
UNIVERSIDADE D
COIMBRA

Samuel Custódio Pereira Neves

# SECURITY AND SIMULATION IN MODERN COMPUTER HARDWARE

Tese no âmbito do Programa Doutoral em Ciências e Tecnologias da Informação, orientada pelo Prof. Doutor Filipe João Boavida de Mendonça Machado de Araújo e apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Dezembro de 2022

Department of Informatics Engineering
Faculty of Sciences and Technology of the University of Coimbra

# Security and Simulation in Modern Computer Hardware

Samuel Custódio Pereira Neves

Tese no âmbito do Programa Doutoral em Ciências e Tecnologias da Informação, orientada pelo Prof. Doutor Filipe João Boavida de Mendonça Machado de Araújo e apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Dezembro de 2022

1 2 9 0

UNIVERSIDADE D
COIMBRA

# Abstract

In the last few decades many aspects of our lives have been moved online. Banking and payments, health care, private messaging, or critical infrastructure monitoring are just a small subset of sensitive activities that are now commonplace on the Internet, and yet it would be catastrophic if they were not adequately secured.

One core tool for securing both data at rest and communications is cryptography. Cryptography offers tools for preserving the confidentiality and integrity of data, as well as for authentication and other more advanced uses like zero-knowledge proofs.

But for cryptography to be useful it needs to be usable. One significant component of usability is speed: if cryptography is—or is perceived to be—slow, this will slow down its adoption and incentivize poor alternatives to replace it. This is not theoretical; there are many such cases in the past where weak or ineffective cryptography was used because better options were deemed to incur too much overhead. One of the reasons that cryptographic functions can be slow is that they are not designed with the hardware in which they are going to run in mind. Hardware which, as Moore's law comes closer to its end, is becoming increasingly parallel.

The main focus of this thesis is the development of fast cryptographic primitives that fit modern hardware—fast enough that there should be no reason to prefer weaker ones. Our main contribution consists of two primitives with different applications.

The first one, BLAKE2, is a hash function designed to replace legacy hash functions such as MD5 and SHA-1 that, despite being long known to be broken, continue to be used due to their speed and availability. BLAKE2 was derived from the original BLAKE, one of the NIST SHA-3 competition finalists, and is expected to have a large security margin while, at the same time, being faster than MD5 and SHA-1. BLAKE2 also supports tree hashing, making it possible to accelerate operations on parallel hardware at a coarse grained level as well.

The second main contribution is NORX, an authenticated encryption primitive. While existing authenticated encryption primitives already existed, such as AES-GCM or EAX, they were based on the AES block cipher. The AES block cipher is widely believed to be secure. However, AES-based primitives suffer from a drawback: for AES to be both fast and secure it must be hardware accelerated. As one of the submissions

to the CAESAR competition for new authenticated encryption designs, NORX aims to eliminate this kind of tradeoff: it is fast, parallelizable, and easy to implement in pretty much every platform. In chips without hardware AES acceleration, such as the ARM Cortex-A7, NORX can be up to three times faster than AES-GCM.

As a byproduct of the development of the primitives above, we also developed Tyche, a small nonlinear pseudorandom generator targeted towards massively parallel simulations. Another byproduct of our research was the cryptanalysis not only of NORX, but also of other primitives such as the CAESAR candidates McMambo and Wheesht, as well as the authenticated encryption scheme found in the widespread Open Smart Grid Protocol standard.

**Keywords** cryptography, cryptanalysis, pseudorandom generator, hash function, authenticated encryption.

# Resumo

Nas últimas décadas muitos aspectos da nossa vida têm sido migrados para a Internet. Desde pagamentos online até saúde, comunicações privadas, ou monitorização de infraestrutura, são imensas as actividades de cariz sensível que não só são actualmente comuns na Internet, mas seria desastroso se não fossem adequadamente protegidas.

Uma ferramenta fundamental para a protecção de dados é a criptografia. A criptografia oferece mecanismos para assegurar a confidencialidade e integridade dos dados, assim como para a sua autenticação e ainda outros usos mais avançados como provas de conhecimento zero.

Mas para a criptografia ser útil tem de ser usável. Uma componente significativa da usabilidade é o desempenho: se a criptografia for lenta—ou tiver a percepção de ser lenta—isto irá impedir a sua adopção ou incentivar o uso de alternativas inseguras. Isto não é apenas uma preocupação teórica; existem muitos casos em que funções criptográficas inseguras continuaram a ser usadas devido ao custo imposto por transitar para alternativas seguras. Uma das razões para uma função criptográfica ser lenta é por não ser desenhada com o hardware em mente, hardware este que à medida que a lei de Moore caminha para o seu fim, se torna cada vez mais paralelo.

O foco principal desta tese é o desenvolvimento de funções criptográficas rápidas que se adaptam ao hardware moderno—rápidas o suficiente de forma a que não haja razão para preferir soluções inseguras. A minha contribuição consiste principalmente em duas funções com aplicações diferentes.

A primeira, BLAKE2, é uma função de dispersão desenhada para substituir funções mais antigas como a MD5 e SHA-1 que, apesar de terem sido quebradas há mais de uma década, continuam a ser utilizadas porque tipicamente também são as mais rápidas. A BLAKE2 deriva da função BLAKE, uma das finalistas da competição SHA-3 do NIST, é considerada ter uma grande margem de segurança e, simultaneamente, é mais rápida que a MD5 e a SHA-1. A BLAKE2 também suporta dispersão em árvore, o que torna possível aproveitar o paralelismo em maior escala dos processadores modernos.

A segunda contribuição é a função de cifração autenticada NORX. Apesar de já existirem funções de cifração autenticada, como a AES-GCM ou a EAX, estas são baseadas da cifra de bloco AES. Apesar desta cifra ser unanimemente considerada

segura, para ser tanto segura como rápida tem de ser implementada em hardware. Uma submissão para a competição CAESAR para novas funções de cifração autenticada, a NORX ambiciona eliminar este dilema por ser rápida, paralelizável, e fácil de implementar na maioria das plataformas. Em processadores sem aceleração de AES, como o ARM Cortex-A7, a NORX pode ser até três vezes mais rápida do que a AES-GCM.

Como subproduto do desenvolvimento destas funções, também desenvolvemos o Tyche, um gerador de números aleatórios destinado a simulações paralelas de grande escala. Outro subproduto foi a nossa criptanálise da NORX, que também se tornou útil para a análise de outras funções como a McMambo e Wheesht, ambas submetidas para a CAESAR, assim como a função de cifração autenticada usada no padrão Open Smart Grid Protocol.

**Palavras-chave** criptografia, criptanálise, gerador de números aleatórios, função de dispersão, cifração autenticada.

# Acknowledgments

This thesis would never had been initiated, let alone completed, without the help and support of numerous people.

Most of all I would like to thank my advisor, Filipe Araújo, for the encouragement to pursue a PhD, giving me the time and freedom to figure out what it would even be about, encouraging my endeavours even when they significantly veered off from our research group's core specialties, and for all of the advice. And finally, for tirelessly pushing me to complete it.

I would also like to thank Tanja Lange for inviting me, out of the blue, to the "Hash$^3$: Proofs, Analysis, and Implementation" ECRYPT II event, which ended up being my first introduction to the cryptographic community. And also Jean-Philippe Aumasson, for encouraging and enabling my early efforts to improve the AVX and XOP BLAKE implementations and for the subsequent numerous and fruitful collaborations, which directly lead to at least two of the chapters in this thesis.

I am also grateful to have met my frequent collaborators Philipp Jovanovic and Bart Mennink, from whom I learned so much, and whose collaborations I consider to be some of the most interesting and challenging work I have ever been involved with.

I would also like to show my gratitude to all my other coauthors and also to the Centre for Informatics and Systems of the University of Coimbra (CISUC), whose Software and Systems Engineering (SSE) group I have been a member of.

# Contents

# Chapter 1

# Introduction

Cryptology is the science of securing communications, melding together *cryptography*—the study of secure communication systems design—and *cryptanalysis*, the search for flaws in such designs. More art than science for most of its existence, cryptology has existed for almost as long as the written word. Throughout history, many schemes to render text undecipherable have been devised, ranging from the famous Caesar cipher [253] to the more elaborate constructions of the Renaissance, e.g., [388, 442, 472].

It was not until the 20th century, however, that cryptology truly became a science. It famously played a major part in the second World War, with the break of the Lorenz, Enigma, and Purple ciphers, and additionally was present at the dawn of the computing age, as some of the first computers were built to aid in the breaking of enemy messages [469]. It was also during this period that the fundamentals of cryptology were being built, such as the seminal work of Shannon [421].

The advent of computers and digital communications, fueled by the decreasing cost of computers and the Internet, brought with it the need to secure more and more data. This was the reason for the standardization of the DES block cipher, over 40 years ago [163], and this trend has only accelerated since then. As of 2021, it is estimated that 5 billion people regularly use the Internet [447], and countless companies do their business exclusively by digital means. It is therefore of the utmost importance that cryptographic algorithms not only work as advertised, i.e., are secure, but also that they are sufficiently efficient to serve the increasing number and variety of applications that require them.

The increased awareness of pervasive Internet surveillance by major intelligence agencies [212] has also acted to accelerate the pace and motivation to adopt encryption at all levels. This has been reflected in the success of projects like "Let's Encrypt" [1], which aim to make the Web encrypted by default.

## 1.1 Motivation

Mass adoption of cryptography is not so simple, however, as it involves a number of social, technical, legal, and political obstacles [2, 3, 40]. At the technical level, cryptography is often slow, or perceived to be slow [57, 291], and this tends to impact adoption in multiple ways:

- When cryptography incurs a noticeable overhead there is a tradeoff between security and usability, leading implementers to be incentivized to make cryptography optional or opt-in, greatly reducing the amount of protected communications. For example, full-disk encryption on Android devices has been a requirement since Android 6.0, released in 2015; but devices with poor AES performance, such as the Cortex-A7 used in many low-end phones, were exempt from it, effectively leaving millions of users unprotected [141, 142].

- Insecure but faster primitives (e.g., RC4 [7, 198, 202, 378, 450], MD5 [194, 267, 431, 433, 434, 466], SHA-1 [296, 432, 465]) are often chosen instead of safer but slower ones (e.g., AES-GCM without hardware acceleration, SHA-2, SHA-3).

There is thus a need to design cryptographic primitives that can not only replace existing weaker ones, but also that can do it without incurring a performance penalty. This is the core tenet of our work.

### 1.1.1 Hardware Trends

Before getting into the main topic of this thesis, we will take a short look at the devices that run most cryptographic algorithms: microprocessors.

Microprocessors have historically doubled in performance every 18 months, following Moore's law [341, 342][1]. This exponential increase was largely possible due to the so-called Dennard scaling. Dennard et al. [160, 161] devised a formula for scaling that consisted in reducing transistor size by 30%, while preserving electric current constant. This worked for decades, until process size shrank so much that electric leakage became a serious problem. This lead to the so-called "power wall" around the 4 GHz mark, which marked the beginning of the end of Dennard scaling.

