathcal{A})\right\}$$

as the maximum advantage over all nonce-respecting *CCA3-adversaries that run in time at most t, ask total maximum of q queries to the encryption and decryption oracles, and whose total query length is at most ℓ blocks.*

It is easy to see that we can rewrite the term given in Definition 4.1 as
$\mathbf{Adv}_{\Pi}^{\mathit{CCA3}}(\mathcal{A}) =$

$$\Big|\Pr\left[K \leftarrow \mathcal{K} : \mathcal{A}^{\mathcal{E}_K(\cdot,\cdot),\mathcal{D}_K(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[K \leftarrow \mathcal{K} : \mathcal{A}^{\mathcal{E}_K(\cdot,\cdot),\perp(\cdot,\cdot,\cdot)} \Rightarrow 1\right] \tag{4.1}$$

$$+ \Pr\left[K \leftarrow \mathcal{K} : \mathcal{A}^{\mathcal{E}_K(\cdot,\cdot),\perp(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot,\cdot),\perp(\cdot,\cdot,\cdot)} \Rightarrow 1\right]\Big|. \tag{4.2}$$

One can interpret (4.1) as the advantage that an adversary has on the integrity of the ciphertext and (4.2) as the advantage an adversary has on the privacy. We use this decomposition as a motivational starting point to define ciphertext integrity and what we mean by an Indistinguishability under Chosen-Plaintext Attack (IND-CPA) adversary against authenticated encryption schemes.

Indistinguishability under Chosen-Plaintext Attack (IND-CPA). Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an authenticated encryption scheme and $\mathcal{A}$ an IND-CPA adversary. The task of $\mathcal{A}$ is to distinguish the *real* world, where it is given oracle access to $\mathcal{E}_K(\cdot,\cdot)$ under a secret key $K \in \{0,1\}^k$, from the *random* world, where $\mathcal{A}$ has access to a random oracle $\$(\cdot,\cdot)$ which returns, consistent, random ciphertexts, as described earlier in this section. If no such adversary $\mathcal{A}$ can perform significant better than random guessing, then, Π protects the privacy of encrypted messages. The IND-CPA advantage is defined as follows:

Definition 4.2 (IND-CPA Advantage). *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an authenticated encryption scheme as described in Section 4.1. Then, the IND-CPA advantage of a* nonce-respecting *adversary $\mathcal{A}$ is defined as*

$$\mathbf{Adv}_{\Pi}^{\mathit{IND\text{-}CPA}}(\mathcal{A}) = \left|\Pr\left[K \leftarrow \mathcal{K} : \mathcal{A}^{\mathcal{E}_K(\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot,\cdot)} \Rightarrow 1\right]\right|,$$

and

$$\mathbf{Adv}_{\Pi}^{\mathit{IND\text{-}CPA}}(q,\ell,t) = \max_{\mathcal{A}} \left\{\mathbf{Adv}_{\Pi}^{\mathit{IND\text{-}CPA}}(\mathcal{A})\right\}$$

as the maximum advantage over all nonce-respecting *IND-CPA-adversaries that run in time at most t, ask a total maximum of q queries to the encryption oracle, and whose total query length is not more than ℓ blocks.*

Integrity of Ciphertext (INT-CTXT). The security notion INT-CTXT is defined by the game-playing approach [29], where the advantage of an adversary is measured

Algorithm 2 INT-CTXT Game

Initialize()
  $K \xleftarrow{\$} \mathcal{K}$
  $\mathtt{win} \leftarrow \mathbf{false}$
  $\mathfrak{Q} \leftarrow \emptyset$
Finalize()
  return $\mathtt{win}$

Encrypt(H,M)
  $(C,T) \leftarrow \mathcal{E}_K(H,M)$
  $\mathfrak{Q} \cup \{(H,C,T)\}$
  return (C,T)

Verify(H,C,T)
  $M \leftarrow \mathcal{D}_K(H,C,T)$
  if $((H,C,T) \notin \mathfrak{Q}) \wedge (M \neq \perp)$ **then**
    $\mathtt{win} \leftarrow \mathbf{true}$
  end if
  return M

as the success probability of winning a (cryptographic) *game* G. Each game consists of three functions: An initialization function **Initialize()**, a finalization function **Finalize()**, and oracle functions. Any adversary $\mathcal{A}$ that is playing a game calls the **Initialize()** function first. In the following, $\mathcal{A}$ then makes some queries to the encrypt and decrypt oracles, and finally, $\mathcal{A}$ ends the game by invoking **Finalize()**.

To $\mathcal{A}$, every function of a game is a black box, *i.e.,* it has no access to internal variables. An adversary wins the game if and only if **Finalize()** returns **true**. We denote $\Pr[\mathcal{A}^G \Rightarrow 1]$ as the probability that the adversary wins the Game G.

An AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ protects the ciphertext integrity against an adversary $\mathcal{A}$ when it is not able to come up with a fresh *authentic ciphertext* tuple (H, C, T), *i.e.,* $\mathcal{D}_K(H, C, T) \neq \perp$, where (H, C, T) is not the result of a previous query of $\mathcal{A}$. The INT-CTXT advantage based on the the Game $G_{\text{INT-CTXT}}$ (cf. Algorithm 2) is formally defined as follows:

Definition 4.3 (INT-CTXT Advantage). *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an authenticated encryption scheme as introduced in Section 4.1, and let $G_{\mathit{INT\text{-}CTXT}}$ denote the game from Algorithm 2. Then, the INT-CTXT advantage of a* nonce-respecting *adversary $\mathcal{A}$ is defined as*

$$\mathbf{Adv}_{\Pi}^{\mathit{INT\text{-}CTXT}}(\mathcal{A}) = \Pr\left[\mathcal{A}^{G_{INT-CTXT}} \Rightarrow 1\right],$$

and

$$\mathbf{Adv}_{\Pi}^{\mathit{INT\text{-}CTXT}}(q, \ell, t) = \max_{\mathcal{A}} \left\{\mathbf{Adv}_{\Pi}^{\mathit{INT\text{-}CTXT}}(\mathcal{A})\right\}$$

as the maximum advantage over all nonce-respecting *INT-CTXT-adversaries that run in time at most t, ask a total maximum of q queries to the encryption and decryption oracles, and whose total query length is at most ℓ blocks.*

Upper-Bounding the CCA3 Advantage. Bellare and Namprempre showed in [22] that an authenticated encryption scheme that is both IND-CPA- and INT-CTXT-secure, is also CCA3-secure. This notable observation is formalized as follows:

Theorem 4.4 (CCA3 Advantage [22]). *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an authenticated encryption scheme as introduced in Section 4.1, and let $\mathcal{A}$ be a* nonce-respecting *CCA3$_\Pi$ adversary that runs in time t, and makes q queries with a total length of at most ℓ blocks. Then, there exists an IND-CPA$_\Pi$-adversary $\mathcal{A}_p$ and an INT-CTXT$_\Pi$ adversary $\mathcal{A}_c$ such that*

$$\mathbf{Adv}_{\Pi}^{\textit{CCA3}}(\mathcal{A}) \leq \mathbf{Adv}_{\Pi}^{\textit{IND-CPA}}(\mathcal{A}_p) + \mathbf{Adv}_{\Pi}^{\textit{INT-CTXT}}(\mathcal{A}_c),$$

where both $\mathcal{A}_p$ and $\mathcal{A}_c$ run in time $O(t)$ and make at most q queries.

Proof Sketch. By applying the *triangle inequality* on Definition 4.1, we have $\mathbf{Adv}_{\Pi}^{\textit{CCA3}}(\mathcal{A}) =$

$$\begin{aligned}
&\left|\Pr\left[K \leftarrow \mathcal{K} : \mathcal{A}^{\mathcal{E}_K(\cdot,\cdot),\mathcal{D}_K(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot,\cdot),\perp(\cdot,\cdot,\cdot)} \Rightarrow 1\right]\right| \\
\leq &\left|\Pr\left[K \leftarrow \mathcal{K} : \mathcal{A}^{\mathcal{E}_K(\cdot,\cdot),\mathcal{D}_K(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[K \leftarrow \mathcal{K} : \mathcal{A}^{\mathcal{E}_K(\cdot,\cdot),\perp(\cdot,\cdot,\cdot)} \Rightarrow 1\right]\right| \\
+ &\left|\Pr\left[K \leftarrow \mathcal{K} : \mathcal{A}^{\mathcal{E}_K(\cdot,\cdot),\perp(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot,\cdot),\perp(\cdot,\cdot,\cdot)} \Rightarrow 1\right]\right|
\end{aligned}$$

For a key $K \stackrel{\$}{\leftarrow} \mathcal{K}$, we design two adversaries $\mathcal{A}_p$ and $\mathcal{A}_c$ so that

$$\begin{aligned}
&\left|\Pr\left[\mathcal{A}^{\mathcal{E}_K(\cdot,\cdot),\perp(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot,\cdot),\perp(\cdot,\cdot,\cdot)} \Rightarrow 1\right]\right| \leq \mathbf{Adv}_{\Pi}^{\textit{IND-CPA}}(\mathcal{A}_p) \\
&\left|\Pr\left[\mathcal{A}^{\mathcal{E}_K(\cdot,\cdot),\mathcal{D}_K(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathcal{E}_K(\cdot,\cdot),\perp(\cdot,\cdot,\cdot)} \Rightarrow 1\right]\right| \leq \mathbf{Adv}_{\Pi}^{\textit{INT-CTXT}}(\mathcal{A}_c).
\end{aligned}$$

$\mathcal{A}_p$: Adversary $\mathcal{A}_p$ runs $\mathcal{A}$ and answers $\mathcal{A}$'s queries to the function **Encrypt** and **Decrypt** by using its own **Encrypt** oracle or returning $\perp$, respectively. $\mathcal{A}_p$ outputs whatever $\mathcal{A}$ outputs.

$\mathcal{A}_c$: Adversary $\mathcal{A}_c$ runs $\mathcal{A}$, and answers $\mathcal{A}$'s queries to the function **Encrypt** by using its own **Encrypt** oracle. It submits $\mathcal{A}$'s queries to the **Decrypt** oracle to its own **Verify** oracle and, regardless of the response, returns $\perp$. Note that the **Verify** oracle sets `win` to **true** if and only if a fresh **Decrypt** query of $\mathcal{A}_c$ is valid. ■

4.4. Game-Based Proofs

The majority of the upcoming proofs in this paper are based on common game-playing arguments. In this thesis, all games are written in a language similar to $\mathcal{L}$ that was introduced by Bellare and Rogaway in [28]. The basic concept of this proof technique is called *game hopping*. It is a formalized way to transform a cryptographic scheme into an ideal scheme, *e.g.,* a random function by a series of *minor* modifications. We denote $G_0, \ldots, G_n$ as a series of games, where G_0 denotes the initial game and G_n the final game. As usual, our adversary $\mathcal{A}$ has only black-box access to any Game G_i. Thus, the advantage of $\mathcal{A}$ to distinguish Game G_i from Game G_j is given by

$$\mathbf{Adv}_{G_i}^{G_j}(A) = \left|\Pr\left[\mathcal{A}^{G_i} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{G_j} \Rightarrow 1\right]\right|.$$

Game-playing proofs become handy when it is hard to compute $\mathbf{Adv}_{G_0}^{G_n}(\mathcal{A})$ in a straightforward manner. The difference between subsequent games G_i and G_{i+1} is, by construction, easy to compute. Finally, from the common triangle inequality, we have $\mathbf{Adv}_{G_i}^{G_{i+2}}(\mathcal{A}) \leq \mathbf{Adv}_{G_i}^{G_{i+1}}(\mathcal{A}) + \mathbf{Adv}_{G_{i+1}}^{G_{i+2}}(\mathcal{A})$, and thus,

$$\mathbf{Adv}_{G_0}^{G_n}(\mathcal{A}) \leq \sum_{i=1}^{n} \mathbf{Adv}_{G_{i-1}}^{G_i}(\mathcal{A}).$$

5 Misusing Authenticated Encryption Schemes

> You are never too old to set another goal or to dream a new dream.
>
> C. S. Lewis

During the past decade, many AE schemes were proposed – usually with a formal proof of their respective CCA3 security. Up to now, CCA3 proofs used to rely on two common assumptions: (1) *nonce-respecting* adversaries, and (2) secure underlying primitives. While both aspects are well-understood in theory, they are hard to guarantee in practice. Thus, security issues were overlooked or ignored in various cases and security applications were put at high risk. In this thesis we highlight two *blind spots* in the established security definitions: *nonce misuse* and *decryption misuse*.

5.1. Nonce Misuse

The standard requirement for encryption schemes – authenticated or not – is to prevent leakage of any information about the plaintext except for its length. A stateless deterministic authenticated encryption scheme cannot fulfill this security requirement since an adversary can easily detect, if a plaintext was encrypted multiple times or not. Thus, the user must provide a fresh additional auxiliary input (called nonce) for each encryption. We speak of a *nonce misuse*, if a nonce value is reused.

In theory, the concept of nonces is simple. In practice, it is challenging to ensure that nonces never repeat. Flawed implementations of nonces are ubiquitous [51, 122, 146, 214, 239], but, apart from implementation failures, cases exist where software developers cannot always prevent nonce reuse. For example, a persistently stored counter that is increased and written back each time a new nonce is needed may be reset by a backup – usually after some previous data loss. Similarly, the internal and persistent state of an application may be duplicated when a virtual machine is cloned, etc.

Our analysis in Section 6.1 shows that almost all previously published OAE schemes cannot longer ensure the privacy, integrity, or both for encrypted messages when threatened by a *nonce-ignoring* adversary.

Ideally, an adversary that is given the encryption of two (equal-length) plaintexts M^1 and M^2 cannot even decide if $M^1 = M^2$ or not. When a nonce is used more than once, deciding if $M^1 = M^2$ becomes easy. Deterministic encryption schemes, such as SIV [209], ensure that they do not leak any other additional information about plaintexts, even when exposed to a nonce-reusing adversary. In the case of on-line encryption, where the i-th ciphertext block is independend of all message blocks M_j with $j > i$, it is unavoidably to leak information beyond $M^1 = M^2$. The adversary can compare any pair of ciphertexts for their Longest Common Prefix (LCP), and then derive the longest common prefix of their corresponding plaintexts. We propose to call an (on-line) AE scheme *misuse resistant* if the only information an adversary can obtain from ciphertexts are their lengths, and the LCP of its plaintexts. In the following we first formally define the length of the LCP.

Definition 5.1 (Length of the Longest Common Prefix (LLCP)). *Let $M, M' \in (\{0,1\}^n)^*$ denote two messages. Then, we define the* length of the longest common n-prefix *of M and M' as*

$$\mathrm{LLCP}_n(M, M') = \max_i \left\{M_1 = M'_1, \ldots, M_i = M'_i\right\}.$$

For a non-empty set $\mathfrak{Q}$ of elements of $(\{0,1\}^n)^*$, we define

$$\mathrm{LLCP}_n(M, \mathfrak{Q}) = \max_{X \in \mathfrak{Q}} \{\mathrm{LLCP}_n(M, X)\}.$$

On-line Permutation (OPerm). We aim for larger permutations that not only permute single blocks but can handle messages of multiple blocks. Such a permutation, from $\{0,1\}^{na}$ to $\{0,1\}^{na}$ for $a > 1$, is (n-)on-line if the i-th block of the output is completely determined by the first i blocks of the input. Let $\mathbf{OPerm}_n$ denote the set

Algorithm 3 Random On-Line Permutation Implemented via Lazy Sampling

Init()	$\mathfrak{P}(M)$	$\mathfrak{P}^{-1}(M)$
$X_I \leftarrow \perp$	**for** $i \leftarrow 1, \ldots, \lvert M\rvert/n$ **do**	**for** $i \leftarrow 1, \ldots, \lvert C\rvert/n$ **do**
$Y_I \leftarrow \perp$	$I \leftarrow M_1, \ldots, M_{i-1}$	$I \leftarrow C_1, \ldots, C_{i-1}$
$\mathfrak{D}_I \leftarrow \emptyset$	$Z \leftarrow X_I[M_i]$	$Z \leftarrow Y_I[C_i]$
$\mathfrak{R}_I \leftarrow \emptyset$	**if** $Z = \perp$ **then**	**if** $Z = \perp$ **then**
	$Z \stackrel{\$}{\leftarrow} \{0,1\}^n \setminus \mathfrak{D}_I$	$Z \stackrel{\$}{\leftarrow} \{0,1\}^n \setminus \mathfrak{R}_I$
	$X_I[M_i] \leftarrow Z$	$Y_I[C_i] \leftarrow Z$
	$Y_I[Z] \leftarrow M_i$	$X_I[Z] \leftarrow C_i$
	$\mathfrak{R}_I \leftarrow \mathfrak{R}_I \cup \{Z\}$	$\mathfrak{D}_I \leftarrow \mathfrak{D}_I \cup \{Z\}$
	$\mathfrak{D}_I \leftarrow \mathfrak{D}_I \cup \{M_i\}$	$\mathfrak{R}_I \leftarrow \mathfrak{R}_I \cup \{C_i\}$
	end if	**end if**
	$C_i \leftarrow Z$	$M_i \leftarrow Z$
	end for	**end for**
	return $(C_1, \ldots, C_{\lvert M\rvert/n})$	**return** $(M_1, \ldots, M_{\lvert C\rvert/n})$

of all on-line permutations from $(\{0,1\}^n)^*$ to $(\{0,1\}^n)^*$. It is easy to extend the definition with a state space $\{0,1\}^v$. Let $\mathbf{OPerm}_n^v$ denote the set of all functions from $\{0,1\}^v \times (\{0,1\}^n)^* \to (\{0,1\}^n)^*$. Then, for each $\mathfrak{G} \in \mathbf{OPerm}_n^v$ and $N \in \{0,1\}^v$, the function $\mathfrak{G}(N, \cdot)$ is an (n-)on-line permutation. We define an On-line Pseudo-random Permutation (OPRP) as a family of n^*-bit on-line permutations with the property that the input-output behaviour of a randomly chosen member of this family is *computationally indistinguishable* from a set of 2^v n^*-bit random permutations $\mathfrak{P}(\cdot,\cdot) \stackrel{\$}{\leftarrow} \mathbf{OPerm}_n^v$. An efficient lazy-sampling implementation of a random on-line permutation is given in Algorithm 3. Note that in the first iteration the prefix I is always set to the empty string ϵ since neither M_0 nor C_0 exists.

Since AE schemes do not only output ciphertexts but also authentication tags, our encryption oracle has to simulate the computation of an authentication tag by returning a random bitstring that matches the length of the tag. Thus, our encryption oracle $\mathcal{O}^{\mathfrak{P}}$ processes a header-message tuple (H, M) as follows:

1. Compute $C \leftarrow \mathfrak{P}(H, M)$.
2. Compute $T \stackrel{\$}{\leftarrow} \{0,1\}^r$.
3. Append some random bits to C if necessary so that the equation $\lvert\mathcal{E}_K(H, M)\rvert = \lvert C\rvert + \lvert T\rvert$ always holds.

4. Finally, output the ciphertext-tag tuple (C, T).

To achieve length preserving encryption, *i.e.,* $|M| = |C|$ for all messages $M \in \{0,1\}^*$, OAE schemes usually have a special treatment for the last message block, *e.g.,* ciphertext stealing where the final message block M_m is padded with the ciphertext block C_{m-1}. Thus, it is quite easy to distinguish such an OAE scheme from the encryption oracle $\mathcal{O}^{\mathfrak{P}}$. At first, we can send any encryption query (H, M) with $M = M_1, \ldots, M_m$ to the encryption oracle. Then we query (H, M') where $M' = M \mid\mid Z$ for any $Z \in \{0,1\}^n$. Let (C, T) denote the output of our first query and (C', T') the output of the second query. Then we output 1 if $C_m \neq C'_m$, and 0 otherwise. Thus, we have to update the definition of the random encryption oracle $\mathcal{O}^{\mathfrak{P}}$ by an intermediate Step 1b: Replace the final ciphertext block (C_m) by a random bitstring when $\mathcal{E}_K$ treats the final message block special.

Definition 5.2 (IND-OPRP Advantage). *Let* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *be an OAE scheme, and let* $\mathfrak{P} \xleftarrow{\$} \boldsymbol{OPerm}_n^{n+}$. *Then, we define the* IND-OPRP advantage *of a* nonce-ignoring *adversary* $\mathcal{A}$ *as*

$$\mathbf{Adv}_{\Pi}^{\textit{IND-OPRP}}(\mathcal{A}) = \left|\Pr\left[K \leftarrow \mathcal{K} : \mathcal{A}^{\mathcal{E}_K(\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathcal{O}^{\mathfrak{P}}(\cdot,\cdot)} \Rightarrow 1\right]\right|,$$

and

$$\mathbf{Adv}_{\Pi}^{\textit{IND-OPRP}}(q, \ell, t) = \max_{\mathcal{A}} \left\{\mathbf{Adv}_{\Pi}^{\textit{IND-OPRP}}(\mathcal{A})\right\}$$

as the maximum advantage over all IND-OPRP adversaries that run in time at most t*, ask a total maximum of* q *queries to the encryption oracles, and whose total query length is at most* ℓ *blocks.*

In the spirit of the CCA3 security definition (cf. Definition 4.1), we introduce the notion of On-line Chosen-Ciphertext Attack 3 (OCCA3) security.

Definition 5.3 (OCCA3 Advantage). *Suppose* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *is an OAE scheme, and let* $\mathfrak{P} \xleftarrow{\$} \boldsymbol{OPerm}_n$ *be a random on-line permutation. Then, we define the* OCCA3 advantage *of a nonce-ignoring adversary* $\mathcal{A}$ *as*

$$\mathbf{Adv}_{\Pi}^{\textit{OCCA3}}(\mathcal{A}) = \left|\Pr\left[K \leftarrow \mathcal{K} : \mathcal{A}^{\mathcal{E}_K(\cdot,\cdot),\mathcal{D}_K(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathcal{O}^{\mathfrak{P}}(\cdot,\cdot),\perp(\cdot,\cdot,\cdot)} \Rightarrow 1\right]\right|,$$

and

$$\mathbf{Adv}_{\Pi}^{\textit{CCA3}}(q, \ell, t) = \max_{\mathcal{A}} \left\{\mathbf{Adv}_{\Pi}^{\textit{CCA3}}(\mathcal{A})\right\}$$

as the maximum advantage over all nonce-ignoring *OCCA3 adversaries that run in time at most t, ask a total maximum of q queries to the encryption and decryption oracles, and whose total query length is at most ℓ blocks.*

Using similar arguments as in the proof of Theorem 4.4, one can show that for any (q, ℓ, t)-bounded adversary $\mathcal{A}$, there exists a $(q, \ell, O(t))$-bounded $\mathcal{A}_p$ such that

$$\left| \Pr\left[K \leftarrow \mathcal{K} : \mathcal{A}^{\mathcal{E}_K(\cdot,\cdot), \perp(\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathcal{O}^P(\cdot,\cdot), \perp(\cdot,\cdot)} \Rightarrow 1\right]\right| \leq \mathbf{Adv}_{\Pi}^{\mathit{IND\text{-}OPRP}}(\mathcal{A}).$$

Corollary 5.4 (Bound for the OCCA3 Advantage). *Suppose* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *is an OAE scheme. Then, it holds that*

$$\mathbf{Adv}_{\Pi}^{\mathit{OCCA3}}(q, \ell, t) \leq \mathbf{Adv}_{\Pi}^{\mathit{IND\text{-}OPRP}}(q, \ell, t) + \mathbf{Adv}_{\Pi}^{\mathit{INT\text{-}CTXT}}(q, \ell, t).$$

5.2. Decryption Misuse

The decryption algorithm of an authenticated encryption scheme either outputs a plaintext, or the bot symbol $\perp$, depending on whether a ciphertext is authentic or not. A *decryption misuse* describes the event that information about the *would-be* plaintext of an invalid ciphertext leaks. An adversary might use this leaked information to break the privacy (or integrity) of an AE scheme. A generic way to get rid of this problem is the *Decrypt-Then-Mask* approach by Fouque *et al.* [105], where the would-be plaintext is blinded after decryption by XORing it with a pseudorandom sequence of bits generated by a Pseudorandom Number Generator (PRNG). After successful authentication, the blinding is removed. Unfortunately, this technique is not applicable in low-end environments since required temporary storage for the decrypted data may just not exist. In high-speed environments, *e.g.,* optical networks, the increased latency for the waiting period that is required until the plaintext authenticity has been established may be prohibitive.

We strive for an authenticated encryption scheme where any change to a valid ciphertext causes its entire post-decryption plaintext to be pseudorandom. Such a scheme is clearly *decryption-misuse resistant* since the decryption of a manipulated ciphertext results in uncontrollable random noise. Unfortunately, this strong definition of *decryption-misuse resistance* and on-line encryption are mutually exclusive:

If an adversary manipulates the i-th block of a ciphertext, an OAE scheme leaves the previous $(i-1)$ blocks unchanged. Therefore, we will introduce two flavours of decryption-misuse resistance, one for deterministic AE schemes and one for OAE schemes.

Indistinguishability under Chosen-Ciphertext Attack (IND-CCA). For deterministic authenticated encryption schemes, we can adapt the CCA3 notion (see Definition 4.1) by slightly modifying the behavior of the decryption oracle. Let $\widehat{\mathcal{D}}$ denote the *faulty* version of the decrypt and verify algorithm $\mathcal{D}$, *i.e.,* $\widehat{\mathcal{D}}$ omits the verification and always returns the decryption for authentic as well as for unauthentic ciphertexts. Thus, in the decryption-misuse setting, an adversary has to distinguish $(\mathcal{E}_K, \widehat{\mathcal{D}}_K)$ with $K \leftarrow \mathcal{K}$ from two independent random oracles $(\$^{\mathcal{E}}, \$^{\mathcal{D}})$. Note that the equation $|\mathcal{E}_K(H,M)| = |\$^{\mathcal{E}}(H,M)|$ holds for header-message tuple (H,M) and $|\widehat{\mathcal{D}}_K(H,C,T)| = |\$^{\mathcal{D}}(H,C,T)|$ holds for all header-ciphertext-tag tuples (H,C,T). Furthermore, we assume that an adversary never asks a query for which the corresponding answer is already known. Then, we define the IND-CCA advantage as follows:

Definition 5.5 (IND-CCA Advantage). *Let* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *be a determinisitc AE scheme, and let the faulty decryption oracle* $\widehat{\mathcal{D}}$ *be defined as above. Then, we define the* IND-CCA advantage *of a* nonce-respecting *adversary* $\mathcal{A}$ *in breaking* Π *with* $K \leftarrow \mathcal{K}$ *as*

$$\mathbf{Adv}_{\Pi}^{\textit{IND-CCA}}(\mathcal{A}) = \left|\Pr\left[\mathcal{A}^{\mathcal{E}_K(\cdot,\cdot),\widehat{\mathcal{D}}_K(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$^{\mathcal{E}}(\cdot,\cdot),\$^{\mathcal{D}}(\cdot,\cdot,\cdot)} \Rightarrow 1\right]\right|$$

and

$$\mathbf{Adv}_{\Pi}^{\textit{IND-CCA}}(q,\ell,t) = \max_{\mathcal{A}} \left\{\mathbf{Adv}_{\Pi}^{\textit{IND-CCA}}(\mathcal{A})\right\}$$

as the maximum advantage over all nonce-respecting *IND-CCA adversaries that run in time at most* t*, ask a total maximum of* q *queries to the encryption and decryption oracles, and whose total query length is at most* ℓ *blocks.*

The following Lemma discloses the relation between the IND-CPA and the IND-CCA security notions.

Lemma 5.6 (IND-CCA $\implies$ IND-CPA). *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a deterministic AE scheme, and let $\mathcal{A}$ be an IND-CPA adversary that runs in time t, and makes q queries with a total length of at most ℓ blocks. Then, there exists an IND-CCA adversary $\mathcal{A}'$ such that*

$$\mathbf{Adv}_{\Pi}^{\textit{IND-CPA}}(\mathcal{A}) \leq \mathbf{Adv}_{\Pi}^{\textit{IND-CCA}}(\mathcal{A}').$$

Proof. The adversary $\mathcal{A}'$ runs $\mathcal{A}$ and answers $\mathcal{A}$'s queries to its encryption oracle. Furthermore, $\mathcal{A}'$ outputs whatever $\mathcal{A}$ outputs. ■

Indistinguishability under On-Line Chosen-Ciphertext Attack (IND-OCCA). We can adapt the IND-CCA notion by replacing the random oracle for decryption ($\$^{\mathcal{D}}$) by a random permutation-based oracle $\widehat{\mathcal{O}}^P$ with $\mathfrak{P} \xleftarrow{\$} \mathbf{OPerm}_n^{n+}$. The construction of this new oracle is quite similar to the construction of the OPerm-based encryption oracle $\mathcal{O}^{\mathfrak{P}}$ from Section 5.1. Thus, $\widehat{\mathcal{O}}^{\mathfrak{P}}$ processes a header-ciphertext-tag tuple (H, C, T) as follows: (1) It computes the plaintext $M \leftarrow \mathfrak{P}(H, C)$ and if necessary, replaces the final message block with random bits. (2) It appends random bits to M if necessary, so that the equation $|\widehat{\mathcal{D}}_K(H, C, T)| = |M|$ always holds, and finally, outputs M.

Definition 5.7 (IND-OCCA Advantage). *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an OAE scheme. Then, we define for $\mathfrak{P} \xleftarrow{\$}$ $\boldsymbol{OPerm}_n^{n+}$ the* IND-OPRP advantage *of a* nonce-respecting *adversary $\mathcal{A}$ as*

$$\mathbf{Adv}_{\Pi}^{\textit{IND-OCCA}}(\mathcal{A}) = \left| \Pr\left[K \leftarrow \mathcal{K} : \mathcal{A}^{\mathcal{E}_K(\cdot,\cdot), \widehat{\mathcal{D}}(\cdot,\cdot,\cdot)} \Rightarrow 1 \right] - \Pr\left[\mathcal{A}^{\$(\cdot,\cdot), \widehat{\mathcal{O}}^{\mathfrak{P}}(\cdot,\cdot,\cdot)} \Rightarrow 1 \right] \right|,$$

and

$$\mathbf{Adv}_{\Pi}^{\textit{IND-OCCA}}(q, \ell, t) = \max_{\mathcal{A}} \left\{ \mathbf{Adv}_{\Pi}^{\textit{IND-OCCA}}(\mathcal{A}) \right\}$$

as the maximum advantage over all IND-OCCA adversaries that run in time at most t, ask a total maximum of q queries to the encryption and decryption oracles, and whose total query length is at most ℓ blocks.

Remark. In 2014, Andreeva *et al.* introduced the notion of *integrity of unverified plaintext release* INT-RUP [6]. In their model, an adversary $\mathcal{A}$ has access to an AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{V})$ with separate decryption and verification algorithms $\mathcal{D}$ and $\mathcal{V}$, respectively. $\mathcal{A}$ wins if it can forge. Furthermore, the authors also introduced the notion of *plaintext awareness* PA, where an adversary has to distinguish between the real and a *simulated world*, where the decryption oracle is replaced by a simulator S that has no access to the secret key. In addition, Andreeva *et al.* propose two different notion of plaintext awareness, named PA1 and PA2. Both differ in the fact that the simulator has aceess to the query history of $\mathcal{E}_K$ in the former notion. Note that PA2 security implies PA1 security since every PA2-simulator is also a PA1-simulator.

We want to emphasize that despite the close relations of our work to that by [6], both are results of completely independent efforts. Analyzing the relations among our and their notions and unifying them is still an open research topic.

5.3. Robustness

This section concludes this chapter by answering the question whether an authenticated encryption scheme in this thesis is considered robust or not.

Deterministic Authenticated Encryption Schemes. It is not hard to tell if a deterministic AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is robust or not. It is robust *iff* it is CCA3-secure in the nonce-respecting and nonce-misuse setting, and IND-CCA-secure in the decryption-misuse setting. The following corollary upper bounds the Nonce- and Decryption-Misuse Attack (NDMA) advantage of a *nonce-ignoring* adversary $\mathcal{A}$. It can be derived from Theorem 4.4, which tells us that IND-CPA plus INT-CTXT security implies CCA3 security, and Lemma 5.6, which states that IND-CCA security implies IND-CPA security. Thus, NDMA security implies robustness.

Corollary 5.8 (NDMA Advantage). *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a deterministic AE scheme. Let $\mathcal{A}$ be a* nonce-ignoring *NDMA adversary that runs in time t, and asks at most q queries with a total length of at most ℓ blocks. Then, there exists a* nonce-ignoring *IND-CCA adversary $\mathcal{A}_p$ and a* nonce-ignoring *INT-CTXT adversary $\mathcal{A}_c$ such that*

$$\mathbf{Adv}_{\Pi}^{NDMA}(\mathcal{A}) \leq \mathbf{Adv}_{\Pi}^{IND\text{-}CCA}(\mathcal{A}_p) + \mathbf{Adv}_{\Pi}^{INT\text{-}CTXT}(\mathcal{A}_c),$$

where both $\mathcal{A}_p$ and $\mathcal{A}_c$ run in time $O(t)$ and make at most q queries with a total length of at most ℓ blocks.

On-Line Authenticated Encryption Schemes. We denote an OAE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ robust *iff* it is (1) CCA3-secure in the nonce-respecting setting, (2) OCCA3-secure in the nonce-misuse setting, and (3) IND-OCCA-secure in the decryption-misuse setting. In the following we introduce the definition of an Indistinguishability under On-Line Chosen-Ciphertext Attack 2 (IND-OCCA2) advantage, which is a generalisation of the IND-OCCA advantage by replacing the *nonce-respecting* adversary with a *nonce-ignoring* adversary. It is basically the same as the generalisation of the IND-CPA advantage by introducing the IND-OPRP advantage. For the following definition, we borrow the notion of the encryption oracle $\mathcal{O}^P$ and the decryption oracle $\widehat{\mathcal{D}}$ from the Sections 5.1 and 5.2, respectively.

Definition 5.9 (IND-OCCA2 Advantage). *Let* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *be an OAE scheme. Then, we define for* $\mathfrak{P} \xleftarrow{\$} \boldsymbol{OPerm}_n^{n+}$ *the* IND-OCCA2 advantage *of a* nonce-ignoring *adversary* $\mathcal{A}$ *as*

$$\mathbf{Adv}_{\Pi}^{\textit{IND-OCCA2}}(\mathcal{A}) = \left| \Pr\left[K \leftarrow \mathcal{K} : \mathcal{A}^{\mathcal{E}_K(\cdot,\cdot), \widehat{\mathcal{D}}(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathcal{O}^P(\cdot,\cdot), \widehat{\mathcal{O}}^P(\cdot,\cdot,\cdot)} \Rightarrow 1\right]\right|,$$

and

$$\mathbf{Adv}_{\Pi}^{\textit{IND-OCCA2}}(q, \ell, t) = \max_{\mathcal{A}} \left\{\mathbf{Adv}_{\Pi}^{\textit{IND-OCCA2}}(\mathcal{A})\right\}$$

as the maximum advantage over all IND-OCCA2 adversaries that run in time at most t, ask a total maximum of q queries to the encryption and decryption oracles, and whose total query length is at most ℓ blocks.

It is easy to see that IND-OCCA2 securtiy implies IND-OPRP security by using similar arguments as in the proof of Lemma 5.6. Therefore, the notion of IND-OCCA2 covers nonce-misuse restricted to data privacy. Now, we put all bits and pieces of this Chapter together and unite them to the following definition of the On-line Nonce- and Decryption-Misuse Attack (ONDMA) advantage:

Definition 5.10 (ONDMA Advantage). *Let* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *be an OAE scheme. Then, we define the* ONDMA advantage *of a* nonce-ignoring *adversary* $\mathcal{A}$ *as*

$$\mathbf{Adv}_{\Pi}^{\textit{ONDMA}}(\mathcal{A}) = \mathbf{Adv}_{\Pi}^{\textit{IND-OCCA2}}(\mathcal{A}) + \mathbf{Adv}_{\Pi}^{\textit{INT-CTXT}}(\mathcal{A}),$$

and

$$\mathbf{Adv}_{\Pi}^{\textit{ONDMA}}(q, \ell, t) = \max_{\mathcal{A}} \left\{\mathbf{Adv}_{\Pi}^{\textit{ONDMA}}(\mathcal{A})\right\}$$

as the maximum advantage over all ONDMA adversaries that run in time at most t, ask a total maximum of q queries to the encryption and decryption oracles, and whose total query length is at most ℓ blocks.

Note that any random OPerm which is only queried once cannot be distinguished from a random function since it is impossible to exploit the common-prefix characteristic of an on-line permutation. Thus, for any *nonce-respecting* adversary, it is impossible to distinguish the IND-CPA from the IND-OPRP setting. This implies that we can call an ONDMA-secure OAE scheme robust.

Part II

Design, Analysis, and Usage of Authenticated Encryption Schemes

6

Robustness of Authenticated Encryption Schemes

> To invent, you need a good imagination and a pile of junk.
>
> Thomas A. Edison

6.1. Nonce-Misuse Resistance

In this section we analyze the nonce-misuse resistance of existing OAE schemes. Note that none of them claims nonce-misuse resistance, and efficient misuse attack does not automatically invalidate the security of those schemes. Nevertheless, ensuring that an implemented authenticated encryption scheme is resistant against misuse attacks is a difficult task for implementors, especially when the software is running in environments like inside a virtual machine.

A summary of our analysis including a brief discussion is given at the end of this chapter.

6.1.1. Generic Attacks

In the following we introduce two generic attack patterns on which the majority of the nonce-misuse attacks are based on.

Repeated-Keystream Attack Pattern. Assume that the encryption routine $\mathcal{E}$ of an on-line authenticated encryption scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ generates a keystream $S = F_K(N)$ of length $|M|$, *i.e.,* $|S| = |M|$, depending on a secret key K and a nonce

N. The ciphertext of a message M is typically computed by $C = S \oplus M$, where S is generated by applying a block cipher in counter mode [30, 147, 164]. Assume that $\mathcal{A}$ is a nonce-ignoring IND-OPRP adversary, with access to an encryption oracle, that tries to distinguish real from random by comparing the difference of two single-block messages with that of their ciphertexts. It easy to see that $\mathcal{A}$'s advantage is almost $1 - 2^{-n}$. A formal definition of $\mathcal{A}$ is given in Algorithm 4.

Algorithm 4 Repeated-Keystream Adversary

$(C, T) \leftarrow \mathcal{O}(M, N)$ $\{$ first encryption query with $(N, M) \xleftarrow{\$} \{0,1\}^v \times \{0,1\}^n)$ $\}$
$(C', T') \leftarrow \mathcal{O}(M', N)$ $\{$ second encryption query with $(M' \xleftarrow{\$} \{0,1\}^n \setminus M)$ $\}$
return $(M \oplus M' = C \oplus C')$

Linear-Tag Attack Pattern. Common stateful authenticated encryption schemes such as Galois/Counter Mode (GCM) [164] or Counter with CBC-MAC (CCM) [85], apply the Encrypt-then-Mac paradigm (cf. Section 4.2), *i.e.,* they compute a tag T by:

$$T = F_K(N) \oplus G_K(M),$$

where N is the nonce, M is the plaintext, F_K is a key-dependent function and G_K is a key-dependent permutation with $C = G_K(M)$. Suppose $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is such a stateful scheme. This enables an efficient nonce-ignoring adversary $\mathcal{A}$ to mount an INT-CTXT attack with an advantage of 1 by solving a simple linear equation system. A formal definition of $\mathcal{A}$ is given in Algorithm 5.

Algorithm 5 Linear-Tag Adversary

$(C, T) \leftarrow \mathcal{E}(N, M)$ $\{$ first encryption query with $(N, M) \xleftarrow{\$} \{0,1\}^v \times \{0,1\}^n)$ $\}$
$(C', T') \leftarrow \mathcal{E}(N', M')$ $\{$ second encryption query with $(N' \neq N) \wedge (M' \neq M)$ $\}$
$(C'', T'') \leftarrow \mathcal{E}(N, M')$ $\{$ third encryption query with $T'' = F_K(N) \oplus G_K(M')$ $\}$
return $\mathcal{D}(N', C, T \oplus T' \oplus T'')$ $\{$ forgery since $T \oplus T' \oplus T'' = F_K(N') \oplus G_K(M)$ $\}$

6.1.2. Misuse Attacks against Previously Published Authenticated Encryption Schemes

CWC, GCM, CCM, EAX, and CHM. Usually, common two-pass OAE schemes, Carter–Wegman Counter (CWC) [147], GCM [164], CCM [85], EAX [30], and CENC with Hash-based MAC (CHM) [124], use the Counter (CTR) mode as their underlying encryption operation [30, 85, 124, 164]. These schemes are vulnerable to repeated-keystream attacks. Four of them, CHM, CWC, GCM, and EAX, are designed according to the Encrypt-then-Mac paradigm, and are thus vulnerable to the linear-tag attacks. The designers of CCM follow the Mac-then-Encrypt approach, which seems to defend against linear-tag attacks. Though, forgery attacks against CCM were presented in [106].

RPC. Related Plaintext Chaining (RPC) [55] combines two coommen modes of operations: CTR and Electronic Codebook (ECB). Given an n-bit block cipher E under a key K and a v-bit nonce N, RPC takes an $(n-v)$-bit plaintext block M_i and computes the ciphertext block

$$C_i \leftarrow E_K(M_i \mid\mid (N+i) \bmod 2^v).$$

Authentication is performed locally for each ciphertext block: During decryption, RPC computes $(M_i \mid\mid X_i) = E_K^{-1}(C_i)$ and accepts M_i as authentic *iff*

$$X_i = (N+i) \bmod 2^v.$$

In the nonce-misuse setting, the same sequence of counter values is used for different messages. This makes it easy to attack the privacy – especially when encrypting messages of $m \cdot (n-v)$-bit blocks. Then, RPC degrades into m independent electronic code books.

More precisely, any adversary that obtains two authentic m-block ciphertexts, $(C_1^0, \ldots, C_m^0)$ and $(C_1^1, \ldots, C_m^1)$ with the same nonce N, can forge 2^m new authentic ciphertexs $(C_1^{\sigma(1)}, \ldots, C_m^{\sigma(m)})$ with $\sigma(i) \in \{0,1\}$ since authenticity is verified locally for each $C_i^{\sigma(i)}$.

CCFB. Similar to RPC, the Counter-CipherFeedback (CCFB) mode [156] is a combination of CTR and Cipher Feedback (CFB) mode. Given an $(n-a)$-bit nonce N and $(n-a)$-bit plaintext blocks $M_1 \ldots, M_m$ CCFB works as follows:

Initial Step: $C_0 \leftarrow N$

Encryption: For $i \in \{1, \ldots, m\}$: $(Z_i \mid\mid T_i) \leftarrow E_K(C_{i-1} \mid\mid i), \quad C_i \leftarrow M_i \oplus Z_i$

Authentication: $(*, T_{m+1}) \leftarrow E_K(C_m \mid\mid m+2), \quad T \leftarrow \bigoplus_{1}^{m+1} T_i$

Note that the first ciphertext block C_1 is essentially the plain encryption of M_1 in CTR Mode. Thus, a variant of the repeated-keystream attack (cf. Algorithm 4) is also applicable to CCFB. Moreover, the following variant of the linear-tag attack pattern (cf. Algorithm 5) applies to CCFB:

1. Encrypt the plaintext M_1 under N to (C_1, T).
2. Encrypt the plaintext $M_1' \neq M_1$ under $N' \neq N$ to (C_1', T').
3. Set $M_1'' \leftarrow M_1' \oplus C_1' \oplus C_1$. Encrypt M_1'' under N' to (C_1'', T''). Observe $C_1'' = C_1$.
4. The triple $(N, C_1', T \oplus T' \oplus T'')$ is a valid forgery.

IAPM, OCB1–3, and TAE. Given a nonce N and a secret key K, Integrity Aware Parallelizable Mode (IAPM) [130] encrypts a message $M = (M_1, \ldots, M_m)$ to a ciphertext $C = (C_1, \ldots C_m)$ and an authentication tag T as follows.

Initial Step: Generate $m + 2$ pseudorandom values $s_0, s_1, \ldots, s_{m+1}$ depending on N and K, but not on the message M.

Encryption: For $i \in \{1, \ldots, m\}$: $C_i \leftarrow E_K(M_i \oplus s_i) \oplus s_i$.

Authentication: $T \leftarrow E_K(s_{m+1} \oplus \widehat{M}) \oplus s_0$ with $\widehat{M} = \bigoplus_{i=1}^{m} M_i$.

When encrypting messages of m blocks, IAPM behaves like a set of m independent instances of the common ECB mode. Hence, IAPM is vulnerable to the same forgery attack as the one that applies to RPC. An adversary who can encrypt two messages M and M' under the same nonce only has to take care that they produce the same checksum $\widehat{M} = \widehat{M'}$ to create a valid forgery since than we have

$$T = E_K(s_{m+1} \oplus \widehat{M}) \oplus s_0 = E_K(s_{m+1} \oplus \widehat{M'}) \oplus s_0.$$

All three versions of the Offset Codebook (OCB) mode family (*i.e.,* OCB1 [208], OCB2 [205], and OCB3 [148]) and Tweakable Authenticated Encryption (TAE) [153] work similar to IAPM, and thus the same attack also applies to them.

IACBC. Given a nonce N and a secret key K, Integrity Aware Cipher Block Chaining Mode (IACBC) [130] encrypts the message $M = (M_0, \ldots, M_m)$ to $C = (C_1, \ldots, C_m)$ and an authentication tag T as follows:

Initial Step: Generate $m+1$ values $s_0, s_1, \ldots s_m$ depending on N and K, but not on the Message M and compute $C_0 \leftarrow x_0 \leftarrow E_K(N)$.

Encryption: For $i \in \{1, \ldots, m\}$: $x_i \leftarrow E_K(M_i \oplus x_{i-1})$, $C_i \leftarrow x_i \oplus s_i$.

Authentication: $T \leftarrow E_K(x_m \oplus \widehat{M}) \oplus s_0$ with $\widehat{M} = \bigoplus_{i=1}^{m} M_i$.

The following attack distinguishes IACBC from a random on-line permutation and also provides an existential forgery.

1. Encrypt M_1 under the nonce 0 to (C_0, C_1, T).
2. Encrypt the nonce 0 under $M = (C_0, C_0, C_0, C_0)$ to (C', T').
3. Set $C'' = (C_0, C'_1, C'_2, T')$
   Note that (C'', T) is a valid encryption of $M = (C_0, C_0)$ since it holds that $C_0 \oplus C_0 = C_0 \oplus C_0 \oplus C_0 \oplus C_0$.

XCBC-XOR. Given a nonce N and secret keys K and K', eXtended Ciphertext Block Chaining with XOR (XCBC-XOR) [113] encrypts a message $M = (M_1, \ldots, M_m)$ to a ciphertext $C = (C_1, \ldots, C_m)$ and an authentication tag T as follows:

Initial Step: Generate $m+1$ values $s_0, \ldots, s_m$ depending on N and K', but not on the plaintext $(M_1, \ldots, M_m)$.

Encryption:

1. $C_0 \leftarrow E_K(N)$; $x_0 \leftarrow E_{K'}(N)$;
2. For $i \in \{1, \ldots, m\}$: $x_i \leftarrow E_K(M_i \oplus x_{i-1})$, $C_i \leftarrow (x_i + s_i) \bmod 2^n$.

Authentication: $T \leftarrow E_K(\widehat{M} \oplus x_m) + s_0 \pmod{2^n}$ with $\widehat{M} = x_0 \cdot \bigoplus_{i=1}^{m} M_i$.

The following attack provides an existential forgery:

1. Encrypt the message $(0^n, 0^n, 0^n)$ under the nonce N to (C_0, C_1, C_2, C_3, T).
2. Then, $(C_0, C_1, C_2, T' = C_3)$ is a valid forgery.

The best IND-PRP attack we found for XCBC-XOR has a workload of $O(2^{n/4})$ instead of $O(1)$. Note that for this reuse-nonce chosen-plaintext attack, we ignore the authentication tag.

1. Generate $2^{n/4}$ encryptions of messages M_1^α under the same nonce N to C_1. Let denote C_1^α the encryption of the α-th message M_1^α with $\alpha = 1, \ldots, n/4$.

   Statistically, we can expect one pair (M_1^i, M_1^j) with $i \neq j$ such that the least significant $n/2$ bits of C_1^i and C_1^j are equal.

2. Generate $2^{n/4}$ encryptions of messages (M_1^i, M_2^α) and (M_1^j, M_2^α) under N, where the $n/2$ least significant bits of all message blocks M_2^α are equal.

   Statistically, we can expect one pair (M_2^k, M_2^ℓ) with $k \neq \ell$ such that $C_2^k = C_2^\ell$ holds.

3. Choose an arbitrary M_3. Encrypt (M_1^i, M_2^k, M_3) and (M_1^j, M_2^ℓ, M_3) under N to $(C_1^i, C_2^k, C_3^{i,k})$ and $(C_1^j, C_2^\ell, C_3^{j,\ell})$.

   Observe $C_3^{i,k} = C_3^{j,\ell}$.

6.2. Decryption-Misuse Resistance

In this section we analyze the decryption-misuse resistance of previously published authenticated encryption schemes. Similar to the nonce-misuse setting, none of them claims decryption-misuse resistance. Hence, our presented attacks do not invalidate their claimed security.

BTM, CCM, CHM, CWC, EAX, GCM, HBS, and SIV These schemes use the CTR mode as their underlying encryption operation. Thus, they are vulnerable to *decryption-misuse* attacks. Assume (C, T) is the encryption of M. Then, we can determine the would-be plaintext M' of the unauthentic tuple (C', T) since – for the same counter value – it must hold that $C \oplus C' = M \oplus M'$. This observation can be exploited by an efficient IND-CCA adversary.

IAPM, OCB1–3, RPC, and TAE. These OAE schemes behave like the ECB mode and are therefore vulnerable in the decryption-misuse setting. Assume two unauthentic ciphertexts that only differ in the i-th block. The decryptions of those produce two messages that also differ only in the i-th block. Thus, these schemes are not IND-CCA-secure.

CCFB. This mode generates a keystream based on the previous ciphertext block and a counter:

$$C_i = E_K(C_{i-1} \mid\mid i) \oplus M_i.$$

Thus, it must hold that $C_i \oplus C'_i = M_i \oplus M'_i$ for $C_{i-1} = C'_{i-1}$. This observation can be exploited by an IND-CCA adversary.

IACBC and XCBC-XOR. Both schemes are based on the concept of the Cipher Block Chaining (CBC) mode, *i.e.,* the i-th message block M_i depends only on the ciphertext blocks C_{i-1} and C_i. Therefore, the decryption of the two nonce-ciphertext-tag triples $(N, C_1 \mid\mid C_2 \mid\mid C_3, T)$ and $(N, C'_1 \mid\mid C_2 \mid\mid C_3, T)$ with $C_1 \neq C'_1$ produces two messages that share the same final message block. In general, schemes based on the CBC approach are not IND-CCA secure in the decryption-misuse setting.

COPA. In 2013, Andreeva *et al.* introduced COPA [7], a nonce-misuse resistant and parallelizable OAE scheme inspired by McOE (cf. Chapter 8). It combines the XOR-Encrypt-XOR (XEX) encryption with the Encrypt-Mix-Encrypt (EME) approach. Therefore, two block-cipher calls are needed to process a single message block.

Initial Step: $Y_0 \leftarrow E_K(N), L \leftarrow E_K(0), \Delta_0 = 3L$, and $\Delta_1 = 2L$.

Encryption: For $i \in \{1, \ldots, m\}$: $X_i \leftarrow E_K(M_i \oplus 2^{i-1}\Delta_0),\ Y_i \leftarrow X_i \oplus Y_{i-1}$,

$$C_i \leftarrow E_K(Y_i) \oplus 2^{i-1}\Delta_1.$$

Tag Generation: $X_{m+1} \leftarrow E_K(\widehat{M} \oplus 2^{m-1}3^2L),\ Y_{m+1} \leftarrow X_{m+1} \oplus Y_m$ with $\widehat{M} = \bigoplus_{i=1}^{m} M_i$, and

$$T \leftarrow E_K(Y_{m+1}) \oplus 2^{m-1}7L.$$

Let $M_a \neq M_b$ be two distinct message blocks. Then, we define $Y_a = E_K(M_a \oplus \Delta_0) \oplus L \oplus Y_0$ and $Y_b = E_K(M_b \oplus \Delta_0) \oplus L \oplus Y_0$.

1. Encrypt (N, M_a, M_c) to $(C_a, C_{(a,c)})$ with

$$X_c = E_K(M_c \oplus 2\Delta_0),$$
$$Y_{(a,c)} = Y_a \oplus X_c, \text{ and}$$
$$C_{(a,c)} = E_K(Y_{(a,c)}) \oplus 2\Delta_1.$$

2. Encrypt (N, M_b, M_c) to $(C_b, C_{(b,c)})$ with

$$X_c = E_K(M_c \oplus 2\Delta_0),$$
$$Y_{(b,c)} = Y_b \oplus X_c, \text{ and}$$
$$C_{(b,c)} = E_K(Y_{(b,c)}) \oplus 2\Delta_1.$$

3. Decrypt $(C_a, C_{(b,c)})$ to $(M_a, M_{(a,bc)})$. It applies that

$$Y_{(b,c)} = E_K^{-1}(C_{(b,c)} \oplus 2\Delta_1), \text{ and}$$
$$X_{(a,bc)} = Y_{(b,c)} \oplus Y_a = Y_b \oplus X_c \oplus Y_a.$$

4. Decrypt $(C_b, C_{(a,c)})$ to $(M_b, M_{(b,ac)})$. It applies that

$$Y_{(a,c)} = E_K^{-1}(C_{(a,c)} \oplus 2\Delta_1), \text{ and}$$
$$X_{(b,ac)} = Y_{(a,c)} \oplus Y_b = Y_a \oplus X_c \oplus Y_b = X_{(a,bc)}.$$

From $X_{(a,bc)} = X_{(b,ac)}$ follows that $M_{(a,bc)} = M_{(b,ac)}$. This observation can be used to distinguish COPA from a random OPERM with probability $1 - 2^{-n}$.

In the following we extend the IND-OPRP attack on COPA into an INT-CTXT attack. The first three queries of this attack are identical to those in the IND-OPRP attack. With their help, we can form a collision in the chaining values for two messages $(M_a, M_{(a,bc)})$ and $(M_b, M_{(a,bc)})$ since it must hold that $X_{(a,bc)} = X_{(b,ac)}$. Thus, we can apply a common length-extension attack to create an existential forgery:

4. Encrypt $(M_a, M_{(a,bc)}, M_d)$ to $(C_a, C_{(a,bc)}, C_d, T)$, where T is the authentication tag.

5. Then, craft the existential forgery $(C_a, C_{(b,ac)}, C_d, T)$.

6.3. Results Summary

As it turned out, we actually found nonce and decryption-misuse attacks for *all* previously published OAE schemes. Table 6.1 summarizes our results. For over a decade, it has been common knowledge how to design a secure, an efficient, and stateless MAC [17, 125]. Therefore, it is surprising that none of the analyzed online schemes provide integrity protection. Only CCM offers some weak integrity protection against nonce-ignoring adversaries, but it does not provide any privacy protection.

Outlook. In the following we present two novel families of OAE scheme: COFFE and McOE. Their designs were inspired by the results of our robustness studies that we summarized in this chapter. The former scheme is the first OAE scheme based on a hash function. It suits very well for resource-restricted devices and offers INT-CTXT security even in the nonce-misuse scenario. McOE is the first published robust OAE scheme.

Scheme		Nonce Misuse		Decryption Misuse
		privacy	integrity	
on-line				
CCFB	[156]	$O(1)$	$O(1)$	$O(1)$
CHM	[124]	$O(1)$	$O(1)$	$O(1)$
COPA	[7]	N/A	N/A	$O(1)$
CWC	[147]	$O(1)$	$O(1)$	$O(1)$
EAX	[30]	$O(1)$	$O(1)$	$O(1)$
GCM	[164]	$O(1)$	$O(1)$	$O(1)$
IACBC	[130]	$O(1)$	$O(1)$	$O(1)$
IAPM	[130]	$O(1)$	$O(1)$	$O(1)$
OCB1	[208]	$O(1)$	$O(1)$	$O(1)$
OCB2	[205]	$O(1)$	$O(1)$	$O(1)$
OCB3	[148]	$O(1)$	$O(1)$	$O(1)$
RPC	[55]	$O(1)$	$O(1)$	$O(1)$
TAE	[153]	$O(1)$	$O(1)$	$O(1)$
XCBC-XOR	[113]	$O(2^{n/4})$	$O(1)$	$O(1)$
off-line				
BTM	[126]	N/A	N/A	$O(1)$
CCM	[85]	$O(1)$	$\ll 2^{(n/2)}$ [106]	$O(1)$
HBS	[127]	N/A	N/A	$O(1)$
SIV	[209]	N/A	N/A	$O(1)$

Table 6.1.: Workloads of our robustness studies on previously published authenticated encryption schemes. Almost all attacks achieve an advantage close to 1. The workloads cover the computational effort, the amount of required memory, as well as the time complexity. Note that we classify CCM as off-line because the message-encryption process requires prior knowledge of the message length.

7 COFFE: Ciphertext Output Feedback Faithful Encryption

> Insight is the first condition of Art.
>
> George Henry Lewes

In this Chapter we aim to provide authenticated encryption in constrained implementation environments where communication security is required, such as devices connected to the Internet of Things (IoT). Designing a sound authenticated encryption scheme is in fact challenging, but designing sound authenticated encryption scheme for constraint environments is a science of its own. Typically, restricted IoT devices have (very) limited computational power and no direct hardware support for any cryptographic primitives, so that all cryptography must be implemented in software. There is only limited memory available to hold executable object code on these processors, so it is imperative to provide the needed cryptographic services in the most compact way possible. One way to achieve this compactness is through the careful implementation of cryptographic primitives. However, it is also possible to facilitate compactness for an overall system by minimizing the number of primitives that must be included in an implementation. In this work we present a design for an on-line authenticated encryption (AE) scheme suitable for restricted devices using a standardized or soon-to-be standardized hash function, *e.g.,* SHA-1 [183], SHA-2 [184], or SHA-3 [34]. Implementations of this scheme can omit a block cipher mode of operation; this is a useful approach since the code size for the block cipher is typically greater than that of the hash function, and hash functions are used in public key cryptography as well.

We focus on the challenge of providing an authenticated encryption scheme that is easily accessible to developers. To provide this accessibility, we take the approach of defining a hash function mode of operation. That is, our AE scheme uses a cryptographic hash function as its only primitive, and does not require direct access to any hash function internals such as the compression function. We chose this approach based on feedback from the practice community. Hash function implementations are widely available, but these implementations do not provide interfaces to the compression function.

Note that to provide data privacy and data integrity, we transform the given hash function into a keyed hash function (PRF. On systems using restricted devices, due to the limited resources, it is desirable to minimize the cost of the code and the circuits for encrypting a message block [11], *i.e.,* keep the size of the cryptographic footprint small. This means that we want only one costly operation per message block. We denote a scheme satisfying this property as a Rate-1 scheme. For example, the GCM authenticated encryption mode is not a Rate-1 AE scheme, since it needs not only one block cipher call per message block, but also an additional galois field multiplication per message block, rendering GCM to be a Rate-2 AE scheme. Another example would be a Feistel-based scheme which requires at least three or four block cipher calls rendering such a scheme to a low-performance Rate-3 or Rate-4 scheme.

Since the implementation of an encryption scheme can be error prone (*e.g.,* [51, 122, 146, 214, 239] it would be desirable to provide a second line of defense to minimize the security fallout. A further preferable goal for our construction is to provide built-in resistance against side-channel attacks. Actually, the overlaying protocols, using an AE scheme, are responsible to provide this goal in an adequate form, *e.g.,* TLS [78] and IPsec [120, 137] generate a new key for each session minimizing the number of measurements which can be done on the secret key. Obviously, an adversary can do a certain amount of measurements (depending on the size of the message) on the session key, but revealing the session key only compromises security for this specific encryption/decryption/authentication. Note that it does not compromise the currently used secret key. But, nevertheless, we provide side-channel resistance even if a protocol may fail to provide this kind of security.

We started our research by analyzing existing authenticated encryption schemes, where the block cipher within these schemes can be easily replaced by a keyed hash function. Unfortunately, none of those fulfill our requirements (see Table 7.1). As one can see, SpongeWrap [38] seems to be a very promising candidate, since it only lacks of built-in side-channel resistance. But, it belongs to the class of compression function based AE schemes, which yields to the fact that the internal used compression function can be seen as the real primitive to be used both for hashing and for authenti-

Scheme	**On-line**	**Side-Channel Res.**	**Rate-1**
COFFE (**this work**)	✓	✓	✓
CHM [124]	✓	X	X
CWC [147]	✓	X	X
EAX [30]	✓	X	X
GCM [164]	✓	X	X
Generic Composition [22]	✓	X	X
HBS [127]	X	X	X
SIV [209]	X	X	X

Table 7.1.: Comparison of selected authenticated encryption schemes that can be instantiated with a hash function.

cated encryption. This is basically not a technical problem, but, while cryptographers know what is meant by *the internal compression function*, typical standards, such as the SHA-2 standard [184], do not formally define it. So, without an explicit specification of a "new" cryptographic primitive, engineers (non-cryptographers) would not be likely to properly implement the authenticated encryption scheme. Also, while on many constrained devices "jumping" to the address of the internal compression function may be easy, this may be not the case for all such devices. In fact, we did consider this approach at the beginning of our research. It would even allow us to design a more efficient AE scheme than the one we actually propose. But, due to the reasons discussed here, we made a decision against a purely compression function based AE scheme in favour of a hash function based AE scheme.

Outlook. In Section 7.1 we give a formal specification of Ciphertext Output Feedback Faithful Encryption (COFFE). In Section 7.2 we introduce a practical instantiation based on SHA-224. In Section 7.3 we show that COFFE is secure in the standard PRF model and in addition, it provides INT-CTXT security in the nonce-misuse setting. Finally, Section 7.4 summarize our contribution.

7.1. Specification

Ciphertext Output Feedback Faithful Encryption (COFFE) is inspired by the CFB and Output Feedback (OFB) modes of operation [84]. It uses the chaining value V_i

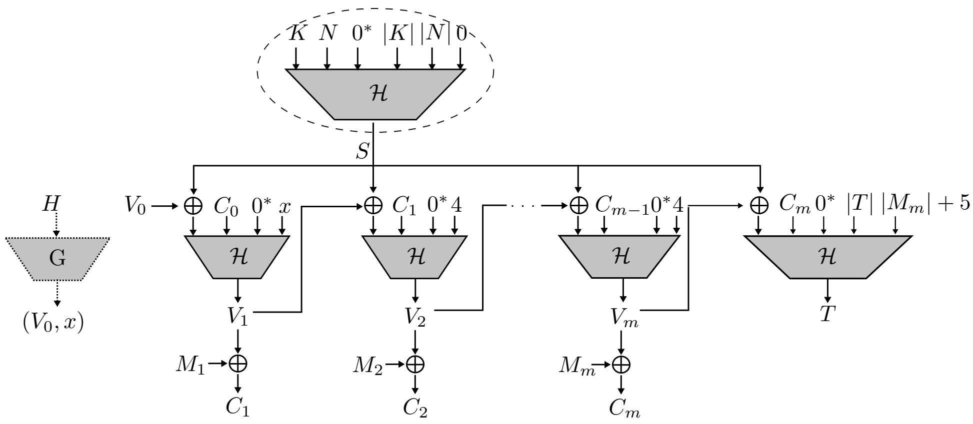

Figure 7.1.: Illustration of the encryption and authentication process of COFFE, where C_0 denotes the first $\alpha/4$ post-decimal hex digits of the number π.

and the previous ciphertext block C_{i-1} as inputs for the computation of the subsequent ciphertext block C_i (see Figure 7.1). The integrity of the ciphertext does not depend on the uniqueness of a nonce, but only on the security of the underlying n-bit hash function $\mathcal{H}$. The definition of COFFE is given in Algorithm 6; both functions **Encrypt** and **Decrypt** consist of the following four steps:

Step 1: Session Key Generation. COFFE is following the domain separation approach. Domain 0 is used to generate the session key S (*short-term key*) which is derived from the secret key K (*long-term key*) and the nonce N as shown in Lines 10 and 20 of Algorithm 6. Note that the lengths of the key $|K|$ and nonce $|N|$ are encoded as single- or two-byte values. The actual encoding depends on the size of the key. Nevertheless, the domain always describes the least significant byte of the input. For practical applications, we recommend to use a key size of n bits, where n denots the output size of the underlying hash function $\mathcal{H}$. The session-key generation provides a built-in side-channel resistance since hash function do not have key schedules. Thus, COFFE can update K for every new message without additional performance costs.

The term '0^*' – which is used in each call to the hash function $\mathcal{H}$ – denotes a zero-padding, where the number of zeros depends on (1) the input size of the underlying compression function and (2) the internal message padding. Thus, it is always chosen such that one needs only one compression function call for one hash function invocation, which complies with our Rate-1 design goal. Therefore, we consider a

Algorithm 6 COFFE

Encrypt(N, H, M)

$S \leftarrow \mathcal{H}(K \mid\mid N \mid\mid 0^* \mid\mid |K| \mid\mid |N| \mid\mid \mathtt{0})$
$(x, V_0) \leftarrow$ **ProcessHeader**(H)
$(C, V_m) \leftarrow$ **ProcessMessage**(S, V_0, M, x)
$T \leftarrow \mathcal{H}(S \oplus V_m \mid\mid C_m \mid\mid 0^* \mid\mid L_T \mid\mid |M_m| + 5)$
return (C, T)

Decrypt(N, H, C, T)

$S \leftarrow \mathcal{H}(K \mid\mid N \mid\mid 0^* \mid\mid |K| \mid\mid |N| \mid\mid \mathtt{0})$
$(x, V_0) \leftarrow$ **ProcessHeader**(H)
$(M, V_m) \leftarrow$ **ProcessCiphertext**(S, V_0, C, x)
$T' \leftarrow \mathcal{H}(S \oplus V_m \mid\mid C_m \mid\mid 0^* \mid\mid L_T \mid\mid |C_m| + 5)$
if $T \neq T'$ **then**
    $M \leftarrow \perp$
end if
return M

constant σ-bit input for $\mathcal{H}$, producing an n-bit output with $\sigma \geq n$.

Step 2: Header Processing. In this step we describe the processing of the associated data H, which can be of arbitrary length. For the sake of optimization, we invoke the hash function $\mathcal{H}$ only if indispensable, *i.e.,* when the header is larger than the output length of $\mathcal{H}$. This allows us to process messages with a small header, *e.g.,* IP packets, much faster by simply applying the 10^*-padding. The domain x indicates the length of the original header before the processing. A formal description of our length-dependent header processing is given next:

$$(x, V_0) \leftarrow G(H) := \begin{cases} \mathtt{1}, H \mid\mid 10^* & \text{if } |H| < n, \\ \mathtt{2}, H & \text{if } |H| = n, \\ \mathtt{3}, \mathcal{H}(H) & \text{else.} \end{cases}$$

The goal of G is to achieve pair-wise distinct tuples (x, V_0) for pair-wise distinct values H and H'. Under the assumption that there is no collision for the hash function G, we have

$$H \neq H' \implies (x, V_0) = G(H) \neq G(H') = (x', V_0{}'),$$

not necessarily meaning that $x \neq x'$.

Algorithm 7 ProcessMessage/ProcessCiphertext

ProcessMessage(S, V_0, M, x)	**ProcessCiphertext**(S, V_0, C, x)
$C_0 \leftarrow$ **ToHex**(π)	$C_0 \leftarrow$ **ToHex**(π)
$V_1 \leftarrow \mathcal{H}(S \oplus V_0 \mid\mid C_0 \mid\mid 0^* \mid\mid x)$	$V_1 \leftarrow \mathcal{H}(S \oplus V_0 \mid\mid C_0 \mid\mid 0^* \mid\mid x)$
$C_1 \leftarrow M_1 \oplus V_1$	$C_1 \leftarrow C_1 \oplus V_1$
for $i = 2, \ldots, m$ **do**	**for** $i = 2, \ldots, m$ **do**
$V_i \leftarrow \mathcal{H}(S \oplus V_{i-1} \mid\mid C_{i-1} \mid\mid 0^* \mid\mid 4)$	$V_i \leftarrow \mathcal{H}(S \oplus V_{i-1} \mid\mid C_{i-1} \mid\mid 0^* \mid\mid 4)$
$C_i \leftarrow M_i \oplus V_i$	$M_i \leftarrow C_i \oplus V_i$
end for	**end for**
return C	**return** M

Step 3: Plaintext/Ciphertext Processing. COFFE is generating a keystream for either encryption or decryption. Since our scheme is designed to comply with the requirements of the use of standardized building blocks, it works with hash functions like SHA-1 and SHA-2. Thus, the input of the compression function is usually limited to less than $2n$ bits, due to the message padding. Note that the n-bit session key S and the domain separation value are mandatory inputs and hence, we have only less then n bits remaining for the message input. To provide adequate security against forgery attacks, we need to additionally process two out of three of the following values: keystream block V_{i-1}, message block M_{i-1}, and ciphertext block C_{i-1}. More precisely, if we only use V_{i-1} in the next iteration step, the tag would become message-independent, *i.e.,* the tag would not provide any integrity at all. Furthermore, if we use only C_{i-1} or M_{i-1}, omitting V_{i-1}, the tag value would only depend on the last ciphertext or plaintext block, respectively. We decided to use the inputs to $\mathcal{H}$ in the following manner:

- n-bit value $S \oplus V_{i-1}$
- δ-bit domain-separation value
- $(\alpha < n - \delta)$-bit ciphertext block C_{i-1}.

Our approach puts the hash function under a lot of stress since it violates the PRF-independency assumption. Thus, we require $\mathcal{H}$ to be indistinguishable from a PRF in the related-key model. More precisely, an adversary has partial control over the key-input to $\mathcal{H}$, resulting in a chance to produce a collision $S \oplus V_{i-1} = S' \oplus V_{j-1}$ for two distinct keys $S \neq S'$. Our security analysis in Section 7.3 shows that our approach still satisfies the birthday-bound security.

Let $M = M_1, \ldots, M_m$ denote the message, where $m = \lceil |M|/\alpha \rceil$ is the number of message blocks processed. Here, all but the last blocks of M and C are of size α bits. The final blocks of M and C consist of at most α bit. Then, the encryption and decryption process of COFFE is defined in Algorithm 7, where $\mathbf{ToHex}(\pi)$ (see Lines 10 and 20) outputs the first $\alpha/4$ post-decimal numbers of π interpreted as hex values ($C_0 = 0x1415926\ldots$).

Step 4: Tag Generation. In the final step we derive the authentication tag from the final chaining value V_m and the final ciphertext block C_m as shown in Lines 13 and 23 of Algorithm 6. Note that the length of the tag is constrained by the output size of $\mathcal{H}$, *e.g.,* at most n bits. The last domain allows a user to authenticate the header without any message to encrypt. Thus, $|M_m|$ can become zero, but for $\mathcal{H}$, $|M_m| + 5$ is always in the range $[5, \ldots, n + 5]$.

7.2. COFFE-SHA-224 – A Practical Instantiation

In this section we discuss a practical instantiation of COFFE using SHA-224 as the underlying hash function – called COFFE-SHA-224. First, we justify our usage of SHA-224 over SHA-256.

Hash Function Choice. For the practical instantiation of COFFE, we searched for a standardized hash function which is suitable for restricted devices, where the usual size of a register is at most 32 bits. Thus, we made our choice in favour of a 32-bit-optimized hash function, which renders SHA-224 and SHA-256 reasonable candidates. Both SHA-224 and SHA-256 share the same compression function $f : \{0,1\}^{256} \times \{0,1\}^{512} \rightarrow \{0,1\}^{256}$. It compresses a 256-bit chaining value and a 512-bit message block into a 256-bit output value. These two hash-function standards differ in two properties: 1) they use different initial values, and 2) SHA-224 truncates the output of the final compression function invocation while SHA-256 does not. Following the Merkle-Damgård paradigm [72, 171], SHA-224 and SHA-256 apply the secure 10*-padding followed by a 64-bit value encoding the message length. Thus, the maximum possible input size to fit our requirements would be $512 - 1 - 64 = 447$ bits. Due to the sake of simplification, we consider only byte-aligned values and we assume all values to be encoded as octet-strings. Thus, we can only process message blocks with a size up to only 440 bits, *i.e.,* 55 bytes. Using SHA-256 implies a 256-bit chaining value and thus, *only* 184 bits were left for the remaining input, including the domain separation byte and the previous ciphertext block. Furthermore, the tag generation step requires two additional input bytes – the length of the last message block β and the tag length $|T|$. Hence, we can process 160-bit message blocks. Since the size of the hash value of SHA-224 is reduced by 32 bits in comparison to the usage of SHA-256, we can process message blocks of 192 bits, which leads to an estimated performance speedup of about 20% in comparison to SHA-256. Furthermore, the 224-bit session key used in SHA-224 is sufficient to make practical attacks infeasible. This makes SHA-224 a logical choice for COFFE.

Parameter Choice. Here, we introduce a sound parameter choice for COFFE-SHA-224 depending on the used hash function SHA-224. The first step is to replace the function $\mathcal{H}$ from Algorithm 6 by SHA-224. This obviously leads to a size of 224 bits for the chaining values V_i. Based on our discussion above, we can process message blocks of up to 192 bits, *i.e.,* we need only one byte to encode the domain specifier for the tag generation ($|M_m| + 5 < 256$). On one hand, the internal state of COFFE is larger than those of other common published authenticated encryption schemes like GCM, OCB, or EAX, which usually support a block size of 128 bits. On the other hand, COFFE employs a slightly worse ratio between the block size and the size of the internal state. Nevertheless, due to the larger block size, the performance of COFFE is still reasonable, *i.e.,* approximately 85% of SHA-224. To ensure an adequate security, we set the default parameter of the size of the secret key to 224 bits. Therefore, we have up to 192 bits left for the nonce.

7.3. Security

This section describes the CCA3 security (cf. Definition 4.1) of COFFE considered under the reasonable assumption that the size of the secret key K can be larger or equal to the size of the session key S, *i.e.,* $|K| \geq |S|$. Therefore, at first we show the IND-CPA security of COFFE when considering a nonce-respecting adversary (cf. Definition 4.2), and then, we proof the INT-CTXT-security of COFFE against generalize the adversary by allowing it to reuse a nonce (cf. Definition 4.3).

Note that the length of the secret key K can differ from the length of the session key S. If this is the case, we can partition $\mathcal{H}$ into two keyed hashfunctions: $\mathcal{H}_1 : \{0,1\}^{|K|} \times \{0,1\}^* \to \{0,1\}^n$ for the generation of the session key and $\mathcal{H}_2 : \{0,1\}^{|S|} \times \{0,1\}^* \to \{0,1\}^n$ for the processing of the message and the generation of the authentication tag. Due to the domain separation, the partitioning of $\mathcal{H}$ is still valid if $|K| = |S|$, *i.e.,* the domain of the session key generation is always 0 and the domain of the message processing is always 4. In this section, for simplification, we define

$$\mathbf{Adv}_{\mathcal{H}_*}^{PRF}(q+\ell, O(t)) = \max\left\{\mathbf{Adv}_{\mathcal{H}_1}^{PRF}(q, O(t)), \mathbf{Adv}_{\mathcal{H}_2}^{PRF\text{-}RKA}(q+\ell, O(t))\right\}.$$

Theorem 7.1 (CCA3 Security of COFFE). *Suppose* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *is the COFFE scheme as defined in Algorithm 6,* i.e., $\mathcal{K}$ *is the key derivation function,* $\mathcal{E} =$ **EncryptAndAuthenticate** *and* $\mathcal{D} =$ **DecryptAndVerify**. *Then,*

$$\begin{aligned}\mathbf{Adv}_{\Pi}^{CCA3}(q, \ell, t) &\leq \frac{5\ell^2 + 3q^2}{2^n} + 2 \cdot \mathbf{Adv}_{\mathcal{H}_*}^{PRF}(q+\ell, O(t)) \\ &+ \frac{2\ell^2 + 3q^2}{2^n} + \frac{q}{2^{|T|}} + 2 \cdot \mathbf{Adv}_{\mathcal{H}_*}^{PRF}(q+\ell, O(t)) \\ &\leq \frac{7\ell^2 + 6q^2}{2^n} + \frac{q}{2^{|T|}} + 4 \cdot \mathbf{Adv}_{\mathcal{H}_*}^{PRF}(q+\ell, O(t)).\end{aligned}$$

Proof. The proof follows from Lemma 7.2 and Lemma 7.3. ∎

Lemma 7.2 (IND-CPA-security of COFFE). *Let* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *denote the COFFE scheme as defined in Algorithm 6. Then,*

$$\begin{aligned}\mathbf{Adv}_{\Pi}^{IND\text{-}CPA}(q+\ell, t) &\leq \frac{(\ell+q)^2 + 2\ell^2 + 2q^2}{2^n} + 2 \cdot \mathbf{Adv}_{\mathcal{H}_*}^{PRF}(q+\ell, O(t)) \\ &\leq \frac{5\ell^2 + 3q^2}{2^n} + 2 \cdot 2 \cdot \mathbf{Adv}_{\mathcal{H}_*}^{PRF}(q+\ell, O(t))\end{aligned}$$

Proof. This proof is using common game-playing arguments. At first, we replace the function $\mathcal{H}_1$ by a random n-bit function. The advantage therefore can be upper bounded by

$$\mathbf{Adv}_{\mathcal{H}_1}^{PRF}(q, O(t)).$$

Furthermore, we can also replace the function $\mathcal{H}_2$ by a random n-bit function since the adversary has partial control over the key $S \oplus V_i$. The advantage of this can be upper bounded by the PRF-RKA Advantage which is defined similar to the PRP-RKA Advantage (see Definition 3.3). Thus, we have

$$\mathbf{Adv}_{\mathcal{H}_2}^{PRF\text{-}RKA}(q+\ell, O(t)).$$

In the following we always consider the full output length n of the tag generation step, *i.e.,* even if $|T|$ is smaller than n, we skip the truncation step for the proof. This is valid since showing IND-CPA security for the tag generation step without truncation implies IND-CPA security for the tag generation with truncation. From each adversary $\mathcal{A}$ with an advantage of ϵ attacking the truncated version we can construct an adversary $\mathcal{A}'$ with the same advantage ϵ attacking the untruncated version. The algorithm $\mathcal{A}'$ is a simulator that forwards all queries from $\mathcal{A}$ to the encryption oracle, truncates the tag output of the oracle responses before forwarding them to $\mathcal{A}$, and returns the same result as $\mathcal{A}$.

In the following we denote V_i^j as the i-th keystream block of the j-th query and m^j as the length of the j-th message in blocks. Let $\mathfrak{Q}$ be the query history of the adversary, where the subset $\mathfrak{Q}_{|V_i,T}$ consists of all the output values of $\mathcal{H}_2$, *i.e.,* all chaining values V_i^j and authentication tags T^j with $i = 1, \ldots, m^j$ and $j = 1, \ldots, q$. We can say that COFFE is IND-CPA-secure if the produced keystream and the tag values within the query history are indistinguishable from a sequence of distinct n-bit random values, where the length of this sequence is limited to $\ell + q$. It is easy to see that the probability of a collision between two values can be upper bounded by

$$\frac{(\ell + q)^2}{2^n}.$$

To complete our proof, we have to estimate the probability $\Pr[Dist]$ that all values within the list $\mathfrak{Q}_{|V_i,T}$ are distinct. Therefore, we upper bound the probability $\Pr[Coll]$ for a collision of at least two of the values within this list since

$$\Pr\left[Dist\right] = 1 - \Pr\left[Coll\right].$$

To upper bound $\Pr[Coll]$, we first consider the input parameter of $\mathcal{H}_2$ represented by the quadruple $z_i^j = (S^j, V_i^j, C_i^j, d_i^j)$ where the domain d_i^j is either 4 or $|M_m| + 5$. Note that we ignore the 0^*-padding which leads to a higher success probability for an adversary. Let (i, j) and (i', j') be two distinct input tuples. In the following, we refer to the event $z_i^j = z_{i'}^{j'}$ as *input collision.* This event implies that either a collision for $\mathcal{H}_2$ occured or we have found a collision for the values $S^j \oplus T_i^j = S^{j'} \oplus T_{i'}^{j'}$.

For our case analysis (cf. Table 7.2), we encode the difference between two input tuples z_i^j and $z_{i'}^{j'}$ using a five-bit value. For example, the value "10110" defines the

Case	Event	Case	Event	Case	Event	Case	Event
00000	trivial	01000	–	10000	1	11000	2,4
00001	3	01001	–	10001	1	11001	3
00010	3	01010	–	10010	1	11010	3
00011	3	01011	–	10011	1	11011	3
00100	–	01100	3	10100	3	11100	1,3
00101	–	01101	3	10101	3	11101	3
00110	–	01110	3	10110	3	11110	3
00111	–	01111	3	10111	3	11111	3

Table 7.2.: This table illustrates the case analysis for the proof of Lemma 7.2, where each case with a non-zero probability is covered by at least one event. The case "11000" is covered by two events depending on the considered domains (Event 2 covers the domain 1,2, and 3; Event 4 covers all other domains). The second special case "11100" is covered by Event 1 if $S^j = S^{j'}$ and by Event 3 if $S^j \neq S^{j'}$.

following case:

$$10110 := \begin{cases} j \neq j' \\ V_i^j = V_{i'}^{j'} \\ S^j \oplus V_i^j \neq S^{j'} \oplus V_{i'}^{j'} \\ C_i^j \neq C_{i'}^{j'} \\ d_i^j = d_{i'}^{j'}. \end{cases}$$

Note that Table 7.2 contains a complete case analysis since all possible cases are covered. The cases which occur with a zero probability are obviously impossible and marked by '–'. The reason for the occurrence of these cases is a violation of the XOR-relation between the values S^j and V_i^j or $S^{j'}$ and $V_{i'}^{j'}$, respectively. For example, $(S^j = S^{j'} \wedge V_i^j = V_{i'}^{j'}) \wedge (S^j \oplus V_i^j \neq S^{j'} \oplus V_{i'}^{j'})$ is an impossible case. Case "00000" implies that a collision must have happened before in the same query and is already covered by other cases. In the following we analyze four events which cover all cases with a non-zero probability from Table 7.2.

After asking at most q queries, we check the query history $\mathfrak{Q}$ of the adversary – which contains all queries and their results – for the occurrence of bad events. We let the adversary win immediately if one of the bad events becomes true. Let denote A_α the α-th event. The occurrence of an event A_α implies that no event A_β with $\beta \in \{1, \ldots, \alpha - 1\}$ occurred before. Hence, the order of the events matters.