Besides manufacturing process improvements, other techniques to use the available transistors in microprocessor circuits in a smarter way have been in use for a long time:

[1] Strictly speaking, Moore predicted that the number of transistors in integrated circuits would double every 2 years; this was later adjusted to 18 months, which is now the most common statement of Moore's law.

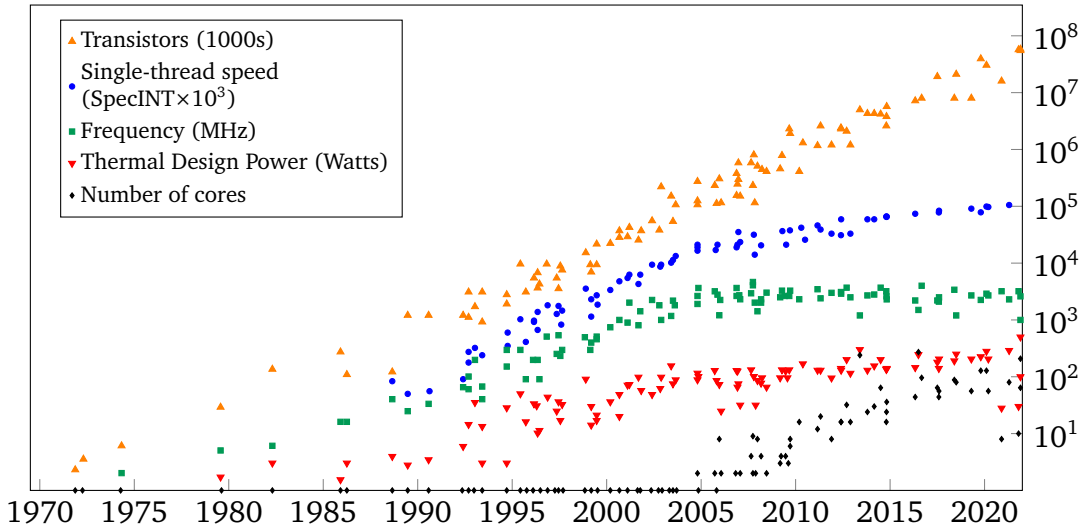



Figure 1.1: Microprocessor trend data, ranging from 1971 to 2022. Data sourced from [154, 409].

- Cache memory [301]
- Pipelining [379]
- Out-of-order execution [440, 441]
- Superscalar execution [428]
- Simultaneous multi-threading [445]
- Vector execution units [280]

Most microprocessors running today use many or all of these techniques—most of which exploiting the instruction-level parallelism present in many applications—to improve performance. However, all of these were not enough to keep CPU performance steadily increasing. In response, manufacturers came up with a coarse but effective strategy: multi-core processors.

Figure 1.1 displays a graph of microprocessor data ranging from 1971—when the Intel 4004 was released—to the current year. It is apparent that starting around 2005, the frequency, single-threaded performance and power consumption started to plateau. This is consistent with the end of Dennard scaling, as noted above. Meanwhile, it is also evident that the transistor count and number of cores continued to grow exponentially[2] until the current day.

[2]Note that the graph's $y$ axis scale is logarithmic.

Whether Moore's law will continue to apply for much longer is still unclear. But it is clear that increases in software performance are increasingly required to tap into this increase in transistors within a chip, be it via instruction-level parallelism, vectorization, or multiple cores.

### 1.1.2 Simulation

The efficient generation of random-looking numbers for Monte Carlo simulations was a research problem as early as the 1940s, with such methods as von Neumann's [461] middle-square method, Lehmer's [293] linear congruential generator, or Green, Smith, and Klem's [210] additive "Fibonacci" generators being among the first proposed schemes. In the following decades, theory surrounding pseudorandom sequences, that is, sequences generated *deterministically* that share many statistical properties with truly random sequences, began to develop [275].

The theory of pseudorandom sequences shares many aspects with cryptographic stream ciphers. Both constructions take a seed (resp. key) and stretch it out to an arbitrarily long stream of bits which, ideally, cannot be distinguished from random. Of course, unlike stream ciphers, non-cryptographic pseudorandom generators do not assume "intelligent" attackers, and have less stringent requirements.

Today some of the most popular generators in use are some variant of linear congruential generator [275], Mersenne Twister [326], or xorshift [319]. While those generators aim to be fast, and by and large they are, they suffer from some drawbacks:

- Linear congruential generators are, as the name implies, linear, and they suffer from the lattice structure identified early in their history [211, 318]. Furthermore, they force a painful tradeoff between period and speed—doubling the modulus bit size (resp. squaring the period) will generally quadruple the number of multiplications necessary per generated number[3].
- The Mersenne Twister has a very large state of $\approx 2.4$ KB. While this is not, in general, a large amount of memory by most standards, it makes it difficult to scale it out on many-core chips such as GPUs, where local on-chip memory is scarce. Thus the Mersenne Twister also forces a choice between space or synchronization of a single generator shared among several threads.
- The xorshift generator and its variants solve both problems raised above—they are very fast, have long periods and small states. However, they are $\mathbb{F}_2$-linear by design and this means that they fail any statistical tests that measure linearity, most notably the binary matrix rank test [320].

[3] While the complexity of $n$-bit integer multiplication is $O(n \log n)$ [224], at practical sizes quadratic multiplication is most efficient.

Nonlinear generators do exist, such as the quadratic [275, §3.2.2] or inversive congruential generators [181]. However, they require more costly multiplications or the computation of modular inverses, which makes them unacceptably slow for practical purposes. Cryptographic stream ciphers can also double as noncryptographic random number generators. They are, however, generally deemed to be too slow for noncryptographic applications, and are rarely used in that capacity. Ultimately, there is an underexplored region of the design space—fast, nonlinear, small-state noncryptographic generators.

### 1.1.3 Security

Cryptography is now a large and maturing field, with many categories of cryptographic algorithms in use. A common dichotomy is to separate them by how key material is used:

- Symmetric algorithms require the same key to be shared by all parties privy to the message;
- Asymmetric algorithms split the key into private and public components, each allowing different capabilities (e.g., the private key can be used for decryption and the public key for encryption.)
- Keyless algorithms do not require key material at all.

While most real-world protocols make use of every type of algorithm described above, we shall focus solely on symmetric and keyless primitives.

Many cryptographic algorithms in use today were designed a long time ago, when the trends pointed out in Section 1.1.1 were not yet in effect. For example, several blockciphers in common use, such as DES [163] and AES [4], use *S-boxes*[4] as a core component. While this may have been a reasonable choice at the time, it has some disadvantages today. Due to the increased disparity between CPU and memory speed in modern hardware, S-boxes inherently limit the peak speed of constructions to the maximum throughput of unpredictable memory accesses. Worse yet, such unpredictable accesses can, in certain attack models, be exploited to recover information about the key! This was found to be the case with the AES [443] and other ciphers with similar design [478]. Cache memory is not the only resource that can be exploited to recover information about the key; other CPU instructions that may not always run in a fixed amount of time may also be targeted. Examples could be IDEA's [286] modular multiplication or data-dependent rotation, as used in several ciphers [315, 316, 397, 398].

[4] An S-box, also known as substitution box, is an arbitrary function from $n$ to $m$ bits, usually implemented as a table lookup.

To avoid such risks, and to make better use of the CPU, different designs are required. One particularly successful approach has been the "add-rotate-xor" (ARX) design—the overlapping of addition modulo $2^n$, bit rotation, and xor ensures that the result is highly nonlinear, since addition is nonlinear relatively to xor. Rotation ensures that bit changes do not only affect more significant bits, but *every* other bit. Moreover, this overlapping is known to be able to generate every possible function [261, 481]. Notable ciphers using this sort of design are FEAL [424], TEA [470], Salsa20 [59], Threefish [192], and others. ARX designs are effectively the current speed leaders in modern hardware[5].

## 1.2 Contributions

This thesis consists of the design and analysis of ARX-like designs for noncryptographic pseudorandom generation and cryptographic primitives. My goal was to achieve primitives that are both secure, simple, and make the most out of the available hardware today, which includes high instruction-level parallelism, vector units, and multiple cores. In particular, I present 3 concrete ARX-like designs: Tyche, BLAKE2, and NORX.

**Tyche (Chapter 3)** Tyche is a small, high-performance nonlinear pseudorandom generator designed for noncryptographic usage. It repurposes a building block from the ChaCha stream cipher [54] to create a fast noncryptographic pseudorandom generator with a very small internal state: 4 32-bit registers. Additionally, Tyche supports the creation of independent streams, which coupled with the small state make Tyche particularly suitable for usage in graphics processing units (GPUs) or other SIMD hardware. The main results of Chapter 3 were published in PPAM 2011 [357] and PPAM 2013 [356].

**BLAKE2 (Chapter 4)** BLAKE2 is a hash function derived from the SHA-3 finalist BLAKE [28], itself also derived from ChaCha building blocks that is very friendly towards implementation in processors with SIMD capabilities. BLAKE2 extends BLAKE's "mechanical sympathy" by removing some components that revealed to be unnecessary to its security, and additionally adds explicit support for coarse-grained parallelism via *tree hashing*. The work presented in Chapter 4 was the result of a collaboration with Jean-Philippe Aumasson, Zooko Wilcox-O'Hearn, and Christian Winnerlein, and was published at ACNS 2013 [32].

[5] Excluding primitives making use of hardware-accelerated instructions.

**NORX (Chapter 5)** NORX was designed as a submission for the CAESAR [114] competition to identify authenticated encryption schemes suitable for widespread use. It combines well-understood building blocks: duplex Sponge-like design [71], a BLAKE2-like permutation that notably replaces the addition in its ARX structure by exclusively bitwise logic operations, and a coarse-grained parallel structure inspired by BLAKE2. Despite replacing addition by boolean logic—which has slower diffusion—NORX is nevertheless one of the fastest CAESAR candidates that does not rely on hardware-accelerated cryptographic instructions, such as Intel's AES-NI or SHA instructions. It advanced to round 3 of the competition. The contents of Chapter 5 were the product of a collaborative effort with Jean-Philippe Aumasson and Philipp Jovanovic, and were published at ESORICS 2014 [30].

**Analysis of NORX and other primitives (Chapters 6 and 7)** As a byproduct of the design of Chapter 5's NORX, I also present some positive cryptanalytic results on it, and use the techniques therein developed to break 2 other CAESAR candidates, McMambo [284] and Wheesht [329]. I also analyze the authenticated encryption scheme used by the Open Smart Grid protocol in Chapter 7 and demonstrate that it is utterly broken. Sections 6.1, 6.2 and 6.4 were the continuation of Chapter 5's collaboration with Jean-Philippe Aumasson and Philipp Jovanovic; this work was published at LATINCRYPT 2014 [29]. The results of Section 6.5 were published in Information Processing Letters [355]. The results of Chapter 7 were joint work with Philipp Jovanovic and published at FSE 2015 [252].

## 1.3 Publications

The following papers were published during my doctoral work.