Event 1: Collision of two Session Keys. The first case describes the scenario where an adversary finds two values S^j and $S^{j'}$, generated using $\mathcal{H}_1$, with $j \neq j'$ and $S^j = S^{j'}$. The probability for this event can be upper bounded by

$$q^2/2^n.$$

Event 2: Input Collision – Associated Data. In this case we consider an adversary which finds two colliding pairs $(V_0^j \oplus S^j, x^j)$ and $(V_0^{j'} \oplus S^{j'}, x^{j'})$ with $j \neq j'$. A pair collides if it holds that $(V_0^j \oplus S^j = V_0^{j'} \oplus S^{j'}) \wedge (x^j = x^{j'})$. The occurence of this event leads to two colliding inputs for $\mathcal{H}_2$ in the first iteration. Observe that if no collision occurs, all V_i^j are independent random values. The probability for this case can be upper bounded by

$$\frac{q^2}{2^n}.$$

Event 3: Output Collision. For this case we consider an adversary which finds two values $V_i^j = V_{i'}^{j'}$ with $(i, j) \neq (i', j')$. The probability for this event can be upper bounded by

$$\ell^2/2^n.$$

Event 4: Input Collision – Message and Tag. Here, we consider an adversary which finds two tuples (V_i^j, S^j) and $(V_{i'}^{j'}, S^{j'})$ with $V_i^j \oplus S^j = V_{i'}^{j'} \oplus S^{j'}$. This leads to two colliding inputs for $\mathcal{H}_2$. Note that we assume that the adversary did not find an output collision before. The probability for this event can be upper bounded by

$$\ell^2/2^n.$$

Our claim follows by adding up the individual bounds. ■

Lemma 7.3 (INT-CTXT Security of COFFE). *Let* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *be the COFFE scheme as defined in Algorithm 6. We assume the adversary to be nonce-ignoring,* i.e., *it is able to choose two nonces* $N^j = N^{j'}$ *with* $j \neq j'$*. Then,*

$$\mathbf{Adv}_{\Pi}^{\textit{INT-CTXT}}(q, \ell, t) \leq \frac{2\ell^2 + 3q^2}{2^n} + \frac{q}{2^{|T|}} + 2 \cdot \mathbf{Adv}_{\mathcal{H}_*}^{\textit{PRF}}(q + \ell, O(t)).$$

Proof. Our bound is derived by game-playing arguments. Consider Games G_1-G_3 of Figure 7.2 and Figure 7.3, and a fixed adversary $\mathcal{A}$ asking at most q queries with a

```
1 Initialize()
2   K ←$ 𝒦()
3   𝔔, 𝔅0, 𝔅1, 𝔅2, 𝔅3, 𝔅4, 𝔅5 ← ∅
4   win ← false

100 Encrypt(N, H, M)  Game G1
101   S ← ℋ(K || N || 0* || |K| || |N| || 0)
102   (x, V0) ← G(H)
103   C0 ← ToHex(π)
104   I ← S ⊕ V0
105   V1 ← ℋ2(I || C0 || 0* || x)
106   C1 ← V1 ⊕ M1
107   for i = 2, ..., m do
108     I ← S ⊕ V_{i−1}
109     Vi ← ℋ2(I || C_{i−1} || 0* || 4)
110     Ci ← Vi ⊕ Mi
111   I ← S ⊕ Vm
112   T ← ℋ2(I || Cm || 0* || |T| || |Mm| + 5)
113   𝔔 ← (N, H, C, T)
114   return (C, T)
```

```
5 Finalize()
6   return win

120 Decrypt(N, H, C, T)  Game G1
121   S ← ℋ(K || N || 0* || |K| || |N| || 0)
122   (x, V0) ← G(H)
123   C0 ← ToHex(π)
124   I ← S ⊕ V0
125   V1 ← ℋ2(I || C0 || 0* || x)
126   M1 ← V1 ⊕ C1
127   for i = 2, ..., m do
128     I ← S ⊕ V_{i−1}
129     Vi ← ℋ2(I || C_{i−1} || 0* || 4)
130     Mi ← Vi ⊕ Ci
131   I ← S ⊕ Vm
132   T' ← ℋ2(I || Cm || 0* || |T| || |Cm| + 5)
133   if (T = T') ∧ (N, H, C, T) ∉ 𝔔 then
134     win ← true
135   return ⊥
```

Figure 7.2.: Game G_1 for the proof of Lemma 7.3.

total length of at most ℓ blocks. We assume that the adversary never asks a query for which the answer is already known. The functions **Initialize** and **Finalize** are identical for all games in this proof. Let G_0 denote the INT-CTXT Game as defined in Algorithm 2 (cf. Section 4.3). Therefore, we have

$$\mathbf{Adv}_{\Pi}^{INT\text{-}CTXT}(\mathcal{A}) \leq \Pr\left[\mathcal{A}^{G_0} \Rightarrow 1\right].$$

In G_1, the encryption- and decryption-placeholders are replaced by their generic COFFE counterparts as of Algorithm 6 and, using similar arguments as in the proof for Lemma 7.2, we can partition $\mathcal{H}$ into two independent PRFs $\mathcal{H}_1$ and $\mathcal{H}_2$. Thus,

$$\Pr\left[A^{G_0} \Rightarrow 1\right] \leq \Pr\left[A^{G_1} \Rightarrow 1\right] + 2 \cdot \mathbf{Adv}_{\mathcal{H}_*}^{PRF}(q + \ell, O(t)),$$

where

$$\mathbf{Adv}_{\mathcal{H}_*}^{PRF}(q + \ell, O(t)) = \max\left\{\mathbf{Adv}_{\mathcal{H}_1}^{PRF}(q, O(t)), \mathbf{Adv}_{\mathcal{H}_2}^{PRF\text{-}RKA}(q + \ell, O(t))\right\}.$$

We now discuss the differences between G_1 and G_2. The sets $\mathfrak{B}_0, \ldots, \mathfrak{B}_5$ are initialized as empty sets (cf. Line 3 of Figure 7.2) and collect fresh values as follows:

- $\mathfrak{B}_0$ collects all fresh values V_0, where $|H| > n$ in Lines 207 and 247.

Algorithm 8 LLCP'

Input: $\mathfrak{Q}$ {Query History}, N {Nonce}, H {Header}, M {Message}

```
p ← 0
for all (N′, H′, M′) ∈ 𝔔 do
    if (N = N′) ∧ (H = H′) then
        p ← max{p, LLCP(M, M′)}
    end if
end for
return p
```

- $\mathfrak{B}_1$ collects all fresh pairs (V_0, S, x) in Lines 212 and 252.
- $\mathfrak{B}_2$ collects all fresh values $I = V_0 \oplus S$ in Lines 213 and 253.
- $\mathfrak{B}_3$ collects all fresh pairs $(S \oplus V_i, C_i)$ with $i = 1, \ldots, m-1$. This is done in Lines 221 and 261
- $\mathfrak{B}_4$ collects all fresh values V_i with $i = 1, \ldots, m$ in Lines 215, 225, 255, and 265.
- $\mathfrak{B}_5$ collects all fresh pairs $(S \oplus V_m, C_m)$. This is done in Lines 233 and 270.

In Lines 201 and 241, the $LLCP'$ oracle is inquired as defined in Algorithm 8. Finally, the variable `bad` is set to `true` if one of the if-conditions in Lines 205, 210, 219, 223, 228, 245, 250, 259, 263, or 268 is **true**. *None* of these modifications affect the values returned to the adversary and therefore,

$$\Pr[\mathcal{A}^{G_1} \Rightarrow 1] = \Pr[\mathcal{A}^{G_2} \Rightarrow 1].$$

It follows that

$$\begin{aligned} \Pr\left[\mathcal{A}^{G_2} \Rightarrow 1\right] &\leq \Pr\left[\mathcal{A}^{G_3} \Rightarrow 1\right] + \left|\Pr\left[\mathcal{A}^{G_2} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{G_3} \Rightarrow 1\right]\right| \\ &\leq \Pr\left[\mathcal{A}^{G_3} \Rightarrow 1\right] + \Pr\left[\mathcal{A}^{G_3} \mathit{sets}\ \mathtt{bad}\right]. \end{aligned} \tag{7.1}$$

We now proceed to upper bound the two terms contained in Equation (7.1) – in right to left order.

The success probability of Game G_3 does not differ from the success probability of Game G_2 unless one of the following cases occur, where each case causes a bad event, *i.e.,* the variable `bad` is set to **true**. In the following, the indices j and j' denote the j-th and j'-th query with $j, j' = 1, \ldots, q$, respectively.

Encrypt(N, H, M) Game G_2 **and** $\boxed{G_3}$
  $p \leftarrow \mathbf{LLCP'}(\mathfrak{Q}_{|N,H,M}, (N, H, M))$
  $S \leftarrow \mathcal{H}_1(K \mid\mid N \mid\mid 0^* \mid\mid |K| \mid\mid |N| \mid\mid 0)$
  $(x, V_0) \leftarrow G(H)$
  if $(x = 3)$ **then**
    if $(H \notin \mathfrak{Q}_{|H}$ **and** $V_0 \in \mathfrak{B}_0)$ **then**
      bad $\leftarrow$ **true** $\boxed{V_0 \xleftarrow{\$} \{0,1\}^n \setminus \mathfrak{B}_0}$
  $\mathfrak{B}_0 \leftarrow \mathfrak{B}_0 \cup \{V_0\}$
  $C_0 \leftarrow \mathbf{ToHex}(\pi)$
  $I \leftarrow S \oplus V_0$
  if $((V_0, S, x) \notin \mathfrak{B}_1$ **and** $I \in \mathfrak{B}_2)$
    bad $\leftarrow$ **true** $\boxed{I \xleftarrow{\$} \{0,1\}^n \setminus B_2}$
  $\mathfrak{B}_1 \leftarrow \mathfrak{B}_1 \cup \{(V_0, S, x)\}$
  $\mathfrak{B}_2 \leftarrow \mathfrak{B}_2 \cup \{I\}$
  $V_1 \leftarrow \mathcal{H}_2(I \mid\mid C_0 \mid\mid 0^* \mid\mid x)$
  $\mathfrak{B}_4 \leftarrow \mathfrak{B}_4 \cup \{V_1\}$
  $C_1 \leftarrow V_1 \oplus M_1$
  for $i = 2, ..., m$ **do**
    $I \leftarrow S \oplus V_{i-1}$
    if $((I, C_{i-1}) \in \mathfrak{B}_3$ **and** $i > p)$ **then**
      bad $\leftarrow$ **true** $\boxed{I \xleftarrow{\$} \{0,1\}^n \setminus \mathfrak{B}_3}$
    $\mathfrak{B}_3 \leftarrow \mathfrak{B}_3 \cup \{(I, C_{i-1})\}$
    $V_i \leftarrow \mathcal{H}_2(I \mid\mid C_{i-1} \mid\mid 0^* \mid\mid 4)$
    if $(V_i \in \mathfrak{B}_4$ **and** $i > p)$ **then**
      bad $\leftarrow$ **true** $\boxed{V_i \xleftarrow{\$} \{0,1\}^n \setminus \mathfrak{B}_4}$
    $\mathfrak{B}_4 \leftarrow \mathfrak{B}_4 \cup \{V_i\}$
    $C_i \leftarrow V_i \oplus M_i$
  $I \leftarrow S \oplus V_m$
  if $((I, C_m) \in \mathfrak{B}_5)$ **then**
    bad $\leftarrow$ **true** $\boxed{I \xleftarrow{\$} \{0,1\}^n \setminus \mathfrak{B}_5}$
  $\mathfrak{B}_5 \leftarrow \mathfrak{B}_5 \cup \{(I, C_m)\}$
  $T \leftarrow \mathcal{H}_2(I \mid\mid C_m \mid\mid 0^* \mid\mid |T| \mid\mid |M_m| + 5)$
  $\mathfrak{Q} \leftarrow (N, H, C, T)$
  return (C, T)

Decrypt(N, H, C, T) Game G_2 **and** $\boxed{G_3}$
  $p \leftarrow \mathbf{LLCP'}(\mathfrak{Q}_{|,N,H,C}, (N, H, C))$
  $S \leftarrow \mathcal{H}_1(K \mid\mid N \mid\mid 0^* \mid\mid |K| \mid\mid |N| \mid\mid 0)$
  $(x, V_0) \leftarrow G(H)$
  if $(x = 3)$ **then**
    if $(H \notin \mathfrak{Q}_{|H}$ **and** $V_0 \in \mathfrak{B}_0)$ **then**
      bad $\leftarrow$ **true** $\boxed{V_0 \xleftarrow{\$} \{0,1\}^n \setminus \mathfrak{B}_0}$
  $\mathfrak{B}_0 \leftarrow \mathfrak{B}_0 \cup \{V_0\}$
  $C_0 \leftarrow \mathbf{ToHex}(\pi)$
  $I \leftarrow S \oplus V_0$
  if $((V_0, S, x) \notin \mathfrak{B}_1$ **and** $I \in \mathfrak{B}_2)$
    bad $\leftarrow$ **true** $\boxed{I \xleftarrow{\$} \{0,1\}^n \setminus \mathfrak{B}_2}$
  $\mathfrak{B}_1 \leftarrow \mathfrak{B}_1 \cup \{(V_0, S, x)\}$
  $\mathfrak{B}_2 \leftarrow \mathfrak{B}_2 \cup \{I\}$
  $V_1 \leftarrow \mathcal{H}_2(I \mid\mid C_0 \mid\mid 0^* \mid\mid x)$
  $\mathfrak{B}_4 \leftarrow \mathfrak{B}_4 \cup \{V_1\}$
  $M_1 \leftarrow V_1 \oplus C_1$
  for $i = 2, ..., m$ **do**
    $I \leftarrow S \oplus V_{i-1}$
    if $((I, C_{i-1}) \in \mathfrak{B}_3$ **and** $i > p)$ **then**
      bad $\leftarrow$ **true** $\boxed{I \xleftarrow{\$} \{0,1\}^n \setminus \mathfrak{B}_3}$
    $\mathfrak{B}_3 \leftarrow \mathfrak{B}_3 \cup \{(I, C_{i-1})\}$
    $V_i \leftarrow \mathcal{H}_2(I \mid\mid C_{i-1} \mid\mid 0^* \mid\mid 4)$
    if $(V_i \in \mathfrak{B}_4$ **and** $i > p)$ **then**
      bad $\leftarrow$ **true** $\boxed{V_i \xleftarrow{\$} \{0,1\}^n \setminus \mathfrak{B}_4}$
    $\mathfrak{B}_4 \leftarrow \mathfrak{B}_4 \cup \{V_i\}$
    $M_i \leftarrow V_i \oplus C_i$
  $I \leftarrow S \oplus V_m$
  if $((I, C_m) \in \mathfrak{B}_5)$ **then**
    bad $\leftarrow$ **true** $\boxed{I \xleftarrow{\$} \{0,1\}^n \setminus \mathfrak{B}_5}$
  $\mathfrak{B}_5 \leftarrow \mathfrak{B}_5 \cup \{(I, C_m)\}$
  $T' \leftarrow \mathcal{H}_2(I \mid\mid C_m \mid\mid 0^* \mid\mid |T| \mid\mid |C_m| + 5)$
  if $(T = T')$ **and** $(N, H, C, T) \notin \mathfrak{Q}$ **then**
    win $\leftarrow$ **true**
  return $\perp$

Figure 7.3.: Games G_2 and G_3 for the proof of Lemma 7.3.

Case 1 (Collision – Initial Chaining Value): In Lines 206 and 246 the initial chaining value V_0 is set to a new random value if the function G returns the same V_0 twice for two distinct values $H^j \neq H^{j'}$ with $j \neq j'$ and $H^j, H^{j'} > n$, *i.e.,* in the case when $x = 3$. The probability for such a collision can be upper bounded by

$$q^2/2^n.$$

Case 2 (Input Collision – Domain $1, \ldots, 3$): In Lines 211 and 251 the value I is set to a new random value if there is a non-trivial input collision between two input values $I^j = S^j \oplus V_0^j$ and $I^{j'} = S^{j'} \oplus V_0^{j'}$ with $x^j = x^{j'}$, so that $I^j = I^{j'}$ with $j \neq j'$. We can upper bound the success probability for this case by

$$q^2/2^n.$$

Case 3 (Input Collision – Domain 4): In Lines 219 and 259 we test for a non-trivial input collision $(S^j \oplus V_i^j, C_i^j) = (S^{j'} \oplus V_{i'}^{j'}, C_{i'}^{j'})$ with $(i,j) \neq (i',j')$. The success probability for this case can be upper bounded by

$$\ell^2/2^n.$$

Case 4 (Output Collision – Domain 4): In Lines 223 and 263 we test if the adversary has found a non-trivial collision of the form $V_i^j = V_{i'}^{j'}$ with $(i,j) \neq (i',j')$. The success probability can be upper bounded by

$$\ell^2/2^n.$$

Case 5 (Input Collision – Domain 5): In Lines 228 and 268 we test for a non-trivial input collision $(S^j \oplus V_m^j, C_m^j) = (S^{j'} \oplus V_{m'}^{j'}, C_{m'}^{j'})$ with $j \neq j'$. We can upper bound the success probability for this case by

$$q^2/2^n.$$

By adding up the individual bounds, it follows that

$$\Pr\left[\mathcal{A}^{G_3} \mathit{sets}\ \mathtt{bad}\right] \ \leq \ \frac{2\ell^2 + 3q^2}{2^n}.$$

The adversary wins Game G_3 *iff* the variable `win` is set to `true`, *i.e.,* the if-condition in Line 272 holds. This implies that the adversary can win only with a fresh query to the **Decrypt** oracle, which leads to $T = T'$, where T' is computed as shown in

Line 271. Lines 268 and 269 ensure that the input for the hash function $\mathcal{H}_2$ in Line 271 is always a fresh value, *i.e.*, it was never asked before. Since $\mathcal{H}_2$ is a PRF, the probability for $T = T'$ can be upper bounded by

$$1/2^{|T|}.$$

As we allow the adversary to ask at most q queries, the success probability for Game G_3 can be upper bounded by

$$\Pr\left[\mathcal{A}^{G_3} \Rightarrow 1\right] \;\leq\; q/2^{|T|}.$$

Our claim follows by adding up the individual bounds. ■

7.4. Results Summary

In this Chapter we presented COFFE, a novel hash function based OAE scheme which, to the best of our knowledge, is the first scheme that fulfills our stated requirements. It can be part of a minimal cryptographic suite that includes hashing and digital signatures. Because it is an AEAD scheme, it could be used in the AEAD interface of the Datagram Transport Layer Security (TLS) protocol [198] that has been identified by the *IETF Constrained Application Working Group* as suitable for applications for the IoT. In the standard model, COFFE provides the regular INT-CTXT and IND-CPA security plus INT-CTXT security in the nonce-misuse setting. Finally, it is resistant against side-channel attacks, which is usually a matter of the implementation of a cryptosystem, rather than of the cryptosystem itself. Nevertheless, we provide side-channel resistance even if an implementation lacks to provide this kind of security.

8 McOE: A Family of Robust On-Line Authenticated Encryption Schemes

If I have seen further it is by
standing on the shoulders of Giants.

Isaac Newton

In recent years, cryptographers developed misuse-resistant schemes for authenticated encryption [126, 127, 210]. These guarantee excellent security even against general adversaries which are allowed to reuse nonces. Their disadvantage is that encryption can be performed in an off-line way, only. In this chapter we introduce a novel family of robust OAE schemes called McOE. Apart from the generic composition Encrypt-then-Mac (EtM), none of the ISO/IEC 19772:2009 schemes – in fact, no previously published authenticated encryption scheme at all – achieves both to be on-line and robust (cf. Table 8.1). In this table we classify a variety of provably secure block cipher based authenticated encryption schemes with respect to their *on-line-ability* and against which adversaries (nonce-respecting vs. nonce-ignoring) they are proven to be secure.

Encrypt-then-Mac (EtM). Since EtM is not a concrete scheme but a generic construction, there are some challenges left in order to make it fully on-line-secure: First, an appropriate on-line cipher has to be chosen. Second, a suitable, on-line-computable, secure, and deterministic MAC must be selected. And, third, the generic EtM scheme requires at least two *independent* keys to be secure. Since two schemes are used in parallel, it is likely to squander resources in terms of run time and – im-

Type	CCA3-secure	Robust
on-line	CCFB CHM **COFFE** CWC EAX GCM IACBC IAPM **McOE-G** **McOE-X** OCB1-3 RPC TAE XCBC	**McOE-X** **McOE-G**
off-line	CCM BTM HBS SIV	

Table 8.1.: Classification of provably secure block cipher based authenticated encryption schemes.

portant for hardware designers – in terms of space. Since EtM first has to be turned into an OAE scheme by making the appropriate choices, we do not consider it in our analysis.

Design Principles for AE Schemes. The question of how to provide authenticated encryption (without stating that name), when given a secure on-line cipher, is studied in [15], the revised and full version of [14]. The first approach in [15] only provides security if all messages are of the same length. The second approach repairs that by prepending the length of the message, at the cost of being off-line since the length must be known at the beginning of the encryption process. Their approach is to prepend and append a random value N to a message M and then to perform the on-line encryption of $(N||M||N)$. This looks promising, but the same N is used for two different purposes, putting different constraints on the generation of N. For privacy, it suffices that N behaves like a nonce, not requiring secrecy or unpredictability. Even if N is not a nonce, but the same N is used for the encryption of several messages, all the adversary can determine are the lengths of common plaintexts prefixes, as we required for nonce reuse. On the other hand, authenticity actually assumes a *secret or unpredictable* N rather than a nonce. If the adversary $\mathcal{A}$ can guess N before choosing a message, $\mathcal{A}$ asks for the authenticated encryption of $(M||N)$. Then, $\mathcal{A}$ can predict the authenticated encryption of M without actually asking for it.

The general structure of McOE is based on the Tweak Chain Hash (TCH) from [153] which itself is adapted from the Matyas-Meyer-Oseas (MMO) construction [170]. Thereby, McOE replaces the *random* N by a proper nonce and the *key-dependent* tag computation value τ, performing a nonce-dependent on-line encryption of $(M||\tau)$. The encryption can also depend on some associated data, which turns McOE into a family of schemes for on-line AEAD.

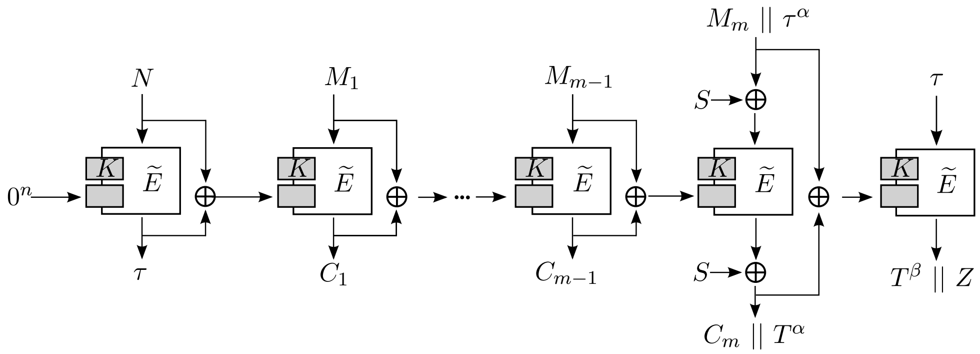

Figure 8.1.: The generic McOE construction. T^α and τ^α denote the lower α bits and T^β and τ^β denote the upper β bits of T, respectively. Furthermore, S is given by $\widetilde{E}_K(1^n, |M|)$.

Outlook. In Section 8.1 we introduce the generic specification of the McOE family based on a tweakable block cipher, and in Section 8.2 we analyze its security. Section 8.3 introduces two practical instance of the McOE family, called McOE-X and McOE-G. In Section 8.4 we present some performance benchmarks of both instances, and finally, Section 8.5 gives a brief summary about our contribution.

8.1. Generic Specification

The structure of McOE bases on the TC3 construction, a tweakable block cipher based encryption scheme, (cf. Figure 8.1), which was presented by Rogaway and Zhan [211]. A formal definition of McOE is given in Algorithm 9. Both functions **Encrypt** and **Decrypt** consist of the following three steps:

Step 1: ProcessHeader. In this step we describe the processing of the header H, which can be of arbitrary length. In the case when the length of the header is not a multiple of the block size n, we apply the common 10*-padding. Furthermore, H has to consist of at least one block since the tag computation value τ depends on it. Hence, the whole header can be seen as a nonce. In the following, the lowest $n - M_m$ bits of τ are denoted by τ^α. A formal definition of the header processing is given in Algorithm 10

Step 2: Plaintext/Ciphertext Processing. The plaintext/ciphertext blocks (except for the final block, which is discussed in step 3) are processed in a straightforward way

Algorithm 9 McOE

Encrypt(H, M)

$m \leftarrow |M|/n$
$(U, \tau) \leftarrow$ **ProcessHeader**(H)
for $i = 1, \ldots, m-1$ **do**
  $C_i \leftarrow \widetilde{E}_K(U, M_i)$
  $U \leftarrow M_i \oplus C_i$
end for
$S \leftarrow \widetilde{E}_K(1^n, |M_m|)$
$X \leftarrow (M_m \mid\mid \tau^\alpha) \oplus S$
$Y \leftarrow \widetilde{E}_K(U, X)$
$(C_m \mid\mid T^\alpha) \leftarrow Y \oplus S$
$U \leftarrow X \oplus Y$
$(T^\beta \mid\mid Z) \leftarrow \widetilde{E}_K(U, \tau)$
$T \leftarrow T^\alpha \mid\mid T^\beta$
return $(C_1, \ldots, C_m, T)$

Decrypt(H, C, T)

$m \leftarrow |C|/n$
$(U, \tau) \leftarrow$ **ProcessHeader**(H)
for $i = 1, \ldots, m-1$ **do**
  $M_i \leftarrow \widetilde{E}_K^{-1}(U, C_i)$
  $U \leftarrow M_i \oplus C_i$
end for
$S \leftarrow \widetilde{E}_K(1^n, |C_m|)$
$Y \leftarrow (C_m \mid\mid T^\alpha) \oplus S$
$X \leftarrow \widetilde{E}_K^{-1}(U, Y)$
$(M_m \mid\mid \tau') \leftarrow X \oplus S$
$U \leftarrow X \oplus Y$
$(T' \mid\mid Z) \leftarrow \widetilde{E}_K(U, \tau)$
if $\tau' = \tau^\alpha$ **and** $T^\beta = T'$ **then**
  return $(M_1, \ldots, M_m)$
end if
return $\perp$

Algorithm 10 ProcessHeader

Input: H {Header}
Output: U {Tweak}, τ {Tag Computation Value}

$U \leftarrow 0^n$
for $i = 1, \ldots, |H|/n$ **do**
  $\tau \leftarrow \widetilde{E}_K(U, H_i)$
  $U \leftarrow H_i \oplus \tau$
end for
return (U, τ)

by the underlying tweakable block cipher $\widetilde{E}$ or its inverse $\widetilde{E}^{-1}$ (cf. Algorithm 9, Lines 3–6 and Lines 23–26). The tweak U is computed by XORing the previous ciphertext block C_{i-1} and plaintext block M_{i-1}. The length of the final plaintext/ciphertext block is between 1 bit and n bits since McOE allows to process arbitrary length messages. A tweakable block cipher allows only to process n-bit message blocks. Therefore, we process the final plaintext block M_m as follows: At first we pad it to n bits by appending τ, before we XOR the padded value with the encryption of M_m encoded as n-bit value with 1^n as tweak (see Algorithm 9, Line 7). The final message block can be computed by inverting this procedure (see Algorithm 9, Lines 27–28).

Step 3: Tag Generation/Verification. A common technique to support length-preserving encryption is Ciphertext Stealing (CTS) [69]. Unfortunately, this approach contradicts the on-line property of McOE since it requires to process the final block before its predecessors. Therefore, we introduce a novel method, called Tag Splitting (TS), where the n-bit tag T is split into an upper part T^α consisting of the α Most Significant Bits (MSBs) of T and a lower part T^β consisting of the β Least Significant Bits (LSBs) of T. Note that $\alpha + \beta = n$ always holds. Furthermore, α can be 0 if M_m is already an full n-bit block.

The final ciphertext block C_m together with the upper part of the authentication tag T^α is derived from the padded message block $M'_m = M_m \mid\mid \tau^\alpha$ by applying (the XEX block cipher mode introduced by Liskov *et al.* in [152]), *i.e.*, $(C_m \mid\mid T^\alpha) = \widetilde{E}_K(U, M'_m \oplus S) \oplus S$, where U is the current chaining value and $S = \widetilde{E}_K(1^n, |M_m|)$ (see Algorithm 9, Lines 7–10). This step implements length-preserving encryption since it ensures that $|M| = |C|$ always holds. Moreover, for the sake of optimization, the masking value S can often be computed in advance for all possible message lengths. This allows to process the final blocks without an additional invocation of the tweakable block cipher. The lower part of the authentication tag is computed by the encryption of τ (see Algorithm 9, Line 12). The remaining α bits of the encryption (Z) are discarded. The straightforward verification of the authentication tag is given in Lines 29–34 of Algorithm 9.

8.2. Security Analysis

In this section we show that McOE is a robust OAE scheme, in the standard model.

Theorem 8.1 (ONDMA Security). *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be the* McOE *scheme as defined in the previous section with $\mathcal{E} =$ **Encrypt** and $\mathcal{D} =$ **Decrypt** from Algorithm 9. Then, for $q \leq 2^{n/2-2}$, we have*

$$\begin{aligned}\mathbf{Adv}_{\Pi}^{ONDMA}(q, \ell, t) \leq & \frac{(q+\ell+3)^2 + 2(q+\ell) + q^2}{2^{n-1} - q} + \frac{q}{2^{n/2} - q} \\ & + 2\mathbf{Adv}_{\widetilde{E}, \widetilde{E}^{-1}}^{IND\text{-}PRP}(2q+\ell, O(t)).\end{aligned}$$

Proof. The proof follows from Lemmas 8.2 and 8.3. ∎

Lemma 8.2 (IND-OCCA2 Security). *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a* McOE *scheme as defined in the previous section with $\mathcal{E} =$ **Encrypt** and $\mathcal{D} =$ **Decrypt** from Algorithm 9 where the tag verification process (Lines 31–33 and 35–36) is omitted. Then, for all $M \in (\{0,1\}^n)^*$, we have*

$$\mathbf{Adv}_{\Pi}^{IND\text{-}OCCA2^1}(q, \ell, t) \leq \frac{(q+\ell+3)^2 + 2(q+\ell) + q^2}{2^n - q} + \mathbf{Adv}_{\widetilde{E}, \widetilde{E}^{-1}}^{IND\text{-}PRP}(2q+\ell, O(t)).$$

Proof. The basic idea of this proof is to show that the absence of non-trivial collisions in the tweak values (U) implies IND-CCA-security.

This proof borrows ideas from [211], proof of Theorem 3, and moreover, uses common game-playing arguments. The advantage of an adversary $\mathcal{A}$ to distinguish G_i from G_j is given by

$$\mathbf{Adv}_{G_i}^{G_j}(\mathcal{A}) = \left|\Pr\left[\mathcal{A}^{G_i} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{G_j} \Rightarrow 1\right]\right|.$$

Let the tuple ($\mathcal{E}$, $\mathcal{D}$) denote the initial Game G_0. The Game G_1 is equal to the game G_0 except that the tweakable block cipher $\widetilde{E}$ is replaced by a set of pseudo random permutations $\mathfrak{P} \xleftarrow{\$} \mathbf{Perm}_n^n$ – which can be implemented efficiently via lazy sampling. To process a total amount of q encryption and decryption queries requieres $\ell + 2q$ unique invocations of the tweakable block cipher. Thus, we have

$$\mathbf{Adv}_{G_0}^{G_1}(\mathcal{A}) \leq \mathbf{Adv}_{\widetilde{E}}^{IND\text{-}PRP}(2q+\ell, O(t)).$$

Game G_1 is transformed into Game G_2 as follows. First, we add a set $\mathfrak{Q}$, the query history, collecting all output tuples (H, M, C, T) consisting of a header H, the authentication tag T, the message M, and the corresponding ciphertext C (see Figure 8.2

Lines 226 and 251). Then, the LLCP-oracle is called to compute the LCP between the current query and all previously asked queries (Figure 8.2 Lines 203 and 233). This is required to determine if a specific part of the message is a common prefix. Furthermore we add three additional sets $\mathfrak{B}_1 - -\mathfrak{B}_3$ which are initialized as follows: $\mathfrak{B}_1 = \{0^n, 1^n\}, \mathfrak{B}_2 = \emptyset$, and $\mathfrak{B}_3 = \emptyset$. Set $\mathfrak{B}_1$ collects all tweaks U that are computed during the encryption or decryption process (see Figure 8.2 Lines 207, 213, 224, 237 and 243). Set $\mathfrak{B}_2$ collects the input tuple (U, X) for the encryption of the final message block – consisting of the masked final message block X and the corresponding tweak U (see Figure 8.2, Line 219). Similarly, the set $\mathfrak{B}_3$ collects the input tuples (U, Y) for the decryption of the final message block – consisting of the masked final ciphertext block Y and the corresponding tweak U (see Figure 8.2 Line 249). Finally, the variable `bad` is set to **true** if one of the if-conditions in lines 205, 211, 217, 222, 236, 241 or 247 is **true**. None of these modifications affect the values returned to the adversary, and therefore, we have

$$\Pr\left[\mathcal{A}^{G_1} \Rightarrow 1\right] = \Pr\left[\mathcal{A}^{G_2} \Rightarrow 1\right].$$

In Game G_3 we eliminate the effects of bad events – immediately after setting the variable `bad` to **true**. After every collision between two chaining values occurs, the current value is replaced by a fresh value (see Figure 8.2 Lines 206, 212, 223, 236, and 242). Furthermore, the encryption of the masked final plaintext block is replaced when an input collision for $\mathfrak{P}$ occurs (see Figure 8.2 Line 218). Finally, the decryption of the masked final ciphertext block is replaced when an input collision for $\mathfrak{P}^{-1}$ occurs (see Figure 8.2 Line 248). Since G_2 and G_3 only differ when the variable `bad` is set to **true**, we have

$$\mathbf{Adv}_{G_2}^{G_3}(\mathcal{A}) = \Pr\left[\mathcal{A}^{G_3} sets\ \texttt{bad}\right].$$

We upper bound the probability for this event by a case analysis.

Collision in $\mathfrak{B}_1$. In this case the adversary must have found either a collision for $\mathfrak{P}(V, W) \oplus W$ (*i.e.,* it has found two input tuples $(V, W) \neq (V', W')$ such that $\mathfrak{P}(V, W) \oplus W = \mathfrak{P}(V', W') \oplus W'$) or it must have found a preimage of 0^n or 1^n. In both cases the variable `bad` would have been set to **true**, and it follows from [47] that the success probability for this event can be upper bound by

$$\frac{(q + \ell + 2)(q + \ell + 3)}{2^n - q} + \frac{2(q + \ell)}{2^n - q}$$

Encrypt(H, M) Game G_2 **and** $\boxed{G_3}$
$m \leftarrow |M|/n$
$h \leftarrow |H|/n$
$p \leftarrow \mathrm{LLCP}(\mathfrak{Q}_{|H,M}, (H, M))$
$(U, \tau) \leftarrow$ **ProcessHeader**(H)
if $((p < h)$ **and** $(U \in \mathfrak{B}_1))$
  bad $\leftarrow$ **true**; $\boxed{U \xleftarrow{\$} \{0,1\}^n \setminus \mathfrak{B}_1}$
$\mathfrak{B}_1 \leftarrow \mathfrak{B}_1 \cup \{U\}$
for $i = 1, \ldots, m-1$
  $C_i \leftarrow \mathfrak{P}(U, M_i)$
  $U \leftarrow M_i \oplus C_i$
  if $((p < h + i)$ **and** $(U \in \mathfrak{B}_1))$
    bad $\leftarrow$ **true**; $\boxed{U \xleftarrow{\$} \{0,1\}^n \setminus \mathfrak{B}_1}$
  $\mathfrak{B}_1 \leftarrow \mathfrak{B}_1 \cup \{U\}$
$S \leftarrow \mathfrak{P}(1^n, |M_m|)$
$X \leftarrow (M_m \mid\mid \tau^\alpha) \oplus S$
$Y \leftarrow \mathfrak{P}(U, X)$
if($(U, X) \in \mathfrak{B}_2$)
  bad $\leftarrow$ **true**; $\boxed{Y \xleftarrow{\$} \{0,1\}^n}$
$\mathfrak{B}_2 \leftarrow \mathfrak{B}_2 \cup \{(U, X)\}$
$(C_m \mid\mid T^\alpha) \leftarrow Y \oplus S$
$U \leftarrow X \oplus Y$
if $(U \in \mathfrak{B}_1)$
    bad $\leftarrow$ **true**; $\boxed{U \leftarrow \{0,1\}^n \setminus \mathfrak{B}_1}$
$\mathfrak{B}_1 \leftarrow \mathfrak{B}_1 \cup \{U\}$
$(T^\beta \mid\mid Z) \leftarrow \mathfrak{P}(U, \tau)$
$\mathfrak{Q} \leftarrow \mathfrak{Q} \cup \{(H, M, C, T)\}$
return (C, T)

Decrypt(H, C, T) Game G_2 **and** $\boxed{G_3}$
$m \leftarrow |C|/n$
$h \leftarrow |H|/n$
$p \leftarrow \mathrm{LLCP}(\mathfrak{Q}_{|H,C}, (H, C))$
$(U, \tau) \leftarrow$ **ProcessHeader**(H)
if $((p < h)$ **and** $(U \in \mathfrak{B}_1))$
  bad $\leftarrow$ **true**; $\boxed{U \xleftarrow{\$} \{0,1\}^n \setminus \mathfrak{B}_1}$
$\mathfrak{B}_1 \leftarrow \mathfrak{B}_1 \cup \{U\}$
for $i = 1, \ldots, m-1$
  $M_i \leftarrow \mathfrak{P}^{-1}(U, C_i)$
  $U \leftarrow M_i \oplus C_i$
  if $((p < h + i)$ **and** $(U \in \mathfrak{B}_1))$
    bad $\leftarrow$ **true**; $\boxed{U \xleftarrow{\$} \{0,1\}^n \setminus \mathfrak{B}_1}$
  $\mathfrak{B}_1 \leftarrow \mathfrak{B}_1 \cup \{U\}$
$S \leftarrow \mathfrak{P}(1^n, |C_m|)$
$Y \leftarrow (C_m \mid\mid T^\alpha) \oplus S$
$X \leftarrow \mathfrak{P}^{-1}(U, Y)$
if($(U, X) \in \mathfrak{B}_3$)
  bad $\leftarrow$ **true**; $\boxed{X \xleftarrow{\$} \{0,1\}^n}$
$\mathfrak{B}_3 \leftarrow \mathfrak{B}_3 \cup \{(U, Y)\}$
$(M_m \mid\mid \tau^\alpha) \leftarrow X \oplus S$
$\mathfrak{Q} \leftarrow \mathfrak{Q} \cup \{(H, M, C, T)\}$
return M

Figure 8.2.: The IND-OCCA2 games G_2 and G_3 for the proof of Lemma 8.2. Game G_3 contains the code in the box while G_2 does not.

Collision in $\mathfrak{B}_2$. In this case we can assume that no bad event has occurred so far. Therefore, the adversary can only win if it finds two colliding message blocks M_m and M'_m sharing the same common prefix, *i.e.*, $M = M_1, ..., M_{m-1}, M_m$ and $M = M_1, ..., M_{m-1}, M'_m$ with $M_m \neq M'_m$. Now, we have to upper bound the success probability for the event

$$(M_m \mid\mid \tau^\alpha) \oplus \mathfrak{P}(1^n, |M_m|) = (M'_m \mid\mid \tau^\alpha) \oplus \mathfrak{P}(1^n, |M'_m|).$$

Note that the equation above can hold only for $|M_m| \neq |M'_m|$. Since $\mathfrak{P}(1^n, \cdot)$ is a random permutation, we can upper bound the success probability of $\mathcal{A}$ by

$$\frac{q^2}{2^{n-1} - q}.$$

Collision in $\mathfrak{B}_3$. This case is similar to the previous one, and therefore, we can upper bound the success probability of an adversary by

$$\frac{q^2}{2^{n-1} - q}.$$

By adding up the individual bounds it follows that

$$\Pr\left[\mathcal{A}^{G_3} sets\ \texttt{bad}\right] \leq \frac{(q+\ell+3)^2 + 2(q+\ell) + q^2}{2^n - q}.$$

Let the tuple $(\mathcal{O}^1_{P,1}, \widehat{O}^1_{P^{-1},1})$ denote the final game G_4 where $\mathcal{O}^1_P$ and $\widehat{\mathcal{O}}^1_{P^{-1}}$ comply with the encryption and decryption oracles from Definition 5.5. Note that in G_3 the chaining value U cannot collide and it is not possible to compute a preimage for any query. This implies that $\mathfrak{P}$ is always invoked with a fresh tweak input, except two queries share a common prefix. Furthermore, we ensure by Lines 218 and 223 that both the final message block tag value is always a fresh random value. The same arguments hold for the decryption oracle. Thus, G_3 and G_4 have identical input-output behaviours and we have,

$$\mathbf{Adv}^{G_4}_{G_3}(\mathcal{A}) = 0.$$

Our claim follows by adding up the individual bounds. ■

Lemma 8.3 (INT-CTXT Security). *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a* McOE *scheme as in the previous section with $\mathcal{E} =$* **Encrypt** *and $\mathcal{D} =$* **Decrypt** *from Algorithm 9. Then, for $q \leq 2^{n/2-2}$ we have*

$$\mathbf{Adv}^{\textit{INT-CTXT}}_{\Pi}(q, \ell, t) \leq \frac{(q+\ell+3)^2 + 2(q+\ell) + q^2}{2^n - q} + \frac{q}{2^{n/2} - q} + \mathbf{Adv}^{\textit{IND-PRP}}_{\widetilde{E},\widetilde{E}^{-1}}(2q+\ell, O(t)).$$

Proof. Our bound is derived with the help of game-playing arguments. Consider Games G_1-G_3 of Figures 8.3 and 8.4 and a fixed adversary $\mathcal{A}$ asking at most q queries

Initialize()
  $K \xleftarrow{\$} \mathcal{K}()$
  $\mathfrak{B}_1 \leftarrow \{0^n, 1^n\}$

Encrypt(H, M) Game G_1
  $m \leftarrow |M|/n$
  $h \leftarrow |H|/n$
  $U \leftarrow 0^n$;
  for $i = 1, \ldots, h$ **do**
    $\tau \leftarrow \widetilde{E}_K(U, H_i)$
    $U \leftarrow H_i \oplus \tau$
  for $i = 1, \ldots, m-1$ **do**
    $C_i \leftarrow \widetilde{E}_K(U, M_i)$
    $U \leftarrow M_i \oplus C_i$
  $S \leftarrow \widetilde{E}_K(1^n, |M_m|)$
  $X \leftarrow (M_m \mid\mid \tau^\alpha) \oplus S$
  $Y \leftarrow \widetilde{E}_K(U, X)$
  $(C_m \mid\mid T^\alpha) \leftarrow Y \oplus S$
  $(T^\beta \mid\mid Z) \leftarrow \widetilde{E}_K(U, \tau)$
  $\mathfrak{Q} \leftarrow \mathfrak{Q} \cup \{(H, M, C, T)\}$
  return $(C_1 \mid\mid \ldots \mid\mid C_m, T^\alpha \mid\mid T^\beta)$

Finalize()
  return `win`

Verify(H, C, T) Game G_1
  $m \leftarrow |C|/n$
  $h \leftarrow |H|/n$
  $U \leftarrow 0^n$
  for $i = 1, \ldots, h$ **do**
    $\tau \leftarrow \widetilde{E}_K(U, H_i)$
    $U \leftarrow H_i \oplus \tau$
  for $i = 1, \ldots, m-1$ **do**
    $M_i \leftarrow \widetilde{E}_K^{-1}(U, C_i)$
    $U \leftarrow M_i \oplus C_i$
  $S \leftarrow \widetilde{E}_K(1^n, |M_m|)$
  $Y \leftarrow (C_m \mid\mid T^\alpha) \oplus S$
  $X \leftarrow \widetilde{E}_K^{-1}(U, Y)$
  $(M_m \mid\mid \tau') \leftarrow X \oplus S$
  $U \leftarrow X \oplus Y$
  $(T' \mid\mid Z) \leftarrow \widetilde{E}_K(U, \tau)$
  if $(\tau' = \tau^\alpha$ **and** $T^\beta = T'$ **and**
      $(H, C, T) \notin \mathfrak{Q}_{|H,C,T})$
    `win` $\leftarrow$ **true**
  $\mathfrak{Q} \leftarrow \mathfrak{Q} \cup \{(H, \perp, C, \perp)\}$
  return $\perp$

Figure 8.3.: Game G_1 for the proof of Lemma 8.3.

with a total length of at most ℓ blocks. The functions **Initialize** and **Finalize** are identical for all games in this proof. Let G_0 denote as the INT-CTXT Game defined in Algorithm 2. Definition 4.3 states that

$$\mathbf{Adv}_{\Pi}^{\mathit{INT\text{-}CTXT}}(\mathcal{A}) \leq \Pr\left[\mathcal{A}^{G_0} \Rightarrow 1\right].$$

In Game G_1, the encrypt and verify placeholders are replaced by their McOE counterparts including two mentionable tweaks which does not effect the success probability of any adversary. First, the oracle **Verify** returns `win` instead of $(M \neq \perp)$, and second, the query history collects additional data which is required to compute the LCP in the Games G_2 and G_3.