1. Samuel Neves and Filipe Araujo. "Fast and Small Nonlinear Pseudorandom Number Generators for Computer Simulation". In: *Parallel Processing and Applied Mathematics - 9th International Conference, PPAM 2011, Torun, Poland, September 11-14, 2011. Revised Selected Papers, Part I*. Ed. by Roman Wyrzykowski, Jack Dongarra, Konrad Karczewski, and Jerzy Wasniewski. Vol. 7203. Lecture Notes in Computer Science. Springer, 2011, pp. 92–101.
2. Samuel Neves and Filipe Araujo. "On the performance of GPU public-key cryptography". In: *22nd IEEE International Conference on Application-specific Systems, Architectures and Processors, ASAP 2011, Santa Monica, CA, USA, Sept. 11-14, 2011*. Ed. by Joseph R. Cavallaro, Milos D. Ercegovac, Frank Hannig, Paolo Ienne, Earl E. Swartzlander Jr., and Alexandre F. Tenca. IEEE Computer Society, 2011, pp. 133–140.

3. Samuel Neves and Jean-Philippe Aumasson. “BLAKE and 256-bit advanced vector extensions”. In: *The Third SHA-3 Candidate Conference*. Mar. 2012.
4. Samuel Neves and Jean-Philippe Aumasson. *Implementing BLAKE with AVX, AVX2, and XOP*. Cryptology ePrint Archive, Report 2012/275. 2012.
5. Samuel Neves and Filipe Araujo. “Representing sparse binary matrices as straight-line programs for fast matrix-vector multiplication”. In: *2012 International Conference on High Performance Computing & Simulation, HPCS 2012, Madrid, Spain, July 2-6, 2012*. Ed. by Waleed W. Smari and Vesna Zeljkovic. Runner up for the Best Paper Award. IEEE, 2012, pp. 520–526.
6. Samuel Neves and Filipe Araujo. “Binary code obfuscation through C++ template metaprogramming”. In: *INForum 2012*. Ed. by Antónia Lopes and José Orlando Pereira. Universidade Nova de Lisboa, Portugal, Sept. 2012, pp. 28–40.
7. Antonio Casimiro, Paulo Veríssimo, Diego Kreutz, Filipe Araujo, Raul Barbosa, Samuel Neves, Bruno Sousa, Marília Curado, Carlos Silva, Rajeev Gandhi, and Priya Narasimhan. “TRONE: Trustworthy and Resilient Operations in a Network Environment”. In: *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, DSN 2012, Boston, MA, USA, June 25-28, 2012*. IEEE, 2012, pp. 1–6.
8. Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O’Hearn, and Christian Winnerlein. “BLAKE2: Simpler, Smaller, Fast as MD5”. In: *ACNS 13: 11th International Conference on Applied Cryptography and Network Security*. Ed. by Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini. Vol. 7954. Lecture Notes in Computer Science. Springer, Heidelberg, June 2013, pp. 119–135.
9. Samuel Neves and Filipe Araujo. “Engineering Nonlinear Pseudorandom Number Generators”. In: *Parallel Processing and Applied Mathematics - 10th International Conference, PPAM 2013, Warsaw, Poland, September 8-11, 2013, Revised Selected Papers, Part I*. Ed. by Roman Wyrzykowski, Jack Dongarra, Konrad Karczewski, and Jerzy Wasniewski. Vol. 8384. Lecture Notes in Computer Science. Springer, 2013, pp. 96–105.
10. Samuel Neves and Filipe Araujo. “Straight-line programs for fast sparse matrix-vector multiplication”. In: *Concurrency and Computation: Practice and Experience* 27.13 (2015), pp. 3245–3261.
11. Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. *NORX v1*. CAESAR Proposal. 2014.

12. Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. “NORX: Parallel and Scalable AEAD”. In: *ESORICS 2014: 19th European Symposium on Research in Computer Security, Part II*. Ed. by Miroslaw Kutylowski and Jaideep Vaidya. Vol. 8713. Lecture Notes in Computer Science. Springer, Heidelberg, Sept. 2014, pp. 19–36.
13. Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. “Analysis of NORX: Investigating Differential and Rotational Properties”. In: *Progress in Cryptology - LATINCRYPT 2014: 3rd International Conference on Cryptology and Information Security in Latin America*. Ed. by Diego F. Aranha and Alfred Menezes. Vol. 8895. Lecture Notes in Computer Science. Springer, Heidelberg, Sept. 2015, pp. 306–324.
14. Philipp Jovanovic and Samuel Neves. “Practical Cryptanalysis of the Open Smart Grid Protocol”. In: *Fast Software Encryption – FSE 2015*. Ed. by Gregor Leander. Vol. 9054. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2015, pp. 297–316.
15. Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. *NORX8 and NORX16: Authenticated Encryption for Low-End Systems*. Cryptology ePrint Archive, Report 2015/1154. 2015.
16. Samuel Neves and Mehdi Tibouchi. “Degenerate Curve Attacks - Extending Invalid Curve Attacks to Edwards Curves and Other Models”. In: *PKC 2016: 19th International Conference on Theory and Practice of Public Key Cryptography, Part II*. Ed. by Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang. Vol. 9615. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2016, pp. 19–35.
17. Robert Granger, Philipp Jovanovic, Bart Mennink, and Samuel Neves. “Improved Masking for Tweakable Blockciphers with Applications to Authenticated Encryption”. In: *Advances in Cryptology – EUROCRYPT 2016, Part I*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9665. Lecture Notes in Computer Science. Springer, Heidelberg, May 2016, pp. 263–293.
18. Atul Luykx, Bart Mennink, and Samuel Neves. “Security Analysis of BLAKE2’s Modes of Operation”. In: *IACR Transactions on Symmetric Cryptology* 2016.1 (2016), pp. 158–176. ISSN: 2519-173X.
19. Anne Canteaut, Eran Lambooij, Samuel Neves, Shahram Rasoolzadeh, Yu Sasaki, and Marc Stevens. “Refined Probability of Differential Characteristics Including Dependency Between Multiple Rounds”. In: *IACR Transactions on Symmetric Cryptology* 2017.2 (2017), pp. 203–227. ISSN: 2519-173X.

20. Bart Mennink and Samuel Neves. “Encrypted Davies-Meyer and Its Dual: Towards Optimal Security Using Mirror Theory”. In: *Advances in Cryptology – CRYPTO 2017, Part III*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10403. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2017, pp. 556–583.

21. Samuel Neves and Mehdi Tibouchi. “Degenerate curve attacks: extending invalid curve attacks to Edwards curves and other models”. In: *IET Information Security* 12.3 (2018), pp. 217–225.

22. Bart Mennink and Samuel Neves. “Optimal PRFs from Blockcipher Designs”. In: *IACR Transactions on Symmetric Cryptology* 2017.3 (2017), pp. 228–252. ISSN: 2519-173X.

23. Samuel Neves and Filipe Araujo. “An observation on NORX, BLAKE2, and ChaCha”. In: *Information Processing Letters* 149 (2019), pp. 1–5. ISSN: 0020-0190.

24. Filipe Araujo and Samuel Neves. “The circulant hash revisited”. In: *Journal of Mathematical Cryptology* 15.1 (2020), pp. 250–257. ISSN: 1862-2976.

25. André Lizardo, Raul Barbosa, Samuel Neves, Jaime Correia, and Filipe Araújo. “End-to-end secure group communication for the Internet of Things”. In: *J. Inf. Secur. Appl.* 58 (2021), p. 102772.

26. Bart Mennink and Samuel Neves. “On the Resilience of Even-Mansour to Invariant Permutations”. In: *Des. Codes Cryptogr.* 89.5 (2021), pp. 859–893.

# Chapter 2

# Background

## 2.1 Notation

We use standard notation from the cryptographic literature. Table 2.1 summarizes the most common notation used throughout this document. Note that individual chapters may introduce more specific notation if so required.

Table 2.1: Common notation.

| Symbol | Meaning |
|---|---|
| $\{0,1\}^n$ | The set of binary strings of $n$ bits |
| $\mathbb{N}$ | The set of natural numbers |
| $\mathbb{Z}$ | The set of integers |
| $\mathbb{F}_q$ | The finite field with exactly $q$ elements |
| $\lll n$ | Bit rotation towards the most significant bits |
| $\ggg n$ | Bit rotation towards the least significant bits |
| $\vee$ | Bitwise OR |
| $\wedge$ | Bitwise AND |
| $\oplus$ | Bitwise XOR |
| $\neg$ | Bitwise NOT |
| $\mathsf{func}(n,m)$ | Set of all functions from $\{0,1\}^n$ to $\{0,1\}^m$ |
| $\mathsf{perm}(n)$ | Set of all permutations of $\{0,1\}^n$ |
| $(n)_r$ | The falling factorial $n(n-1)\ldots(n-r+1)$ |

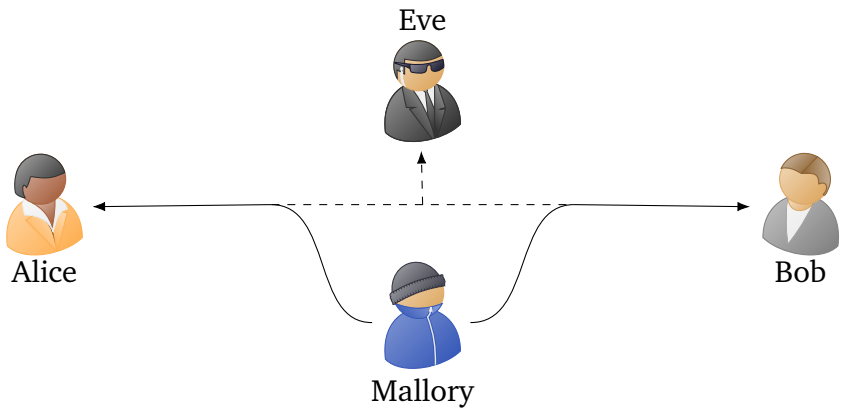



Figure 2.1: The symmetric cryptography setting. Eve is a passive eavesdropper, whereas Mallory is the proverbial active "man in the middle" attacker.

## 2.2 Defining Security

We cannot speak of security without asking the fundamental question "against what?". In this section, we are concerned with the simple setting in which two parties—Alice and Bob—communicate over an open channel, such as the Internet, GSM, or radio, where messages can be eavesdropped or altered in transit, cf. Figure 2.1. They are assumed to have somehow agreed on a secret key $k$ ahead of time; the means by which this happens are out of scope here[1]. In this setting, we may be trying to achieve one or more of the following goals:

**Secrecy** Two parties who share a key $k$ want to communicate in secret over an open channel.

**Authenticity** Two parties who share a key $k$ want to ensure that their communications come from the expected party.

**Integrity** Two parties who share a key $k$ want to ensure that their communications are not tampered with.

Shannon [421, 422] first studied secrecy in the information-theoretic setting, where an attacker has unbounded computational power and is only limited by the information it is able to collect. In this setting it is possible to define what is, exactly, a perfect cipher. Definition 2.1 elaborates.

[1] One may speculate that they agreed on a shared key in person, or that some form of authenticated Diffie-Hellman key exchange [164, 165] may have occurred.

**Definition 2.1** (Perfect secrecy). *Let $\mathscr{E} = \{E, D\}$ be a pair of functions, $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$, $D : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$, and $D(k, E(k,m)) = m, \quad \forall m \in \{0,1\}^n$. $\mathscr{E}$ is perfectly secret if for all $m_0, m_1 \in \{0,1\}^n$ and $c \in \{0,1\}^n$,*

$$\Pr[E(k, m_0) = c] = \Pr[E(k, m_1) = c]$$

*with the randomness taken over the choice of $k$.*

Shannon further proved that the one-time pad, also known as the Vernam cipher [51, 457], is perfectly secret. The one-time pad is quite simple to describe—given a key $k \xleftarrow{\$} \{0,1\}^n$ of the same length as the message $m \in \{0,1\}^n$ to encrypt, obtain the ciphertext $c \in \{0,1\}^n$ as $c = m \oplus k$, and recover the message from the ciphertext as $m = c \oplus k$.

**Theorem 2.2.** *The one-time pad $\mathscr{E} : \{\cdot \oplus k, \cdot \oplus k\}$ is perfectly secret.*

*Proof.* We prove the required property from Definition 2.1 by showing that for any pair $(m, c)$ there is precisely 1 key that results in $E(k, m) = c$. That key is $k = m \oplus c$, and is unique. For a key $k$ taken uniformly at random, then, the probability of $\Pr[E(k, m_0) = c]$ is exactly the same as $\Pr[E(k, m_1) = c]$, for any $m_0, m_1$: $2^{-n}$. □

Although the one-time pad is often touted as the perfect cipher—and rightfully so—it is pretty much never used in the real world. Why? The answer is that the one-time pad requires a separate key for each message, and this key needs to be uniformly random *for each distinct message*. Managing this volume of key material is a logistical nightmare, and if two parties are able to securely exchange keys with the same length as their messages, one wonders why they would need to use the one-time pad in the first place!

When the "one-time" requirement from the one-time pad is ignored, that is, when the same key is reused for more than one message, secrecy breaks down. Eavesdroppers learn the xor of the messages—$m_0 \oplus k \oplus m_1 \oplus k = m_0 \oplus m_1$—and through basic statistical analysis, may learn the content of the messages. This was what happened in the Venona project [225], one of the most famous cryptanalytic successes of modern times. Due to the failure of Soviet agents to ensure unique keys when using one-time pad encryption, British and American cryptanalysts were able to decipher some of their messages.

Given that perfect secrecy is not practical, what can we do? A useful notion is that of *computational secrecy*.

**Definition 2.3** (Computational secrecy). *Let $\mathscr{E} = \{E, D\}$ be a pair of functions, $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$, $D : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$, and $D(k, E(k,m)) = m, \quad \forall m \in \{0,1\}^n$. $\mathscr{E}$ is computationally secret if for all efficient attackers $A$,*

$$|\Pr[A(m_0, m_1, c) = 1 \mid c = E(k, m_0)] - \Pr[A(m_0, m_1, c) = 1 \mid c = E(k, m_1)]| \leq \mathsf{negl}(k) \tag{2.1}$$

*with the randomness taken over the choice of k. The expression (2.1) is known as the advantage* $\mathsf{Adv}_A(k)$.

In other words, instead of demanding that the encryption be secure against every conceivable attacker, we constrain the attackers to those with limited computational power. In particular, an efficient adversary, in the sense of Definition 2.3, is a probabilistic polynomial time algorithm relatively to the security parameter $k$. Accordingly, the advantage $\mathsf{negl}(k)$ is a negligible function of the security parameter $k$.

The observant reader might recognize Definition 2.3 by another name—*semantic security*. This notion, originally invented by Goldwasser and Micali [205, 206], is the standard definition of secrecy in modern practical cryptography. While it is often stated in its asymptotic form, which is not useful to assess the concrete security of a construction—where constant factors *do* matter—it also has concrete counterparts, so-called practice-oriented provable security [43, 405], under which both the computational allowance given to the attacker and the advantage are given by concrete bounds.

## 2.3 Cryptographic Primitives

### 2.3.1 Pseudorandom Number Generators

Cryptographic pseudorandom generators can be defined in terms of the distinguishing advantage of a computationally-constrained attacker, as defined in Definition 2.4.

**Definition 2.4** (Pseudorandom generator)**.** *A* $(t, \epsilon)$*-secure cryptographic pseudorandom generator is a deterministic function* $G : \{0,1\}^n \to \{0,1\}^m$*,* $m > n$*, for which every distinguisher* $\mathsf{D}$*, running in time at most* $t$*,*

$$\mathsf{Adv}^{\mathrm{prg}}_G(\mathsf{D}) = \left|\Pr_{k \leftarrow\$ \{0,1\}^n}[\mathsf{D}(G(k)) = 1] - \Pr_{r \leftarrow\$ \{0,1\}^m}[\mathsf{D}(r) = 1]\right| \leq \epsilon.$$

In other words, a pseudorandom number generator is an algorithm that "stretches" a secret seed of $n$ bits to a possibly much larger size $m$, with this output remaining indistinguishable from random except with very high computational or data requirements, i.e., it passes all statistical tests up to a given cost. Cryptographic pseudorandom generators are rarely constructed from scratch. Instead, they are commonly built in terms of block ciphers or hash functions.

#### Non-cryptographic pseudorandom generators

Although there is plenty of overlap between cryptographic and non-cryptographic generators, they are seldom considered together. Cryptographic generators clearly are suitable for non-cryptographic purposes, but they are often considered too slow

for this purpose. The converse is not true, however; non-cryptographic generators are often completely unsuitable for cryptographic purposes, e.g., [211, 273, 321, 336, 392–394].

In practice, the quality of such generators is often measured by batteries of statistical tests such as Diehard [317] or TestU01 [283]. Sometimes, Monte Carlo simulations themselves act as distinguishers for some particularly bad generators [129, 193, 209, 381]!

To make it possible to use the same definitions, we may adopt a more relaxed definition of pseudorandom number generator (cf. Definition 2.4) for non-cryptographic purposes.

**Definition 2.5** (Non-cryptographic pseudorandom number generator)**.** *A non-cryptographic generator is a function* $G : \{0,1\}^n \to \{0,1\}^m$ *which is also a* $(0.05, 2^{36})$*-secure generator by Definition 2.4.*

In other words, a non-cryptographic generator passes most cheap statistical tests with high probability. The concrete parameters were based on TestU01's "Big Crush" battery, which looks at $2^{36}$ pseudorandom values at the most.

### 2.3.2 Stream Ciphers

Stream ciphers are very much related to pseudorandom number generators. Their main difference is that, while a generator simply expands secret inputs, a stream cipher takes in a key and a nonce. Its definition is given in Definition 2.6.

**Definition 2.6** (Stream Cipher)**.** *A* $(t, \epsilon)$*-secure stream cipher is a deterministic function* $E : \{0,1\}^l \times \{0,1\}^n \to \{0,1\}^m$*,* $m > n$*, for which every distinguisher* $\mathsf{D}$*, running in time at most* $t$*,*

$$\mathsf{Adv}^{\text{stream}}_G(\mathsf{D}) = \left|\Pr_{k \leftarrow\$ \{0,1\}^l}[\mathsf{D}(G(k,\cdot)) = 1] - \Pr_{r \leftarrow\$ \{0,1\}^m}[\mathsf{D}(r) = 1]\right| \leq \epsilon.$$

As with pseudorandom number generators, stream ciphers are not commonly designed from scratch. The most common stream cipher is, in fact, a block cipher in the CTR or OFB mode of operation (cf. Section 2.3.3). Stream ciphers that are *not* built from other primitives are, for example, RC4 [380], Trivium [158], or Grain [226].

### 2.3.3 Block Ciphers

In his famous papers, Shannon [421, 422] described what a perfect $n$-bit block cipher ought to be—each key would select one out of $(2^n)!$ permutations independently at random. When a block cipher is modeled this way, one is said to be operating in the *ideal cipher model*.

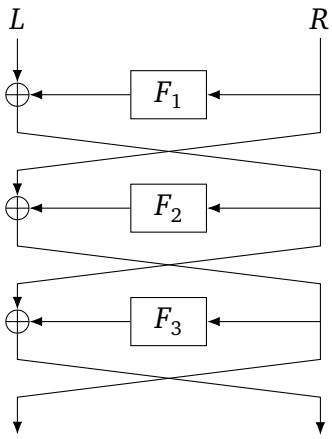



Figure 2.2: 3-round Feistel network.

Unfortunately, this is a very inefficient way to design block ciphers; a block cipher with $n$-bit blocks and $k$-bit keys requires $2^k \log_2 2^n! \approx n2^{k+n}$ bits to represent. In practice, block ciphers use a mixture of *confusion* and *diffusion* primitives, iterated over a sufficient number of rounds, to achieve an approximation of a random permutation. This leads to the standard definition of a pseudorandom permutation, or PRP for short.

**Definition 2.7** (Block Cipher)**.** *A $(t, \epsilon)$-secure block cipher is a deterministic function $B : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ for which every distinguisher $\mathsf{D}$, running in time at most $t$,*

$$\mathsf{Adv}^{\mathrm{prp}}_B(\mathsf{D}) = \left|\Pr_{k \leftarrow\$ \{0,1\}^k}[\mathsf{D}(B(k,\cdot)) = 1] - \Pr_{F \leftarrow\$ \mathsf{perm}(n)}[\mathsf{D}(F(\cdot)) = 1]\right| \leq \epsilon.$$

There are two main approaches to design a block cipher: Feistel networks and substitution-permutation networks. Recently there has been increased interest in a third, simpler, construction—Even-Mansour.

### Feistel Networks

The first publicly known practical block ciphers were developed by Horst Feistel's research group in the 1970s [188–190]. From that work two ciphers resulted: Lucifer [429] and the Data Encryption Standard (DES) [163]. The Soviet Union's GOST 28147-89 [386, 475], alleged to have been developed during the same period, also follows the same high-level structure, and so does the East German SKS V/1 [139]. That structure is usually called the *Feistel network* or *Luby-Rackoff cipher*.

A Feistel network is an iterated permutation that operates over several *rounds*. The block is divided into two equally-sized halves $L$ and $R$, and at each round $i$, the

following transformation is applied (cf. Figure 2.2):

$$L, R = R, f_{k_i}(L) \oplus R, \tag{2.2}$$

where $f_{k_i}$ is a function making use of a *round key* $k_i$.

Luby and Rackoff [305, 306] later abstracted and generalized the Feistel construction to take random and independent functions $F_i \in \mathrm{func}(n, n)$, where (2.2) becomes

$$L, R = R, F_i(L) \oplus R.$$

With this new abstraction in place, Luby and Rackoff [305, 306][2] showed that 3 rounds of this construction are indistinguishable from a random function with probability $\leq q^2/2^n$ for an adversary that performs $q$ *encryption* calls and unbounded amount of computation. The 4-round Luby-Rackoff cipher, on the other hand, is indistinguishable from a random function with the same probability, but allowing encryption *and* decryption queries. These bounds were shown to be tight by Patarin [370] and Aiello and Venkatesan [5], who gave matching attacks with the same complexity.