We now discuss the differences between G_1 and G_2. The set $\mathfrak{B}_1$ is initialized with $\{0^n, 1^n\}$ before it collects all new tweak values U that are computed during the encryption or verification process (in Lines 210, 216, 227, 250, 256, and 267). Furthermore, set $\mathfrak{B}_2$ collects the input tuples (U, X) consisting of the masked final message block X and the corresponding tweak U (see Figure 8.4, Line 222). Similarly, the set $\mathfrak{B}_3$ collects the input tuples (U, Y) consisting of the masked final ciphertext block U and the corresponding tweak U (see Figure 8.4, Line 262).

In Lines 203 and 243, the LLCP oracle is inquired. Finally, the variable `bad` is set to **true** if one of the if-conditions in Lines 208, 214, 220, 225, 248, 254, 260 or 265 holds. *None* of these modifications affect the values returned to the adversary and therefore

$$\Pr\left[\mathcal{A}^{G_1} \Rightarrow 1\right] = \Pr\left[\mathcal{A}^{G_2} \Rightarrow 1\right].$$

For our further discussion, we require another Game G_4 which is explained in more detail later in this proof[1]. It follows that

$$\begin{aligned}
\Pr\left[\mathcal{A}^{G_2} \Rightarrow 1\right] &\leq \Pr\left[\mathcal{A}^{G_3} \Rightarrow 1\right] + \left|\Pr\left[\mathcal{A}^{G_2} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{G_3} \Rightarrow 1\right]\right| \\
&\leq \Pr\left[\mathcal{A}^{G_3} \Rightarrow 1\right] + \Pr\left[\mathcal{A}^{G_3} \mathit{sets}\ \texttt{bad}\right] \\
&\leq \Pr\left[\mathcal{A}^{G_4} \Rightarrow 1\right] + \left|\Pr\left[\mathcal{A}^{G_3} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{G_4} \Rightarrow 1\right]\right| \\
&\quad + \Pr\left[\mathcal{A}^{G_3} \mathit{sets}\ \texttt{bad}\right].
\end{aligned}$$

Next, we upper bound the three terms in the line above from right to left. Using similar arguments as in the proof of Lemma 8.2 we can upper bound $\Pr\left[\mathcal{A}^{G_3} \mathit{sets}\ \texttt{bad}\right]$ by

$$\frac{(q+\ell+3)^2 + 2(q+\ell) + q^2}{2^n - q}.$$

Now, we describe the new Game G_4, which is equal to G_3 *except* that the block cipher $\widetilde{E}$ is replaced by a set of random permutations $\mathfrak{P} \xleftarrow{\$} \mathbf{Perm}_n^n$ – which can be implemented efficiently via lazy sampling. To process a total amount of q encryption and decryption queries implies at most $\ell + 2q$ unique invocations of the tweakable block cipher. Thus, we have

$$\mathbf{Adv}_{G_0}^{G_1}(\mathcal{A}) \leq \mathbf{Adv}_{\widetilde{E},\widetilde{E}^{-1}}^{\mathit{IND\text{-}PRP}}(2q+\ell, O(t)).$$

Finally, we have to upper bound the advantage for the adversary $\mathcal{A}$ to win the Game G_4. $\mathcal{A}$ can win this game only if the condition in Line 269 (resp. Line 469 for Game G_4) holds. Without loss of generality, we assume that $\mathcal{A}$ does not ask a question if the answer is already known. This implies that $(H, C, T) \notin \mathfrak{Q}_{|H,C,T}$. We formally adjust Line 269 (*i.e.,* choose as the tag computation operation either $\mathfrak{P}$ or $\mathfrak{P}^{-1}$) such that we always have enough randomness left for our result. For the sake of simplicity, we denote the two final chaining values by U_m and U_{m+1}. For our analysis, we distinguish between two main scenarios: (1) $|M_m| = n$, and (2) $|M_m| \neq n$

[1] Since the difference is minor, we do not provide an extra figure.

```
Encrypt(H, M)  Game G₂ and [G₃]
  m ← |M|/n
  h ← |H|/n
  p ← LLCP(𝔔_{|H,M}, (H, M))
  U ← 0^n
  for i = 1, ..., h
      τ ← Ẽ_K(U, H_i)
      U ← H_i ⊕ τ
      if (U ∈ 𝔅₁ and p < i) then
          bad ← true; [U ←$ {0,1}^n \ 𝔅₁]
      𝔅₁ ← 𝔅₁ ∪ U
  for i = 1, ..., m − 1
      C_i ← Ẽ_K(U, M_i)
      U ← M_i ⊕ C_i
      if ((p < h + i) and (U ∈ 𝔅₁))
          bad ← true; [U ←$ {0,1}^n \ 𝔅₁]
      𝔅₁ ← 𝔅₁ ∪ {U}
  S ← Ẽ_K(1^n, |M_m|)
  X ← (M_m || τ^α) ⊕ S
  Y ← Ẽ_K(U, X)
  if( (U, X) ∈ 𝔅₂ )
    bad ← true; [Y ←$ {0,1}^n]
  𝔅₂ ← 𝔅₂ ∪ {(U, X)}
  (C_m || T^α) ← Y ⊕ S
  U ← X ⊕ Y
  if (U ∈ 𝔅₁)
      bad ← true; [U ←$ {0,1}^n \ 𝔅₁]
  𝔅₁ ← 𝔅₁ ∪ {U}
  (T^β || Z) ← Ẽ_K(U, τ)
  𝔔 ← 𝔔 ∪ {(H, M, C, T)}
  return (C₁ || ... || C_m, T^α || T^β)
```

```
Verify(H, C, T)  Game G₂ and [G₃]
  m ← |C|/n
  h ← |H|/n
  p ← LLCP(𝔔_{|H,C,T}, (H, C, T))
  U ← 0^n
  for i = 1, ..., h
      τ ← Ẽ_K(U, H_i)
      U ← H_i ⊕ τ
      if (U ∈ 𝔅₁ and p < i) then
          bad ← true; [U ←$ {0,1}^n \ 𝔅₁]
      𝔅₁ ← 𝔅₁ ∪ U
  for i = 1, ..., m − 1
      M_i ← Ẽ_K^{-1}(U, C_i)
      U ← M_i ⊕ C_i
      if ((p < h + i) and (U ∈ 𝔅₁))
          bad ← true; [U ←$ {0,1}^n \ 𝔅₁]
      𝔅₁ ← 𝔅₁ ∪ {U}
  S ← Ẽ_K(1^n, |M_m|)
  Y ← (C_m || T^α) ⊕ S
  X ← Ẽ_K^{-1}(U, Y)
  if((p < h + m) and ((U, Y) ∈ 𝔅₃))
      bad ← true; [X ←$ {0,1}^n \ 𝔅₁ ⊕ Y]
  𝔅₃ ← 𝔅₃ ∪ {(U, Y)}
  (M_m || τ') ← X ⊕ S
  U ← X ⊕ Y
  if ((p < h + m) and (U ∈ 𝔅₁))
      bad ← true; [U ←$ {0,1}^n \ 𝔅₁]
  𝔅₁ ← 𝔅₁ ∪ {U}
  (T'||Z) ← Ẽ_K(U, τ)
  if (τ' = τ^α and T^β = T' and
  (H, C, T) ∉ 𝔔_{|H,C,T})
      win ← true
   𝒬 ← 𝒬 ∪ {(H, ⊥, C, ⊥)}
   return win
```

Figure 8.4.: Games G_2 and G_3 for the proof of Lemma 8.3. Game G_3 contains the code in the box while G_3 does not.

Scenario 1: ($|M_m| = n$). For our analysis, we distinguish between two mutually exclusive cases.

Case 1 ($U_{m+1} \in \mathfrak{B}_1$):
In this case we first consider that U_{m+1} is not fresh. This implies that the ciphertext $(C_1, \ldots, C_m)$ must be part of a common prefix of a previous query. The adversary can win only if T is a fresh value, *i.e.,* not a part of a previously occurred prefix. Since $\mathfrak{P}$ is a set of random permutations, the upper bound is then given by

$$\Pr\left[\mathfrak{P}^{-1}(U_{m+1}, T) = \tau\right] = 0.$$

Case 2 ($U_{m+1} \notin \mathfrak{B}_1$):
If U_{m+1} is fresh, we can upper bound the success probability for one query by $1/(2^n - q)$. Hence, for q queries, we can upper bound the success probability by

$$\frac{q}{2^n - q}.$$

Due to the fact that Case 1 and Case 2 are mutually exclusive, we can upper bound the success probability for q queries by

$$\max\left\{0, \frac{q}{2^n - q}\right\} \leq \frac{q}{2^n - q}.$$

Scenario 2: ($|M_m| \neq n$). As in Scenario 1 we analyze two mutually exclusive cases.

Case 1 ($U_{m+1} \in \mathfrak{B}_1$):
This case implies that the ciphertext-tag tuple $(C_1, \ldots, C_m, T^\alpha)$ must be part of a prefix previously occured query. Hence, the adversary can win only if T^β is new. Though, it is impossible, *i.e.,* for all $\forall Z \in \{0,1\}^\alpha$ it holds that

$$\Pr\left[\mathfrak{P}^{-1}(U_{m+1}, T^\beta \mid\mid Z) = \tau\right] = 0$$

since $\mathfrak{P}(U_{m+1}, \cdot)$ is a random permutation.

Case 2 ($U_{m+1} \notin \mathfrak{B}_1$):
This case implies that $C_m || T^\alpha$ must be fresh. The probability that the condition $\tau' = \tau^\alpha$ from Line 469 holds, can be upper bounded by

$$\Pr_\alpha = \max_{M_m}\left\{\Pr\left[\mathfrak{P}^{-1}(U_m, C_m \mid\mid T^\alpha) = (M_m \mid\mid \tau^\alpha)\right]\right\} \leq \frac{1}{2^{(n-|C_m|)} - q}.$$

Hence, the probability for q queries can be upper bounded by $\frac{q}{2^{(n-|C_m|)}-q}$. From the assumption $U_{m+1} \notin B_1$ follows that U_{m+1} is new. Since $\mathfrak{P}$ is a set of random permutations, we can upper bound the probability that the condition $T^\beta = T'$ from Line 469 holds by

$$\Pr_\beta = \max_Z \left\{ \Pr\left[\mathfrak{P}(U_{m+1}, \tau) = T^\beta \mid\mid Z \right] \right\} \leq \frac{1}{2^{|C_m|} - q}.$$

Then, the probability for q queries can be upper bound by $q/(2^{|C_m|} - q)$.

The (total) success probability of this case depends on the length of $|C_m|$. So we can distinguish between the following three subcases:

Subcase 2.1 ($|C_m| < n/2$):
In this case we can upper bound $\Pr_\alpha$ by $\frac{1}{2^{n/2}-q}$ and $\Pr_\beta$ by 1. Hence, the total success probability for q queries is at most $\frac{q}{2^{n/2}-q}$.

Subcase 2.2 ($|C_m| = n/2$):
In this case we can upper bound $\Pr_\alpha$ by $\frac{2}{2^{n/2}-2q}$ and $\Pr_\beta$ by $\frac{1}{2^{n/2}-q}$. Hence, the total success probability for q queries is at most $\frac{2q^2}{2^{n-1}-q^2}$.

Subcase 2.3: ($|C_m| > n/2$):
In this case we can upper bound $\Pr_\alpha$ by 1 and $\Pr_\beta$ by $\frac{1}{2^{n/2+1}-q}$. Hence, the total success probability for q queries is at most $\frac{q}{2^{n/2+1}-q}$.

Since all three subcases are mutually exclusive, we can upper bound the success probability for $q \leq 2^{n/2-2}$ queries by

$$\max\left\{ \frac{q}{2^{n/2} - q}, \frac{2q^2}{2^{n-1} - q^2}, \frac{q}{2^{n/2+1} - q} \right\} \leq \frac{q}{2^{n/2} - q}.$$

Due to the fact that Case 1 and Case 2 are mutually exclusive, we can upper bound the success probability for q queries by

$$\max\left\{ 0, \frac{q}{2^{n/2} - q} \right\} \leq \frac{q}{2^{n/2} - q}.$$

Since both scenarios are mutually exclusive, we can upper bound the success probability for q queries by

$$\frac{q}{2^{n/2} - q}.$$

Our claim follows by adding up the individual bounds. ∎

For simplification, we provide an upper bound that is easier to grasp than the original bound, but not as tight as the original bound given above.

Corollary 8.4 (Simplified ONDMA Bound). *Lets assume that $16 \leq q \leq \ell$ and the IND-PRP advantage is at most ϵ for an adversary which amount of queries is at most $2^{n/2-2}$. Then the following bound holds:*

$$\mathbf{Adv}_{\Pi}^{ONDMA}(q, \ell, t) \leq \frac{6\ell^2 + 9}{2^{n-1} - q} + \frac{q}{2^{n/2} - q} + 2\epsilon.$$

Discussion. The proofs in this chapter show that any IND-CCA-secure on-line encryption scheme can be easily transformed into a full-fleged robust OAE scheme by simply (a) prepending the associated data and (b) appending the tag generation procedure to the message.

8.3. Practical Instances

The generic McOE scheme is based on a tweakable block cipher $\widetilde{E} \in \texttt{Block}(k, n, n)$. Usually, an (efficient) tweakable block cipher is constructed out of a standard n-bit block cipher [15, 63, 152, 205]. Threefish [90] is the only native tweakable block cipher, published so far. Since the tweak size (128-bit) of Threefish is smaller than its block size (256, 512 or 1024 bit) it does not match our requirements. In the following we introduce two block cipher based instances of McOE: McOE-X and McOE-G.

8.3.1. McOE-X

The McOE-X scheme uses a regular block cipher $E \in \texttt{Block}(k, n)$ (*e.g.,* AES [71]) which is converted to a tweakable block cipher, namely TX, by mixing the tweak (*i.e.,* the chaining value) into the key K by using the XOR operation (cf. Figure 8.5). A formal definition of TX follows.

Definition 8.5 (TX). *Let $E \in \texttt{Block}(k, n)$ be a block cipher, $M, C, T \in \{0,1\}^n$, and $K \in \{0,1\}^k$. Then, the tweakable block cipher* TX-$E \in \texttt{Block}(k, n, n)$ *is defined by*

$$\text{TX-}E_K(U, M) := E_{K \oplus U}(M),$$

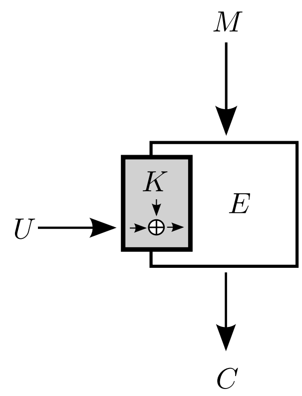

Figure 8.5.: Constructed tweakable block cipher TX.

.

and its inverse is defined by

$$\text{TX-}E_K^{-1}(U, C) := E_{K \oplus U}^{-1}(C).$$

Note that we can generalize the XOR operation between the key and the tweak by a function $\varphi : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$. For any fixed key K, $\varphi(K, \cdot)$ and the XOR operation are injective. Therefore, we can replace the XOR operation by the function φ. It is easy to see that a secure instance of TX requires related-key resistance for the block cipher E since the adversary can *partially control* some relations among the keys used in the computation. Thus, we have

$$\mathbf{Adv}^{PRP}_{(\text{TX-}E, \text{TX-}E^{-1})}(q, t) = \mathbf{Adv}^{PRP\text{-}RKA}_{E, E^{-1}}(q, O(t)).$$

Therefore, we can deduce the OCCA3 security of McOE-X from Theorem 8.1 and thus, from Corollary 8.4.

Corollary 8.6 (McOE-X Security). *For $16 \leq q \leq \ell$ and $q \leq 2^{n/2-2}$ we have*

$$\mathbf{Adv}^{ONDMA}_{\text{McOE-X}}(q, \ell, t) \leq \frac{6\ell^2 + 9}{2^{n-1} - q} + \frac{q}{2^{n/2} - q} + 2\mathbf{Adv}^{PRP\text{-}RKA}_{E, E^{-1}}(2q + \ell, O(t)).$$

Key-Recovery Attack. In [165], Mendel *et al.* showed a key-recovery attack on McOE-X with a birthday-bound complexity. The adversary is allowed to query the

McOE-X encryption oracle $\mathcal{E}$ and to access the block cipher E itself. In general, this is a reasonable assumption since we can assume E to be a common block cipher like AES. The attack works as follows:

1. Choose an arbitrary value a, compute $b_i = E_{K_i}(a)$ for $i = 1, \ldots, q$ and store all pairs (b_i, k_i) in a list $\mathfrak{L}$.

2. Choose an arbitrary constant value M_1, and set $M_2 = a$. Then, choose an arbitrary nonce value N_i. Thus, set $M = M_1 \,||\, M_2$ and ask for $C = \mathcal{E}(N_i, M)$ with $C = C_1 \,||\, C_2$.

3. If $C_2 \in \mathfrak{L}_{|b}$, compute the secret key by $K = k_i \oplus M_1 \oplus C_1$, otherwise, go back to Step 2.

Let us denote q' as the number of iterations for the loop described in Steps 2 and 3. The total number of block cipher invocations ℓ is restricted to $\ell \leq q + 4q'$ since for one query to $\mathcal{E}$, four block cipher calls are necessary to compute the pair (C, T). If we choose $\ell \approx 2^{n/2}$, there exists an adversary with a success probability of at most $1/2$, which is able to recover the secret key K using the presented attack. Note that this attack does only confirm with our security claim, but also requires more queries then our security bound. Nevertheless, this attack shows that it is very crucial to change the cipher key after $\ll 2^{n/2}$ invocations of the block cipher $\mathcal{E}$. The proposed attack can be avoided by increasing the key size to, *e.g.,* $2n$.

8.3.2. McOE-G

The McOE instance McOE-G updates the chaining value by applying an almost-XOR-universal (ϵ-AXU) hash function $\mathcal{H}$ to the XOR result of the previous message block and ciphertext block (see Figure 8.6). In our practical implementation, we use the Galois-Field multiplication for $\mathcal{H}$, *i.e.,* the key K_2 is multiplied with the chaining value over GF(2^{128}) defined by the low-weight irreducible polynomial $g(x) = x^{128} + x^7 + x^2 + x + 1$ as used in OCB [208] and GCM [164].

Definition 8.7 (TG). *Let $E \in \mathtt{Block}(k, n)$ be a block cipher, and $\mathcal{H} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be an ϵ-AXU hash function. Suppose $M, C, T \in \{0,1\}^n$ and $(K_1, K_2) \in \{0,1\}^k \times \{0,1\}^n$ with $K = K_1 \,||\, K_2$. Then, the tweakable block cipher* TG-$E \in \mathtt{Block}(k + n, n, n)$ *is defined by*

$$\text{TG-}E_K(U, M) = E_{K_1}(M \oplus \mathcal{H}_{K_2}(U)) \oplus \mathcal{H}_{K_2}(U),$$

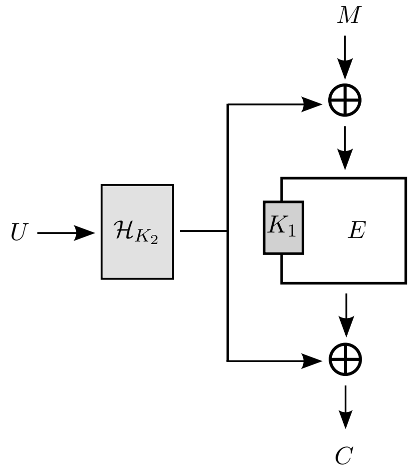

Figure 8.6.: Constructed tweakable block cipher TG.

.

and its inverse is defined by

$$\text{TG-}E_K^{-1}(U, C) = E^{-1}(C \oplus \mathcal{H}_{K_2}(U)) \oplus \mathcal{H}_{K_2}(U).$$

Liskov *et al.* showed in Theorem 2 of [153] that TG is a secure tweakable block cipher with

$$\mathbf{Adv}^{PRP}_{\text{TG-}E,\text{TG-}E^{-1}}(q, t) \leq 3q^2\epsilon \cdot \mathbf{Adv}^{PRP}_{E,E^{-1}}(q, O(t)).$$

Therefore, we can deduce the ONDMA security of McOE-X from Theorem 8.1 and from Corollary 8.4.

Corollary 8.8 (McOE-G Security). *For* $16 \leq q \leq \ell$ *and* $q \leq 2^{n/2-2}$, *we have*

$$\mathbf{Adv}^{OCCA3}_{\text{McOE-G}}(q, \ell, t) \leq \frac{6\ell^2 + 9}{2^{n-1} - q} + \frac{q}{2^{n/2} - q} + 6q^2\epsilon \cdot \mathbf{Adv}^{PRP}_{E,E^{-1}}(2q + \ell, O(t)).$$

Remark. McOE-G is not secure if an adversary has oracle access to the internal building blocks (*i.e.*, the block cipher E and the ϵ-AXU hash function $\mathcal{H}$) [45]. Hence, it is crucial that an adversary has only black-box oracle access to the tweakable block cipher $\widetilde{E}$.

Block cipher	Impl.	Message length in bytes 64	256	512	1024	4096	16384	32768
McOE-X-AES	software	27.0	20.6	19.5	19.0	18.6	18.6	18.6
McOE-X-AES	AES-NI	13.7	10.9	10.5	10.2	10.0	10.0	10.0
McOE-X-Threefish	software	15.6	7.9	6.7	6.1	5.6	5.5	5.5
McOE-G-AES	software	32.3	24.9	23.9	23.1	22.6	22.5	22.5
McOE-G-AES	GF-NI/AES-NI	12.5	9.9	9.4	9.2	9.0	9.0	9.0
AES-CBC encryption	software	18.6	18.0	16.4	15.5	14.2	11.4	11.4
AES-CBC encryption	AES-NI	5.1	5.9	5.7	5.4	4.1	4.1	4.1

Table 8.2.: Performance values (cpb, single core), measured on a Core i5 540M for AES-128 and Threefish-512.

8.4. Benchmarks

This section provides software-performance benchmarks of the two presented members of the McOE family, *i.e.,* McOE-X and McOE-G. All measurement results are based on the real-time clock (RTC) and obtained by the median of 5,000 measurements of the target function. The performance values are given in Cycles per Byte (cpb). For the sake of comparison, we also provide performance benchmarks for AES-CBC, a common encryption scheme without authentication, standardized by the National Institute of Standards and Technology (NIST) [84]. The AES software implementation is based on Gladman [111], whereas the hardware implementation is based on the Intel AES-NI Sample Library [66]. The Threefish implementation is based on the NIST/SHA-3 reference source as provided by the Skein authors [178]. Finally, the implementation of Galois-Field multiplication uses Intels carry-less multiplication instruction `PCLMULQDQ` that allows to compute the carry-less product of two 64-bit operands in about 3.54 cpb [119]. Note that all performance benchmarks are based on *naive* implementations based on reference code. Therefore, it is most likely that the benchmarks can be further improved by the usage of sophisticated optimization methods.

Target Platform. The benchmarks were performed on a single core of an Intel Core i5-3210M CPU 2.50GHz computer. All software benchmarks were written in C or ASM and compiled with the GNU C compiler (gcc) version 4.8.2 using the optimization flag -O3.

Results. The results of the 64-bit performance benchmarks are summarized in Table 8.2.

Further Implementation Results. Bogdanov *et al.* recently published a high-speed implementation of McOE-G optimized for Haswell CPUs [50]. Their implementation runs at about 6.24 cpb on a single core of an Intel Core i5-4300U CPU (1900 MHz) processor. In addition, Bogdanov *et al.* also performed benchmarks in the multi-message scenario. Here, McOE-G matches the performance of GCM at about 1.45 cpb. Their results show that scenarios exist where a conventional OAE scheme can be replaced by a robust one without noticeable performance loss.

8.5. Results Summary

Originally, our research was inspired by the search for an authenticated encryption scheme that can be used in a general-purpose cryptographic library. It should offer by default a huge failure tolerance for practical software developers and still allow being used in an on-line manner.

Since the well-known schemes (such as OCB and SIV) did not fit our requirements, we developed McOE– the first robust OAE scheme. Furthermore, it is provably secure in the standard model and fast enough for most common applications like (full) hard-disk encryption or secure network communication. This renders McOE a well-suited candidate for any general-purpose cryptographic library.

Part III

Design and Usage of Cryptographic Hash Functions

9 Twister$_\pi$ – A Framework for Fast and Secure Hash Functions

> A person who never made a mistake never tried anything new.
>
> Albert Einstein

In this chapter we present TWISTER$_\pi$, a framework for hash functions. It is an improved version of TWISTER, one of the 51 accepted fist-round participants of the SHA-3 Competition. TWISTER$_\pi$ is built upon the ideas of wide-pipe and sponge functions. The core of this framework is a – very easy to analyze – `Mini-Round`, providing both fast diffusion as well as collision-freeness. The total security level is claimed to be not below $2^{n/2}$ for collision attacks and 2^n for (2nd-) preimage attacks. TWISTER$_\pi$ instantiations are secure against all known generic attacks. We also propose two instances TWISTER$_\pi$-n for hash output sizes $n = 256$ and $n = 512$. These instantiations are highly optimized for 64-bit architectures and run fast in hardware and software. Furthermore, TWISTER$_\pi$ scales very well on low-end platforms.

Related Work. In the last decade, design flaws in popular hash functions such as MD5 [200] and SHA-0 [185] were exposed, leading to a huge amount of attacks for SHA-1 [183] [39, 40, 62, 80, 199, 234–236]. But, also newer hash functions, *e.g.,* [12, 121, 140, 141] – which try to take care for weaknesses in the *Merkle-Damgård* construction itself – were broken soon after their publications [118, 162, 168, 192, 193].

Back then, some cryptographers were worried that the standardized SHA-2 [183, 184] family could also be vulnerable to state-of-the-art techniques in cryptanalysis.

Therefore, in 2008, the NIST started the SHA-3 Competition [181] with the goal to find a successor for the SHA-2 family. On October 2, in 2012, the NIST announced Keccack [34], a sponge-function based software, as the winner of the SHA-3 Competition. The concept of sponge functions [36] is given for example by a big internal state that absorbs a message of infinite length and that later squeezes out a hash value of variable size. In 2014, it will become the official SHA-3 standard.

Our Contribution. The design of secure and practical hash functions is of great interest. Due to the SHA-3 Competition, many new proposals for hash functions have been published during this process. In this chapter we present a new hash function framework called TWISTER$_\pi$. Our proposal is based on a sponge construction [37] as well as on the wide-pipe approach [155]. The main goal is to present a fast and secure hash function which is flexible to use and easy to analyze.

More precise, it uses XOR-sponges with a big internal state as proposed in [155]. The GRINDAHL design [141], which is the closest to our approach, but contains some flaws which cannot be exploited in TWISTER$_\pi$. We take advantage of the well studied basic operations of AES [71] and adopt several of them, including some optimization techniques.

Due to recent breakdowns of many proposed hash functions, we analyze the resistance of TWISTER$_\pi$ against all known generic attacks on hash functions. We show that the TWISTER$_\pi$ framework resists all of them if the size of the internal chaining value is at least double the size of the hash output.

In 2009, Mendel *et al.* discovered a serious design flaw of TWISTER [166], the predecessor of TWISTER$_\pi$, leading to several attacks against TWISTER-512 [166]. Those attacks are not transferable to TWISTER$_\pi$-512 since it was especially designed to resist them.

Outline. Section 9.1 points out the principles of TWISTER$_\pi$. Section 9.2 specifies TWISTER$_\pi$ and Section 9.3 briefly discusses its resistance against generic attacks. Section 9.4 shows some optimization techniques related to the implementation of TWISTER$_\pi$ on different platforms, and Section 9.5 provides software-performance benchmarks. Section 9.6 introduces the differences between TWISTER$_\pi$ and its predecessor TWISTER [93]. Finally, Section 9.8 summarizes our contribution.

9.1. Design Principles of TWISTER$_\pi$

In this section we explain our design purpose and point out the principles of the TWISTER$_\pi$ design.

Security. The concept of Maximum Distance Separable (MDS) matrices allows us to obtain a maximum of diffusion inside each column of the state matrix. Since the message input is orthogonal with the diffusion of the state, *i.e.,* a message word is always injected into the last row of our state matrix, we allow a minimum of control on the state for an adversary. Introducing a local feed-forward as well as the `Blank-Round` (`Twister-Round` with no message input), furthe reduces the influence of an adversary on the state.

Evolutionary. Throughout the last decade, a lot of hash functions using many different concepts have been broken. And often we had to learn that using newly developed techniques lead to stronger attacks, which render pretended strong hash functions weak.

The well-studied and analyzed block ciphers that were in the final round of the AES Competition[1] lead to some well established design principles offering a high level of cryptographic knowledge. Therefore, Rijndael [70] can be seen as one of the most studied block ciphers during this process and also in the time after. Its concepts of simple byte-wise operations `SubBytes`, `ShiftRows`, and `MixColumns`, are well-analyzed and it turns out that their combination can offer a high level of speed and some form of provable security. We adopt a few of these concepts for TWISTER$_\pi$ and we also learn from recent hash function breakouts.

Simplicity. A strong hash function should not be hard to analyze since not finding an attack due to the algorithmic complexity does not mean that there is no simple attack which breaks the whole function in an easy way. We therefore only use simple and few components as building blocks for the `Twister-Round`. These components have been studied before and are well known – but combining these components to obtain a good hash function is new. Our clear design and straightforward structure of TWISTER$_\pi$ makes cryptanalysis easy and serves the purpose that there are no simple attacks which cannot be found due to a complex and unreadable algorithm.

[1] `http://csrc.nist.gov/archive/aes/`

Portability and Scalability. A main design criteria of TWISTER$_\pi$ is its application to a wide range of applications. Due to its byte-wise operations, it scales very neat on 8-, 16-, 32-, and 64-bit platforms. TWISTER$_\pi$ can be very efficiently applied on smart cards with small 8-bit processors. We also offer an optimized version for 32-bit and 64-bit environments. The portability will be enhanced by its low-memory requirements, which makes TWISTER$_\pi$ even valuable for low-end platforms.

Analyzability. TWISTER$_\pi$ consists of well-known and well-analyzed components. The security level of TWISTER$_\pi$ can be proven for the inner components, which is more worth than just a security claim. Using the concept of an MDS matrix, lead to a very fast diffusion. After only two `Twister-Round` invocations, a full diffusion is guaranteed (if no feed-forward has taken place). This high level of diffusion makes TWISTER$_\pi$ very close to a randomized hash function offering a high level of security.

9.2. Specification of the TWISTER$_\pi$ Hash-Function Family

Algorithm 11 TWISTER$_\pi$

Input: M {Message to Hash}, n {Output Length}
Output: Y {Hash Value}
  $T \leftarrow 0$
  $S \leftarrow$ `Init`(π)
  $S_{(1\rightarrow)} \leftarrow S_{(1\rightarrow)} \oplus n$
  if $n \leq 256$ **then**
    $T \leftarrow$ `null`
  end if
  $M' \leftarrow M \mid\mid 10^*$
  for $i = 1, \ldots, |M'|/512$ **do**
    $(S, T) \leftarrow$ `Twister-Round`(S, T, M'_i)
  end for
  $S \leftarrow$ `State-Finalization`$(S, T, |M|, n)$
  return $Y \leftarrow$ `Output-Round`(S, n)

In this section we specify the overall structure and the individual building blocks of TWISTER$_\pi$, a byte-oriented hash function family, operating on a square state matrix S. The core primitive of each individual TWISTER$_\pi$ hash function is the underlying compression function, `Twister-Round`, processing 512-bit message blocks together with a 512-bit state (*i.e.,* the chaining value) and outputs a 512-bit value. Messages are padded using the 10*-Padding-Rule (cf. Definition 2.7) to become a multiple of 512 bits. The compression function is based on an AES-like round function `Mini-Round`, processing a 64-bit message word. The 512-bit checksum T can be seen as an optional parameter, only needed for computing a message digest greater than 256 bits, otherwise it is set to `null`. It serves as an additional state to preserve our wide-pipe approach where the state is at least twice as large as the message digest. In the following, `Twister-Round` (S,M_i) denotes the invocation of `Twister-Round` where the parameter T is either `null` or omitted. Thus,

$$\texttt{Twister-Round}(S, M_i) = \texttt{Twister-Round}(S, \texttt{null}, M_i).$$

The finalization phase starts after the padded message was fully processed, *i.e.,* the message is completely absorbed by the state S. At first, the message length and the checksum – if present – is absorbed into the state by the means of the `State-Finalization`. Finally, the message digest is computed by following the design ideas of the sponge function [36]; instead of presenting the complete internal state to the attacker, the `Output-Round` computes as many 64 bit output slices as needed. A description of TWISTER$_\pi$ in pseudo-code notation is shown in Algorithm 11. Next, we give an in-depth description of the individual components of TWISTER$_\pi$.

9.2.1. Context

The State S. TWISTER$_\pi$ operates on a square state matrix $S = (S_{i,j})$, $1 \leq i, j \leq 8$, consisting of eight rows and columns, where each cell $S_{i,j}$ represents one byte.

$S_{1,1}$	$S_{1,2}$	$\dots$	$S_{1,8}$
$S_{2,1}$	$S_{2,2}$	$\dots$	$S_{2,8}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$
$S_{8,1}$	$S_{8,2}$	$\dots$	$S_{8,8}$

Note, $S_{(i\rightarrow)} := (S_{i,1}, \dots, S_{i,8})$ denotes the i-th row vector and $S_{(j\downarrow)} := (S_{1,j}, \dots, S_{8,j})$ the j-th column vector. similar to Blowfish [220], the initial state of TWISTER$_\pi$ is given by the first 64 hex digits of the fractional portion of π. After the initialization the internal state S is given by the following matrix:

24	3F	6A	88	85	A3	08	D3
13	19	8A	2E	03	70	73	44
A4	09	38	22	29	9F	31	D0
08	2E	FA	98	EC	4E	6C	89
45	28	21	E6	38	D0	13	77
BE	54	66	CF	34	E9	0C	6C
C0	AC	29	B7	C9	7C	50	DD
3F	84	D5	B5	B5	47	09	17

Checksum T**.** The checksum enlarges the state of Twister$_\pi$-384 and Twister$_\pi$-512 to stick to our wide-pipe design [155] decision. In other words: using the checksum, we can double size of the internal state.

similar to the state S, the checksum T is represented by a square matrix $T = (T_{i,j})$, $1 \leq i, j \leq 8$, consisting out of eight rows and columns, where each cell $T_{i,j}$ represents one byte.

$T_{1,1}$	$T_{1,2}$	$\dots$	$T_{1,8}$
$T_{2,1}$	$T_{2,2}$	$\dots$	$T_{2,8}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$
$T_{8,1}$	$T_{8,2}$	$\dots$	$T_{8,8}$

We define a checksum-update operation as

$$T_{(i\downarrow)} = (3T_{(i\downarrow)}) \oplus (T_{(i+1\downarrow)} \boxplus S_{(i\downarrow)}),$$

where $\boxplus$ denotes an addition modulo 2^{64}. The initial checksum state is given by the all zero state, *i.e.*, $T_{i,j} = 0$ with $1 \leq i, j \leq 8$

9.2.2. Mini-Round

The `Mini-Round` (cf. Algorithm 12) is the basic building block of any Twister$_\pi$ hash function. The design of this primitive follows the lead of the AES round transformation and thus, prefers simple components over complex ones. The main purpose of this non-linear permutation is to inject a message word W, (`InjectMessage`) and to take care of the diffusion of the state matrix S. The core of the `Mini-Round` is the `MixColumns` operation where S is multiplied with an MDS matrix to achieve a

Algorithm 12 `Mini-Round`

Input: S {State}, W {Message word}
Output: S {Updated state}
$S \leftarrow$ `InjectMessage`(S, W)
$S \leftarrow$ `AddTwistCounter`(S)
$S \leftarrow$ `SubBytes`(S)
$S \leftarrow$ `ShiftRows`(S)
$S \leftarrow$ `MixColumns`(S)
return S

proper diffusion. The `Mini-Round` is visualized in Figure 9.1. TWISTER$_\pi$ can handle at most 2^{64} `Mini-Rounds`. This limitation is caused by the `AddTwistCounter` operation where a 64-bit counter is added. Each `Mini-Round` can process 64 bits of message data. Therefore, with a native usage of a `Mini-Round` it is possible to process up to $2^{64} \cdot 64$ message bits. If this limitation becomes a real world issue in the future, it is possible to increase the size of the `TwistCounter` to 128s bit with almost no performance loss.

InjectMessage (IM). A 64-bit message word W is XOR-injected into the last row. Let $W = W[1], \ldots, W[8]$ where $W[i]$ denotes the i-th significant byte of W, *e.g.,* $W[8]$ denotes the most significant byte of W. Then, we define the message-injection process by

$$S_{(8\rightarrow)} \oplus W := S_{(8,1)} \oplus W[1] \mid\mid \cdots \mid\mid S_{(8,8)} \oplus W[8].$$

AddTwistCounter (AC). The `TwistCounter` ctr is an unsigned 64-bit integer initialized by the maximum value, *i.e.,* `0xFFFF_FFFF_FFFF_FFFF`. It is XORed byte by byte into the first row of the state S.

$$S_{(1\rightarrow)} \oplus ctr := S_{(1,1)} \oplus ctr[1] \mid\mid \ldots \mid\mid S_{(1,8)} \oplus ctr[8]$$

After successful addition, ctr is decreased by 1.

SubBytes (SB). This function is defined as a bijection

$$\texttt{SubBytes} : \{0,1\}^8 \rightarrow \{0,1\}^8$$

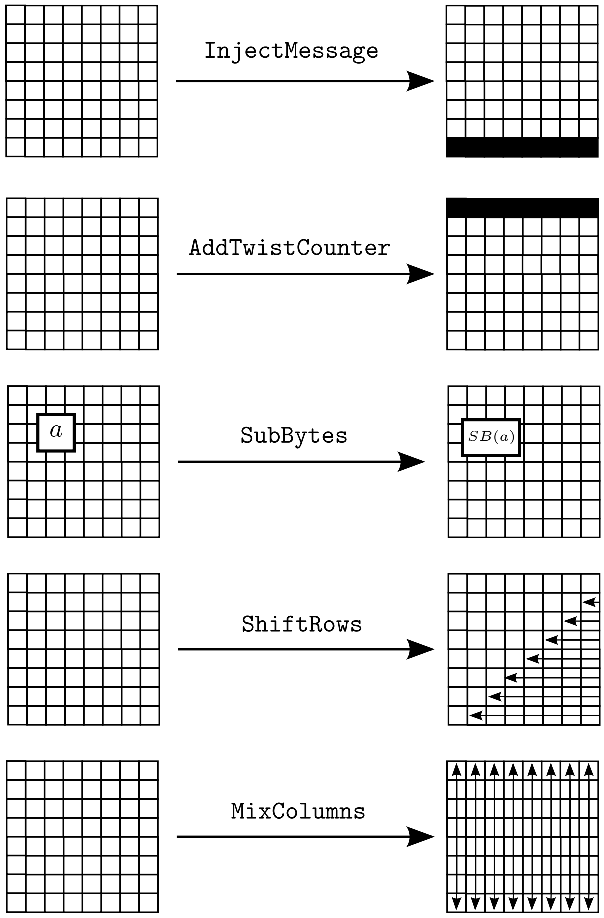

Figure 9.1.: Illustration of a `Mini-Round`.

and is used as an S-box for each byte. It should, among other properties, be highly non-linear. A discussion on how to obtain such cryptographically strong S-boxes (for $8x8$ S-boxes) can be found in [241]. Twister$_\pi$ uses the well-known and intensively studied AES S-Box which can be found in [71].

We define the `SubBytes` operation by

$$S_{(i,j)} := SB(S_{(i,j)}) \quad \text{with} \quad 1 \leq i, j \leq 8.$$

ShiftRows (SR). ShiftRows is a cyclic left shift similar to the ShiftRows operation of AES. It rotates Row j by $(j-1) \bmod 8$ bytes to the left. Suppose $S_{\leftarrow j(i\rightarrow)}$ denotes the i-th row rotated by j bytes to the left. Then, the `ShiftRows` operation is defined by

$$S_{(i\rightarrow)} := S_{\leftarrow i-1(i\rightarrow)} \quad \text{with} \quad 1 \leq i \leq 8.$$

MixColumns (MC). The `MixColumns` step is a permutation operation on the state. It applies a $N \times N$-MDS A (a maximum distance separable matrix as defined below) to each column, *i.e.,* $A \cdot S_{(j\downarrow)}$ for $1 \leq j \leq 8$.