The Luby-Rackoff construction, for varying number of rounds, is now well understood; it is known [350, 369–371, 373, 374] that 5 rounds are sufficient for security up to approximately $2^n$ encryption/decryption queries.

Several generalizations of the Luby-Rackoff construction have also been proposed and studied:

- Replace some random round functions by weaker functions [309, 352, 353, 375];
- Require only one random round function [351, 368, 385, 412].
- Unbalanced Feistel networks, where $L$ and $R$ can have differing lengths [352, 353, 413];
- Generalized Feistel networks with more than 2 halves [233, 365, 479, 480];
- Misty, KASUMI, and Lai-Massey schemes [244, 245, 256, 257, 292, 311, 312, 453, 474].

### Substitution-Permutation Networks

A different approach to building block ciphers is the so-called substitution-permutation network (SPN). In this construction there are two separate layers: the substitution layer and the permutation layer.

[2] See also [327, 372] for simplified treatments of the same result.

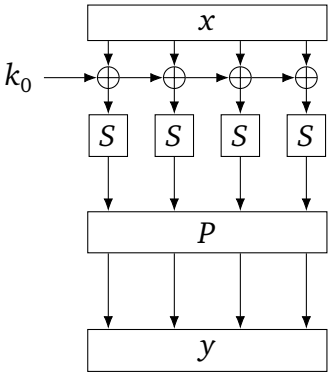



Figure 2.3: One round of a substitution-permutation network with 4 lanes.

The substitution layer splits the input into a number of lanes, and to each lane applies a nonlinear bijection $S$. This bijection may be itself keyed, or key material might be xored into each lane before applying $S$. The permutation layer then concatenates the lanes together again, and applies an arbitrary linear transformation to the state[3]. Figure 2.3 depicts one round of a general substitution-permutation network with 4 lanes.

In SPNs each layer serves a purpose. The substitution layer, along with keying, creates *confusion*—the input and key are now mixed in a complicated manner. The permutation layer is there for diffusion—spreading changes quickly throughout the entire block.

The most famous and successful SPN today is by far the Advanced Encryption Standard, or AES [150]. The AES was the winner of a cryptographic competition held by NIST in 1997 where many candidate ciphers, most of which Feistel networks, were submitted. Its security has held admirably throughout more than 20 years of scrutiny.

Although very successful, there are relatively few results on the security of SPNs when abstracted away in the Luby-Rackoff model. Vaudenay and Baignères [35] showed that replacing the AES S-boxes by random permutation results in a cipher provably secure against linear and differential cryptanalysis, but not against every class of attacks. Cogliati et al. [130] recently proved a stronger result, but the proof is only useful for unrealistically large S-boxes. Dodis et al. [168] get closer to meaningful results.

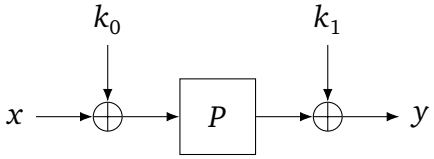



Figure 2.4: The Even-Mansour blockcipher.

**Even-Mansour**

Even and Mansour [185, 186] considered the simplest way to build a block cipher from a single randomly chosen permutation.

Let $P \xleftarrow{\$} \text{perm}(n)$, and $K_1, K_2 \xleftarrow{\$} \{0,1\}^n$. Then the Even-Mansour pseudorandom permutation is given by (cf. Figure 2.4)

$$E_{K_1,K_2}(M) = P(M \oplus K_1) \oplus K_2. \tag{2.3}$$

Despite its simplicity, this construction is secure. An attacker that is tasked with distinguishing $E$ from a uniformly random permutation can do so with probability at most $2qp/2^n$, where $q$ is the number of calls to $E$, and $p$ the number of calls to $P$. This bound is tight; Daemen [144] gave a chosen-plaintext attack with matching complexity, and Dunkelman et al. [172, 173] gave known-plaintext attacks with matching complexity.

A closely related construction is DESX, which was invented by Rivest to strengthen DES again bruteforce attacks (cf. Section 2.4):

$$\text{DESX}_{k,k_1,k_2}(m) = \text{DES}_k(m \oplus k_1) \oplus k_2.$$

Kilian and Rogaway [264, 265] studied this construction, and suggested $k_1 = k_2$ without any meaningful loss of security. The same observation applies to Even-Mansour [172, §4].

Like Luby-Rackoff, Even-Mansour can be iterated as well to improved its security against attacks. This is alternatively called *key-alternating cipher*, and is described as follows:

$$KAC_{K_0,K_1,\ldots,K_t}(m) = P_{t-1}(P_1(P_0(m \oplus K_0) \oplus K_1) \oplus \ldots) \oplus K_t.$$

There has been a recent flurry of research on the security of key-alternating ciphers [106, 126, 127, 288], culminating in the following result by Hoang and Tessaro [234].

[3] Although originally the permutation layer in substitution-permutation networks did refer to a permutation of the bits of the block [255], nowadays this layer refers to more general linear transformation. Only a few designs, such as PRESENT [105] or GIFT [36], do actually use bit permutations as a linear layer.

**Theorem 2.8.** *Let KAC be the $t$-round key-alternating cipher using $t$ public permutations $P_i \in \mathsf{perm}(n)$ and $t+1$ keys $K_i \in \{0,1\}^n$. Then any attacker performing $q$ calls to KAC and $p_i$ calls to $P_i$ can be distinguished from a random permutation with probability at most $\frac{4^t q \prod_{i=0}^{t-1} p_i}{2^{nt}}$.*

### Modes of Operation

So far, we have surveyed how to build a fixed-size block cipher. But this is not sufficient to encrypt variable-length data, as is necessary in practice.

To make the DES more useful, NBS[4] published FIPS 81 [162], which defined some popular modes that are still in use today:

**Electronic Codebook** $C_i = E_K(M_i)$

**Cipher Block Chaining** $C_i = E_K(M_i \oplus C_{i-1})$

**Output Feedback Mode** $C_i = E_K(C_{i-1}) \oplus M_i$

**Counter Mode** $C_i = E_K(i) \oplus M_i$

The security of these modes was first formally analyzed in [45]. Note that the OFB and counter modes are a simple way to turn a block cipher into a stream cipher, and do not require evaluating its inverse.

The usage of the above modes without adequate authentication has been the source of many issues. The electronic codebook mode is easily distinguished from random; the counter and output feedback modes are easily malleable—xoring the ciphertext propagates the difference to the deciphered plaintext. Cipher block chaining is also malleable, and given adequate decryption oracles plaintext recovery becomes quite practical [37, 176, 249, 455].

## 2.3.4 Message Authentication Codes

Message authentication codes, or MACs, are algorithms used to validate the integrity of a message.

**Definition 2.9** (Message authentication code)**.** *A $(t,\epsilon)$-secure message authentication code is a deterministic function $H : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^n$ for which every algorithm $\mathsf{P}$ with blackbox access to $H(k,\cdot)$, running in time at most $t$,*

$$\mathsf{Adv}^{\mathrm{mac}}_H(\mathsf{P}) = \Pr_{k \leftarrow\$ \{0,1\}^k}[\mathsf{P}(H(k,m)) = H(k,m)] \leq \epsilon\,,$$

*where $m$ is an input that was not queried by $\mathsf{P}$.*

[4] Better known today as NIST.

That is, an attacker with oracle access to the message authentication code is unable to produce a tag for a new message[5] with significantly better probability than by guessing.

Most MACs are created by using a pseudorandom function, itself usually built from a block cipher or a hash function. Any pseudorandom function is a secure MAC, but the converse is not necessarily the case. The best known and most widely used MAC is probably HMAC [44], which takes an existing hash function and turns it into a PRF.

A different class of MACs are the so-called one-time authenticators. These types of authenticators were pioneered by Carter and Wegman [468] and are generally of the form

$$\mathsf{WC}(k_0, k_1, n, m) = H(k_0, m) + F(k_1, n),$$

where $H$ is a differentially uniform hash function, $F$ is a fixed-length pseudorandom function, and $n$ is a unique value. The advantage of this type of authenticator is that the bulk of the processing is done by the cheaper hash function, and the pseudorandom function can operate in parallel on a much smaller number.

One-time authenticators are the current speed champions in authentication and include such functions as Poly1305 [58], GMAC [331], UMAC [97], and many others.

Yet another class of authenticators making use of universal hashing is the hash-then-PRF construction:

$$\mathsf{HtF}(\mathsf{k}_0, \mathsf{k}_1, \mathsf{m}) = F(k_1, H(k_0, m)).$$

These authenticators do away with the need for a unique nonce, whose uniqueness may be hard to guarantee in real protocols. In exchange, their security bounds degrade faster than Wegman-Carter. The protected counter sum [56], PMAC [98, 156], and LightMAC [314] functions belong to this category.

### 2.3.5 Authenticated Encryption

Historically, messages were encrypted using a block cipher in some secure mode of operation, cf. Section 2.3.3, and independently authenticated using some message authentication code, cf. Section 2.3.4. This is the so-called *generic composition* setting. There are essentially 3 main ways in which this can be done:

**Encrypt-then-MAC** First encrypt the message, and then authenticate the resulting ciphertext.

**MAC-then-encrypt** First authenticate the message, and then encrypt the message *and* the authentication tag.

[5] Or an existing one, in some attack models.

**MAC-and-encrypt** Encrypt and authenticate the message, and output each result independently of each other.

Bellare and Namprempre [48] analyzed each of these possibilities abstracting away the details of encryption and authentication. They found out that of these three options, only the first—encrypt-then-MAC—was generically secure under the most assumptions.

The problem with this is that most real-world protocols at the time did not apply generic composition properly, to disastrous effects:

- The SSH protocol used the MAC-and-encrypt order [6, 47];
- SSL and TLS used MAC-then-encrypt, which combined with padding oracles [455] has led to many attacks [117, 376, 377, 455].

Some cryptographers concluded that the abstraction layer of encryption schemes and MACs was imperfect; what users really needed was authenticated encryption as an atomic primitive [403].

Authenticated encryption is thus a primitive that combines an encryption algorithm and an authenticator in a single package. Additionally, it has been suggested, and is now commonplace, to have an *associated data* input that also authenticates metadata associated with the message, such as session information [403].

**Definition 2.10** (Authenticated encryption with associated data)**.** *An authenticated encryption with associated data scheme is a pair of functions*

$$\begin{aligned}\mathsf{Enc} &: \{0,1\}^k \times \{0,1\}^n \times \{0,1\}^* \times \{0,1\}^* \to (\{0,1\}^*, \{0,1\}^t)\\ \mathsf{Dec} &: \{0,1\}^k \times \{0,1\}^n \times \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^t \to \{0,1\}^* \cup \{\bot\}\,.\end{aligned}$$

*Given a $k$-bit key $K$, $n$-bit nonce $N$, a message $M$, and associated data $A$,* $\mathsf{Enc}(K,N,M,A)$ *returns a pair $(C,T)$ consisting of a ciphertext and a $t$-bit authentication tag.* $\mathsf{Dec}(K,N,C,A)$ *returns the decrypted message $M$ if the tag verification succeeds, otherwise it returns $\bot$.*

The security of an authentication scheme is then defined by the indistinguishability of its ciphertexts and the unforgeability of its tags, as defined in previous sections.

Authenticated encryption is generally the most appropriate tool for message encryption; without authentication, encryption can often be negated or subverted by active attackers. While for a long time authenticated encryption was stitched together via composition of unrelated primitives, e.g., AES-CBC combined with HMAC-SHA1, there are dedicated authenticated encryption primitives that are more efficient, namely AES-GCM [331], EAX [50], or OCB [406].

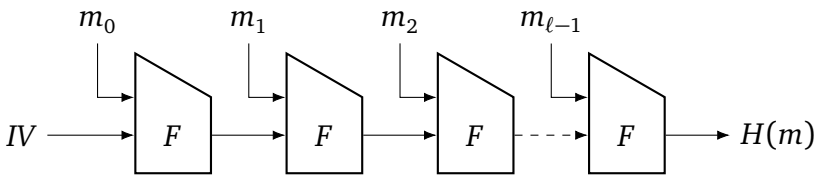



Figure 2.5: The Merkle-Damgård mode of operation on a 4-block padded message.

### 2.3.6 Hash Functions

Cryptographic hash functions compress a message $m$ of arbitrary length $\{0,1\}^*$ down to a fixed-size string $\{0,1\}^n$. Historically, they were expected to satisfy the following properties [407]:

**Preimage resistance** Given a string $h \in \{0,1\}^n$, it should be computationally unfeasible—on the order of $2^n$ operations—to obtain a message that hashes to $h$.

**Second-preimage resistance** Given a message $m$ and its respective hash $h$, it should be computationally unfeasible to find another message $m'$ that hashes to the same value $h$.

**Collision resistance** It should be computationally unfeasible—on the order of $2^{n/2}$ operations—to find two messages $m_1, m_2$ that hash to the same value.

Another notion that hash functions are often expected to satisfy is that of the random oracle [49]—a theoretical construct that responds do each new query with a independent and uniformly random value in the output range. Unfortunately, it has been shown that there are schemes that are secure with random oracles but insecure with any concrete instantiation of one [115]. This raised doubts about the validity of the random oracle methodology. These constructions are quite contrived, however, and it is arguable that the random oracle model does more good than harm, and constructions designed to avoid the random oracle model are often worse [277].

But how does one show that a hash function is, or behaves like, a random oracle? With another layer of indirection it is possible to show that a hash function can safely be replaced by a random oracle. Suppose one has a hash function H making use of a fixed-width primitive F, e.g., a compression function or a block cipher. If one assumes instead that F is ideal, it is possible to show that H is as good as a random oracle. The idea is roughly the same as with indistinguishability. But indistinguishability relies on there being secret information the adversary is not privy to, e.g., a key or a random function. Distinguishing a random oracle from H is trivial—since the adversary has unfettered access to F, it can cross-check the inputs to H with F and easily tell each case apart.

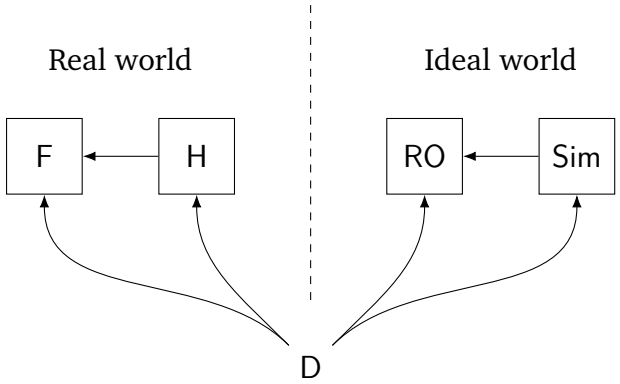



Figure 2.6: The indifferentiability framework. A distinguisher D with either access to the real world, where it queries a primitive H which calls an ideal component F, or the ideal world, where it queries an ideal RO and a simulator Sim which may call RO, must decide which world it is dealing with.

Maurer et al.'s indifferentiability framework [328], specialized to hashing by Coron et al. [134], adds a simulator to the mix. In particular, a distinguisher can talk to one of two worlds. The *real world* comprises the hash function H and its associated compression function F *modeled as an ideal object*. The *ideal world* consists of a random oracle RO with the same domain and range as H, and a *simulator* Sim. This simulator "pretends" to be F based only off of responses from RO and its own internal bookkeeping. Figure 2.6 depicts this scenario.

The main idea is that if there is a simulator that can pretend to be F well enough that the distinguisher cannot tell apart which world it is talking to, the availability of F cannot do much to help the distinguisher, and thus the hash function H can be treated as random oracle in further analyses. Definition 2.11 formalizes this notion.

**Definition 2.11.** *Let* $\mathsf{H} : \{0,1\}^* \to \{0,1\}^n$ *be a hash function with oracle access to an ideal primitive* $\mathsf{F} : \{0,1\}^m \to \{0,1\}^n$. *Let* RO *be a random oracle with the same domain and range as* H*, and* Sim *a simulator with oracle access to* RO*. The advantage of* H *with respect to differentiability of a distinguisher* D *making at most* $q$ *queries is given by*

$$\mathsf{Adv}^{\text{indiff}}_{\mathsf{H},\mathsf{RO}}(q) = |\Pr[\mathsf{D}(\mathsf{H},\mathsf{F}) = 1] - \Pr[\mathsf{D}(\mathsf{RO},\mathsf{Sim}) = 1]| \ .$$

This may seem pointless. After all, we are replacing one ideal component, the hash function, by another ideal component—its compression function. But this has its advantages: first, it ensures that the way the hash function iterates through its input is *sound* and leaves no room for unexpected behavior. Secondly it focuses the attention on the smaller, simpler, component that can be more easily analyzed by cryptanalysts.

### Merkle-Damgård

The most notable hash function construction was put forward by Merkle and, independently, Damgård [153, 334][6]. Given a fixed-size compression function, after suitably padding the message to a multiple of the block size, one simply chains the output of one call to the input of the next. Figure 2.5 depicts this process for a 4-block message. Most classic hash functions, such as MD4 [254, 396], MD5 [395], or SHA-1 [418], follow this design principle.

Merkle-Damgård is collision resistant if the compression function is also collision resistant [153, 334]. It is vulnerable to length-extension attacks, however, which makes it clearly differentiable from a random oracle. Consider Definition 2.11. A distinguisher D can make three queries:

1. $h_0 = \mathsf{H}(m_0)$;
2. $c_0 = \mathsf{F}(h_0, m_1)$;
3. $h_1 = \mathsf{H}(m_0, m_1)$;
4. Return 1 if $c_0 = h_1$, 0 otherwise.

This distinguisher always returns 1 for the real world. But a simulator that tries to simulate the query $\mathsf{F}(h_0, m_1)$ does not know what $m_0$ is, and therefore cannot easily query $\mathsf{RO}(m_0, m_1)$ to keep its story straight. As such, the best it can do is guess either $m_0$ or $\mathsf{RO}(m_0, m_1)$. This makes the differentiability advantage of this distinguisher $1 - 2^{-\min(n,m)}$.

Additionally, finding multicollisions is much easier in Merkle-Damgård than it would be in a random oracle [12, 13, 248, 259, 260]. To salvage the construction from the above flaws, several variants have been proposed: chopMD [123], using a wider block size [307, 308], using a permutation in the last compression function call [230, 231], or using a counter [81].

### Sponges

Sponge functions [66] were initially proposed for hashing as a way to specify a function behaving very much like a random oracle, but with the inevitable possibility of internal collisions. However, it quickly became apparent that sponge functions were actually a practical way to design hash functions, and they became the basis for Keccak [68], the SHA-3 competition winner.

Sponge functions make use of a single public permutation $\mathsf{P} : \{0,1\}^b \to \{0,1\}^b$, unlike most designs up to that point which were almost entirely based on block

[6] Though it would appear that Rabin came up with the same design principle first in [391].

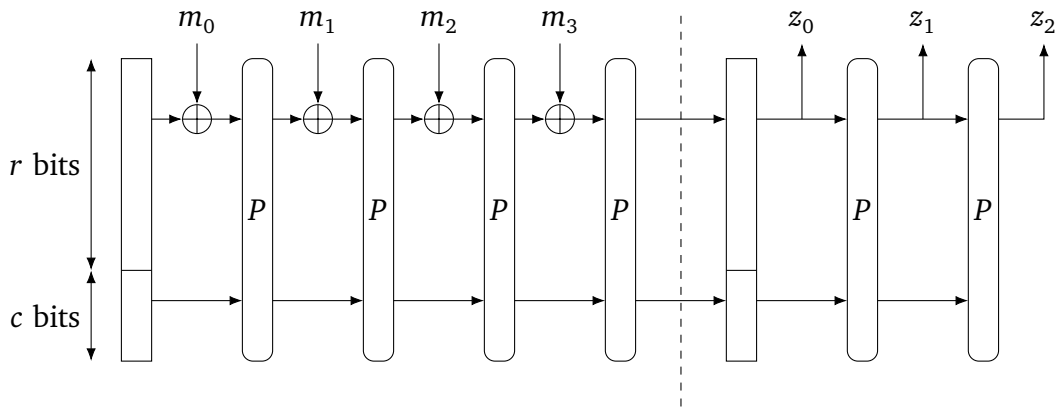



Figure 2.7: Sponge hashing.

ciphers[7]. The basic layout for a sponge function is given in Figure 2.7. The permutation is split into two halves, one of which is "public"—the $r$-bit rate—and the other one private—the $c$-bit capacity. Message blocks are xored into the rate during the absorption phase, followed by an application of the permutation. The squeezing phase follows, in which an arbitrary number of output bits are extracted from the rate, separated by permutation calls if necessary.

The security of sponge functions is tied to two parameters: the size of the permutation $b$, as well as the capacity $c$. Bertoni et al. [72] showed that the differentiability advantage of any distinguisher for the sponge construction using an ideal permutation is, for $q \ll 2^c$,

$$\mathsf{Adv}^{\text{indiff}}_{\text{sponge,RO}}(q) \leq \frac{\binom{q+1}{2}}{2^c} - \frac{\binom{q}{2}}{2^b}\,. \tag{2.4}$$

Despite being originally designed for hashing, sponge functions have proved to be highly versatile, and have found applications in many other areas, such as pseudorandom generation [73], authenticated encryption [65, 71, 332], randomness extractors and key-derivation functions [203, 238], protocols [220, 411], and more.

## 2.4 Cryptanalysis

Cryptanalysis is the subfield of cryptology that deals with the analysis of cryptosystems, confirming or disproving their security claims. Simply put, cryptanalysts try to break cryptosystems.

[7] An exception is Merkle's Snefru [333], built from a fixed-key block cipher, effectively the same as a public permutation.

As seen in the previous section, many schemes try to provably reduce to the security of a smaller core primitive, with concrete bounds [43, 405]. In most cases, this primitive is a fixed-length block cipher or keyless permutation. Cryptanalytic efforts and techniques thus mainly focus on these primitives, and most attack techniques have been developed for the analysis of block ciphers and/or permutations. In the following sections some of the most useful attacks are presented.