Definition 9.1 (MDS Matrix). *An* $[n, k, d]$*-code with a generator matrix*

$$G = \left[I_{k\times k} \; A_{k\times(n-k)}\right]$$

is an MDS code if every square submatrix of A *is non-singular,* i.e., $d \neq 0$*, where* d *denotes the determinant of* A*. The matrix* A *is called an MDS matrix.*

The MDS matrix chosen for $\textsc{Twister}_\pi$ is cyclic, *i.e.,* its i-th row can be obtained by a cyclic right rotation of (02 01 01 05 07 08 06 01) by i entries. It has a branch number of 9 meaning that if two 8-byte input vectors differ in $1 \leq k \leq 8$ bytes, the output of the `MixColumns` operation differs in at least $9 - k$ bytes. More precisely, the approximate probability that two 8-byte input words with D_I different bytes on predefined positions maps to two 8-byte output words with D_O different bytes on predefined positions by the `MixColumns` operation is given in Table 9.1. The 8×8-MDS matrix used for all proposed instances of $\textsc{Twister}_\pi$ is:

$$\text{MDS} = \begin{pmatrix} 02 & 01 & 01 & 05 & 07 & 08 & 06 & 01 \\ 01 & 02 & 01 & 01 & 05 & 07 & 08 & 06 \\ 06 & 01 & 02 & 01 & 01 & 05 & 07 & 08 \\ 08 & 06 & 01 & 02 & 01 & 01 & 05 & 07 \\ 07 & 08 & 06 & 01 & 02 & 01 & 01 & 05 \\ 05 & 07 & 08 & 06 & 01 & 02 & 01 & 01 \\ 01 & 05 & 07 & 08 & 06 & 01 & 02 & 01 \\ 01 & 01 & 05 & 07 & 08 & 06 & 01 & 02 \end{pmatrix}$$

All of the byte-entries are considered to be elements of $GF(2^8)$. An element of $GF(2^8)$ is represented by $\sum_{i=0}^{7} a_i 2^i$. The reduction polynomial $R(x)$ of $GF(2^8)$ is defined by

$$R(x) = x^8 + x^6 + x^3 + x^2 + 1.$$

A detailed discussion about the properties of MDS matrices/codes is given in [158].

D_I/D_O	0	1	2	3	4	5	6	7	8
0	1	- ∞	- ∞	- ∞	- ∞	- ∞	- ∞	- ∞	- ∞
1	- ∞	- ∞	- ∞	- ∞	- ∞	- ∞	- ∞	- ∞	1
2	- ∞	- ∞	- ∞	- ∞	- ∞	- ∞	- ∞	2^{-8}	0.99
3	- ∞	- ∞	- ∞	- ∞	- ∞	- ∞	2^{16}	2^{-8}	0.99
4	- ∞	- ∞	- ∞	- ∞	- ∞	2^{-24}	2^{16}	2^{-8}	0.99
5	- ∞	- ∞	- ∞	- ∞	2^{32}	2^{-24}	2^{16}	2^{-8}	0.99
6	- ∞	- ∞	- ∞	2^{-40}	2^{32}	2^{-24}	2^{16}	2^{-8}	0.99
7	- ∞	- ∞	2^{-48}	2^{-40}	2^{32}	2^{-24}	2^{16}	2^{-8}	0.99
7	- ∞	2^{-56}	2^{-48}	2^{-40}	2^{32}	2^{-24}	2^{16}	2^{-8}	0.99

Table 9.1.: Column properties of the state matrix after multiplication with an MDS matrix.

9.2.3. Compression Function

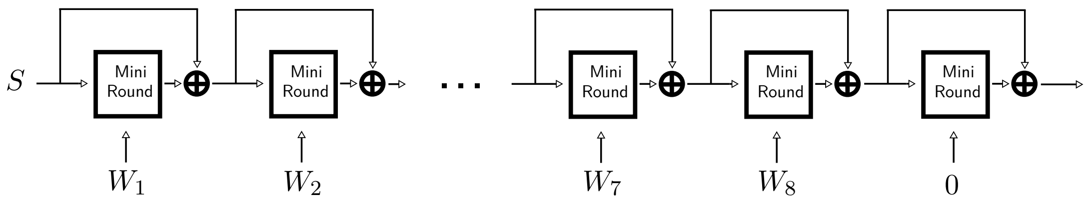

Figure 9.2.: Illustration of a `Twister-Round`.

The compression function

$$\texttt{Twister-Round} : \{0,1\}^{512} \times [\{0,1\}^{512}] \times \{0,1\}^{512} \to \{0,1\}^{512}$$

maps three 512-bit input values (*i.e.,* the state, an optional checksum and a message block) to a 512-bit output value. It consists of nine `Mini-Rounds`. Each of the first eight rounds absorbs a 64-bit word (*i.e.,* $W_1, \ldots, W_8$) of the message block into the state. The last `Mini-Round` is invoked with a virtual all-zero message word to limit an adversaries control over the internal state of the hash function. Such a `Blank-Round` is a common building component in the design of cryptographic hash functions [35, 141, 180]. The individual `Mini-Rounds` are separated by a feed-forward operation with the state before its invocation (cf. Figure 9.2) to guarantee the one-wayness of `Twister-Round` since the remaining `Mini-Round` operations are invertible. The feed-forward operation is defined by

$$S := S \oplus \texttt{Mini-Round}(S, X),$$

Algorithm 13 `Twister-Round`

Input: S {State}, T {Checksum}, W {Message Block}
Output: S {Updated State}, T {Updated Checksum}

for $i = 1, \ldots, 8$ **do**
  if $T \neq$ `null` **then**
    $T_{(i\downarrow)} \leftarrow 3 \cdot T_{(i\downarrow)} \oplus (T_{(i+1)\downarrow} \boxplus S_{(i\downarrow)})$
  end if
  $S \leftarrow$ `Mini-Round`$(S, W_i) \oplus S$
end for
$S \leftarrow$ `Mini-Round`$(S, 0) \oplus S$
return (S,T)

where X is either a message word or a sequence of 64 zero-bits. The optional checksum for computing a message digest longer than 256 bits is updated before processing a message word. A description of `Twister-Round` in pseudo-code notation is given in Algorithm 13.

9.2.4. Post-Processing

This section describes the TWISTER$_\pi$ finalization process. It starts after the message is completely processed by iterating the compression function over all message blocks. The post-processing consists of the following two steps:

State Finalization. At first, to prevent length-extension attacks (see Section 2.3), the state is updated by processing the bit-length $|M|$ of the unpadded message M, encoded as a 64-bit value, together with the current state by means of a `Mini-Round` (*i.e.*, $S \leftarrow$ `Mini-Round`$(S, |M|)$). Afterwards, the state finalization ends with either a `Blank-Round` or the processing of the checksum with the state by means of a `Twister-Round`, depending on the length of the message digest. In the latter case, the checksum T is transformed in a 64-byte message block $M = M[1], \ldots, M[64]$ column by column where

$$(M[(i \cdot 8) - 7], \ldots, M[i \cdot 8]) \leftarrow T_{(i\downarrow)} \text{ with } 1 \leq i \leq 8.$$

A formal definition of `State-Finalization` is given in Algorithm 14.

Message Digest Computation. The task of `Output-Round` is the computation of the actual message digest from a given state S. A description in pseudo-code notation

Algorithm 14 `State-Finalization`

$S \leftarrow$ `Mini-Round`$(S, |M|)$
if $n \leq 256$ **then**
  $S \leftarrow$ `Mini-Round`$(S, 0)$
else
  $S \leftarrow$ `Twister-Round`$(S,$ `null`$, T)$
end if

is shown in Algorithm 15. For every 64 bits of the message digest, the current state is first saved, and then updated by a `Blank-Round` followed by a feed-forward operation with the saved state, and finally, an additional invocation of a `Blank-Round` (see Lines 3-6). A 64-bit output value is then obtained by XOR-ing the first column of S with the first column of the saved state (see Line 7). This procedure is repeated until the needed amount of message digest bits is obtained. The last output stream can be varied between 32 bits and 64 bits by taking only the first half of the output value. This allows to vary the output size for a huge amount of applications. Note that due to `Output-Round`, TWISTER$_\pi$ can theoretically produce hash values up to 2^{64} bits. This limitation results from the initial step where the output length is written into to first row of the state (cf. Algorithm 11). In some particular scenarios, long hash values can become handy, *e.g.,* full domain hashing [26]. Anyway, the security of TWISTER$_\pi$ is limited by the state size.

Algorithm 15 `Output-Round`

Input: S {State}, n {Output Length}
Output: Y {Hash Value}
1: $Y \leftarrow \emptyset$
2: **for** $i = 1, \ldots, \lceil n/64 \rceil$ **do**
3:   $X \leftarrow S$
4:   $S \leftarrow$ `Mini-Round`$(S, 0) \oplus S$
5:   $S \leftarrow$ `Mini-Round`$(S, 0)$
6:   $Y \leftarrow Y \mid\mid (S_{(1\downarrow)} \oplus X_{(1\downarrow)})$
7: **end for**

9.3. Security against Generic Attacks

In this section we give a brief discussion why TWISTER$_\pi$ is resistant to existing generic attacks.

Length-Extension Attacks. The combination of the TWISTER$_\pi$ padding rule and the processing of the message length in the post-processing phase avoids such type of attacks. Another possible attack can be as follows:

For a known hash value $\mathcal{H}(M)$, one can compute the hash value $\mathcal{H}(M \mid\mid Y \mid\mid Z)$ for any suffix Z if the length of an unknown message M is known as well as the padding Y of M. TWISTER$_\pi$ is secure against such attacks due to two countermeasures: (1) By knowing only the hash value, an attacker cannot easily determine the state S after the last compression function call as it has only access to the hash value generated by the `Output-Round`, which squeezes out some bits of the state by applying the output transformation. The bits of the squeezing process do not leave enough information to recover the internal state; (2) The multiple feed-forward does also prevent any attacker to successfully gain any knowledge about prior state information. In each squeezing process, one feed-forward is applied.

Multi-Collision Attacks. An instance of TWISTER$_\pi$ fully resists a multi-collision attack if $8 \cdot 8^2 = 512 \geq n$ since the complexity is determined by $k \cdot 2^{512/2}$. All instances of TWISTER$_\pi$ have this feature, although the state of TWISTER$_\pi$-384 and TWISTER$_\pi$-512 is not big enough to prevent this attack by itself, but including the checksum can be viewed as an enlargement of the state which then provides resistance against this kinds of attacks.

Herding Attacks. For TWISTER$_\pi$-256 (TWISTER$_\pi$-512), we have an internal state of 512 bits ($|S| + |T| = 1024$) and with $512 > \frac{3 \cdot 256 - 5}{2} = 381.5$ bits ($1024 > \frac{3 \cdot 512 - 5}{2} = 765.5$). The attack has the same complexity as for a (2nd-) preimage attack on a random oracle. The complexity of this attack decreases with increased size of the message. If the message is of size about 2^ℓ, the complexity of the attack is $2^{(2n-5)/3-\ell}$. It is easy to see that all of our proposed instances of TWISTER$_\pi$ provides resistance against this kind of attacks.

Long 2nd-preimage Attacks. Long 2nd-preimage attacks cannot be applied to the TWISTER$_\pi$ framework for three reasons. First, in each `Mini-Round`, the `Twist-Counter` *ctr* is added to the second column of the state S which does not allow to find expandable messages. Second, TWISTER$_\pi$ uses multiple feed-forwards and third, the internal chaining value is in general much larger than n. This makes it harder to find collisions and fix points since we essentially have constructions similar to the wide-pipe design [155].

Slide Attacks. The `TwistCounter` *ctr* prevents slide attacks since in each iteration of the `Mini-Round`, a fresh value is injected into the state matrix which does not allow an adversary to find slid pairs of messages. Furthermore, the last inserted message block cannot be the all-zero block due to the padding rules. Thus, slide attacks are not possible for TWISTER$_\pi$.

9.4. Implementation Details

In this section we discuss issues related to the implementation of TWISTER$_\pi$ on different platforms. Test vectors to verify a specific implementaion are given in Appendix A.

In essence, our techniques for implementing this cryptographic hash function rely on the following key sources of information:

- Optimization techniques as given in [71] and
- some of the new techniques on how to reduce the number of instructions for an AES implementation as given in [33].

Most of the discussed issues are relevant for more than one platform.

Note that there are no multiplications of two arbitrary values of $GF(2^8)$, but only multiplications of a variable with some fixed constants. The latter is easier to implement than the former – especially in the context of hardware and high-speed software implementations.

9.4.1. 64-Bit Platforms

All steps of the round transformation, (*i.e.,*`SubBytes`, `ShiftRows`, and `MixColumns`) can be combined in a single set of lookup tables, allowing for very fast implementations on processors with word length greater equal 64 bits. The following notations will be used for the elements at matrix position (i, j) and for (for $1 \leq i, j < 8$):

$a_{i,j}$ input state matrix element,

$b_{i,j}$ state matrix element after `SubBytes`,

$c_{i,j}$ state matrix element after `ShiftRows`,

$d_{i,j}$ state matrix element after `MixColumns`.

After finishing the `MixColumns` operation, we have for $1 \leq j \leq 8$:

$$\begin{bmatrix} d_{1,j} \\ d_{2,j} \\ d_{3,j} \\ d_{4,j} \\ d_{5,j} \\ d_{6,j} \\ d_{7,j} \\ d_{8,j} \end{bmatrix} = \begin{bmatrix} 02 & 01 & 01 & 05 & 07 & 08 & 06 & 01 \\ 01 & 02 & 01 & 01 & 05 & 07 & 08 & 06 \\ 06 & 01 & 02 & 01 & 01 & 05 & 07 & 08 \\ 08 & 06 & 01 & 02 & 01 & 01 & 05 & 07 \\ 07 & 08 & 06 & 01 & 02 & 01 & 01 & 05 \\ 05 & 07 & 08 & 06 & 01 & 02 & 01 & 01 \\ 01 & 05 & 07 & 08 & 06 & 01 & 02 & 01 \\ 01 & 01 & 05 & 07 & 08 & 06 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} SB[a_{1,j}] \\ SB[a_{2,j+1}] \\ SB[a_{3,j+2}] \\ SB[a_{4,j+3}] \\ SB[a_{5,j+4}] \\ SB[a_{6,j+5}] \\ SB[a_{7,j+6}] \\ SB[a_{8,j+7}] \end{bmatrix}$$

where $SB : \{0,1\}^8 \rightarrow \{0,1\}^8$ denotes the S-Box operation, and where '+' denotes a *wraparound addition*, *e.g.,* $j + 6 \equiv 2$ for $j = 4$. This matrix multiplication can be interpreted as a linear combination of all eight column vectors:

$$\begin{bmatrix} d_{1,j} \\ d_{2,j} \\ d_{3,j} \\ d_{4,j} \\ d_{5,j} \\ d_{6,j} \\ d_{7,j} \\ d_{8,j} \end{bmatrix} = \begin{bmatrix} 02 \\ 01 \\ 06 \\ 08 \\ 07 \\ 05 \\ 01 \\ 01 \end{bmatrix} SB[a_{1,j}] \oplus \begin{bmatrix} 01 \\ 02 \\ 01 \\ 06 \\ 08 \\ 07 \\ 05 \\ 01 \end{bmatrix} SB[a_{1,j+1}] \oplus \ldots \oplus \begin{bmatrix} 01 \\ 06 \\ 08 \\ 07 \\ 05 \\ 01 \\ 01 \\ 02 \end{bmatrix} SB[a_{1,j+7}].$$

We now define eight V-tables: $V_1, V_2, \ldots, V_8$:

$$V_1[\alpha] = \begin{bmatrix} 02 \\ 01 \\ 06 \\ 08 \\ 07 \\ 05 \\ 01 \\ 01 \end{bmatrix} SB[\alpha], \quad V_2[\alpha] = \begin{bmatrix} 01 \\ 02 \\ 01 \\ 06 \\ 08 \\ 07 \\ 05 \\ 01 \end{bmatrix} SB[\alpha], \quad \ldots \quad V_8[\alpha] = \begin{bmatrix} 01 \\ 06 \\ 08 \\ 07 \\ 05 \\ 01 \\ 01 \\ 02 \end{bmatrix} SB[\alpha].$$

It follows that we can write the combined operation of `SubBytes`, `ShiftRows`, and `MixColumns` as

$$\begin{bmatrix} d_{1,j} \\ d_{2,j} \\ d_{3,j} \\ d_{4,j} \\ d_{5,j} \\ d_{6,j} \\ d_{7,j} \\ d_{8,j} \end{bmatrix} = V_1[a_{1,j}] \oplus V_2[a_{1,j+1}] \oplus \quad \ldots \quad \oplus V_1[a_{1,j+7}].$$

So, there are only 64-bit XOR operations involved in the computation of a `Twister-Round` that can be implemented quite efficiently on most platforms.

9.4.2. 32-Bit Platforms

By splitting the 64-bit lookup tables $V_1, \ldots, V_8$ into 32-bit chunks, it takes twice as much operations as compared to the 64-bit variant. More general, this `Twister-Round` implementation has the desirable feature of scaling down linearly in terms of speed depending on the available word size of the platform.

9.4.3. Specific Remarks for 8-Bit Platforms

The performance on 8-bit processors is an important issue since most smart cards with cryptographic applications are restricted to their usage. There are several options for implementing Twister$_\pi$, depending on whether the requirements demand for minimum space (*i.e.,* low memory for storing lookup tables) or maximum speed. If minimum space is requested, the multiplication of two elements in $GF(2^8)$ has to be performed in software and should not be stored as a lookup table. Specific details for such issues can be found in [71, Chapter 4.1.1]. If space limitations are not an issue, the technique for implementing Twister$_\pi$ via lookup tables should be chosen as discussed in Section 9.4.1 or by splitting them up into single operations as discussed in Section 9.4.2. As all operations linearly scale down in terms of speed, *i.e.,* a 64-bit XOR can be easily implemented via eight times an 8-bit XOR, the running time is eight times the running time on a 64-bit platform.

9.4.4. Dedicated Hardware

Twister$_\pi$ is suited to be implemented in dedicated hardware. There are several tradeoffs between chip area and speed possible. Since the implementation in software on general-purpose processors is already very fast, the need for hardware implementation will probably be limited to very specific cases like:

1. Extreme high-speed chips with no area restrictions: The tables $V_1, \ldots, V_8$ can be hard-wired and the XOR operations can be conducted in parallel.

2. Compact coprocessors on smart cards: There can either be only the S-Box hard-wired or, additionally (and if enough memory is available), the tables $V_1, \ldots, V_8$ be generated at runtime.

3. If there is essentially no space to hard-wire anything, even the S-Box can be generated at runtime. Since Twister$_\pi$ uses the Rijndael S-Box, one can assemble it using two transformations:

   $$SB[\alpha] = f(g(\alpha)),$$

   where $g(\alpha)$ is defined as

   $$\alpha \to \alpha^{-1} \text{ in } GF(2^8)$$

   and $f(\alpha)$ is an affine transformation.

Note that there are finite-field multiplier over $GF(2^n)$ available in hardware that execute in a single clock cycle. More information is available in , *e.g.,* [188] or, for a short summary, in [71].

9.5. Benchmarks

This section provides software-performance benchmarks for the Twister$_\pi$ reference implementation. All measurement results are based on the real-time clock (RTC) and obtained by the median of 5,000 measurements of the target function. The performance values are given in cpb. For the sake of comparison, we also provide performance benchmarks for SHA-256 and SHA-512, where we used the implementation from *OpenSSL*[2] version 1.0.1e.

[2] `http://www.openssl.org`, last access: July 2013

Bytes	**SHA-256**	Twister$_\pi$**-256**	**SHA-512**	Twister$_\pi$**-512**
20	40.2	86.6	49.8	160.2
64	22.8	41.0	15.6	64.1
256	13.5	20.5	10.4	26.4
512	12.0	17.1	8.4	20.1
576	11.8	16.7	7.6	19.4
1024	11.5	15.3	7.5	16.9
1500	10.7	14.5	6.8	15.6
4096	10.6	14.0	6.8	14.4
10000	10.5	13.8	6.6	14.0
16384	10.3	13.7	6.6	13.9
32768	10.3	13.7	6.6	13.8

Table 9.2.: The benchmark results for **64-bit** platforms in cpb on an Intel Core i5-3210M CPU 2.50GHz; OS: Linux 3.9-1-amd64; Compiler: *gcc* 4.7.3.

Bytes	**SHA-256**	Twister$_\pi$**-256**	**SHA-512**	Twister$_\pi$**-512**
20	137.3	223.9	456.0	415.4
64	71.7	105.3	142.7	165.8
256	39.4	53.0	92.6	68.6
512	34.0	44.3	74.9	52.5
576	33.4	43.3	66.6	50.7
1024	31.4	39.9	66.0	44.3
1500	30.0	37.9	59.7	41.1
4096	29.5	36.7	59.3	38.3
10000	29.1	36.0	57.9	37.0
16384	29.0	35.9	57.7	36.8
32768	29.0	35.8	57.4	36.5

Table 9.3.: The benchmark results for **32-bit** platforms in cpb on an Intel Core i5-3210M CPU 2.50GHz; OS: Linux 3.9-1-amd64; Compiler: *gcc* 4.7.3.

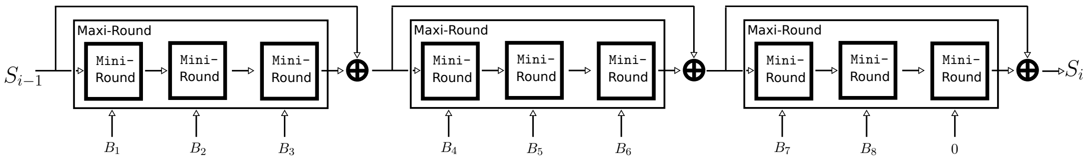

Figure 9.3.: The compression function `Twister-Round`-256.

Target Platform. The benchmarking took place on a single core of an Intel Core i5-3210M CPU 2.50GHz processor. All of the software benchmarking was written in C or ASM and compiled with the GNU C compiler (*gcc*) version 4.7.3 using the optimization flag `-O3`.

Implementation Remarks. TWISTER$_\pi$ was especially designed with 64-bit platforms in mind by making it possible to aggregate eight times an 8-bit table lookup into one single 64-bit table lookup.

Results. The results of the 64-bit and 32-bit performance benchmarks are summarized in Tables 9.2 and 9.3, respectively.

9.6. From TWISTER to TWISTER$_\pi$

The task of this section is to introduce the differences between TWISTER$_\pi$ and its predecessor TWISTER [93]. All modifications have been taken into account as a results of the external TWISTER$_\pi$ cryptanalysis from Mendel *et al.* [166]. A detailed discussion on their findings is given in Section 9.7. The major change between the two hash function families lies in the structure of the compression function. All members of the TWISTER$_\pi$ family use the same compression function, independent of the length of the message digest. On the other hand, the TWISTER family uses two similar compression functions, (1) `Twister-Round`-256, for the computation of message digest up to 256 bits and (2) `Twister-Round`-512 to compute message digests longer than 256 bits. Both hash functions follow the *Min-Max* approach, *i.e.,* either three or four `Mini-Rounds` are pooled to a `Maxi-Round`. The feed-forward operation (as visualized in Figure 9.3 and 9.4) is only performed after each `Maxi-Round` and not after each `Mini-Round` since two consecutive `Mini-Rounds` without a feed-forward in between guarante full diffusion – a nice property which unfortunately does not imply either collision or preimage security. In TWISTER$_\pi$, we perform a feed-forward after each `Mini-Round` to thwart rebound attacks. Furthermore, in TWISTER$_\pi$, we

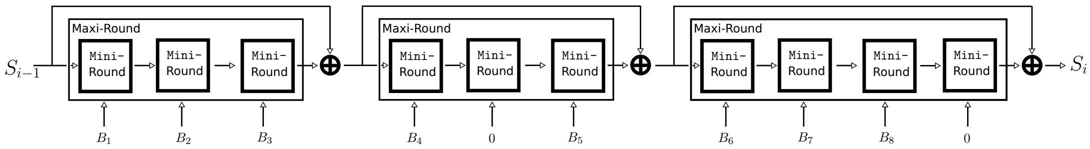

Figure 9.4.: The compression function `Twister-Round`-512.

Type	Compression Function Calls	Memory Requirement
collision attack	2^{251}	2^9
2nd-preimage attack	2^{384}	2^{64}
preimage attack	2^{456}	2^{10}

Table 9.4.: Cryptanalytic results for Twister-512 [166].

improved the checksum algorithm by a non-linear operation, namely multiplication by 3. Finally, Twister injects the `TwistCounter` into the second column. Twister$_\pi$ injects the `TwistCounter` into first row to circumvent a cancellation of a non-zero difference between two message words W and W'.

9.7. Untwisting the Myth – Cryptanalysis of Twister

This section consists of three parts. The first part introduces the preliminaries needed for understanding differential cryptanalysis. The second part presents the cryptanalytic results for Twister-512 from Mendel *et al.* [166] – summarized in Table 9.4 – and the third part discusses why those attacks are not applicable to Twister$_\pi$ anymore.

9.7.1. Preliminaries of Differential Cryptanalysis

Differential cryptanalysis, introduced by Biham and Shamir at Crypto'90 [42], turned out to be one of the most powerful techniques to attack cryptographic primitives like hash functions and block ciphers. It follows differential trails that occur with a significant probability, instead of looking at specific values. Next, we give a brief introduction about the basic definitions that are needed in the following cryptanalysis of Twister. The notions are borrowed from [219].

Definition 9.2 (XOR Difference). *Suppose x_1 and x_2 are two n-bit values. Then, the (n-bit) XOR difference is defined by*

$$\Delta x = x_1 \oplus x_2.$$

Definition 9.3 (Differential Probability). *Let $F : \{0,1\}^n \rightarrow \{0,1\}^m$ be a function. Suppose Δx is an n-bit input difference and Δy is an m-bit output difference. Then, we define the differential probability by*

$$\Pr\left[\Delta x \xrightarrow{F} \Delta y\right] = 2^{-m} \sum_{i=0}^{2^n-1} (F(i) \oplus F(i \oplus \Delta x) = \Delta y).$$

In this thesis we name Δx input difference, Δy output difference and $\Delta x \xrightarrow{F} \Delta y$ differential. Note that differentials with a differential probability of zero are called impossible differentials.

Difference-Distribution Table (DDT). The number of right pairs of a differential $\Delta x \xrightarrow{F} \Delta y$ denoted as $N_F(\Delta x \xrightarrow{F} \Delta y)$ is the number of pairs which satisfy that an input difference Δx leads to an output difference Δy. Usually, cryptanalysts are interested in the number of right pairs for all possible input and output values of a non-linear function, *e.g.,* an S-box, which can be encoded as DDT.

Definition 9.4 (Difference-Distribution Table (DDT)). *Let F be an $n \times m$ S-Box. Then, the DDT of F is a $2^n \times 2^m$ table whose entries are the number of right pairs $N_F(\Delta x \xrightarrow{F} \Delta y)$ for all differentials $\Delta x \xrightarrow{F} \Delta y$. The rows and columns of the DDT are indexed by the input differences Δx and output differences Δy, respectively.*

Due to the fact that the XOR operation is commutative, all entries of a DDT are even values, and the sum of each row must be 2^n. A toy example is given in Figure 9.5.

$$F(x) = \begin{cases} 1, & x = 0 \\ 3, & x = 1 \\ 2, & x = 2 \\ 0, & x = 3 \end{cases}$$

$\Delta x / \Delta y$	0	1	2	3
0	4	0	0	0
1	0	2	2	0
2	0	2	0	2
3	0	4	0	0

Figure 9.5.: Example computation of a DDT for a $2^2 \times 2^2$ S-box.

AES S-box. The $2^8 \times 2^8$ AES S-box SB – which is also used in Twister$_\pi$ and Twister – provides a nearly uniform distribution of XOR differentials. More precisely, the 65,536 entries of the SB-DDT are given by 33,150 entries 0, 32,130 entries 2, 255 entries 4, and one entry of 256 [219]. So, the probability that an entry chosen uniformly at random contains a value greater than zero is about 1/2. A more formal description of this observation is given in the following proposition:

Proposition 9.5 (Non-Zero Probability of the AES S-Box). *Suppose SB is the AES S-box, and let $(\Delta x, \Delta y)$ a fixed tuple of input-output differences. Then, it holds that*

$$\Pr\left[N_{SB}(\Delta x \overset{SB}{\to} \Delta y) > 0\right] = \frac{32386}{65536} \approx 1/2.$$

9.7.2. Collision Attack on Twister$_\pi$-512

The core of the collision attack is a semi-free-start collision attack for the compression function based on a rebound attack presented at FSE'09 by Mendel *et al.* [167] – about four months after the SHA-3 submission deadline. This attack was then developed into a collision attack for the hash function using Wagner's generalized birthday attack [233].

Semi-Free-Start Collision Attack. This attack only considers the first `Maxi-Round` of the `Twister-Round`-512 compression function. This operation updates the state S^0 by processing the first message words W_1, W_2, and W_3 of a 512-bit message block M_i by three consecutive invocations of the `Mini-Round`, *i.e.*, $S' = (\texttt{Mini-Round}_{W_3} \circ \texttt{Mini-Round}_{W_2} \circ \texttt{Mini-Round}_{W_1})(S^0) = \texttt{Maxi-Round}(S^0, W_1, W_2, W_3)$. Note that we can further decompose a `Mini-Round` into its individual basic operations:

$$\texttt{Mini-Round}(S^0, W_i) = (MC \circ SR \circ SB \circ AC \circ IM_{W_i})(S^0).$$

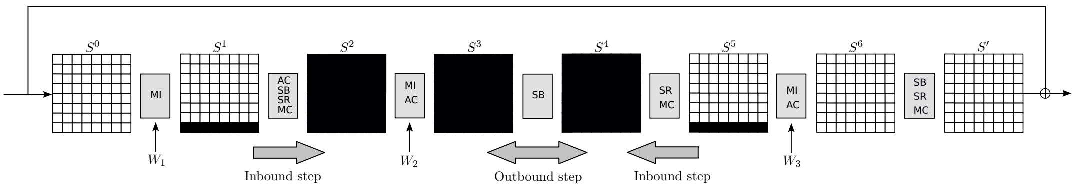

Figure 9.6.: A schematic view of the semi-free-start collision attack of Twister-512. Black state bytes are active.

Then, we have

$$S' = \texttt{Maxi-Round}(S^0, W_1, W_2, W_3) = (MC \circ SR \circ SB \circ AC \circ IM_{W_3} \circ \\ MC \circ SR \circ SB \circ AC \circ IM_{W_2} \circ \\ MC \circ SR \circ SB \circ AC \circ IM_{W_1})(S^0).$$

The basic idea of the attack is to inject a message-word difference ΔW_1 into the first `Mini-Round`, which can be canceled by the message-word difference ΔW_3 in the third `Mini-Round`. The rebound attack can then be described by an inbound step and an outbound step (cf. Figure 9.6). The inbound step propagates differences in W_1 and W_3 forwards and backwards through the `MixColumns` operation with a probability of 1. The goal of the outbound step is to find matches for the resulting differences of the `SubBytes` operation of the second round and propagate them outwards.

Inbound Step. Let $S^0, \ldots, S^6$, and S' be the internal states of Twister as shown in Figure 9.6. First, a message word of eight active bytes, *i.e.,* an XOR difference unequal zero, is injected into the last row of the State S^1 by means of the `InjectMessage` operation followed by another message injection into the state S^5 to cancel out the remaining active bytes in the last row. Then, the two active States S^2 and S^4 are computed by forward and backward computation, respectively. The column property of the `MixColumns` operation and its inverse (see Section 9.2.2) ensures that all 64 bytes of S^3 and S^4 are active.

Outbound Step. In this step, the adversary has to find a match for the input and output differences of the `SubBytes` operation of the second `Mini-Round`. Note that all 64 bytes of S^3 and S^4 are active. So, we have to estimate the effort for the adversary in finding 64 matches. For a single S-box call, the probability that a fixed $\Delta x \overset{SB}{\to} \Delta y$ exists is about 2^{-1} (see Proposition 9.5). For such a differential, it is possible to assign at least two possible values to the S-box, due to the symmetry of XOR differences. Next, the adversary chooses a random difference for the active Byte $S^2_{(8,1)}$ and then, computes the first column of the interim state S^3. The probability of finding non-zero differentials for all entries of this column is 2^{-8}. After finding a match, the adversary continues with the next column, until all columns are successfully processed. The complexity of this step is less than 2^8 compression function calls.

After executing the outbound step, the adversary has found a differential match for the `SubBytes` operation and can choose from at least 2^8 possible states for S^3. Each of those states can be computed forwards or backwards and produces a semi-free-start collision for a `Maxi-Round`. Each computation determines as well the State S^0 as the values and differences of W_1 and W_3, where the value of the message word W_2 can be freely chosen.

Next, we show how this semi-free-start collision attack can be extended to a collision attack on Twister-512.

Collision Attack on the Compression Function. At first, the adversary $\mathcal{A}$ computes 2^{224} semi-free-start collisions for the last `Maxi-Round` of `Twister-Round`-512 and stores them in a list $\mathfrak{L}$. This has a time complexity of about 2^{224} compression function calls. By varying the values for the massage word W_7, $\mathcal{A}$ can gain additional 2^{64} degree of freedom. After this pre-computations step, $\mathcal{A}$ randomly chooses some values for the message words $W_1 - W_5$ and computes the input of the last `Maxi-Round`. Statistically, the adversary finds a match in $\mathfrak{L}$ after 2^{224} tries. The complexity for this step of the attack is about $2/3 \cdot 2^{224}$ compression function calls. In total, this collision attack has a time complexity of at most 2^{225} compression function calls and a memory complexity of 2^{224}. The adversary can get rid of the memory complexity by applying the memory-less variant of the meet-in-the-middle attack introduced by Quisquater and Delescaille [195].

In the next paragraph we discuss how $\mathcal{A}$ can extend a collision for the compression function into a genuine collision for Twister-512.

Collision Attack on the Hash Function. In addition to the 8×8 state matrix S, Twister also has an 8×8 checksum matrix T which is updated immediately before the processing of a message word by a `Mini-Round`. In the post-processing, T is absorbed into the state via the compression function. Therefore, to construct a collision for Twister-512, an adversary has to implement a collision for both the chaining value represented by the state S and the checksum T. This can be done by applying the multi-collision attack introduced by Joux [129]. The effort to construct 2^t multi-collisions is $t \cdot \alpha$ where α denotes the complexity for constructing a single collision. A collision for the compression function of TWISTER-512 can be constructed with a time complexity of about 2^{225}. Thus, an adversary can construct 2^{256} collision with a time complexity of about $256 \cdot 2^{225} = 2^{233}$ evaluations of the compression function. The memory complexity for this attack is about 2^9 to store the 2^{256} multi-collisions. Due to the birthday paradox, the probability that two out of 2^{256} corresponding checksums are equal is greater than 1/2. Thus, we can assume that the set of multi-collisions also contains a collision for TWISTER-512. The time complexity to find a colliding checksum pair is about 2^{256} checksum computations, *i.e.,* 16 integer operations consisting of eight XOR operations and eight modular additions per checksum computation. In contrast, a `Twister-Round`-512 evaluation, omitting the checksum computation, can be done in 684 integer operations and 64 load/store operations. The individual numbers can be computed by adding up the cost of the individual operations which are

- $3 \cdot 8$ XOR operations for the feed-forward operations,
- $10 \cdot 1$ XOR operations for the `InjectMessage` step,
- $10 \cdot 64$ XOR operations for a combined operation that consists of the `SubBytes`, `ShiftRows`, and `MixColumns` step, and
- $10 \cdot 64$ table lookups for such a combined operation.

Thus, we can assume that the cost of evaluating the compression function is at least 32 times higher than the computation of a checksum, so, 2^{256} checksum computations has an effort of at most 2^{251} compression function evaluations. The memory complexity for this step is negligible when applying the memory-less variant of the birthday attack [195].

9.7.3. 2nd-Preimage Attack on TWISTER-512

Let S^i denote the state S after the invocation of the i-th `Mini-Round` within a `Twister-Round`-512. The following 2nd-preimage attack has a minor limitation; it only works for a given message-hash tuple (M,TWISTER-512(M)), where the message consists of at least 513 message blocks, *i.e.,* $|M| > 512 \cdot 513$. Without loss of generality, we assume that the message M consists of 513 message blocks. The process of the 2nd-preimage adversary $\mathcal{A}$ can be described by the following six consecutive steps:

1. $\mathcal{A}$ applies the multi-collision attack to construct 2^{512} multi-collisions with a time complexity of about $512 \cdot 2^{225} = 2^{234}$ evaluations of the compression function and a memory complexity of 2^9. After this step, $\mathcal{A}$ gets 2^{512} messages leading to the same chaining value V_{512}, *i.e.,* the state after 512 iterations of the compression function.

2. $\mathcal{A}$ chooses arbitrary values for the last message block M'_{513} with correct padding and computes the chaining value V_{513}, *i.e.,* the state which is used as input for the post-processing step.

3. $\mathcal{A}$ chooses arbitrary values for the first five columns of the checksum T', *i.e.,* $T'_{(1\downarrow)}, \ldots, T'_{(5\downarrow)}$. Then, it computes the interim state $S_F^6 = V'_{513} \oplus S^3 \oplus S^6$ of the last compression function call, $\texttt{Twister-Round-256}(V'_{513}, T')$. From the first preimage, $\mathcal{A}$ can compute $S' = \texttt{Twister-Round-256}(S^0, T)$, and then the value $S^{10} = S' \oplus S_F^6$.

4. For each of the 2^{64} possible values of $T'_{(8\downarrow)}$, $\mathcal{A}$ computes **backwards** the values $S'^7 = \texttt{InjectMessage}(S'^7, T'_{(7\downarrow)})$ and stores them in the list $\mathfrak{L}$.

5. For each of the 2^{64} possible values of $T'_{(6\downarrow)}$, $\mathcal{A}$ computes **forwards** from S^6 to the injection of the checksum word $T'_{(7\downarrow)}$, and then checks if the result matches any element of the list $\mathfrak{L}$. Since $\mathcal{A}$ is still able to choose an arbitrary value for $T'_{(7\downarrow)}$, it is sufficient to match all rows except the last. The probability that all of those 448 bits – from the remaining seven rows – match, is about $2^{448-128}$ since $\mathcal{A}$ has 2^{128} pairs to check. Statistically, $\mathcal{A}$ finds a match after 2^{320} iterations of the Steps 3-5. So, we can upper bound the costs of finding such a pair by about $2^{320+64} = 2^{384}$ compression function evaluations.

6. Once $\mathcal{A}$ found a 2nd-preimage for the iterative part of Twister-512, it has to ensure that the checksum T' is valid. From the computation of Step 1, $\mathcal{A}$ has access to 2^{512} checksums leading to the same chaining values V_{512},V_{513}, and V_{514}. By applying a memory-less meet-in-the-middle-attack [195], the adversary can construct the needed checksum value. The complexity for this attack is about 2^{257} checksum operations.

The introduced 2nd-preimage attack for Twister-512 has a time complexity of about 2^{384} and a memory complexity of 2^{9+64}. Due to the output transformation `Output-Round`, the attack cannot be extended to a preimage attack on Twister-512 in a straightforward way. But, in the next section we will present a sophisticated way to use this attack as a key element of a preimage attack.

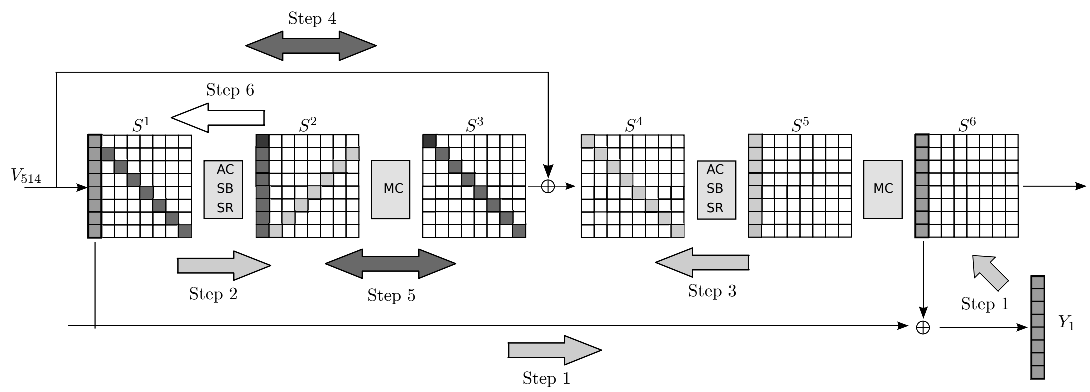

Figure 9.7.: Inversion of the first 64-bit word of a Twister-512 hash value.

9.7.4. Preimage Attack on Twister-512

To construct a preimage for Twister-512, an adversary $\mathcal{A}$ has to invert the output transformation `OutputRound` (see Section 9.2.4). Afterwards, it can apply the 2nd-preimage attack former presented to construct a preimage. Suppose the adversary has to find a preimage for the value $Y = Y_1 \mid\mid \ldots \mid\mid Y_8$ where Y_i denotes the i-th hash word which was generated by the `OutputRound` function. Furthermore, we assume without loss of generality that $\mathcal{A}$ produces a preimage M of 513 message blocks with correct padding, such that $Y =$ Twister-512(M). Let S^1-S^6 be the internal states of Twister as shown in Figure 9.7, where S^1 denotes the state after the invocation of the function `State-Finalization`, *i.e.,* V_{514}. The `OutputRound` inversion attack can be described by the following six consecutive steps:

1. $\mathcal{A}$ chooses an arbitrary value for the first column of S^1 and sets the first column of S^6 using the first 64-bit hash word, *i.e.,* $S^6_{(1\downarrow)} = S^1_{(1\downarrow)} \oplus Y_1$.

2. $\mathcal{A}$ computes forwards 64 bits from the state S^1, the Byte $S^2_{(1,1)}$, and the seven Bytes $S^2_{(i,j)}$ for $2 \leq i \leq 8$ where $j = 10 - i$.

3. $\mathcal{A}$ computes backwards from State S^6 the first column of S^5 and the diagonal bytes of S^4.

4. $\mathcal{A}$ chooses arbitrary values for the seven remaining diagonal bytes of State S^1, *i.e.,* $S^1_{(2,2)}, \ldots, S^1_{(8,8)}$. This determines the first column of S^2. Then, $\mathcal{A}$ computes backwards the eight diagonal bytes $S^3_{(i,i)} = S^1_{(i,i)} \oplus S^4_{(i,i)}$ for $1 \leq i \leq 8$.

5. Next, $\mathcal{A}$ has to find a match of S^2 and S^3 through the `MixColumns` operation

(cf. Figure (9.7). Note that the first column of S^2 is already determined by the previous step. First off all, $\mathcal{A}$ tests if the first Byte $S^3_{(1,1)}$ matches, if not, $\mathcal{A}$ starts over again from Step 1. Statistically, $\mathcal{A}$ finds a match after 2^8 iterations of Steps 1-4. Then, it find matches for the remaining columns of S^2 by executing the following two steps:

a) $\mathcal{A}$ chooses for each column $2 \le i \le 8$ arbitrary values for the remaining seven Bytes $S^2_{(j,i)}$ with $1 \le j \le 8$ and $j \ne 10 - i$ since $S^2_{(10-i,i)}$ is already fixed due to Step 2.

b) Then, $\mathcal{A}$ computes the `MixColumns` operation and checks if Byte $S^3_{(i,i)}$ matches, and repeates the previous step if not. This step has a time complexity of 2^8 `Mini-Round` operations.

Note that each column can be modified independently. Therefore, this step has a time complexity of about $8 \times 2^8 = 2^{11}$ `Mini-Round` operations. This corresponds to about 2^8 compression function evaluations.

6. After $\mathcal{A}$ has found a match for all columns of S^2, it computes S^1 backwards from S^2. Note that the values fixed in Step 1 and Step 4 do not change anymore.

$\mathcal{A}$ can invert the `OutputRound` of Twister-512 by repeating Steps 1-6 about 2^{448} time. Thus, this inversion attack has a time complexity of about $2^{448+8} = 2^{456}$ compression function evaluations and a total memory complexity of 2^{10}. Now, $\mathcal{A}$ can apply the 2nd-preimage attack of the previous section to the State $S^1 = V_{514}$ to construct a preimage for Twister-512 which consists of 513 message blocks. This preimage attack has a total time complexity of $2^{448}+2^{456} \approx 2^{456}$ compression function operations and a total memory complexity of 2^{10}.

Remarks. None of the introduced collision, preimage, and 2nd-preimage attacks on Twister-512 are practical due to the time complexity of at least 2^{384} compression function calls. Nevertheless, they reveal non-random properties that are not present in SHA-512. Due to the publication of those attacks, the committee of the SHA-3 Competition did not pass Twister-512 to the second round; a rightful choice.

9.7.5. Twister$_\pi$ Security Discussion

In this section we argue why the presented attacks on Twister-512 are not applicable to Twister$_\pi$-512.

Collision Attacks. The semi-free-start collision attack exploits the invertibility of a `Maxi-Round`. TWISTER$_\pi$ abandoned the concept of `Maxi-Rounds` and applies a feed-forward after each `Mini-Round`. Thus, following the notations of Figure 9.6, we have $S^6 = (AC \circ MI)(S^5 \oplus S^2)$ instead of $S^6 = (AC \circ MI)(S^5)$. Since S^2 is a full active state and the message word is injected in the last row, the interim state S^6 has at least 56 active bytes. Therefore, it is no longer possible to achieve a state containing an all-zero difference after three invocations of a `Mini-Round`. So, the additional feed-forward operations thwart the proposed rebound attack. Furthermore, this modification also thwarts the presented collision attack on TWISTER-512 since it is just a sophisticated extension of the semi-free-start collision attack.

(2nd-) Preimage Attack. The preimage attack on TWISTER-512 presented in Section 9.7.4 invokes the 2nd-preimage attack from Section 9.7.3. Thus, it is sufficient to show why the 2nd-preimage attack is no longer applicable to TWISTER$_\pi$-512.

Steps 3 and 4 of the 2nd-preimage attack on TWISTER-512, the adversary also exploits the invertibility of a `Maxi-Round` to compute the interim states S^{10} and S'^7. The extra feed-forward operation would not allow to compute S^{10} without determining $T_{(6\downarrow)}, T_{(7\downarrow)}$, and $T_{(8\downarrow)}$. But this would take away the freedom from an adversary to choose an arbitrary value for $T_{(7\downarrow)}$ and $T_{(8\downarrow)}$, increasing the time complexity by 2^{128} from 2^{384} to 2^{512}, which renders this attack not better than exhausting search.

9.8. Results Summary

We proposed a family of hash functions which overcomes several identified weaknesses of the commonly used MD4/5 family of hash functions (MD4, MD5, SHA-0/1). By using some of the well-analyzed building blocks and ideas of Rijndael, we obtained a design for which we claim that no efficient differential collision structure exists. In addition, we limit access to the internal structure and take care that any possible difference quickly diffuses into the internal state. Furthermore, it is highly scalable as there are – as proposed in, *e.g.,*TWISTER$_\pi$-256 and TWISTER$_\pi$-512 – many possible ways to adopt our main building block, the `Twister-Round`.

Furthemore, we proposed two specific instantiations of the TWISTER$_\pi$ framework, TWISTER$_\pi$-256 and TWISTER$_\pi$-512. The claimed security level for TWISTER$_\pi$-256 with respect to collision resistance is 2^{128} and with respect to (2nd-) preimage resistance 2^{256}. For TWISTER$_\pi$-512, the claimed security level for collision resistance is 2^{256} and for (2nd-) preimage resistance 2^{512}. The TWISTER$_\pi$ family of hash functions exploits mathematical structures (*i.e.,* MDS matrices) and, at the same time, has comparable speed to the SHA-2 family. Thus, instances of the Twister$_\pi$ family are suitable for a huge range of applications from low-end 8-bit microcontroller platforms up to high-end 64-bit software architectures.

10

Catena: A Memory-Consuming Password Scrambler

> Talent hits a target no one else can hit; Genius hits a target no one else can see.
>
> Arthur Schopenhauer

From the early 1960s [218] till now, the concept of textural passwords are dominant in terms of human-computer authentication. In the context of this thesis we define a password as a user-chosen secret, and thus, we also consider both a passphrase and a Personal Identification Number (PIN) as a *password*. As observed by Wilkes in the late 1960s [237], storing plain authentication passwords is insecure. Everybody that is granted access to the password storage of a specific multi-user system, immediately learns all user passwords, and can just impersonate any user by a legitimate login. About a decade later, the UNIX system integrated some of Wilkes ideas [174] by deploying a DES-based [175] one-way encryption function, called `crypt`. This function is limited to passwords up to eight characters since the seven least significant bits of each of the first eight characters of the password represents the 56-bit key of the 64-bit block cipher DES, which is used to encrypt iteratively – 25 times – a string of 64 zero-bits.

Under the assumption that `crypt` is preimage-secure, there is no efficient way to recover the original password from its output, *i.e.,* the password hash. Nevertheless, this scheme can nowadays not longer be considered secure, due to its very small key space of 56 bits. By the means of modern Graphical Processing Units (GPUs) with hundreds of cores [67] – as embedded in all state-of-the art graphics cards – it is possible to recover the key in feasible time. For example, the advanced password-recovery tool `hashcat` can process about 2^{26} password candidates per second (c/s) on a single *AMD hd6990* graphics card [226]. An adversary with access to a GPU-cluster with 128 nodes can compute a preimage in about four months. Thus, the question of how to slow down such adversaries becomes a pressing one.

Memory is expensive; so, a typical GPU or other cheap massively-parallel hardware with lots of cores can only have a limited amount of memory for each single core. More importantly, each core will have only a very limited amount of fast memory (cache). So, the way to prevent c-core adversaries from gaining some close-to-c-times speed-up is by making a password scrambler not only intentionally slow on standard sequential computers, but also intentionally memory consuming. Under the preimage-security assumption, any adversary using c cores in parallel with less than about c times the memory of a sequential implementation must experience a strong slow-down. A formal definition of this property called *sequential memory-hardness* is given in Section 10.3 (cf. Definition 10.3). The first password scrambler that took this condition into account was `scrypt` [191].

In the light of the current situation, the designer of a modern password scrambler is caught between Scylla and Charybdis. On the one hand, the acceptance of a password scrambler depends on its time and memory usage. Usually, user want to log in without noticeable delay [231], and especially on embedded devices, such as routers or switches, it is unlikely that the developers choose to implement a login process that consumes a significant amount of *expensive* memory. On the other hand, the more time and memory a password scrambler needs to compute a hash from a password, the less efficient are guessing attacks such as exhaustive search. This is the reason why the password processing takes some time for both kinds of users legitimate ones and attackers. Thus, a good password scrambler $\mathcal{P}$ has to satisfy at least the following three basic conditions:

(1) Given a password pwd, computing $\mathcal{P}(pwd)$ should be "fast enough" for the user.

(2) Computing $\mathcal{P}(pwd)$ should be "as slow as possible", without contradicting the previous condition.

(3) Given $h = \mathcal{P}(pwd)$, there must be no significantly faster way to test q password candidates $x_1, x_2, \ldots, x_q$ for $\mathcal{P}(X_i) = Y$ than by actually computing $\mathcal{P}(x_i)$ for each x_i.

Memory-Access Pattern and Outline. Note that a memory-consuming password scrambler may suffer from a new problem. If the memory-access pattern depends on the password, and the adversary can observe that pattern, this may open the way to another kind of shortcut attack. For example, a spy process running on the same machine as the password scrambler $\mathcal{P}$ (without access to the internal memory of $\mathcal{P}$) may gather information about the memory-access pattern by measuring cache timings. This information can be used to greatly speed-up massively-parallel attacks with low memory for each core. In Section 10.4 we show that this is actually an issue for `scrypt`, and then, in Section 10.5 we present a fix by introducing CATENA, a new password scrambler framework which consumes lots of memory (like `scrypt`), but does not have a password-dependend memory-access pattern. In Section 10.6 we formally analyze the security of CATENA framework and its memory consumption. Section 10.7 presents a secure Key Derivation Function (KDF) based on CATENA. In Section 10.8 we introduce an instantiation of CATENA, namely CATENA-DBG, and in Section 10.9 we analyze its security and memory-hardness. Finally, Section 10.10 summarizes our contribution.

10.1. Background

Since the introduction of `crypt`, storing the hash of a password and avoiding to store the plain password itself has become the minimum standard for secure password-based user authentication. But, even as late as 2012, major players like Yahoo and CSDN (China Software Developer Network) seem to store plain user passwords [157].

Two important innovations from `crypt` were *key stretching* and *salts.* Key stretching is the answer to the typically low entropy of user-chosen passwords: The password scrambler is intentionally slow, but not too slow for the regular operation, *e.g.,* a password-based login. This makes exhaustively searching through all likely passwords more expensive.

Salt. A salt refers to an additional random input for the password scrambler and is stored together with the password hash. It enables a password scrambler to derive lots of different password hashes from a single password as an initialization vector enables an encryption scheme to derive lots of different ciphertexts from a single plaintext. Since the salt must be chosen uniformly at random, it is most likely that different users have different salts. Thus, it defends against attacks where password hashes from many different users are known to the attacker, *e.g.,* against the usage of rainbow tables [182].

Pepper. There are different ways to perform key stretching. One is to keep p bits of the salt secret, turning them into *pepper* [159]. Both attackers and legitimate users have to try out all 2^p values the pepper can have (or 2^{p-1} on the average). Note that a careless implementation of this approach could leak a few bits of the pepper via timing information when trying out all possible values in a specific order. Thus, a recommended approach would be to start at a random value and wrap around at 2^p. Kelsey *et al.* [136] analyzed another key stretching approach where a cryptographic operation is iterated n times. Boyen proposed in [52] a user-defined implicit choice of n by iterating until the user presses a "halt" button.

10.2. Related Work

Table 10.1 provides an overview of password scramblers that are or have been in frequent use, compared to Catena. It indicates whether the password scrambler supports *salt*, *server relief*, and *client-independent updates*. Furthermore, the table lists all possible values of the cost factor (security parameter) including the default values, the memory usage, and issues from which the considered password scrambler may suffer from.

Hash Function Based Password Scramblers. Not long ago, `md5crypt` [132] has been used in nearly all Free-BSD and Linux-based systems to scramble user passwords. It is based on the well-known MD5 hash function with a fixed number of 1,000 iterations. Due to the fact that CPUs and GPUs become more and more powerful, `md5crypt` can now be computed too fast, *e.g.,* over 5 million times per second on an *AMD HD 6990* graphics card [226]. Additionally, its own author does not consider `md5crypt` secure anymore [132]. Common Linux distributions nowadays employ `sha512crypt` [82], *e.g.,* Debian, Ubuntu, and Fedora. It provides similar features as `md5crypt`, but uses SHA-512 instead of MD5. Furthermore, the number of iterations can be chosen by the user; default is 5,000 iterations. `NTLMv1` [112] is a fast password scrambler which is deployed to generate hash values for several versions of Microsoft Windows passwords. It is very efficient to compute: One can check over nine billion password candidates per second on a single Commercial Off-The-Shelf (COTS) graphics card [226]. For this and other reasons, we recommend that `NTLMv1` should not be used anymore, if possible.

The Password-Based Key Derivation Function 2 (PBKDF2) has been specified by the NIST [231]. It is widely used either as a KDF (*e.g.,* in Wi-Fi Protected Access (WPA), WPA2, OpenOffice, or WinZip) or as a password scrambler (*e.g.,* in

Password Scrambler		Cost Factor	Memory	Server Relief	Client-Indep. Updates	Issues
`crypt`	[174]	25	small	-	-	"too fast"
`md5crypt`	[132]	1,000	small	-	-	"too fast"
`sha512crypt`	[82]	1,000–999,999	small	-	-	small memory
`NTLMv1`	[112]	1	small	-	-	"too fast"
`PBKDF2`	[231]	1–∞	small	-	-	small memory
`bcrypt`	[194]	2^4–2^{99}	4,168 bytes	-	-	constant memory
`scrypt`	[191]	1–∞	flexible, big	-	-	cache-timing attacks
Catena	(this work)	2^1–2^{∞}	flexible, big	✓	✓	new and untested

Table 10.1.: Comparison of contemporary password scramblers.

Mac OS X, and LastPass). The security of `PBKDF2` is based on c iterations of Hash-Based Message Authentication Code (HMAC) [17] instantiated with SHA-1, where c is a user-chosen value which is given by default with $c = 1,000$.

`bcrypt`. The `bcrypt` algorithm [194] is built upon the Blowfish block cipher [220]. Internally, Blowfish uses a slow key scheduler to generate an internal state of 4,168 bytes for the key-dependent S-boxes ($4 \times 1,024$ bytes) and the round keys (72 bytes). Thus, while `bcrypt` has not been designed with the intention to thwart parallelized attackers by exhaustive memory usage, the state is sufficiently large to slow down `bcrypt` significantly on current GPUs, *e.g.,* it can only be computed about 4,000 times per second on an *AMD HD 7970* graphic card [226]. However, the state size is fixed – so if future GPUs have a larger cache, it may actually run *much faster*. There is no tunable parameter to increase the memory requirement. For key stretching, `bcrypt` invokes the Blowfish key scheduler 2^c times, *e.g.,* OpenBSD uses $c = 6$ for users and $c = 8$ for the superuser.

`scrypt`. Occupying a lot of memory hinders attacks using special-purpose hardware (storage is expensive) and GPUs. We are aware of one single password-scrambler that has been designed to fulfill this requirement: `scrypt` [191]. (There was HEKS [197], but it has been broken by the author of `scrypt` [191].) As its core, `scrypt` uses the sequentially memory-hard function `ROMix`, which can take G units of memory and performs $2G$ operations. With only G/K units of memory, the number of operations goes up to $2G \cdot K$. In [191], Percival recommends $G = 2^{14}$ and $G = 2^{20}$ for password hashing and key derivation, respectively. We will describe and analyze `scrypt` and `ROMix` in Section 10.4.

10.3. Memory-Related Properties

In this section we introduce a listing of desired properties a modern password scrambler should have – beyond salt and pepper. We start, by introducing the security parameter g, called *garlic*. The notion of *garlic* reflects the property that incrementing this parameter by '1' doubles the memory usage and at least doubles the computational time.

Memory-Hardness. To describe memory requirements, we adopt and slightly change the notion from [191]. The intuition is that for any parallelized attack, using c cores, the required memory per core is decreased by a factor of $1/c$, and vice versa.

Definition 10.1 (Memory-Hard Function). *For a* memory-hard function $\mathcal{F}$ *which is computed on a Random Access Machine using $S(g)$ space and $T(g)$ operations, it holds that*

$$T(g) = \Omega\left(\frac{G^2}{S(g)}\right),$$

where $G = 2^g$.

Thus, for $S(g) \cdot T(g) = G^2$ with $G = 2^g$, using c cores, it holds that

$$(1/c \cdot S(g)) \cdot (c \cdot T(g)) = G^2.$$

A formal generalization of this notion is given in the following.

Definition 10.2 ($\lambda-$Memory-Hard Function). *For a* λ-memory-hard function $\mathcal{F}$ *which is computed on a Random Access Machine using $S(g)$ space and $T(g)$ operations, it holds that*

$$T(g) = \Omega\left(\frac{G^{\lambda+1}}{S(g)^\lambda}\right),$$

where $G = 2^g$.

Thus, if one has only $1/c$ of the memory available, one needs c^λ processor units to gain the same time-memory tradeoff, *i.e.*,

$$(1/c \cdot S(g)^\lambda) \cdot (c^\lambda \cdot T(g)) = G^{\lambda+1}.$$

In the following we use $S(g)$ and S as synonyms.

Definition 10.3 (Sequential Memory-Hard Function). *A* sequential memory-hard function *is a function $\mathcal{F}$ with the following properties:*

(a) $\mathcal{F}$ is memory-hard and

(b) there is no $\beta > 0$ such that $\mathcal{F}$ can be computed on a Parallel Random Access machine with $S^*(g)$ processors and $S^*(g)$ space in expected time $T^*(g)$, where $S^*(g)T^*(g) = O(T(g)^{2-\beta})$.

Password Recovery (Preimage Security). For a modern password scrambler it must hold that the advantage of an adversary (modelled as a computationally unbounded but always-halting algorithm) for guessing a valid password should be reasonable small, *i.e.,* not higher than for trying out all possible candidates. Therefore, given a password scrambler $\mathcal{P}$, we define the password-recovery advantage of an adversary $\mathcal{A}$ as follows:

Definition 10.4 (Password Recovery Advantage). *Let s denote a randomly chosen salt value and pwd a password randomly chosen from a source $\mathcal{Q}$ with m bits of min-entropy. Then, given a hash value $h \leftarrow \mathcal{P}(s, pwd)$, it holds that*

$$\mathbf{Adv}^{REC}_{\mathcal{P},\mathcal{Q}}(\mathcal{A}) = \Pr_{s}\left[pwd \leftarrow \mathcal{Q}, h \leftarrow \mathcal{P}(s, pwd) : x \leftarrow \mathcal{A}^{\mathcal{P},s,h} : \mathcal{P}(s, x) = h\right].$$

Furthermore, by $\mathbf{Adv}^{REC}_{\mathcal{P}}(q)$ we denote the maximum advantage taken over all adversaries asking at most q queries to $\mathcal{P}$.

Client-Independent Update. According to Moore's Law [173], the available resources of an adversary increase continually over time – and so do those of the legitimate user. Hence, a security parameter chosen once may be too weak after some time and needs to be updated. This can easily be accomplished immediately after the user has entered its password the next time. However, in many cases, a significant amount of user accounts is inactive or rarely used, *e.g.,* 70.1% of all Facebook accounts experience zero updates per month [177], and 73% of all Twitter accounts do not have at least one tweet per month [215]. Therefore, it is desirable to be able to compute a new password hash (with some higher security parameter) from the old one (with the old and weaker security parameter), without having to involve user interaction or otherwise having to know the password. We call this feature a *client-independent update* of the password hash. When key stretching is done by iterating an operation, client-independent updates may or may not be possible, depending on the details of the inner workings of a password scrambler. For example, when the original password is one of the inputs for the final operation (see [191]), client-independent updates are impossible.

Server Relief. A slow and – even worse – memory-demanding password-based login process may be too much of a burden for many service providers. *Server relief* splits the password-scrambling process into two parts: (1) a slow (and possibly memory-demanding) one-way function $\mathcal{F}$ and (2) an efficient one-way function $\mathcal{H}$. By default, the server computes the password hash $\mathcal{H}(\mathcal{F}(pwd, s))$ from the password pwd and a salt s. Alternatively, the server sends s to the client who responds $x = \mathcal{F}(pwd, s)$. Finally, the server just computes $\mathcal{H}(x)$. While it is probably easy to write a generic server-relief protocol using any password scrambler, none of the existing password scramblers has been designed to naturally support this property.

Key Derivation Function (KDF). Beyond authentication, passwords are also used to derive symmetric keys. Obviously, one can just use the output of the password scrambler as a symmetric key – perhaps after truncating it to the required key size. This is a disadvantage if one either needs a key that is longer than the password hash or has to derive more than one key. Thus, it is prudent to consider a KDF as a tool of its own right – with the option to derive more than one key and with the security requirement that compromising some of the keys does not endanger the other ones. Note that it is required for a KDF that the input and output behaviour cannot be distinguished from that of a set of random functions.

Resistance against Cache-Timing Attacks. Password scramblers with a password-dependent memory-access pattern risk to be vulnerable against cache-timing attacks. Depending on the implementation and under certain circumstances, timing information related to a given machine's cache behavior may enable the adversary to observe which addresses have been accessed. This can be exploited to implement a very efficient password-candidate sieve. Therefore, any password scrambler whose memory-access pattern is independent from the password is not vulnerable against cache-timing attacks.

10.4. The `scrypt` Password Scrambler

Algorithm 16 describes the `scrypt` password scrambler and its core operation `ROMix`. For pre- and post-processing, `scrypt` invokes the one-way function PBKDF2 to support inputs and outputs of arbitrary length. `ROMix` uses a hash function $\mathcal{H}$ with an n-bit output where n is the size of a cache line (at current machines, often 512 bits). To support hash functions with smaller output sizes, [191] proposes to instantiate $\mathcal{H}$ by a function called BlockMix, which we will not elaborate on. For our security analysis of `ROMix`, we modelled $\mathcal{H}$ as a random oracle.

`ROMix` takes two inputs: An initial state x which depends on both salt and password, and the array size G that defines the required storage. One can interpret $\log_2(G)$ as the garlic factor of `scrypt`. In the first phase (Lines 20–23), `ROMix` initializes an array v, *i.e.,* the array variables $v_0, \ldots, v_{G-1}$ are set to $x, \mathcal{H}(x), \mathcal{H}(\mathcal{H}(x)), \ldots,$ $\mathcal{H}(\ldots(\mathcal{H}(x)))$, respectively. In the second phase (Lines 24–27), `ROMix` updates x depending on v_j. The sequential-memory hardness comes from the way how the index j is computed, depending on the current value of x, *i.e.,* $j \leftarrow x \bmod G$. After G updates, the final value of x is returned and undergoes the post-processing.

A minor issue is that `scrypt` uses the password *pwd* as one of the inputs for post-processing (Line 12). Thus, it has to be in storage during the entire password-scrambling process. This is risky if there is any chance that the memory can be

Algorithm 16 The `scrypt` Algorithm and its Core Operation `ROMix` [191].

`scrypt`

Input:
- pwd {Password}
- s {Salt}
- G {Cost Parameter}

Output: x {Password Hash}

10: $x \leftarrow \mathrm{PBKDF2}(pwd, s, 1, 1)$
11: $x \leftarrow$ `ROMix`(x, G)
12: $x \leftarrow \mathrm{PBKDF2}(pwd, x, 1, 1)$
13: **return** x

`ROMix`

Input:
- x {Initial State}
- G {Cost Parameter}

Output: x {Hash Value}

20: **for** $i = 0, \ldots, G-1$ **do**
21: $\quad v_i \leftarrow x$
22: $\quad x \leftarrow \mathcal{H}(x)$
23: **end for**
24: **for** $i = 0, \ldots, G-1$ **do**
25: $\quad j \leftarrow x \bmod G$
26: $\quad x \leftarrow \mathcal{H}(x \oplus v_j)$
27: **end for**
28: **return** x

Algorithm 17 `ROMixMC`

Input:
x {Initial State},
G {1st Cost Parameter},
K {2nd Cost Parameter}

Output: x {Hash Value}

1: **for** $i = 0, \dots, G-1$ **do**
2:   **if** $i \bmod K = 0$ **then**
3:     $v_i \leftarrow x$
4:   **end if**
5:   $x \leftarrow \mathcal{H}(x)$
6: **end for**
7: **for** $i = 0, \dots, G-1$ **do**
8:   $j \leftarrow x \bmod G$
9:   $\ell \leftarrow K(j/K)$
10:   $y \leftarrow v_\ell$
11:   **for** $m = \ell+1, \dots, j$ **do**
12:     $y \leftarrow \mathcal{H}(y)$ { invariant: $y \leftarrow v_m$ }
13:   **end for**
14:   $x \leftarrow \mathcal{H}(x \oplus y)$
15: **end for**
16: **return** x

compromised during the time `scrypt` is running. Compromising the memory should not happen anyway, but this issue could easily be fixed without any bad effect on the security of `scrypt`, *e.g.,* one could replace Line 12 of Algorithm 16 by $x \leftarrow \mathrm{PBKDF2}(s, x, 1, 1)$.

10.4.1. Brief Analysis of `ROMix`

In the following we introduce a way to run `ROMix` with less than G units of storage. Suppose we only have $S < G$ units of storage for the values in v. For convenience, we assume G is a multiple of S and set $K \leftarrow G/S$. As it will turn out, the memory-constrained Algorithm `ROMixMC` (cf. Algorithm 17) generates the same result as `ROMix` with less than G storage units and is $\Theta(K)$ times slower than `ROMix`. From the array v, we will only store the values v_0, v_K, v_{2k}, $\dots$, $v_{(S-1)K}$ – using all the S memory units available.

At Line 9, the variable ℓ is assigned the biggest multiple of K less or equal j. By verifying the invariant at Line 12, one can easily see that `ROMixMC` computes the same hash value as the original `ROMix`, except that v_j is computed on the fly, beginning with v_ℓ. These computations call the random oracle on average $(K-1)/2$ times. Thus, the second phase of `ROMixMC` is about $\Theta(K)$ times slower than the second phase of `ROMix`, and this dominates the workload for `ROMixMC`.

Next, we briefly discuss why `ROMix` is sequentially memory-hard (for the full proof see [191]). The intuition is as follows: The indices j are determined by the output of the random oracle $\mathcal{H}$ and thus, essentially, uniformly distributed random values over $\{0, \dots, G-1\}$. With no way to anticipate the next j, the best approach is to minimize the size of the "gaps", *i.e.,* the number of consecutively unknown v_j. This is indeed what `ROMixMC` does, by storing one v_i every K'th step.

10.4.2. Cache-Timing Attacks

Algorithm 16 (`scrypt/ROMix`) revisited. What could possibly go wrong?

The Spy Process. As it turns out, the idea to compute an unpredictable index j and then ask for the value v_j, which is useful for sequential memory-hardness, is also an issue. Consider a spy process running on the same machine as `scrypt`. This spy process cannot read the internal memory of `scrypt`. But, as it is running on the same machine, it shares its cache memory with `ROMix`. The spy process interrupts the execution of `ROMix` twice:

1. When `ROMix` enters the second phase (Line 24 of Algorithm 17), the spy process reads from a bunch of addresses, to force out all the v_i that are still in the cache. Thereupon, `ROMix` is allowed to run for another very short time.

2. Now, the spy process interrupts `ROMix` again. By measuring access times when reading from different addresses, the spy process can figure out which of the v_i have been read by `ROMix`, in between.

So, the spy process can tell us the indices j for which v_j has been read, and with this information we can mount the following cache-timing attack.

The Preliminary Cache-Timing Attack. Let pwd' denote the current password candidate. Suppose x is the output of PBKDF2(pwd',*salt*,1,1). Then, we can apply the following password candidate sieve:

1. Execute the first phase of `ROMix`, without storing the v_i, *i.e.,* skip Line 21 of Algorithm 16.

2. Compute the index $j \leftarrow x \bmod G$.

3. If v_j is one of the values that have been read by `ROMix`, then store pwd' in a list.

4. Else, conclude that pwd' is a wrong password.

This sieve can run in parallel on any number of cores, each core trying out another password candidate pwd'. Note that each core needs only a small and constant amount of memory – the data structure to decide if j is one of the indices of the value v_j which has been read. Further, this can be shared between all the cores. Hence, we can use exactly the kind of hardware that `scrypt` has been designed to hinder.

Next, we discuss the gain of this attack. Let r denote the number of iterations the loop in Lines 24–27 of `ROMix` has performed before the second interrupt by the spy process. So, there are at most r indices j with v_j being read. That means, we expect this approach to sort out all but r/G candidates. If our spy process manages to interrupt very soon after allowing it to run again, we have $r \ll G$. This may enable us to use conventional hardware to run full `ROMix` to search for the correct password among the candidates on the list.

The Final Cache-Timing Attack. In this attack we allow the second interrupt to arrive very late – maybe even as late as the termination time of `ROMix`. So, the loop in Lines 24–27 of `ROMix` has been run $r = G$ times. As it seems, each v_i has been read once. But actually, this is only true *on the average*; some v_i have been read more than once, and we expect about $(1/e)G \approx 0.37\,G$ array elements v_i not being read at all. So, applying the basic attack allows us to eliminate about 37 % of all password candidates – a rather small gain for such hard work.

In the following we introduce a way to push the attack further, inspired by Algorithm 17, the memory-constrained `ROMixMC`. Our final cache-timing attack on `scrypt` does only need the smallest possible amount of memory: $S = 1, K = G/S = G$, and thus, we only have to store the single value v_0. Like the second phase of `ROMixMC`, we will compute the values v_j on the fly when needed. Unlike `ROMixMC`, we will stop execution whenever one of our values j is such that v_j has not been read by `ROMix` (according to the information from our spy process).

Thus, if the first v_j has not been read, we immediately stop the execution without any on-the-fly computation; if the first v_j has been read, but not the second, we need one on-the-fly computation of v_j, and so forth.

Since a fraction, *i.e.,* $1/e$, of all values v_i has not been read, we will need about $1/(1 - 1/e) \approx 1.58$ on the fly computations of some v_j, each at the average price of $(G - 1)/2$ times calling $\mathcal{H}$. Additionally, each iteration needs one call to $\mathcal{H}$ for computing $x \leftarrow \mathcal{H}(x \oplus v_j)$. Including the work for the first phase, with G calls to $\mathcal{H}$, the expected number of calls to reject a wrong password is about

$$G + 1.58 \cdot \left(1 + \frac{G-1}{2}\right) \approx 1.79\,G.$$

As it turns out, rejecting a wrong password with constant memory is faster than computing ordinary `ROMix` with all the required storage, which actually makes $2G$ calls to $\mathcal{H}$, without computing any v_i on the fly. We stress that the ability to abort the computation, thanks to the information gathered by the spy process, is crucial.

10.4.3. Discussion

At the current point of time, our cache-timing attacks are theoretical. Even if one manages to run a spy process on a machine using `scrypt`, the requirement to interrupt `ROMix` twice at the right points of time is demanding. Nevertheless, even the theoretical ability of mounting such attacks should be seriously taken into account.

The idea of attacking cryptographic algorithms from hardware side (side-channel attacks) is not new [145], neither is the usage of a spy process for theoretical cache-timing attacks [190]. In [31], Bernstein demonstrated practically how to recover AES keys by using cache-timing information:

> The problem lies in AES itself: it is extremely difficult to write constant-time high-speed AES software [...]. Constant time low-speed AES software is fairly easy to write but is also unacceptable for many applications.

Similarly, we argue that there is a problem in `scrypt` itself. One can certainly implement `scrypt` such that cache-timings do not leak information about the password. But, we believe this would drastically reduce the performance of `scrypt`. As a compensation – recall that password scramblers are intentionally slow, but must be "fast enough" for the user – one would have to set the cost parameter G to some smallish value. But, this would only make regular attacks more efficient since attackers can use faster implementations. At the end of the day, this may defeat the entire point of using `scrypt` at all.

Note that this cache-timing attack has even more severe consequences. It does not only speed-up regular password-guessing attacks where the password hash is already in possession of the adversary. It also enables an adversary $\mathcal{A}$ to recover a password without knowing the password hash at all by just verifying the memory-access pattern.

The core of the problem is the fact that `ROMix` reads a value v_j, where the index $j \leftarrow x \bmod G$ depends on x and thus, on the password. It would be very convenient to have a password scrambler which is sequentially memory-hard *and* computes j in some password-independent way, *i.e.,* only depending on the loop index i. In the next section we actually present such a λ-memory-hard password scrambler, Catena.

10.4.4. The Garbage-Collector Attack

Here we introduce another memory-based issue of the ROMix algorithm. Typical attackers try plenty of password candidates in parallel, and this gets a lot more costly if they need a huge amount of memory for each candidate. The defender, on the other hand, will only compute a single hash, and the parameters (especially the "garlic") should be chosen such that the required amount of memory is easily available to the defender.

But, memory-demanding password scrambling may also provide completely new attack opportunities for the adversary. If we allocate a huge block of memory for password scrambling, holding $v_0, v_1, \ldots, v_{G-1}$, this memory becomes "garbage" after the password scrambler has terminated, and will be collected for reuse, eventually. One usually assumes that the adversary learns the hash of the secret. The *garbage-collector attack* assumes that the adversary additionally learns the memory content, *i.e.,* the values v_i, after the termination of the password scrambler.

For ROMix, the value $v_0 = \mathcal{H}(x)$ is a plain hash of the original secret x. Hence, the garbage-collector adversary can bypass ROMix completely and search directly for x with $\mathcal{H}(x) = v_0$, implying that each password candidate can be checked in time and memory $O(1)$. Thus, ROMix does not provide much defense against garbage-collector attacks. As a possible countermeasure, one can simply overwrite $v_0, \ldots, v_{G-1}$ after running ROMix. But, this step might be removed by a compiler due to optimization, since it is algorithmically ineffective.

10.5. Specification of Catena

In this section we introduce our password scrambler Catena. More detailed, we first specify Catena and explain its properties regarding to password hashing, *i.e.,* client-independent update and server relief. Afterwards, we present a instantiations of Catena, called Catena-DBG.

A formal definition is shown in Algorithm 18, based on two building blocks: (1) the cryptographic hash function $\mathcal{H}$ (see Lines 1 and 4) and (2) the memory-consuming n-bit hash function $\mathcal{F}_\lambda$ (see Line 3). Note that we require that the function $\mathcal{F}_\lambda$ is

1. λ-memory hard,

Algorithm 18 CATENA

Input: λ {Depth}, g_0 {Initial Garlic}, pwd {Password},
u {Tweak}, s {Salt}, g {Garlic}

Output: x {hash of the password}

```
1: x ← H(u || pwd || s)
2: for c = g_0, ..., g do
3:     x ← F_λ(c, x)
4:     x ← H(c || x)
5: end for
6: return x
```

2. collision resistant and,

3. its memory-access pattern is independent of the password derived input x.

Note that the for loop (Line 2–5) is required to provide client-independent updates. For the initial deployment of CATENA, we recommend to set the initial garlic value g_0 to g to achieve the best ratio between running time and memory usage. For the sake compatibility λ and g_0 should never be updated.

Note that a secure password scrambler must satisfy preimage security. It is easy to see that CATENA inherits the preimage security from the underlying hash function $\mathcal{H}$.

Next, we discuss the tweak and two further novel features of CATENA.

Tweak. The tweak u is an additional multi-byte value which is given by:

$$u \leftarrow d \mid\mid \lambda \mid\mid n \mid\mid |s| \mid\mid \mathcal{H}(H),$$

where the first byte d denotes the mode (domain) for which CATENA is used: $d = 0$ when used as a password scrambler, and $d = 1$ when used as a KDF (see Section 10.7). All remaining possible values for d are reserved for future applications. The second byte λ (depth) defines together with the memory cost parameter g (garlic) the security parameters for CATENA. The next 16-bit value n denotes the output length of the underlying hash function $\mathcal{H}$ in bits. The 32-bit value $|s|$ denotes the total length of the salt in bits. The last n-bit value $\mathcal{H}(H)$ is the hash of the associated data H, which can contain additional information like hostname, user-ID, name of the company, or the IP address of the host, with the goal to customize the password hashes. The tweak is processed together with the secret password and the salt (see

Algorithm 18, Line 1). Thus, the tweak u can be seen as a weaker version of a salt, increasing the additional computational effort for an adversary when using different values. Furthermore, it allows to differentiate between password hashing and key derivation.

Client-Independent Update. Its sequential structure does enable Catena to provide client-independent updates. Let $h = \text{Catena}_\lambda(pwd, u, s, g)$ be the hash of a specific password pwd, where u, s, and g denote the tweak, the salt, and the garlic. After increasing the security parameter from g to $g' = g + 1$, we can update the hash value h without user interaction by computing:

$$h' = \mathcal{H}(g' \mid\mid \mathcal{F}_\lambda(g', h)).$$

It is easy to see that the equation $h' = \text{Catena}_\lambda(pwd, u, s, g')$ holds.

Server Relief. In the last iteration of the **for**-loop in Algorithm 18, the client has to omit the last invocation of the hash function $\mathcal{H}$ (see Line 4) and then transmits the output of Catena to the server. Afterwards, the server computes the password hash by applying the hash function $\mathcal{H}$. Thus, the vast majority of the effort (memory usage and computational time) for computing the password hash is handed over to the client exonerating the server. This enables someone to deploy Catena even under restricted environments or when using constrained devices – or when a single server has to handle a huge amount of authentication requests.

Keyed Password Hashing. To further thwart off-line attacks, we introduce a technique to use Catena for keyed password hashing, where the password hash depends on both a password and a secret key K. Note that K is the same for all users, and thus, it has to be stored on the server. To preserve the server-relief property (see above), we encrypt the output of Catena using the XOR operation with $\mathcal{H}(K \mid\mid \text{userID} \mid\mid g \mid\mid K)$, which, under the reasonable assumption that the value $(\text{userID} \mid\mid g)$ is a nonce, was proven to be CPA-secure in [205]. Then, the keyed password hash y is given by

$$y := \text{Catena}_\lambda(pwd, u, s, g) \oplus \mathcal{H}(K \mid\mid \text{userID} \mid\mid g \mid\mid K),$$

where the userID is a unique and user-specific identification number which is assigned by the server. Now, we show what happens during the client-independent update, *i.e.*, when $g = g + r$ for arbitrary integer $r > 0$. The process takes the following four steps:

1. Given K and userID, compute $w = \mathcal{H}(K \mid\mid \text{userID} \mid\mid g \mid\mid K)$.
2. Compute $x = y \oplus w$, where y denotes the current keyed hash value.
3. Update x, *i.e.*, $x = \mathcal{H}(c \mid\mid \mathcal{F}_\lambda(c, x))$ for $c \in \{g+1, \dots, g+r\}$.
4. Compute the new hash value $y = y \oplus \mathcal{H}(K \mid\mid \text{userID} \mid\mid g+r \mid\mid K)$.

Remark. Obviously, it is a bad idea to store the secret key K on the same place as the password hashes since it can be leaked in the same way as the password-hash database. One possibility to separate the key from the hashes is to securely store the secret key by making use of hardware security modules (HSM), which provide a tamper-proof memory environment with verifiable security. Then, the protection of the secret key depends on the level provided by the HSM (see FIPS140-2 [57] for details). Another possibility is to derive K from a password during the bootstrapping phase. Afterwards, K will be kept in the RAM and will never be on the hard disk drive. Thus, the key and the password-hash database should never be part of the same backup file.

10.6. Security Analysis of Catena

We denote a password scrambler to be secure if it provides at least 1-memory-hardness and preimage security. Furthermore, it should be resistant against cache-timing attacks. It is easy to see that Catena inherits its λ-memory-hardness from $\mathcal{F}_\lambda$. Since the memory-access pattern of Catena is static and therefore, independent from the password, it provides resistance against cache-timing attacks. Finally, we show that Catena is a secure passsword scrambler that behaves like a good random function, which is useful for using Catena as a secure KDF. Before we present our claims, we introduce some essential knowledge, which ease the understanding of our proofs.

Password-Recovery Resistance. In this section we show that Catena is a good password scrambler, *i.e.,* given the hash value h it is infeasible for an adversary to do better than trying out password candidates in likelihood order to obtain the correct password.

Theorem 10.5 (Catena is Password-Recovery Resistant). *Let m denote the min-entropy of a passwords source $\mathcal{Q}$. Then, it holds that*

$$\mathbf{Adv}^{REC}_{\text{Catena},\mathcal{Q}}(q) \leq \frac{q}{2^m} + \mathbf{Adv}^{pre}_{\mathcal{H}}(q, t).$$

Proof. Note that an adversary $\mathcal{A}$ can always guess a (weak) password by trying out about 2^m password candidates. For a maximum of q queries, it holds that the success probability is given by $q/2^m$. Instead of guessing 2^m password candidates, an adversary can also try to find a preimage for a given hash value h. It is easy to see from Algorithm 18 that an adversary thus has to find a preimage for $\mathcal{H}$ in Line 4. More detailed, for a given value h with $h \leftarrow \mathcal{H}(g, x)$, $\mathcal{A}$ has to find a valid value for x. The success probability for this can be upper bounded by $\mathbf{Adv}_{\mathcal{H}}^{pre}(q,t)$. Our claim follows by adding up the individual terms. ∎

Pseudorandomness. In the following we analyze the advantage of an adversary $\mathcal{A}$ in distinguishing the output of Catena from a random bitstring of the same length as the output of Catena. Therefore, we model the internally used hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^n$ as a random oracle.