### 2.4.1 Attack Models

When cryptanalyzing a (symmetric) primitive, it is assumed that the adversary knows its inner workings in their entirety, *except for the key*. This is known as *Kerckhoffs's principle*, named after Auguste Kerckhoffs [363][8]. Furthermore, a cipher can be analyzed under different assumptions of what the attacker is able to do:

**Ciphertext-only attack** The attacker only has access to captured ciphertext and a reasonable guess of the plaintext distribution (e.g., is it English text?);

**Known-plaintext attack** The attacker is given corresponding plaintext-ciphertext pairs, but cannot choose them;

**Chosen-plaintext attack** The attacker can choose plaintexts and receives back their corresponding ciphertexts;

**Chosen-ciphertext attack** The attacker can choose both plaintext and ciphertext, and receive back their corresponding ciphertext or plaintext.

Attacks can be *adaptive* or *non-adaptive*. In adaptive attacks, the attacker chooses the next input after each query; the non-adaptive attacker is forced to choose all their inputs ahead of time. The attacker can further be given more than one target (read: key) to attack, and success is defined as breaking *at least one* of the targets. This is commonly named the multi-target or multi-user setting [74, 75, 343].

The goals of the attacker can also vary. *Key recovery* is the most common—and devastating—goal, but the attacker may also mount *distinguishing* or *forgery* attacks that break the cryptosystem claims without recovering the key. Distinguishers are often used as the first step towards key recovery, cf. Section 2.4.3.

Besides the above common models, other more niche models exist for other scenarios, e.g., related keys [46, 76, 77, 473], known keys [11, 131, 271], chosen keys [94], etc.

[8] Kerckhoffs's principle was in fact the second of six principles delineated in "La Cryptographie Militaire" [363].

### 2.4.2 Generic Attacks

A *generic* attack is one that does not make use of any internal property of the target, treating it more or less like a black box. A generic attack on a block cipher, for example, simply assumes that the block cipher behaves like a random permutation; a generic attack on a hash function assumes that its internal component, e.g., compression function, behaves like a random function.

#### Bruteforce

The better known and rather obvious generic attack is exhaustive search, also known as bruteforce: simply trying every possibility until one works. Examples of bruteforce attacks are

- Given a block cipher and one or more known plaintext-ciphertext pairs, recover the key by trying all possible keys until one matches the pairs;
- Given a hash function output, try many inputs until one of them matches it.

Given a search space of $2^k$, a bruteforce attack will succeed on average after $2^{k-1}$ attempts. Each attempt will usually involve evaluating the primitive in question; for iterated ciphers, it is generally possible to improve this cost by a marginal constant factor using more advanced techniques [104, 237, 263].

In some cases it is possible to trade time for memory. In the extreme case, the attacker precomputes the ciphertext corresponding to a fixed chosen plaintext for every key; after this upfront cost of $2^k$ evaluations and memory, recovering any key is a matter of obtaining the ciphertext and a simple hash table lookup. Hellman [228] came up with a better tradeoff—after a precomputation of $2^{2k/3}$ cipher evaluations and memory, a subsequent key recovery costs $2^{2k/3}$ evaluations. Wiener [471] and Bernstein [60] dispute the effectiveness of such time-memory tradeoffs compared to the cost of 2- and 3-dimensional parallel mesh machines, when the full cost of the computation is considered. In general, because bruteforce is trivially parallelizable, it is preferable to increase the number of processors whose cost is comparable to that of the equivalent amount of memory. For example $2^{2k/3}$ parallel processors will find a $k$-bit key in $2^{k/3}$ time, much faster than Hellman's tradeoff and at comparable cost. One may also look for one out of several keys instead of simply one, with an improved success rate proportional to the number of keys under attack.

#### Collision Search

The birthday paradox states that for there to be a colliding birthday among a set of people with randomly distributed birthdays it suffices to look at approximately 23

people. Though unintuitive, there are $(365)_{23}$ distinct sets of birthdays out of $365^{23}$ total possibilities. This gives a collision probability of $1-\frac{(365)_{23}}{365^{23}} \approx 0.507$.

To find collisions in a random $\{0,1\}^m \to \{0,1\}^n$ function $F(x)$, $m \geq n$, the obvious approach to accomplish this is as follows:

1. Initialize an empty hash table $H$;
2. For $i$ from 0 to $2^n-1$
   a) compute $y = F(i)$;
   b) if $H[y]$ exists, return $(i, H[y])$;
   c) insert $i$ into $H[y]$;

At iteration $i$ the probability of a collision is given by

$$\begin{aligned} & 1-\frac{(2^n)_i}{2^{ni}} \\ =\ & 1-\prod_j\left(1-\frac{j}{2^n}\right) \\ \geq\ & 1-e^{-\sum_{j=1}^{i-1} j} \\ =\ & 1-e^{\frac{\binom{i}{2}}{2^n}}, \end{aligned}$$

which approaches $1/2$ at $2^{n/2}$, and approaches 1 soon after. A more precise estimate for the expected number of iterations is $\sqrt{\pi 2^n/2}$ [448, Appendix A].

The issue with the above algorithm is that it requires as much space ($\approx 2^{n/2}$ entries) as computation. With an extra but constant amount of computation, collisions can be found in negligible space. The so-called rho method uses Floyd's cycle detection [275, §3.1, Exercise 6 and 7]:

1. Let $x_0 = x_1 = 0$, with 0 being an arbitrary starting point;
2. Repeatedly set $x_0 = F(x_0)$ and $x_1 = F(F(x_1))$;
3. If $x_0 = x_1$, a collision has occurred at some point in the sequence. That is, $F^i(0) = F^{2i}(0)$.

While a cycle has at this point been detected, it does not immediately yield a collision.

The idea is that the sequence $0, F^1(0), \ldots, F^i(0)$ has a "tail" and a cycle[9] of length respectively $s$ and $l$. It is known that $F^i(0)$ is inside the cycle of length $l$ and thus $i \geq s$ and because $F^i(0) = F^{2i}(0)$ then $i$ must be multiple of $l$.

[9] Hence the name rho: this trajectory resembles the Greek letter $\rho$.

The attack continues: find the first $j$ such that $F^j(0) = F^{i+j}(0)$. Unless $j = 0$, in which case there is no tail, this will yield a collision when $j = s$: $F^s(0) = F^{i+s}(0)$. One of the inputs is at the tail, the other one at the cycle. The complexity of this method is higher than the above, but is still within $O(2^{n/2})$ and the memory requirements make it much more practical for large $n$.

One drawback of the above method is that, while it improved the memory consumption, it is not amenable to parallelization. This leads us to the state of the art today in collision finding, the Wiener-Oorschot [448] parallel collision search. The Wiener-Oorschot method requires introducing the notion of *distinguished points*, values of the image of $F$ that have some easily identifiable property. A typical choice of distinguished point is a value with $\theta$ leading bits equal to zero. For a random function an output of $F$ is distinguished with probability $2^{-\theta}$.

The method works in a client-server model, and operates as follows:

1. Each client selects a random starting point $x_1$ and iterates $x_{i+1} = F(x_i)$ until a distinguished point $x_n$ is hit. The client sends the tuple $(x_1, x_n, n)$ to the server, and repeats the process.

2. The server, upon receiving a triple $(x_1, x, n)$, verifies whether it already received another triple $(x'_1, x, n')$; if so, and if the sequence generated by $(x_1, x, n)$ does not happen to be a subsequence of $(x'_1, x, n')$ or vice-versa, within both sequences generated by $x_1$ and $x'_1$ lies a collision.

Given $p$ clients, this method runs in expected $\sqrt{\pi 2^n}/p + 2.5 \cdot 2^\theta$ evaluations of $F$ per client and parallelizes trivially with the number of clients. The amount of necessary storage in the server is configurable by the parameter $\theta$.

### 2.4.3 Statistical Attacks

Statistical attacks make up the most powerful class of attacks against iterated ciphers. We remind that an iterated cipher $E_k$ is formed by the composition of $r$ round functions, each possibly using distinct round keys:

$$E_{k_{r-1}} \circ E_{k_{r-2}} \circ \cdots \circ E_{k_1} \circ E_{k_0}\,.$$

A statistical attack hinges on the existence of a distinguisher for part of the cipher. As before, a distinguisher D is an algorithm that, given some inputs, returns 0 or 1 and distinguishes the target from the corresponding ideal object with some success probability. For a blockcipher see Definition 2.7. A high-probability distinguisher for the entire cipher immediately breaks it. But full-round distinguishers are relatively rare.

For block ciphers another popular application of statistical distinguishers is some variation of a last-round attack. In a last-round attack, one hopes to find an efficient

**Algorithm 2.1:** A generic chosen-plaintext last-round attack of order $d$.

**Input:** Chosen plaintext oracle $O$, distinguisher $D$
**Result:** Set $S$ of candidate last round keys.
$S \leftarrow \{\}$;
**for** $k' \in$ *admissible last round keys* **do**
  $c \leftarrow 0$;
  **for** $i \leftarrow 0$ **to** *data requirement* **do**
    $(x_0, \ldots, x_{d-1}) \leftarrow \ldots$; `// Choose inputs`
    $(y_0, \ldots, y_{d-1}) \leftarrow (O(x_0), \ldots, O(x_{d-1}))$; `// Obtain ciphertext`
    $(z_0, \ldots, z_{d-1}) \leftarrow \left(E_{k'}^{-1}(x_0), \ldots, E_{k'}^{-1}(x_{d-1})\right)$; `// Undo last round`
    $c \leftarrow c + D((x_0, \ldots, x_{d-1}), (z_0, \ldots, z_{d-1}))$; `// Test property`
  **end**
  **if** $c \geq$ *some threshold* **then**
    $S \leftarrow S \cup \{k'\}$;
  **end**
**end**
**return** $S$

distinguisher for $E_{k_{r-2}} \circ \cdots \circ E_{k_1} \circ E_{k_0}$, i.e., the cipher minus its last round. It is assumed that the choice of key does not matter, i.e., that the distinguisher works similarly regardless of key choice. This is the so-called stochastic equivalence hypothesis [287], and although it does not necessarily apply [149, 151], it is a useful heuristic.

Having such a distinguisher, an attacker in possession of many plaintext-ciphertext tuples can try to undo the last round and verify whether the distinguisher succeeds. To undo the last round requires knowledge of the last round key $k_{r-1}$; the attacker must then guess every possible value of $k_{r-1}$, and wrong key guesses will likely result in the distinguisher failing—this is the so-called wrong-key randomization hypothesis [222]. Once the last round key is recovered, the attack is either over—the main key can be recovered from it—or the attack proceeds to recover the second-last round in the same fashion.

Distinguishers can be parameterized by the number of inputs they take, as per Vaudenay's treatment [452]. If a distinguisher purely observes relations between plaintext and ciphertext, we say it is of order 1; if it observes relationships between pairs of plaintexts and ciphertexts, it is of order 2; and so on. Algorithm 2.1 lays out the general framework for a last-round attack using a distinguisher of order $d$. To attack a concrete cipher, many more details need to be filled in: the distinguisher to use, the data requirement, the admissible round key set, the threshold to use for valid candidate keys, etc.