Theorem 10.6 (PRF Security of Catena). *Let q denote the number of queries made by an adversary and s a randomly chosen salt value. Furthermore, let $\mathcal{H}$ be modelled as a random oracle and $g \geq g_0 \geq 1$. Then, it holds that*

$$\mathbf{Adv}_{\text{CATENA}_\lambda}^{PRF}(q,t) \leq (q \cdot g + q)^2/2^n + \mathbf{Adv}_{F_\lambda}^{coll}(g \cdot q)$$

Proof. Suppose that $a^i = (pwd^i \mid\mid u^i \mid\mid s^i \mid\mid g)$ represents the i-th query, where pwd^i denotes the password, u^i denotes the tweak, s^i the salt, and g the garlic. For this proof, we impose the reasonable condition that all queries of an adversary are distinct, *i.e.*, $a^i \neq a^j$ for $i \neq j$.

Suppose that y^j denotes the output of $F_\lambda(g, a^j)$ of the j-th query (cf. Algorithm 18, Line 3). Then, $\mathcal{H}(g \mid\mid y^j)$ is the output of $\text{CATENA}_\lambda(a^j)$. In the case that $y^1, \ldots, y^q$ are pairwise distinct, an adversary $\mathcal{A}$ cannot distinguish $\mathcal{H}(g \mid\mid \cdot)$ from a random function $\$(\cdot)$ since in the random-oracle model, both functions return a value chosen uniformly at random from $\{0,1\}^n$.

Therefore, we have to upper bound the probability of the event $y^i = y^j$ with $i \neq j$. Due to the assumption that $\mathcal{A}'s$ queries are pairwise distinct, there must be at least one collision for $\mathcal{H}$ or $\mathcal{F}_\lambda$. For q queries, we have at most $q(g+1)$ invocations of $\mathcal{H}$. Thus, we can upper bound the collision probability by

$$(q \cdot g + q)^2/2^n.$$

Furthermore, we have $q \cdot g$ invocations of the memory-consuming function $\mathcal{F}_\lambda$. We can upper bound the probability of a collision by $\mathbf{Adv}^{coll}_{F_\lambda}(g \cdot q)$. Our claim follows from the union bound. ∎

10.7. The Catena-KG Key-Derivation Function

In this section, we introduce CATENA-KG – a mode of operation based on CATENA, which can be used to generate keys of different sizes (even larger than the natural output size of CATENA (cf. Algorithm 19). To provide uniqueness of the inputs, the domain value d of the tweak is set to 1, *i.e.,* the tweak u' is given by

$$u' \leftarrow \texttt{0x01} \,||\, \lambda \,||\, n \,||\, |s| \,||\, \mathcal{H}(H).$$

Then, the call of CATENA is followed by an output transformation that takes the output x of CATENA, a *key identifier* $\mathcal{I}$, and a parameter ℓ_K for the key length as input, and generates key material of the desired output size. CATENA-KG is even able to handle the generation of extra-long keys (longer than the output size of $\mathcal{H}$) by applying $\mathcal{H}$ in CTR Mode [84]. Note that longer keys do not imply improved security, in that context. The key identifier $\mathcal{I}$ is supposed to be used when different keys are

Algorithm 19 CATENA-KG

Input: pwd {Password}, u' {Tweak}, s {Salt}, g {Garlic}, $\mathcal{I}$ {Key Identifier}
Output: K {ℓ_K-Bit Key Derived from the Password}

```
x ← Catena_λ(pwd, u', s, g)
K ← ∅
for i = 1, ..., ⌈ℓ_K/n⌉ do
    K ← K || H(i || I || ℓ_K || x)
end for
return Truncate(K, ℓ_K) {truncate k to the first ℓ_K bits}
```

generated from the same password. For example, when Alice and Bob set up a secure connection, they may need four keys: An encryption and a message-authentication key for messages from Alice to Bob, and another two keys for the opposite direction. One could argue that $\mathcal{I}$ should also become part of the associated data. Actually, this would be a bad move. Setting up the connection would require legitimate users to run CATENA several times. But, the adversary can search for the password for one key, and just derive the other keys, once that password has been found. Instead, one should rather employ a single call to CATENA with larger security parameters and then run the output transformation for each key. In contrast to the password-hashing scenario, where a user want to log in without noticeable delay, users may tolerate a delay of several seconds to derive an encryption key from a password [231], *e.g.*, when setting up a secure connection, or when mounting a cryptographic file system. Thus, we recommend to use higher values for g for key-derivation.

Security Analysis. It is easy to see that Catena-KG inherits its λ-memory-hardness from Catena since it invokes Catena (Line 1 of Algorithm 19). Next, we show the PRF security of Catena-KG in the random-oracle model.

Theorem 10.7 (Catena-KG Security). *Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^n$ be a random function. Then, for $g \geq g_0 \geq 1$, it holds that*

$$\mathbf{Adv}^{PRF}_{\text{CATENA-KG}_\lambda}(q,t) \leq (q \cdot g + q)^2/2^n + \mathbf{Adv}^{coll}_{F_\lambda}(g \cdot q)$$

Proof. For the sake of simplification, we omit the truncation step and let the adversary always get access to the untruncated key K. Since $\mathcal{H}$ is a random function, the only chance for an adversary to distinguish Catena-KG$_{(\mathcal{H},\lambda,g)}(\cdot)$ from a random n-bit function is an input collision in Line 4 of Algorithm 19. Thus we have to upper bound the probability that two outputs of Catena $_\lambda$ collide. Let $a^i = (pwd^i \mid\mid u^i \mid\mid s^i \mid\mid g^i)$ denote the i-the query, where pwd^i denotes the password, u^i denotes the tweak, s^i the salt, and g^i the garlic. A collision between two distinct queries a^i and a^j, *i.e.*, $\text{CATENA}_\lambda(a^i) = \text{CATENA}_\lambda(a^j)$ with $a^i \neq a^j$, implies a collision in $\mathcal{H}$. The probability for this event can be upper bounded by

$$(q \cdot g + q)^2/2^n + \mathbf{Adv}^{coll}_{F_\lambda}(g \cdot q),$$

using similar arguments as in the proof of Theorem 10.10. ∎

10.8. Catena-DBG

In this section we introduce Catena-DBG, a concrete instantiation of Catena where $\mathcal{F}_\lambda$ is instantiated with the Double Butterfly Hashing (DBH) operation that is based on a stack of λ G-superconcentrators. The following definition of a G-superconcentrator is a slightly adapted version of that introduced in [151].

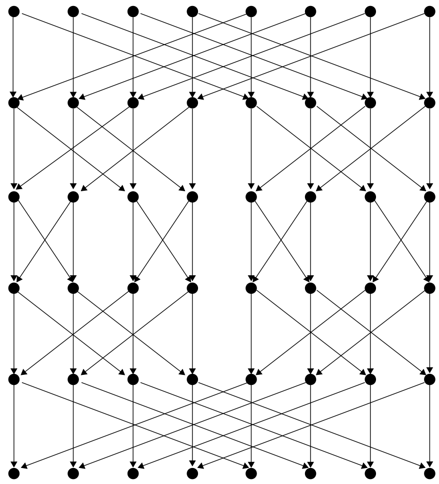

Figure 10.1.: A Cooley-Tukey FFT graph with eight input and output vertices.

Definition 10.8 (G-Superconcentrator). *A Directed Acyclic Graph (DAG) with a set of vertices $\mathfrak{V}$ and a set of edges $\mathfrak{E}$, a bounded indegree, G inputs, and G outputs is called a G-superconcentrator if for every k such that $1 \leq k \leq G$ and for every pair of subsets $\mathfrak{V}_1 \subset \mathfrak{V}$ of k inputs and $\mathfrak{V}_2 \subset \mathfrak{V}$ of k outputs, there are k vertex-disjoint paths connecting the vertices in $\mathfrak{V}_1$ to the vertices in $\mathfrak{V}_2$.*

Double Butterfly Graph (DBG). A DBG is a G-superconcentrator which is defined by the graph representation of two back-to-back placed Fast Fourier Transform (FFT) [53]. More detailed, it is a representation of twice the Cooley-Tukey FFT algorithm [65] omitting one row in the middle (see Figure 10.1 for an example where $g = 3$). Therefore, a DBG consists of $2g$ rows.

Based on the DBG we define the sequential and stacked (DBG_λ^g) where the security parameters λ and g determine the depth (number of stacked superconcentrators) and the width (number of nodes per row, *i.e.,* 2^g), respectively. In the following, we denote $v_{i,j}^k$ as the j-th vertex in the i-th row of the k-th superconcentrator. Note that in this thesis we use the vertices $v_{0,j}^k$ and $v_{2^g-1,j}^{k-1}$ as synonyms since due to the stacking of λ DBGs the last row of the $k-1$-th DBG is identical to the first row of the k-th DBG.

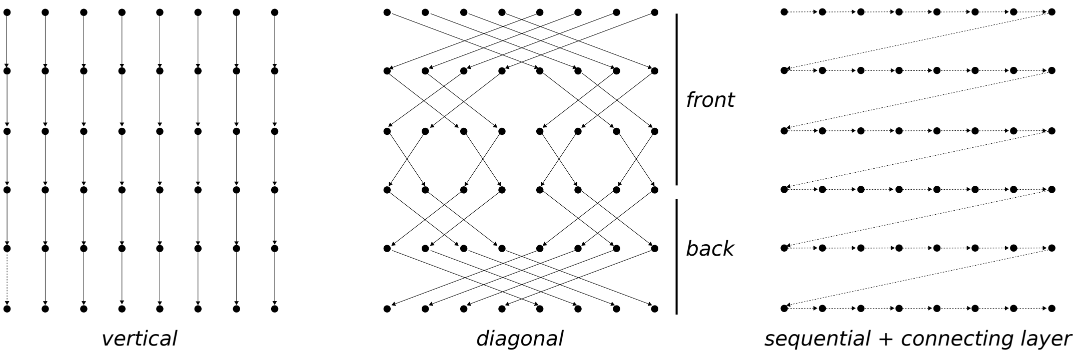

Figure 10.2.: Types of edges of an $(3,1)$-Double Butterfly Graph.

Definition 10.9 ($\mathbf{DBG}^g_\lambda$). *Fix two integers $g, \lambda \geq 1$, then the (g,λ)-Double Butterfly Graph (DBG^g_λ) $\Pi(\mathcal{V},\mathcal{E})$ consists of $2^g(\lambda(2g-1)+1)$ vertices*

$$\left(\bigcup_{i=0}^{2g-2}\bigcup_{j=0}^{2^g-1}\bigcup_{k=1}^{\lambda}\{v^k_{i.j}\}\right) \cup \left(\bigcup_{j=0}^{2^g-1}\{v^\lambda_{2g-1,j}\}\right)$$

and $\lambda \cdot (2g-1) \cdot (3 \cdot 2^g) + 2^g - 1$ edges

- *vertical: $2^g \cdot (\lambda \cdot (2g-1))$ edges*

$$\bigcup_{i=0}^{2g-2}\bigcup_{j=0}^{2^g-1}\bigcup_{k=1}^{\lambda}\{v^k_{i,j}, v^k_{i+1,j}\}$$

- *diagonal: $2^g \cdot \lambda \cdot g \, + 2^g \cdot \lambda \cdot (g-1)$ edges*

$$\bigcup_{k=1}^{\lambda}\bigcup_{j=0}^{2^g-1}\left(\left(\bigcup_{i=0}^{g-1}\{v^k_{i,j}, v^k_{i+1,j\oplus 2^{g-1-i}}\}\right) \cup \left(\bigcup_{i=g}^{2g-2}\{v^k_{i,j}, v^k_{i+1,j\oplus 2^{i-(g-1)}}\}\right)\right)$$

- *sequential: $(2^g-1) \cdot (\lambda \cdot (2g-1)+1)$ edges*

$$\left(\bigcup_{i=1}^{2g-1}\bigcup_{j=0}^{2^g-2}\bigcup_{k=1}^{\lambda}\{v^k_{i,j}, v^k_{i,j+1}\}\right) \cup \left(\bigcup_{j=0}^{2^g-2}\{v^\lambda_{2g-1,j}, v^\lambda_{2g-1,j+1}\}\right)$$

- *connecting layer: $\lambda \cdot (2g-1)$ edges*

$$\bigcup_{i=1}^{2g-2}\bigcup_{k=1}^{\lambda}\{v^k_{i,2^g-1}, v^k_{i+1,0}\}$$

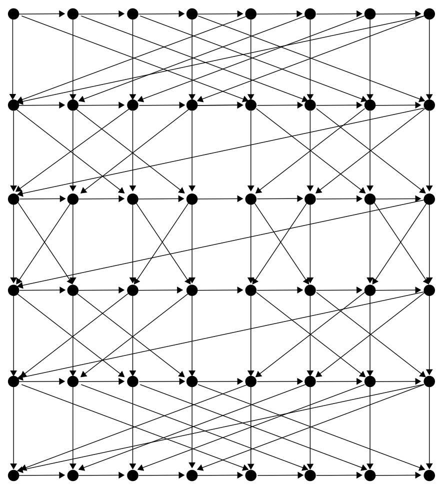

Figure 10.3.: An $(3,1)$-Double Butterfly Graph.

For the parameter set $g = 3$ and $\lambda = 1$ Figure 10.2 illustrates the individual types of edges we use in our Definition above. Moreover, an example for an $(3,1)$-Double Butterfly Graph (DBG) Figure 10.3.

Double Butterfly Hashing (DBH). The DBH_λ^g operation is defined in Algorithm 20. The structure is based of a DBG_λ^g. Note that the function σ (see Lines 7 and 9) is given by

$$\sigma(g,i,j) = \begin{cases} j \oplus 2^{g-1-i} & \text{if } 0 \leq i \leq g-1, \\ j \oplus 2^{i-(g-1)} & \text{otherwise.} \end{cases}$$

Thus, σ determines the indices of the vertices of the diagonal edges.

Since the security of CATENA in terms of password hashing is based on a time-memory tradeoff, it is desired to implement it in an efficient way, making it possible to increase the required memory. We recommend BLAKE2b [13] as the underlying hash function, implying a block size of 1024 bits with 512 bits of output. Thus, it can process two input blocks within one compression function call. For CATENA-DBG, we cannot simply concatenate the inputs to the hash function $\mathcal{H}$ while keeping the same performance per hash function call, *i.e.,* three inputs to $\mathcal{H}$ require two compression function calls. Therefore, we compute $\mathcal{H}(X,Y,Z) = \mathcal{H}(X, \oplus Y \mid\mid Z)$ instead of $\mathcal{H}(X,Y,Z) = \mathcal{H}(X \mid\mid Y \mid\mid Z)$. Obviously, this doubles the probability of input collisions. Nevertheless, for a 512-bit hash function, the advantage for an adversary is still negligible.

Algorithm 20 Double Butterfly Hashing (DBH)

Input: g {Garlic}, x {Value to hash}, λ {Depth}, $\mathcal{H}$ {Hash Function}
Output: x {Password Hash}

$v_0 \leftarrow \mathcal{H}(x)$
for $i = 1, \ldots, 2^g - 1$ **do**
  $v_i \leftarrow \mathcal{H}(v_{i-1})$
end for
for $k = 1, \ldots, \lambda$ **do**
  for $i = 1, \ldots, 2g - 1$ **do**
    $r_0 \leftarrow \mathcal{H}(v_{2^g-1} \oplus v_0 \mid\mid v_{\sigma(g,0,j)})$
    for $j = 1, \ldots, 2^g - 1$ **do**
      $r_i \leftarrow \mathcal{H}(r_{i-1} \oplus v_i \mid\mid v_{\sigma(g,i,j)})$
    end for
    $\vec{v} \leftarrow \vec{r}$
  end for
end for
return $x \leftarrow v_{2^g-1}$

10.9. Analysis of Catena-DBG

Next, we discuss the security of Catena-DBG against side-channel attacks. Furthermore, we discuss the memory-hardness and collision resistance of the DBH^g_λ operation.

10.9.1. Side-Channel Attacks and Collision Resistance

Straightforward implementations of Catena-DBG have neither password-dependent memory-access pattern nor have they password-dependent branches. Therefore, our proposed instantiation of Catena is resistant against cache-timing attacks.

Considering a malicious garbage collector, Algorithm 20 exposes two arrays, namely v and r. Both are overwritten multiple times. Therefore, Catena-DBG is resistant against garbage-collector attacks. Note that Catena-DBG with some $\lambda \geq 2$ is at least as resistant to garbage-collector attacks as the same variant with $\lambda - 1$ without a malicious garbage collector.

Next, we analyze the collsion resistance of DBH^g_λ. Therefore, we model the internally used hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^n$ as a random oracle.

Theorem 10.10 (Collision Security of $\mathbf{DBH}^g_\lambda$). *Let q denote the number of queries. Furthermore, let $\mathcal{H}$ be modelled as a random oracle for some fixed integers $g, g_0, \lambda \geq 1$ with $g \geq g_0$ and $G = 2^g$. Then, it holds that*

$$\mathbf{Adv}^{coll}_{DBH^g_\lambda}(q,t) \leq \frac{(q \cdot \lambda \cdot g)^2}{2^{n-2g-3}}.$$

Proof. From Algorithm 20 it is easy to see that collision $\mathrm{DBH}^g_\lambda(x) = \mathrm{DBH}^g_\lambda(x')$ for $x \neq x'$ implies either a input or output collision for $\mathcal{H}$.

For our analysis, we replace the random oracle $\mathcal{H}$ by $\mathcal{H}'(x) := \mathcal{H}(\mathrm{truncate}_n(x))$ that truncates any input to n bits before hashing. Thus, any collision in the first n bits $\mathcal{H}$ in Line 7 and 9 of Algorithm 20 leads to a collision, regardless of the remaining inputs.

Output Collision. In this case, we can upper bound the collision probability of $\mathcal{H}$ by deducing the total amount of invocations of $\mathcal{H}'$ per query. There are G invocations of $\mathcal{H}'$ in Lines 1–4. of Algorithm 20. In addition, there are $\lambda(2g-1)G$ invocations in Lines 5-14 of Algorithm 20. In total, we have $\lambda 2gG$ invocations. Since $\mathcal{H}$ is modelled as a random oracle, we can upper bound the collision probability for q queries by

$$\frac{(q \cdot \lambda \cdot 2g \cdot G)^2}{2^n} \leq \frac{q^2\lambda^2 g^2}{2^{n-2g-2}}.$$

Input Collision. In this case, we have to take into account that a input collision for distinct queries a and b in Line 7 and 9 can occur:

$$v^a_{2^g-1} \oplus v^a_0 = v^b_{2^g-1} \oplus v^b_0 \qquad \text{(Algorithm 20, Line 7)}$$

or

$$r^a_{i-1} \oplus v^a_i = r^b_{i-1} \oplus v^b_i \qquad \text{(Algorithm 20, Line 9).}$$

For each query this can happen $\lambda \cdot (2g-1) \cdot 2^g$ times. Note that all values v_i, r_i are outputs from the random oracle $\mathcal{H}'$, except the initial value for v_0. Hence, we can upper bound the collision probability for this event by

$$\frac{(q\lambda \cdot (2g-1) \cdot 2^g)^2}{2^n} \leq \frac{q^2\lambda^2 g^2}{2^{n-2g-2}}.$$

Our claim follows from the union bound. ■

10.9.2. Memory Hardness.

In 1970, Hewitt and Paterson introduced a method for analyzing Time-Memory Tradeoffs (TMTOs) on directed acyclic graphs [189], called *pebble game*. While their method has been known for decades, it was recently used in a cryptographic context, see *e.g.,* [86]. In general, a pebble game is a common model to derive and analyze TMTOs as shown in [216, 217, 222, 227, 229].

The pebble-game model is restricted to DAGs with bounded in-degree and can be seen as a single-player game. Let $\Pi(\mathcal{V}, \mathcal{E})$ be a DAG and let $G = |\mathcal{V}|$ be the number of vertices within $\Pi(\mathcal{V}, \mathcal{E})$. In the setup phase of the game, the player gets S pebbles (tokens) with $S \leq G$. A pebble can be placed (*pebble*) or be removed (*unpebble*) from a vertex $v \in \mathcal{V}$ under certain requirements:

1. A pebble may be removed from a vertex v at any time.

2. A pebble can be placed on a vertex v if all predecessors of the vertex v are marked.

3. If all immediate predecessors of an unpebbled vertex v are marked, a pebble may be moved from a predecessor of v to v.

A *move* is the application of either the second or the third action stated above. The goal of the game is to pebble Π, *i.e.,* to mark all vertices of the graph Π at least once. The total amount of moves represent the computational costs.

In [151], Lengauer and Tarjan have already analyzed the TMTO for a stack of λ G-superconcentrators. Since the double-butterfly is a special form of a G-superconcentrators there bound also holds for DBG_λ^g.

Theorem 10.11 (TMTO for a stack of λ G-Superconcentrators [151]). *For pebbling a stack of λ G-Superconcentrators using $S \leq G/20$ pebbles it holds that*

$$T \;\geq\; G\left(\frac{\lambda G}{64S}\right)^{\lambda}.$$

Note that the DBH operation computes a special variations of the DBG_λ^g where each vertex represents the the hash values of its direct predecessors. Thus, DBH_λ^g and therefore Catena-DBG inherits the λ-memory hardness from DBG_λ^g.

Discussion. We have to point out that the computational effort for DBH_λ^g with reasonable values for G, *e.g.,* $G \in [2^{17}, 2^{21}]$, may stress the patience of many users since the number of vertices and edges grows logarithmic with G. Thus, it remains an open research problem to find a G-superconcentrator – or any other λ-memory-hard function – that can be computed more efficiently than a DBH_λ^g.

10.10. Results Summary

We introduced a new class of side-channel attacks, called garbage-collector attack, which bases on a malicious garbage collector. We showed that the common password scrambler `scrypt` is vulnerable to this kind of attacks. Furthermore, we presented a (theoretical) cache-timing attack on `scrypt` that exploits its password-dependent memory-access pattern. Both attacks allows an adversary to construct a *memoryless* password filter that enables massively-parallel password-guessing attacks. Moreover, we show that our attacks work even without knowledge of the password hash. All regular implementations, *i.e.,* implementations that are not hardened against side-channel attacks, of password scramblers with a password-dependent memory-access pattern appear to be vulnerable to these attacks.

As a remedy, we introduced a novel password-scrambler framework Catena, which is based on a λ-memory-hard function. It is the first framework which naturally supports client-independent updates and server relief. It consists of two security parameters λ (depth) and g (garlic), where λ reflects the memory hardness and g the memory consumption. In addition, we have shown that Catena is provably secure in the random oracle model.

Furthermore, we presented a DBH based instantiatation of Catena, Catena-DBG. Note that DBH basically computes a stack of several Double Butterfly Graph where each vertex of the graph is the hash value of its direct predecessors.

Finally, we want to stress out that the limited practicality of this implementation. There is a good chance that the runtime of Catena-DBG might exceed the patience of many users.

Part IV

Epilog

11 Conclusion

> Science never solves a problem without creating ten more.
>
> — George Bernard Shaw

In this section we conclude this thesis by giving a brief summary of the main contributions and emphasize some further work as well as open research topics.

11.1. Summary

Robustness. One of the main topics of this thesis is the analysis and design of misuse-resistant authenticated encryption schemes. The field of robust authenticated encryption schemes was pioneered by Rogaway and Shrimpton [210] who introduced the notion of *(nonce-) misuse resistance* in 2006. During our research we came up with a generalized definition of robustness as well as the security notion of decryption misuse.

Moreover, we introduced two novel on-line authenticated encryption schemes: McOE and COFFE. The latter one is the first provably secure OAE scheme that has been designed for the usage of a hash function rather than a block cipher as the underlying primitive. In contrast to conventional AE schemes, COFFE also provides ciphertext integrity in the nonce-misuse scenario. The former one, McOE, was presented at FSE 2012 [98]. It was the first robust OAE scheme published by then. This academic work inspired fellow researchers to introduce new nonce-misuse resistant OAE schemes [5, 7, 73]. Our work seem to have influenced the submission requirements of the upcoming Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR):

> ... that the cipher is designed to provide the maximum possible robustness against message-number reuse[1].

We expect that our contributions in terms of nonce- and decryption-misuse resistance will foster misuse awareness in future (O)AE scheme designs. Furthermore, we do believe that providing a second line of defence – by applying robust authenticated encryption – helps to make the IT-world a little bit more secure.

Hash Function Design. Another main topic of this work is the presentation of TWISTER$_\pi$, a family of cryptographic hash functions. It is a revised version of the SHA-3 submission TWISTER [88] that has some vulnerabilities against certain rebound attacks [166]. In the revision process we applied effective countermeasures to overcome those weaknesses. Until now, no attacks are known.

Password Scrambler. Inspired by the discovery of cache-timing attacks on `scrypt`, we designed CATENA, the first provably secure and memory-consuming password scrambler that does not only thwart GPU-based attacks, but also provide a password independent memory-access pattern to render cache-timing attacks infeasible. Furthermore, CATENA naturally supports client-independent updates and server relief, and it is provable secure in the random oracle model. The program chair of the Password Hashing Competition (PHC) has serendipitously added support for client-independent update as a functional requirement and cache-timing resistance as a security requirement[2]. Finally, we hope that our contribution lay the groundwork for all subsequent password-hashing schemes.

11.2. Further Research

Due to the CAESAR contest, the design and analysis of authenticated encryption schemes is a hot topic in the field of symmetric cryptography. Inside the cryptographic community, there is a clear consensus about the fact that the notion of secure AE (CCA3 security) is the de facto gold standard for the vast majority of secure channels. Nevertheless, there are still some open research topics.

[1] `http://competitions.cr.yp.to/caesar-call.html`
[2] `https://password-hashing.net/call.html`

- Is it possible to design an integrated tweakable block cipher $\widetilde{E} \in \texttt{Block}(k, u, n)$ which is more efficient than a constructed one? Instances of McOE or TC1 [211] would greatly benefit from such a primitive.

- What is about the software and hardware efficiency of such integrated primitives?

- McOE is highly sequential. Is it possible to construct a provably secure on-line authenticated encryption scheme which is both parallel and robust?

- Are there, apart from nonce and decryption misuse, any other misuse scenarios that should be taken into account by the cryptographic community?

- Shall possible security issues in the case of robustness also be discussed in public-key cryptography, *e.g.,* digital signatures or fully homomorphic encryption?

Cryptographers have almost orphaned the field of password-hashing schemes in spite of the medial omnipresence of leaked password databases. Therefore, designing a good password-hashing scheme is more an art than a science. The PHC tries to raise the awareness of this research topic. Imperatively, a solid theoretic foundation is needed, *i.e.,* rigorous analysis, formal definitions, and security notions.

- Is it possible to design a fast and parameterizable cryptographic hash function which can be turned – by an appropriate parameter choice – into a memory-hard KDF or password scrambler?

- How efficient would such a construction be in soft- or hardware?

- Are there any other relevant properties research should take a look at?

- Which reasonable security notions should become the gold standard for password-hashing schemes?

- Is it possible to construct a λ-memory hard function that is more efficient than our proposed DBG operation? For example, does a scalable G-superconentrator exist, that can be computed more efficently, *i.e.,* that has a linear number of edges and vertices?

12

List of Publications

The lists are ordered by the date of publication.

Lecture Notes in Computer Science

1. **Christian Forler**, Stefan Lucks, and Jakob Wenzel. Memory-Demanding Password Scrambling. In Tatsuaki Okamoto, editor, *ASIACRYPT*, volume 8874 of *Lecture Notes in Computer Science, pages 289–305. Springer, 2014. [104]*

2. *Farzaneh Abed,* ***Christian Forler****, Eik List, Stefan Lucks, and Jakob Wenzel. Counter-bDM: A Provably Secure Family of Multi-Block-Length Compression Functions. In David Pointcheval and Damien Vergnaud, editors,* AFRICACRYPT*, volume 8469 of* Lecture Notes in Computer Science*, pages 440–458. Springer, 2014. [4]*

3. *Farzaneh Abed, Scott Fluhrer,* ***Christian Forler****, Eik List, Stefan Lucks, David McGrew, and Jakob Wenzel. Pipelineable On-Line Encryption. In Carlos Cid and Christian Rechberger, editor,* Fast Software Encryption Lecture Notes in Computer Science*, Springer, 2014.* (Note: to appear.) *[1]*

4. *Farzaneh Abed,* ***Christian Forler****, Eik List, Stefan Lucks, and Jakob Wenzel. A Framework for Automated Independent-Biclique Cryptanalysis. In Shiho Moriai, editor,* FSE*, volume 8424 of* Lecture Notes in Computer Science*, pages 561–581. Springer, 2013. [3]*

5. *Ewan Fleischmann,* ***Christian Forler****, and Stefan Lucks.* McOE*: A Family of Almost Foolproof On-Line Authenticated Encryption Schemes. In Anne Canteaut, editor,* FSE*, volume 7549 of* Lecture Notes in Computer Science*, pages 196–215. Springer, 2012. [98]*

6. *Ewan Fleischmann,* ***Christian Forler****, and Stefan Lucks. The Collision Security of MDC-4. In Aikaterini Mitrokotsa and Serge Vaudenay, editors,* AFRICACRYPT*, volume 7374 of* Lecture Notes in Computer Science*, pages 252–269. Springer, 2012. [99]*

7. *Ewan Fleischmann,* ***Christian Forler****, Stefan Lucks, and Jakob Wenzel. Weimar-DM: A Highly Secure Double-Length Compression Function. In Willy Susilo, Yi Mu, and Jennifer Seberry, editors,* ACISP*, volume 7372 of* Lecture Notes in Computer Science*, pages 152–165. Springer, 2012. [101]*

8. ***Christian Forler****, Stefan Lucks, and Jakob Wenzel. Designing the API for a Cryptographic Library - A Misuse-Resistant Application Programming Interface. In Mats Brorsson and Luís Miguel Pinho, editors,* Ada-Europe*, volume 7308 of* Lecture Notes in Computer Science*, pages 75–88. Springer, 2012. [102]*

9. *Ewan Fleischmann,* ***Christian Forler****, and Stefan Lucks.* Γ-$MAC[H,P]$ *- A New Universal MAC Scheme. In Frederik Armknecht and Stefan Lucks, editors,* WEWoRC*, volume 7242 of* Lecture Notes in Computer Science*, pages 83–98. Springer, 2011. [97]*

10. *Ewan Fleischmann,* ***Christian Forler****, Michael Gorski, and Stefan Lucks. Collision Resistant Double-Length Hashing. In Swee-Huay Heng and Kaoru Kurosawa, editors,* ProvSec*, volume 6402 of* Lecture Notes in Computer Science*, pages 102–118. Springer, 2010. [94]*

11. *Ewan Fleischmann,* ***Christian Forler****, Michael Gorski, and Stefan Lucks. New Boomerang Attacks on ARIA. In Guang Gong and Kishan Chand Gupta, editors,* INDOCRYPT*, volume 6498 of* Lecture Notes in Computer Science*, pages 163–175. Springer, 2010. [95]*

12. *Ewan Fleischmann,* ***Christian Forler****, Michael Gorski, and Stefan Lucks. Twister - A Framework for Secure and Fast Hash Functions. In Feng Bao, Hui Li, and Guilin Wang, editors,* ISPEC*, volume 5451 of* Lecture Notes in Computer Science*, pages 257–273. Springer, 2009. [93]*

International Publications in Journals

1. Ewan Fleischmann, **Christian Forler**, Michael Gorski, and Stefan Lucks. TWISTER$_\pi$ - A Framework for Secure and Fast Hash Functions. International Journal of Applied Cryptography (*IJACT*), Volume 2 Number 1, pages 68–81, Inderscience, 2010. [96]

Further International Publications

1. **Christian Forler**, David A. McGrew, Stefan Lucks, and Jakob Wenzel. COFFE: Ciphertext Output Feedback Faithful Encryption. *IACR Cryptology ePrint Archive*, 2014:1003, 2014.

2. **Christian Forler**, Eik List, Stefan Lucks, and Jakob Wenzel. Overview of the Candidates for the Password Hashing Competition - And their Resistance against Garbage-Collector Attacks. *IACR Cryptology ePrint Archive*, 2014:881, 2014

3. Farzaneh Abed, **Christian Forler**, and Stefan Lucks. Classification of the CAESAR Candidates. *IACR Cryptology ePrint Archive*, 2014:792, 2014.

4. Farzaneh Abed, Scott Fluhrer, **Christian Forler**, Eik List, Stefan Lucks, David McGrew, and Jakob Wenzel. Pipelineable On-Line Encryption (full version of [1]). *IACR Cryptology ePrint Archive*, 2014:297, 2014.

5. Farzaneh Abed, Scott Fluhrer, **Christian Forler**, Eik List,Stefan Lucks, David McGrew, and Jakob Wenzel. The POET Family of On-Line Authenticated Encryption Schemes. Submission to the CAESAR competition, 2014.

6. **Christian Forler**, Stefan Lucks, and Jakob Wenzel. Catena: A Memory-Consuming Password Scrambler. Submission to the PHC, 2014.

7. **Christian Forler**, Stefan Lucks, and Jakob Wenzel. Catena: A Memory-Consuming Password Scrambler (full version of [104]). *IACR Cryptology ePrint Archive*, 2013:525, 2013. [103]

8. Ewan Fleischmann, **Christian Forler**, Stefan Lucks, and Jakob Wenzel. The Collision Security of MDC-4 (full version of [99]). *IACR Cryptology ePrint Archive*, 2012:096, 2012.

9. Farzaneh Abed, **Christian Forler**, Eik List, Stefan Lucks, and Jakob Wenzel. Biclique Cryptanalysis of the PRESENT and LED Lightweight Ciphers. *IACR Cryptology ePrint Archive*, 2012:591, 2012. [2]

10. Ewan Fleischmann, **Christian Forler**, Stefan Lucks, and Jakob Wenzel. McOE: A Foolproof On-Line Authenticated Encryption Scheme (full version of [98]). *IACR Cryptology ePrint Archive*, 2011:644, 2011. [100]

11. Ewan Fleischmann, **Christian Forler**, and Michael Gorski. Classification of the SHA-3 Candidates. *IACR Cryptology ePrint Archive*, 2008:511, 2008. [92]

12. Ewan Fleischmann, **Christian Forler**, and Michael Gorski. The Twister Hash Function Family. Submission to NIST, 2008. [88]

Bibliography

[1] Farzaneh Abed, Scott R. Fluhrer, Christian Forler, Eik List, Stefan Lucks, David A. McGrew, and Jakob Wenzel. Pipelineable On-Line Encryption. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption*, Lecture Notes in Computer Science (to appear). Springer, 2014.

[2] Farzaneh Abed, Christian Forler, Eik List, Stefan Lucks, and Jakob Wenzel. Biclique Cryptanalysis of the PRESENT and LED Lightweight Ciphers. *IACR Cryptology ePrint Archive*, 2012:591, 2012.

[3] Farzaneh Abed, Christian Forler, Eik List, Stefan Lucks, and Jakob Wenzel. A Framework for Automated Independent-Biclique Cryptanalysis. In Shiho Moriai, editor, *Fast Software Encryption - 20th International Workshop, FSE 201 3, Singapore, March 11-13, 2013. Revised Selected Papers*, volume 8424 of *Lecture Notes in Computer Science*, pages 561–581. Springer, 2013.

[4] Farzaneh Abed, Christian Forler, Eik List, Stefan Lucks, and Jakob Wenzel. Counter-bDM: A Provably Secure Family of Multi-Block-Length Compression Functions. In David Pointcheval and Damien Vergnaud, editors, *AFRICACRYPT*, volume 8469 of *Lecture Notes in Computer Science*, pages 440–458. Springer, 2014.

[5] Elena Andreeva, Begül Bilgin, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. APE: Authenticated Permutation-Based Encryption for Lightweight Cryptography. *IACR Cryptology ePrint Archive*, 2013:791, 2013.

[6] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. How to Securely Release Unverified Plaintext in Authenticated Encryption. *IACR Cryptology ePrint Archive*, 2014:144, 2014.

[7] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Elmar Tischhauser, and Kan Yasuda. Parallelizable and Authenticated Online Ciphers. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT*, volume 8269 of *Lecture Notes in Computer Science*, pages 424–443. Springer, 2013.

[8] Elena Andreeva, Charles Bouillaguet, Pierre-Alain Fouque, Jonathan J. Hoch, John Kelsey, Adi Shamir, and Sébastien Zimmer. Second Preimage Attacks on Dithered Hash Functions. In Smart [225], pages 270–288.

[9] Elena Andreeva, Gregory Neven, Bart Preneel, and Thomas Shrimpton. Seven-Property-Preserving Iterated Hashing: ROX. In Kurosawa [149], pages 130–146.

[10] Elena Andreeva and Bart Preneel. A Three-Property-Secure Hash Function. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography*, volume 5381 of *Lecture Notes in Computer Science*, pages 228–244. Springer, 2008.

[11] Jari Arkko, Carsten Bormann, Peter Friess, Cullen Jennings, Antonio Skarmeta, Zack Shelby, and Hannes Tschofenig. Report from the Smart Oject Security Workshop. In *Smart Object Security Workshop*, 2012.

[12] Jean-Philippe Aumasson, Willi Meier, and Raphael C.-W. Phan. The Hash Function Family LAKE. In Kaisa Nyberg, editor, *Fast Software Encryption*, volume 5086 of *Lecture Notes in Computer Science*, pages 36–53. Springer, 2008.

[13] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. BLAKE2: Simpler, Smaller, Fast as MD5. In Michael J. Jacobson Jr. and Reihaneh Safavi-Naini, editors, *Applied Cryptography and Network Security*, volume 7954 of *Lecture Notes in Computer Science*, pages 119–135. Springer, 2013.

[14] Mihir Bellare, Alexandra Boldyreva, Lars R. Knudsen, and Chanathip Namprempre. Online Ciphers and the Hash-CBC Construction. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 292–309. Springer, 2001.

[15] Mihir Bellare, Alexandra Boldyreva, Lars R. Knudsen, and Chanathip Namprempre. On-line Ciphers and the Hash-CBC Constructions. *Journal of Cryptology*, 25(4):640–679, 2012.

[16] Mihir Bellare, Alexandra Boldyreva, and Adriana Palacio. An Uninstantiable Random-Oracle-Model Scheme for a Hybrid-Encryption Problem. In Cachin and Camenisch [56], pages 171–188.

[17] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying Hash Functions for Message Authentication. In Koblitz [143], pages 1–15.

[18] Mihir Bellare, Anand Desai, E. Jokipii, and Phillip Rogaway. A Concrete Security Treatment of Symmetric Encryption. In *Annual Symposium of Foundations of Computer Science*, pages 394–403. IEEE Computer Society, 1997.

[19] Mihir Bellare, Roch Guérin, and Phillip Rogaway. XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions. In Don Coppersmith, editor, *CRYPTO*, volume 963 of *Lecture Notes in Computer Science*, pages 15–28. Springer, 1995.

[20] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The Security of Cipher Block Chaining. In Yvo Desmedt, editor, *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 341–358. Springer, 1994.

[21] Mihir Bellare and Chanathip Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. In Okamoto [187], pages 531–545.

[22] Mihir Bellare and Chanathip Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. *Journal of Cryptology*, 21(4):469–491, 2008.

[23] Mihir Bellare and Thomas Ristenpart. Multi-Property-Preserving Hash Domain Extension and the EMD Transform. In Lai and Chen [150], pages 299–314.

[24] Mihir Bellare and Phillip Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In Denning et al. [75], pages 62–73.

[25] Mihir Bellare and Phillip Rogaway. Optimal Asymmetric Encryption. In Alfredo De Santis, editor, *EUROCRYPT*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111. Springer, 1994.

[26] Mihir Bellare and Phillip Rogaway. The Exact Security of Digital Signatures - How to Sign with RSA and Rabin. In Ueli M. Maurer, editor, *EUROCRYPT*, volume 1070 of *Lecture Notes in Computer Science*, pages 399–416. Springer, 1996.

[27] Mihir Bellare and Phillip Rogaway. Encode-Then-Encipher Encryption: How to Exploit Nonces or Redundancy in Plaintexts for Efficient Cryptography. In Okamoto [187], pages 317–330.

[28] Mihir Bellare and Phillip Rogaway. Code-Based Game-Playing Proofs and the Security of Triple Encryption. *IACR Cryptology ePrint Archive*, 2004:331, 2004.

[29] Mihir Bellare and Phillip Rogaway. The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs. In Vaudenay [232], pages 409–426.

[30] Mihir Bellare and David Wagner. The EAX Mode of Operation. In Roy and Meier [212], pages 389–407.

[31] Daniel J. Bernstein. Cache-timing attacks on AES. `http://cr.yp.to/antiforgery/cachetiming-20050414.pdf`, 2005. Accessed August 10, 2014.

[32] Daniel J. Bernstein. The Poly1305-AES Message-Authentication Code. In Gilbert and Handschuh [110], pages 32–49.

[33] Daniel J. Bernstein and Peter Schwabe. New AES Software Speed Records. In Dipanwita Roy Chowdhury, Vincent Rijmen, and Abhijit Das, editors, *INDOCRYPT*, volume 5365 of *Lecture Notes in Computer Science*, pages 322–336. Springer, 2008.

[34] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. The Keccak SHA-3 submission. Submission to NIST (Round 3), 2011.

[35] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. RadioGatún, a belt-and-mill hash function. *IACR Cryptology ePrint Archive*, 2006:369, 2006.

[36] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Sponge Functions. Ecrypt Hash Workshop, 2007. See `http://gva.noekeon.org/papers/bdpv07.html`.

[37] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. On the Indifferentiability of the Sponge Construction. In Smart [225], pages 181–197.

[38] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications. In Ali Miri and Serge Vaudenay, editors, *Selected Areas in Cryptography*, volume 7118 of *Lecture Notes in Computer Science*, pages 320–337. Springer, 2011.

[39] Eli Biham and Rafi Chen. Near-Collisions of SHA-0. In Franklin [107], pages 290–305.

[40] Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet, and William Jalby. Collisions of SHA-0 and Reduced SHA-1. In Cramer [68], pages 36–57.

[41] Eli Biham and Orr Dunkelman. A Framework for Iterative Hash Functions - HAIFA. *IACR Cryptology ePrint Archive*, 2007:278, 2007.

[42] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In Menezes and Vanstone [169], pages 2–21.

[43] Alex Biryukov, editor. *Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers*, volume 4593 of *Lecture Notes in Computer Science*. Springer, 2007.

[44] John Black. The Ideal-Cipher Model, Revisited: An Uninstantiable Blockcipher-Based Hash Function. In Robshaw [203], pages 328–340.

[45] John Black, Martin Cochran, and Thomas Shrimpton. On the Impossibility of Highly-Efficient Blockcipher-Based Hash Functions. In Cramer [68], pages 526–541.

[46] John Black and Phillip Rogaway. A Block-Cipher Mode of Operation for Parallelizable Message Authentication. In Lars R. Knudsen, editor, *EUROCRYPT*, volume 2332 of *Lecture Notes in Computer Science*, pages 384–397. Springer, 2002.

[47] John Black, Phillip Rogaway, and Thomas Shrimpton. Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In Yung [243], pages 320–335.

[48] John Black, Phillip Rogaway, Thomas Shrimpton, and Martijn Stam. An Analysis of the Blockcipher-Based Hash Functions from PGV. *Journal of Cryptology*, 23(4):519–545, 2010.

[49] Matt Blaze. A Cryptographic File System for UNIX. In Denning et al. [75], pages 9–16.

[50] Andrey Bogdanov, Martin M. Lauridsen, and Elmar Tischhauser. AES-Based Authenticated Encryption Modes in Parallel High-Performance Software. Cryptology ePrint Archive, Report 2014/186, 2014. `http://eprint.iacr.org/`.

[51] Nikita Borisov, Ian Goldberg, and David Wagner. Intercepting Mobile Communications: The Insecurity of 802.11. In Christopher Rose, editor, *MOBICOM*, pages 180–189. ACM, 2001.

[52] Xavier Boyen. Halting Password Puzzles – Hard-to-break Encryption from Human-memorable Keys. In *16th USENIX Security Symposium—SECURITY 2007*, pages 119–134. Berkeley: The USENIX Association, 2007. Available at `http://www.cs.stanford.edu/~xb/security07/`.

[53] William F. Bradley. Superconcentration on a Pair of Butterflies. *CoRR*, abs/1401.7263, 2014.

[54] Gilles Brassard, editor. *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*. Springer, 1990.

[55] Enrico Buonanno, Jonathan Katz, and Moti Yung. Incremental Unforgeable Encryption. In Matsui [161], pages 109–124.

[56] Christian Cachin and Jan Camenisch, editors. *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*. Springer, 2004.

[57] Tom Caddy. FIPS 140-2. In Henk C. A. van Tilborg, editor, *Encyclopedia of Cryptography and Security*. Springer, 2005.

[58] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. OpenPGP Message Format. RFC 4880 (Proposed Standard), November 2007. Updated by RFC 5581.

[59] Ran Canetti, Oded Goldreich, and Shai Halevi. The Random Oracle Methodology, Revisited (Preliminary Version). In Jeffrey Scott Vitter, editor, *Symposium on Theory of Computing*, pages 209–218. ACM, 1998.

[60] Ran Canetti, Oded Goldreich, and Shai Halevi. The Random Oracle Methodology, Revisited. *Journal of the ACM*, 51(4):557–594, 2004.

[61] Christophe De Cannière and Christian Rechberger. Finding SHA-1 Characteristics: General Results and Applications. In Lai and Chen [150], pages 1–20.

[62] Florent Chabaud and Antoine Joux. Differential Collisions in SHA-0. In Hugo Krawczyk, editor, *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 56–71. Springer, 1998.

[63] Debrup Chakraborty and Palash Sarkar. A General Construction of Tweakable Block Ciphers and Different Modes of Operations. *IEEE Transactions on Information Theory*, 54(5):1991–2006, 2008.

[64] Donghoon Chang, Sangjin Lee, Mridul Nandi, and Moti Yung. Indifferentiable Security Analysis of Popular Hash Functions with Prefix-Free Padding. In Lai and Chen [150], pages 283–298.

[65] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Math. Comput.*, 19:297–301, 1965.

[66] Intel Corporation. AES-NI Sample Library v1.2. `http://software.intel.com/en-us/articles/download-the-intel-aesni-sample-library/`, October 2010.

[67] Nvidia Corporation. Nvidia GeForce GTX 680 - Technology Overview, 2012.

[68] Ronald Cramer, editor. *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*. Springer, 2005.

[69] Joan Daemen. *Hash Function and Cipher Design: Strategies Based on Linear and Differential Cryptanalysis*. Ph.D. thesis, Katholieke Universiteit Leuven, Leuven, Belgium, March 1995.

[70] Joan Daemen and Vincent Rijmen. Rijndael for AES. In *AES Candidate Conference*, pages 343–348, 2000.

[71] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.

[72] Ivan Damgård. A Design Principle for Hash Functions. In Brassard [54], pages 416–427.

[73] Nilanjan Datta and Mridul Nandi. Characterization of EME with Linear Mixing. *IACR Cryptology ePrint Archive*, 2014:9, 2014.

[74] Richard D. Dean. *Formal Aspects of Mobile Code Security*. PhD thesis, January 1999.

[75] Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors. *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993*. ACM, 1993.

[76] T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246 (Proposed Standard), January 1999. Obsoleted by RFC 4346, updated by RFCs 3546, 5746, 6176.

[77] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346 (Proposed Standard), April 2006. Obsoleted by RFC 5246, updated by RFCs 4366, 4680, 4681, 5746, 6176.

[78] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176.

[79] Whitfield Diffie and Martin E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[80] Hans Dobbertin. Cryptanalysis of MD4. *Journal of Cryptology*, 11(4):253–271, 1998.

[81] Yevgeniy Dodis, Roberto Oliveira, and Krzysztof Pietrzak. On the Generic Insecurity of the Full Domain Hash. In Shoup [224], pages 449–466.

[82] Ulrich Drepper. Unix crypt using SHA-256 and SHA-512. `http://www.akkadia.org/drepper/SHA-crypt.txt`. Accessed May 16, 2013.

[83] Orr Dunkelman, editor. *Fast Software Encryption, 16th International Workshop, FSE 2009, Leuven, Belgium, February 22-25, 2009, Revised Selected Papers*, volume 5665 of *Lecture Notes in Computer Science*. Springer, 2009.

[84] Morris Dworkin. *Special Publication 800-38A: Recommendation for Block Cipher Modes of Operation*. National Institute of Standards, U.S. Department of Commerce, December 2001.

[85] Morris Dworkin. *Special Publication 800-38C: Recommendation for block cipher modes of operation: the CCM mode for authentication and confidentiality*. National Institute of Standards and Technology, U.S. Department of Commerce, May 2005.

[86] Stefan Dziembowski, Tomasz Kazana, and Daniel Wichs. Key-Evolution Schemes Resilient to Space-Bounded Leakage. In Phillip Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 335–353. Springer, 2011.

[87] Shimon Even and Yishay Mansour. A Construction of a Cipher from a Single Pseudorandom Permutation. *Journal of Cryptology*, 10(3):151–162, 1997.

[88] Christian Forler Ewan Fleischmann and Michael Gorski. The Twister Hash Function Family. Submission to NIST, 2008.

[89] Niels Ferguson. AES-CBC + Elephant diffuser: A Disk Encryption Algorithm for Windows Vista, 2006.

[90] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, and Jesse Walker. The Skein Hash Function Family. Submission to NIST, 2010.

[91] Marc Fischlin and Anja Lehmann. Multi-property Preserving Combiners for Hash Functions. In Ran Canetti, editor, *Theory of Cryptography*, volume 4948 of *Lecture Notes in Computer Science*, pages 375–392. Springer, 2008.

[92] Ewan Fleischmann, Christian Forler, and Michael Gorski. Classification of the SHA-3 Candidates. *IACR Cryptology ePrint Archive*, 2008:511, 2008.

[93] Ewan Fleischmann, Christian Forler, Michael Gorski, and Stefan Lucks. Twister - A Framework for Secure and Fast Hash Functions. In Feng Bao, Hui Li, and Guilin Wang, editors, *Information Security Practice and Experience*,

volume 5451 of *Lecture Notes in Computer Science*, pages 257–273. Springer, 2009.

[94] Ewan Fleischmann, Christian Forler, Michael Gorski, and Stefan Lucks. Collision Resistant Double-Length Hashing. In Swee-Huay Heng and Kaoru Kurosawa, editors, *ProvSec*, volume 6402 of *Lecture Notes in Computer Science*, pages 102–118. Springer, 2010.

[95] Ewan Fleischmann, Christian Forler, Michael Gorski, and Stefan Lucks. New Boomerang Attacks on ARIA. In Guang Gong and Kishan Chand Gupta, editors, *INDOCRYPT*, volume 6498 of *Lecture Notes in Computer Science*, pages 163–175. Springer, 2010.

[96] Ewan Fleischmann, Christian Forler, Michael Gorski, and Stefan Lucks. TWISTER$_{\mathrm{pi}}$ - A Framework for Secure and Fast Hash Functions. *International Journal of Advanced Computer Technology (IJACT)*, 2(1):68–81, 2010.

[97] Ewan Fleischmann, Christian Forler, and Stefan Lucks. Γ-MAC$[H, P]$- A New Universal MAC Scheme. In Frederik Armknecht and Stefan Lucks, editors, *WEWoRC*, volume 7242 of *Lecture Notes in Computer Science*, pages 83–98. Springer, 2011.

[98] Ewan Fleischmann, Christian Forler, and Stefan Lucks. McOE: A Family of Almost Foolproof On-Line Authenticated Encryption Schemes. In Anne Canteaut, editor, *Fast Software Encryption*, volume 7549 of *Lecture Notes in Computer Science*, pages 196–215. Springer, 2012.

[99] Ewan Fleischmann, Christian Forler, and Stefan Lucks. The Collision Security of MDC-4. In Aikaterini Mitrokotsa and Serge Vaudenay, editors, *AFRICACRYPT*, volume 7374 of *Lecture Notes in Computer Science*, pages 252–269. Springer, 2012.

[100] Ewan Fleischmann, Christian Forler, Stefan Lucks, and Jakob Wenzel. McOE: A Foolproof On-Line Authenticated Encryption Scheme (full version). *IACR Cryptology ePrint Archive*, 2011:644, 2011.

[101] Ewan Fleischmann, Christian Forler, Stefan Lucks, and Jakob Wenzel. Weimar-DM: A Highly Secure Double-Length Compression Function. In Willy Susilo and Jennifer Seberry, editors, *Australasian Conference on Information Security and Privacy*, volume 7372 of *Lecture Notes in Computer Science*, pages 152–165. Springer, 2012.

[102] Christian Forler, Stefan Lucks, and Jakob Wenzel. Designing the API for a Cryptographic Library - A Misuse-Resistant Application Programming Interface. In Mats Brorsson and Luís Miguel Pinho, editors, *Ada-Europe*, volume 7308 of *Lecture Notes in Computer Science*, pages 75–88. Springer, 2012.

[103] Christian Forler, Stefan Lucks, and Jakob Wenzel. Catena: A Memory-Consuming Password Scrambler (full version). *IACR Cryptology ePrint Archive*, 2013:525, 2013.

[104] Christian Forler, Stefan Lucks, and Jakob Wenzel. Memory-Demanding Password Scrambling. In Tatsuaki Okamoto, editor, *ASIACRYPT*, Lecture Notes in Computer Science (to appear). Springer, 2014.

[105] Pierre-Alain Fouque, Antoine Joux, Gwenaëlle Martinet, and Frédéric Valette. Authenticated On-Line Encryption. In Mitsuru Matsui and Robert J. Zuccherato, editors, *Selected Areas in Cryptography*, volume 3006 of *Lecture Notes in Computer Science*, pages 145–159. Springer, 2003.

[106] Pierre-Alain Fouque, Gwenaëlle Martinet, Frédéric Valette, and Sébastien Zimmer. On the Security of the CCM Encryption Mode and of a Slight Variant. In Steven M. Bellovin and Moti Yung, editors, *Applied Cryptography and Network Security*, volume 5037 of *Lecture Notes in Computer Science*, pages 411–428, 2008.

[107] Matthew K. Franklin, editor. *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*. Springer, 2004.

[108] Patrick Gallagher, Deputy Director Foreword, and Cita Furlani Director. Federal Information Processing Standards Publication (FIPS PUB) 186-3: Digital Signature Standard (DSS), 2009.

[109] Rosario Gennaro, Hugo Krawczyk, and Tal Rabin. Secure Hashed Diffie-Hellman over Non-DDH Groups. In Cachin and Camenisch [56], pages 361–381.

[110] Henri Gilbert and Helena Handschuh, editors. *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers*, volume 3557 of *Lecture Notes in Computer Science*. Springer, 2005.

[111] Brian Gladman. Brian Gladman's AES Implementation, 19th June 2006. http://gladman.plushost.co.uk/oldsite/AES/index.php.

[112] Eric Glass. The NTLM Authentication Protocol and Security Support Provider. http://davenport.sourceforge.net/ntlm.html. Accessed May 16, 2013.

[113] Virgil D. Gligor and Pompiliu Donescu. Fast Encryption and Authentication: XCBC Encryption and XECB Authentication Modes. In Matsui [161], pages 92–108.

[114] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, 1986.

[115] Shafi Goldwasser and Yael Tauman Kalai. On the (In)security of the Fiat-Shamir Paradigm. In *FOCS*, pages 102–113. IEEE Computer Society, 2003.

[116] Shafi Goldwasser and Silvio Micali. Probabilistic Encryption and How to Play Mental Poker Keeping Secret All Partial Information. In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *Symposium on Theory of Computing*, pages 365–377. ACM, 1982.

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

[118] Michael Gorski, Stefan Lucks, and Thomas Peyrin. Slide Attacks on a Class of Hash Functions. In Josef Pieprzyk, editor, *ASIACRYPT*, volume 5350 of *Lecture Notes in Computer Science*, pages 143–160. Springer, 2008.

[119] Shay Gueron and Michael E. Kounavis. Efficient implementation of the Galois Counter Mode using a Carry-Less Multiplier and a Fast Reduction Algorithm, journal = Inf. Process. Lett. 110(14-15):549–553, 2010.

[120] D. Harkins and D. Carrel. The Internet Key Exchange (IKE). RFC 2409 (Proposed Standard), November 1998. Obsoleted by RFC 4306, updated by RFC 4109.

[121] Deukjo Hong, Donghoon Chang, Jaechul Sung, Sangjin Lee, Seokhie Hong, Jaesang Lee, Dukjae Moon, and Sungtaek Chee. A New Dedicated 256-Bit Hash Function: FORK-256. In Robshaw [203], pages 195–209.

[122] George Hotz. Console Hacking 2010 - PS3 Epic Fail. 27th Chaos Communications Congress, 2010. `http://events.ccc.de/congress/2010/Fahrplan/attachments/1780_27c3_console_hacking_2010.pdf`.

[123] R. Housley. Cryptographic Message Syntax (CMS). RFC 3852 (Proposed Standard), July 2004. Obsoleted by RFC 5652, updated by RFCs 4853, 5083.

[124] Tetsu Iwata. New Blockcipher Modes of Operation with Beyond the Birthday Bound Security. In Robshaw [203], pages 310–327.

[125] Tetsu Iwata and Kaoru Kurosawa. OMAC: One-Key CBC MAC. In Thomas Johansson, editor, *Fast Software Encryption*, volume 2887 of *Lecture Notes in Computer Science*, pages 129–153. Springer, 2003.

[126] Tetsu Iwata and Kan Yasuda. BTM: A Single-Key, Inverse-Cipher-Free Mode for Deterministic Authenticated Encryption. In Michael J. Jacobson Jr. and Reihaneh Safavi-Naini, editors, *Selected Areas in Cryptography*, volume 5867 of *Lecture Notes in Computer Science*, pages 313–330. Springer, 2009.

[127] Tetsu Iwata and Kan Yasuda. HBS: A Single-Key Mode of Operation for Deterministic Authenticat Encryption. In Dunkelman [83], pages 394–415.

[128] Don Johnson, Alfred Menezes, and Scott A. Vanstone. The Elliptic Curve Digital Signature Algorithm (ECDSA). *International Journal of Information Security*, 1(1):36–63, 2001.

[129] Antoine Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In Franklin [107], pages 306–316.

[130] Charanjit S. Jutla. Encryption Modes with Almost Free Message Integrity. *Journal of Cryptology*, 21(4):547–578, 2008.

[131] B. Kaliski. PKCS #5: Password-Based Cryptography Specification Version 2.0. RFC 2898 (Informational), September 2000.

[132] Poul-Henning Kamp. The History of md5crypt. `http://phk.freebsd.dk/sagas/md5crypt.html`. Accessed May 16, 2013.

[133] Jonathan Katz and Moti Yung. Unforgeable Encryption and Chosen Ciphertext Secure Modes of Operation. In Bruce Schneier, editor, *Fast Software Encryption*, volume 1978 of *Lecture Notes in Computer Science*, pages 284–299. Springer, 2000.

[134] John Kelsey and Tadayoshi Kohno. Herding Hash Functions and the Nostradamus Attack. In Vaudenay [232], pages 183–200.

[135] John Kelsey and Bruce Schneier. Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work. In Cramer [68], pages 474–490.

[136] John Kelsey, Bruce Schneier, Chris Hall, and David Wagner. Secure Applications of Low-Entropy Keys. In Eiji Okamoto, George I. Davida, and Masahiro Mambo, editors, *Information Security Workshop*, volume 1396 of *Lecture Notes in Computer Science*, pages 121–134. Springer, 1997.

[137] S. Kent. IP Encapsulating Security Payload (ESP). RFC 4303 (Proposed Standard), December 2005.

[138] S. Kent and R. Atkinson. IP Encapsulating Security Payload (ESP). RFC 2406 (Proposed Standard), November 1998. Obsoleted by RFCs 4303, 4305.

[139] Joe Kilian and Phillip Rogaway. How to Protect DES Against Exhaustive Key Search (an Analysis of DESX). *Journal of Cryptology*, 14(1):17–35, 2001.

[140] Lars R. Knudsen. SMASH - A Cryptographic Hash Function. In Gilbert and Handschuh [110], pages 228–242.

[141] Lars R. Knudsen, Christian Rechberger, and Søren S. Thomsen. The Grindahl Hash Functions. In Biryukov [43], pages 39–57.

[142] Donald E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching.* Addison-Wesley, 1973.

[143] Neal Koblitz, editor. *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science.* Springer, 1996.

[144] Neal Koblitz and Alfred Menezes. Another Look at "Provable Security". II. In Rana Barua and Tanja Lange, editors, *INDOCRYPT*, volume 4329 of *Lecture Notes in Computer Science*, pages 148–175. Springer, 2006.

[145] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Koblitz [143], pages 104–113.

[146] Tadayoshi Kohno. Attacking and Repairing the WinZip Encryption Scheme. In Vijayalakshmi Atluri and Patrick Drew McDaniel, editors, *ACM Conference on Computer and Communications Security*, pages 72–81. ACM, 2004.

[147] Tadayoshi Kohno, John Viega, and Doug Whiting. CWC: A High-Performance Conventional Authenticated Encryption Mode. In Roy and Meier [212], pages 408–426.

[148] Ted Krovetz and Phillip Rogaway. The Software Performance of Authenticated-Encryption Modes. In Antoine Joux, editor, *Fast Software Encryption*, volume 6733 of *Lecture Notes in Computer Science*, pages 306–327. Springer, 2011.

[149] Kaoru Kurosawa, editor. *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, volume 4833 of *Lecture Notes in Computer Science*. Springer, 2007.

[150] Xuejia Lai and Kefei Chen, editors. *Advances in Cryptology - ASIACRYPT 2006, 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006, Proceedings*, volume 4284 of *Lecture Notes in Computer Science*. Springer, 2006.

[151] Thomas Lengauer and Robert Endre Tarjan. Asymptotically Tight Bounds on Time-Space Trade-Offs in a Pebble Game. *Journal of the ACM*, 29(4):1087–1130, 1982.

[152] Moses Liskov, Ron L. Rivest, and David Wagner. Tweakable Block Ciphers. In Yung [243], pages 31–46.

[153] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable Block Ciphers. *Journal of Cryptology*, 24(3):588–613, 2011.

[154] Stefan Lucks. Ciphers Secure against Related-Key Attacks. In Roy and Meier [212], pages 359–370.

[155] Stefan Lucks. A Failure-Friendly Design Principle for Hash Functions. In Bimal K. Roy, editor, *ASIACRYPT*, volume 3788 of *Lecture Notes in Computer Science*, pages 474–494. Springer, 2005.

[156] Stefan Lucks. Two-Pass Authenticated Encryption Faster than Generic Composition. In Gilbert and Handschuh [110], pages 284–298.

[157] T. Alexander Lystad. Leaked Password Lists and Dictionaries - The Password Project. `http://thepasswordproject.com/leaked_password_lists_and_dictionaries`. Accessed May 16, 2013.

[158] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes*. North-holland Publishing Company, 2nd edition, 1978.

[159] Udi Manber. A Simple Scheme to Make Passwords Based on One-Way Functions Much Harder to Crack. *Computers & Security*, 15(2):171–176, 1996.

[160] Luther Martin. XTS: A Mode of AES for Encrypting Hard Disks. *IEEE Security & Privacy*, 8(3):68–69, 2010.

[161] Mitsuru Matsui, editor. *Fast Software Encryption, 8th International Workshop, FSE 2001 Yokohama, Japan, April 2-4, 2001, Revised Papers*, volume 2355 of *Lecture Notes in Computer Science*. Springer, 2002.

[162] Krystian Matusiewicz, Thomas Peyrin, Olivier Billet, Scott Contini, and Josef Pieprzyk. Cryptanalysis of FORK-256. In Biryukov [43], pages 19–38.

[163] Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In Moni Naor, editor, *Theory of Cryptography*, volume 2951 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2004.

[164] David A. McGrew and John Viega. The Security and Performance of the Galois/Counter Mode (GCM) of Operation. In Anne Canteaut and Kapalee Viswanathan, editors, *INDOCRYPT*, volume 3348 of *Lecture Notes in Computer Science*, pages 343–355. Springer, 2004.

[165] Florian Mendel, Bart Mennink, Vincent Rijmen, and Elmar Tischhauser. A Simple Key-Recovery Attack on McOE-X. In Josef Pieprzyk and Mark Manulis, editors, *CANS*, volume 7712, pages 23–31. Springer, 2012.

[166] Florian Mendel, Christian Rechberger, and Martin Schläffer. Cryptanalysis of Twister. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *Applied Cryptography and Network Security*, volume 5536 of *Lecture Notes in Computer Science*, pages 342–353, 2009.

[167] Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In Dunkelman [83], pages 260–276.

[168] Florian Mendel and Martin Schläffer. Collisions for Round-Reduced LAKE. In Yi Mu, Willy Susilo, and Jennifer Seberry, editors, *Australasian Conference on Information Security and Privacy*, volume 5107 of *Lecture Notes in Computer Science*, pages 267–281. Springer, 2008.

[169] Alfred Menezes and Scott A. Vanstone, editors. *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*, volume 537 of *Lecture Notes in Computer Science*. Springer, 1991.

[170] Alfred Menezes and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[171] Ralph C. Merkle. One Way Hash Functions and DES. In Brassard [54], pages 428–446.

[172] Andres Molina-Markham, George Danezis, Kevin Fu, Prashant J. Shenoy, and David E. Irwin. Designing Privacy-Preserving Smart Meters with Low-Cost Microcontrollers. In Angelos D. Keromytis, editor, *Financial Cryptography*, volume 7397 of *Lecture Notes in Computer Science*, pages 239–253. Springer, 2012.

[173] Gordon E. Moore. Cramming more Components onto Integrated Circuits. *Electronics*, 38(8), April 1965.

[174] Robert Morris and Ken Thompson. Password Security - A Case History. *Communications of the ACM*, 22(11):594–597, 1979.

[175] National Institute of Standards and Technology. *FIPS PUB 46-3: Data Encryption Standard (DES)*. National Institute for Standards and Technology, Gaithersburg, MD, USA, October 1999. supersedes FIPS 46-2.

[176] NIST National Institute of Standards and Technology. Federal Information Processing Standards Publication (FIPS PUB) 197: Specification for the Advanced Encryption Standard (AES). November 2001. See `http://csrc.nist.gov`.

[177] Krishna Neelamraju. Facebook Pages: Usage Patterns | Recommend.ly. `http://blog.recommend.ly/facebook-pages-usage-patterns/`. Accessed May 16, 2013.

[178] Niels Ferguson and Stefan Lucks and Bruce Schneier and Doug Whiting and Mihir Bellare and Tadayoshi Kohno and Jon Callas and Jesse Walker. Skein Source Code and Test Vectors. `http://www.skein-hash.info/downloads`. Accessed August 10, 2014.

[179] Jesper Buus Nielsen. Separating Random Oracle Proofs from Complexity Theoretic Proofs: The Non-committing Encryption Case. In Yung [243], pages 111–126.

[180] Ivica Nikolić, Alex Biryukov, and Dmitry Khovratovich. Specification Update of the Hash Family LUX. `http://ehash.iaik.tugraz.at/uploads/c/c6/LUXadd.pdf`, 2009. Accessed August 10, 2014.

[181] NIST National Institute of Standards and Technology. SHA-3 Cryptographic Hash Algorithm Competition (2007-2012). More information available at `http://csrc.nist.gov/groups/ST/hash/sha-3/index.html`. Accessed August 10, 2014.

[182] Philippe Oechslin. Making a Faster Cryptanalytic Time-Memory Trade-Off. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 617–630. Springer, 2003.

[183] NIST National Institute of Standards and Technology. Federal Information Processing Standards Publication (FIPS PUB) 180-1: Secure Hash Standard. April 1995. See http://csrc.nist.gov.

[184] NIST National Institute of Standards and Technology. Federal Information Processing Standards Publication (FIPS PUB) 180-2: Secure Hash Standard. August 2002. See `http://csrc.nist.gov`.

[185] NIST National Institute of Standards and Technology. Federal Information Processing Standards Publication (FIPS PUB) 180: Secure Hash Standard. May 1993. See http://csrc.nist.gov.

[186] NIST National Institute of Standards and Technology. Federal Information Processing Standards Publication (FIPS PUB) 46-3: Data Encryption Standard (DES). October 1999. See `http://csrc.nist.gov`.

[187] Tatsuaki Okamoto, editor. *Advances in Cryptology - ASIACRYPT 2000, 6th International Proceedings*, volume 1976 of *Lecture Notes in Computer Science*. Springer, 2000.

[188] Christof Paar and Martin Rosner. Comparison of Arithmetic Architectures for Reed-Solomon Decoders in Reconfigurable Hardware. In *Field-Programmable Custom Computing Machines*, pages 219–225. IEEE Computer Society, 1997.

[189] Michael S. Paterson and Carl E. Hewitt. Comparative Schematology. In Jack B. Dennis, editor, *Record of the Project MAC conference on concurrent systems and parallel computation*, chapter Computation schemata, pages 119–127. ACM, New York, NY, USA, 1970.

[190] Colin Percival. Cache Missing for Fun and Profit. Presented at BSDCan'05, May 2005.

[191] Colin Percival. Stronger Key Derivation via Sequential Memory-Hard Functions. Presented at BSDCan'09, May 2009.

[192] Thomas Peyrin. Cryptanalysis of Grindahl. In Kurosawa [149], pages 551–567.

[193] Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen. Breaking a New Hash Function Design Strategy Called SMASH. In Bart Preneel and Stafford E. Tavares, editors, *Selected Areas in Cryptography*, volume 3897 of *Lecture Notes in Computer Science*, pages 233–244. Springer, 2005.

[194] Niels Provos and David Mazières. A Future-Adaptable Password Scheme. In *USENIX Annual Technical Conference, FREENIX Track*, pages 81–91. USENIX, 1999.

[195] Jean-Jacques Quisquater and Jean-Paul Delescaille. How Easy is Collision Search. New Results and Applications to DES. In Brassard [54], pages 408–413.

[196] B. Ramsdell. S/MIME Version 3 Message Specification. RFC 2633 (Proposed Standard), June 1999. Obsoleted by RFC 3851.

[197] A.G. Reinhold. HEKS: A Family of Key Stretching Algorithms, 1999.

[198] E. Rescorla and N. Modadugu. Datagram Transport Layer Security Version 1.2. RFC 6347 (Proposed Standard), January 2012.

[199] Vincent Rijmen and Elisabeth Oswald. Update on SHA-1. In Alfred Menezes, editor, *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 58–71. Springer, 2005.

[200] R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321 (Informational), April 1992. Updated by RFC 6151.

[201] Ronald L. Rivest. The MD4 Message Digest Algorithm. In Menezes and Vanstone [169], pages 303–311.

[202] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[203] Matthew J. B. Robshaw, editor. *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, volume 4047 of *Lecture Notes in Computer Science*. Springer, 2006.

[204] Phillip Rogaway. Authenticated-Encryption with Associated-Data. In Vijayalakshmi Atluri, editor, *ACM Conference on Computer and Communications Security*, pages 98–107. ACM, 2002.

[205] Phillip Rogaway. Efficient Instantiations of Tweakable Blockciphers and Refinement to Modes OCB and PMAC. In Pil Joong Lee, editor, *ASIACRYPT*, volume 3329 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2004.

[206] Phillip Rogaway. Nonce-Based Symmetric Encryption. In Roy and Meier [212], pages 348–359.

[207] Phillip Rogaway. Formalizing Human Ignorance. In Phong Q. Nguyen, editor, *VIETCRYPT*, volume 4341 of *Lecture Notes in Computer Science*, pages 211–228. Springer, 2006.

[208] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption. In Michael K. Reiter and Pierangela Samarati, editors, *ACM Conference on Computer and Communications Security*, pages 196–205. ACM, 2001.

[209] Phillip Rogaway and Thomas Shrimpton. A Provable-Security Treatment of the Key-Wrap Problem. In Vaudenay [232], pages 373–390.

[210] Phillip Rogaway and Thomas Shrimpton. Deterministic Authenticated-Encryption: A Provable-Security Treatment of the Key-Wrap Problem. *IACR Cryptology ePrint Archive*, 2006:221, 2006.

[211] Phillip Rogaway and Haibin Zhang. Online Ciphers from Tweakable Blockciphers. In Aggelos Kiayias, editor, *CT-RSA*, volume 6558 of *Lecture Notes in Computer Science*, pages 237–249. Springer, 2011.

[212] Bimal K. Roy and Willi Meier, editors. *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, volume 3017 of *Lecture Notes in Computer Science*. Springer, 2004.

[213] RSA Laboratories. *PKCS #7: Cryptographic Message Syntax Standard*. RSA Data Security, Inc., May 1997.

[214] Todd Sabin. Vulnerability in Windows NT's SYSKEY encryption. *BindView Security Advisory*, 1999. Available at `http://marc.info/?l=ntbugtraq&m=94537191024690&w=4`. Accessed August 10, 2014.

[215] Semiocast SAS. Brazil becomes 2nd country on Twitter, Japan 3rd — Netherlands most active country. `http://goo.gl/QOeaB`. Accessed May 16, 2013.

[216] J. Savage and S. Swamy. Space-Time Trade-Offs on the FFT Algorithm. *Information Theory, IEEE Transactions on*, 24(5):563 – 568, sep 1978.

[217] John E. Savage and Sowmitri Swamy. Space-Time Tradeoffs for Oblivious Interger Multiplications. In Hermann A. Maurer, editor, *ICALP*, volume 71 of *Lecture Notes in Computer Science*, pages 498–504. Springer, 1979.

[218] Allan Lee Scherr. *An Analysis of Time-Shared Computer Systems*. PhD thesis, 1965.

[219] Martin Schläffer. *Cryptanalysis of AES-Based Hash Functions*. PhD thesis, 2011.

[220] Bruce Schneier. Description of a New Variable-Length Key, 64-bit Block Cipher (Blowfish). In Ross J. Anderson, editor, *Fast Software Encryption*, volume 809 of *Lecture Notes in Computer Science*, pages 191–204. Springer, 1993.

[221] Bruce Schneier. *Applied cryptography: Protocols, Algorithms, and Source Code in C (2. ed.)*. Wiley, 1996.

[222] Ravi Sethi. Complete Register Allocation Problems. *SIAM J. Comput.*, 4(3):226–248, 1975.

[223] Victor Shoup. On Fast and Provably Secure Message Authentication Based on Universal Hashing. In Koblitz [143], pages 313–328.

[224] Victor Shoup, editor. *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*. Springer, 2005.

[225] Nigel P. Smart, editor. *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, volume 4965 of *Lecture Notes in Computer Science*. Springer, 2008.

[226] Jens Steube. oclHashcat-plus - Advanced Password Recovery. `http://hashcat.net/oclhashcat-plus/`. Accessed May 16, 2013.

[227] Sowmitri Swamy and John E. Savage. Space-Time Tradeoffs for Linear Recursion. In Alfred V. Aho and Barry K. Rosen, editors, *Principles of Programming Languages*, pages 135–142. ACM Press, 1979.

[228] Trusted Computing Group (TCG). TPM Main Specification Version 1.2 rev. 116. `http://www.trustedcomputinggroup.org/resources/tpm_main_specification`, March 2011.

[229] Martin Tompa. Time-Space Tradeoffs for Computing Functions, Using Connectivity Properties of their Circuits. In Richard J. Lipton and Alfred V. Aho, editors, *Symposium on Theory of Computing*, pages 196–204. ACM, 1978.

[230] Gene Tsudik. Message Authentication with One-Way Hash Functions. In *INFOCOM*, pages 2055–2059. IEEE, 1992.

[231] Meltem Sönmez Turan, Elaine B. Barker, William E. Burr, and Lidong Chen. SP 800-132. Recommendation for Password-Based Key Derivation: Part 1: Storage Applications. Technical report, NIST, Gaithersburg, MD, USA, 2010.

[232] Serge Vaudenay, editor. *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*. Springer, 2006.

[233] David Wagner. A Generalized Birthday Problem. In Yung [243], pages 288–303.

[234] Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In Cramer [68], pages 1–18.

[235] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In Shoup [224], pages 17–36.

[236] Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In *EUROCRYPT*, pages 19–35, 2005.

[237] M.V. Wilkes. *Time-Sharing Computer Systems*. MacDonald computer monographs. American Elsevier Publishing Company, 1968.

[238] Robert S. Winternitz. Producing a One-Way Hash Function from DES. In David Chaum, editor, *CRYPTO*, pages 203–207. Plenum Press, New York, 1983.

[239] Hongjun Wu. The Misuse of RC4 in Microsoft Word and Excel. *IACR Cryptology ePrint Archive*, 2005:7, 2005.

[240] Kan Yasuda. ”Sandwich” Is Indeed Secure: How to Authenticate a Message with Just One Hashing. In Josef Pieprzyk, Hossein Ghodosi, and Ed Dawson, editors, *Australasian Conference on Information Security and Privacy*, volume 4586 of *Lecture Notes in Computer Science*, pages 355–369. Springer, 2007.

[241] Xun Yi, Shi Xing Cheng, Xiao Hu You, and Kwok Yan Lam. A Method for Obtaining Cryptographically Strong 8x8 S-boxes. In *IEEE Global Telecommunications Conference, GLOBECOM 97, Volume 2*, pages 689–693, 1997.

[242] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Transport Layer Protocol. RFC 4253 (Proposed Standard), January 2006. Updated by RFC 6668.

[243] Moti Yung, editor. *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*. Springer, 2002.

[244] Zhibin Zhou and Dijiang Huang. Computing Cryptographic Pairing in Sensors. *SIGBED Review*, 5(1):27, 2008.

A

Twister$_\pi$: Test Vectors

A.1. Twister$_\pi$-256

Input:	`61 62 63`	(3 octets)

Output:	`12 f6 c9 7c 5a 07 22 ad a 16 0d 1c 92`
	`32 f8 9d ed 3e ba e7 f8 39 14 28 3c 91`
	`4f b1 41 17 71 83`
		(32 octets)

Input:	`61 62 63 64 65 66 67 68 62 63 64 65 66`
	`67 68 69 63 64 65 66 67 68 69 6a 64 65`
	`66 67 68 69 6a 6b 65 66 67 68 69 6a 6b`
	`6c 66 67 68 69 6a 6b 6c 6d 67 68 69 6a`
	`6b 6c 6d 6e 68 69 6a 6b 6c 6d 6e 6f 69`
	`6a 6b 6c 6d 6e 6f 70 6a 6b 6c 6d 6e 6f`
	`70 71 6b 6c 6d 6e 6f 70 71 72 6c 6d 6e`
	`6f 70 71 72 73 6d 6e 6f 70 71 72 73 74`
	`6e 6f 70 71 72 73 74 75`
		(112 octets)

Output:	`48 41 3c 68 03 45 7b 8f d9 22 23 10 a8`
	`43 ef 0d 1d 3a 67 9b 1f a3 5e 0d 44 99`
	`37 f9 d3 b7 8c 3e`
		(32 octets)

Input:	`61 61 61 61 61 61 ...61 61 61 61 61 61`	(1,000,000 octets)

Output:	`5e ab cc 39 e6 0e e7 94 42 9d 88 0b 74`
	`9b f3 13 47 58 52 88 37 ac 49 d7 c1 b6`
	`16 50 39 b4 6b 9c`
		(32 octets)

A.2. TWISTER$_\pi$-512

Input:	`61 62 63`	(3 octets)
Output:	`d9 2e 69 c1 86 9e 0c c1 17 06 77 fc fb`	(64 octets)
	`79 b4 33 ea b9 23 93 b6 59 07 bb d1 69`	
	`0e f2 1f 69 d8 3a 72 ae 44 30 84 56 f0`	
	`49 e6 ec 38 64 bc 37 7a 47 76 02 ee 9e`	
	`98 67 48 50 09 66 6f 60 80 1d 16 2a`	

Input:	`61 62 63 64 65 66 67 68 62 63 64 65 66`	(112 octets)
	`67 68 69 63 64 65 66 67 68 69 6a 64 65`	
	`66 67 68 69 6a 6b 65 66 67 68 69 6a 6b`	
	`6c 66 67 68 69 6a 6b 6c 6d 67 68 69 6a`	
	`6b 6c 6d 6e 68 69 6a 6b 6c 6d 6e 6f 69`	
	`6a 6b 6c 6d 6e 6f 70 6a 6b 6c 6d 6e 6f`	
	`70 71 6b 6c 6d 6e 6f 70 71 72 6c 6d 6e`	
	`6f 70 71 72 73 6d 6e 6f 70 71 72 73 74`	
	`6e 6f 70 71 72 73 74 75`	
Output:	`b0 e3 4b aa 3d a1 54 87 0f 1f 7a c4 ef`	(64 octets)
	`a1 5e 33 d6 d3 23 f0 74 c6 2f e1 40 ea`	
	`37 57 9e ee 1a 2e 4b ce 3e be 6c 0e 40`	
	`56 bb 83 57 e8 41 b0 05 0e 3d df ea e3`	
	`5a 02 49 0c ac 0f e0 1b dd 4a 7f f4`	

Input:	`61 61 61 61 61 61 ...61 61 61 61 61 61`	(1,000,000 octets)
Output:	`8b 99 35 5a 36 c6 29 53 62 02 4a de 91`	(64 octets)
	`94 b3 ab a9 d1 d0 b9 18 ce e4 c4 d2 2e`	
	`0d 92 bc ca 74 af 9e ad d3 9e 56 6e 4a`	
	`b6 b7 68 2b c6 14 9e 33 ec 37 d0 69 83`	
	`1e c1 3e fd f5 2e e3 3b 9a d9 36 c3`	

Index

10^* padding rule, 12
G-superconcentrator, 142
λ-memory hardness, 128
2nd-preimage resistance, 11

AEAD, 24
AES S-Box, 113
authenticated encryption, 23

block cipher, 17
brute force attack, 11

Catena, 123, 136
Catena-KG, 141
CCA3, 26, 29
CCA3 advantage, 26
client-independent update, 129
COFFE, 53
- header processing, 57
- parameter choice, 60
- plaintext/ciphertext processing, 58
- session key generation, 56
- tag generation, 59

collision resistance, 10
compression function, 8
concrete security, 2

decryption misuse, 35
deterministic AE, 24
difference distribution table, 113
differential attack, 15
double-butterfly graph, 143
double-butterfly hashing, 145

Encrypt-and-Mac, 25
Encrypt-then-Mac, 25, 71

family of keyed permutation, 17

game-based proofs, 30
garbage-collector-attack, 136
garlic, 128
generic composition, 25

hash function, 7
herding attack, 14
hybrid standard model, 10

ideal cipher model, 18

IND-CCA, 36
IND-CCA advantage, 36
IND-CPA, 27
IND-CPA advantage, 27
IND-OCCA, 37
IND-OCCA advantage, 37
IND-OCCA2 advantage, 39
IND-OPRP advantage, 34
IND-PRP, 19, 21
IND-PRP advantage, 21
INT-CTXT, 27, 28
INT-CTXT advantage, 28
iterated hashfunction, 12

key derivation function, 130
keyed hash function, 15
keyed password hashing, 138

length-extension attack, 13
List of Notations, xvii, xxi
long 2nd-preimage attack, 14
longest common prefix, 32

Mac-then-Encrypt, 25
McOE, 71
- header processing, 73
- plaintext/ciphertext processing, 73
- tag generation/verification, 75

McOE-G, 87
- benchmarks, 89

McOE-X, 85
- benchmarks, 89

MDS Matrix, 101
memory hardness, 128
memory-hardness, 128
message expansion, 12
misuse resistance, 3
multi-collision attack, 13

NDMA advantage, 39
nonce, 23
nonce misuse, 31

OCCA3 advantage, 35
on-line authenticated encryption, 24
on-line permutation, 32
ONDMA advantage, 40

password recovery advantage, 129
password-recovery resistance, 139
pebble game, 148
pepper, 126
polynomial security, 2
preimage resistance, 11
PRF advantage, 16
provable security, 1
PRP-RKA, 20
PRP-RKA advantage, 20
pseudo random function, 15
pseudorandom permutation, 18

random oracle model, 8
random permutation, 18
related key attacks, 19
robustness, 38

salt, 125
sequential-memory hardness, 129
server relief, 130
slide attack, 15
standard model, 9
superconcentrator, 143

tweakable block cipher, 20
Twister$_\pi$, 93
- add twister counter, 99
- benchmarks, 109
- compression function, 102

message digest computation, 103
message injection, 99
Mini-Round, 98
mix columns, 101
shift rows, 100
state, 97, 98
state finalization, 103
substitute bytes, 99