### Differential Cryptanalysis

Differential cryptanalysis was introduced by Biham and Shamir [52, 85–88, 390][10], but see also [348]. A differential attack is, in the language of the previous section, an order-2 statistical attack whose distinguisher detects the presence of high-probability differentials. A differential is defined in Definition 2.12 below.

**Definition 2.12.** *Let $f : \{0,1\}^m \to \{0,1\}^n$ be a vectorial boolean function. A* differential *for $f$ is a pair $(\alpha, \beta) \in \{0,1\}^m \times \{0,1\}^n$ such that*

$$f(x \oplus \alpha) \oplus f(x) = \beta$$

*may occur for some $x \in \{0,1\}^m$. The differential $(\alpha, \beta)$ may also be represented as $\alpha \xrightarrow{f} \beta$.*

In Definition 2.12 we may call $\alpha$ the *input difference* or $\Delta x$, and $\beta$ the *output difference*. A differential $(\alpha, \beta)$ may also not occur for any $x$, in which case we call $(\alpha, \beta)$ an impossible differential. The difference need not be with respect to exclusive-or ($\oplus$): it can be with respect to any group operation, such as addition modulo $2^n$ [63], multiplication modulo $2^n$ [108], multiplication modulo $2^n - 1$ [152], multiplication in $\mathbb{F}_2[x]/(x^n - 1)$ [261], etc. In such cases we make explicit the group operation in question; a differential with respect to the difference operation $g$ will be named a $g$-differential.

The most important property of a differential is how likely it is to occur.

**Definition 2.13.** *Let $f : \{0,1\}^m \to \{0,1\}^n$ be a vectorial boolean function, and $(\alpha, \beta)$ a differential for $f$. The* differential probability *of $(\alpha, \beta)$ is given by*

$$\Pr_{x \leftarrow\$ \{0,1\}^m}[f(x \oplus \alpha) \oplus f(x) = \beta]. \tag{2.5}$$

*If a differential has probability $p$, we may represent this as $\alpha \xrightarrow[p]{f} \beta$. Furthermore, we call $-\log_2 p$ the* weight *of the differential.*

If the probability (2.5) is considerably higher than $2^{1-m}$, the expected probability for a random function or permutation [149], a differential attack is likely to be useful to break the cipher. In particular, if the probability of the differential is $p$, the number of plaintext-ciphertext pairs to mount a differential attack is proportional to $O(1/p)$ [101].

In differential cryptanalysis, the first and often main challenge is to identify differentials with the highest possible probability. The obvious way is to exhaustively

[10] It turns out that this technique was already known by the designers of the DES as the "tickle attack" [132, 133, 190].

generate a table—the differential distribution table—containing the frequencies of every $(\alpha, \beta)$ for all possible inputs. This, however, requires $O(2^{2m})$ time and $O(m2^{m+n})$ space for an $\{0,1\}^m \to \{0,1\}^n$ function, which makes it impractical except for very small block sizes.

A more practical solution is to split the cipher into smaller analyzable components, identify high-probability differentials in those easier to analyze individual components, and then link them together. This leads to so-called differential trails.

**Definition 2.14.** *Let $f_0, \ldots, f_{r-1}$ be vectorial boolean functions from $\{0,1\}^n$ to $\{0,1\}^n$. The sequence of differentials $(\alpha_0, \ldots, \alpha_r)$*

$$\alpha_0 \xrightarrow{f_0} \alpha_1 \ldots \xrightarrow{f_{r-1}} \alpha_r$$

*is called a differential trail (or path, or characteristic) for $f_{r-1} \circ \cdots \circ f_0$.*

For example, consider a 3-round blockcipher

$$E_{k_0,k_1,k_2} = E_{k_2} \circ E_{k_1} \circ E_{k_0}\,.$$

Suppose we find a differential $(\alpha_0, \beta_0)$ with probability $p_0$ for $E_{k_0}$, $(\beta_0, \alpha_1)$ with probability $p_1$ for $E_{k_1}$, and $(\alpha_1, \beta_2)$ with probability $p_2$ for $E_{k_2}$. Then $\alpha_0 \xrightarrow[p_0]{E_{k_0}} \beta_0 \xrightarrow[p_1]{E_{k_1}} \alpha_1 \xrightarrow[p_2]{E_{k_2}} \beta_2$ is a *differential trail, path, or characteristic* for $E_k$. Furthermore, if $E_{k_i}$ is a Markov cipher [287, 367, 454], that is, if

$$\Pr_{k_i}\left[E_{k_i}(x \oplus \alpha) \oplus E_{k_i}(x) = \beta\right] = \Pr_{k_i,x}\left[E_{k_i}(x \oplus \alpha) \oplus E_{k_i}(x) = \beta\right]$$

and $k_0$, $k_1$, and $k_2$ are independent subkeys then the probability of the trail is $p_0 p_1 p_2$, the product of the probabilities of each individual differential. If they are not independent then the probability might deviate, but it often still is a reasonable approximation.

We also note that in the above example, there might exist more than one differential trail for the differential $\alpha_0 \xrightarrow{E_{k_0,k_1,k_2}} \beta_2$. In fact, there often is. Typically the probability of such a differential is dominated by one particularly strong trail, but this need not be the case. When many differential trails add up for the same differential, making its probability higher than it would be expected, we call this differential clustering or the differential effect. In some cases the probability gap between the best trail and its differential probability can be considerable [17, 219].

Besides key-recovery, differential attacks have been very successful in finding hash function collisions. If a compression function F has a high-probability differential of the form $(\alpha, 0)$ it immediately yields a collision. This is called a vanishing differential. Useful vanishing differentials are vanishingly rare, however. A more common strategy

is to mount multi-block collisions: the first message block is used to introduce a difference, and the subsequent blocks cancel it. Consider a simple 2-block hash function

$$\mathsf{H}(x_0, x_1) := \mathsf{F}(x_1, \mathsf{F}(x_0, 0)),$$

where $\mathsf{F} : \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^n$ is a compression function. Suppose there is a high-probability differential $((\alpha_1, \alpha_2), 0)$ with probability $p_0$ for $\mathsf{F}$. Because the second argument of the first $\mathsf{F}$ call is fixed, this differential cannot be used directly to cause a collision. However, suppose we have another differential of the form $((\alpha_0, 0), \alpha_2)$ with probability $p_1$. Then $(x_0, x_1)$ and $(x_0 \oplus \alpha_0, x_1 \oplus \alpha_1)$ are likely to collide. To see this, observe that

$$\mathsf{F}(x_0 \oplus \alpha_0, 0) = \mathsf{F}(x_0, 0) \oplus \alpha_2$$

will happen with roughly $1/p_0$ trials. Then

$$\mathsf{F}(x_1 \oplus \alpha_1, \mathsf{F}(x_0, 0) \oplus \alpha_2) = \mathsf{F}(x_1, \mathsf{F}(x_0, 0))$$

with another $1/p_1$ trials. Heuristically, after $1/p_0 + 1/p_1$ attempts a collision is found.

The same sort of techniques can also be used against message authentication codes to create forgeries. Hash functions are unkeyed, however, which allows for further optimizations. A key observation is that differential probabilities for a particular differential are taken over random choices of input. But clever choices of the inputs themselves, i.e., $x_0$ and $x_1$ in the above example, can greatly boost probabilities to much better than $p_0$ and $p_1$. These are known as *message modification techniques* and they were instrumental in bringing MD5 and SHA-1 collisions to fruition [432, 465–467].

### Linear Cryptanalysis

Linear cryptanalysis was discovered by Matsui when analyzing the FEAL [338, 339, 424] blockcipher and later the DES [322, 324, 325]. However the basic principle of linear cryptanalysis was already present in earlier work [419, 437] and, based on its design principles, known to the SKS V/1 East Germany cipher designers in the 1970s [139, 140]. Linear cryptanalysis only looks at the relationship between an input and its respective output. Thus it is a distinguisher of order 1.

While differential cryptanalysis made use of differentials, linear cryptanalysis exploits linear approximations.

**Definition 2.15.** *Let $f : \{0,1\}^m \to \{0,1\}^n$ be a vectorial boolean function. A linear approximation is a pair $(\alpha, \beta) : \{0,1\}^m \to \{0,1\}^n$ such that*

$$\alpha \cdot x \oplus \beta \cdot f(x),$$

*where $\cdot$ indicates $\mathbb{F}_2$ inner product, holds with probability $\frac{1}{2} + \epsilon$, $\epsilon \neq 0$.*

The value $\epsilon$ in Definition 2.15 is denoted the *bias* of the linear approximation. It is analogous to the concept of differential probability. A cleaner concept, introduced by Daemen et al. [145], is correlation.

**Definition 2.16.** *Let $f : \{0,1\}^m \to \{0,1\}^n$ be a vectorial boolean function, and $(\alpha, \beta)$ a linear approximation. The correlation $C_F$ of $(\alpha, \beta)$ is given by*

$$C_f(\alpha, \beta) = 2\left(\Pr_{x \leftarrow\$\{0,1\}^m}[\alpha \cdot x = \beta \cdot f(x)]\right) - 1\,.$$

*We denote a linear approximation with correlation $p$ as $\alpha \xrightarrow[p]{f} \beta$, and its* weight *as* $-\log_2 |p|$.

Correlation ranges from $-1$, where $\alpha \cdot x$ is the negation of $\beta \cdot f(x)$, to 1, where $\alpha \cdot x = \beta \cdot f(x)$. Sometimes determining the exact correlation is useful to directly recover key bits. But more often, when evaluating the resistance of a cipher, only the magnitude of this value matters. In particular, the *squared correlation* or *linear potential* is particularly useful as a metric, since $C(\alpha, \beta)^{-2}$ is a good approximation of the data requirements to identify a correlation $C(\alpha, \beta)$.

Daemen et al. [145] and independently Chabaud and Vaudenay [119] related the correlation to the Walsh-Hadamard transform of the target function. In particular, for input mask $\alpha$ and output mask $\beta$ one has

$$C_f(\alpha, \beta) = \sum_x -1^{\beta \cdot f(x)} -1^{\alpha \cdot x} = \sum_x -1^{\alpha \cdot x \oplus \beta \cdot f(x)}\,, \tag{2.6}$$

which is exactly the Walsh-Hadamard transform for the vectorial boolean function $f(x)$, or alternatively the Fourier-Hadamard transform of the indicator function $\mathbf{1}_{f(x)=y}$ [118]. Using the Fast Hadamard Transform it is possible to compute the correlation for every linear mask in $O((m+n)2^{m+n})$ space and time, which is faster than the naïve $O(2^{3n})$ time algorithm. The interpretation of linear cryptanalysis as exploiting large Fourier coefficients of the cipher also enables its generalization to other, non-binary, domains [34].

Much like in differential cryptanalysis, finding the specific linear masks with the highest correlation is nontrivial for a large enough block size, and is most easily done by breaking up the cipher into simpler components and then stitching them together.

**Definition 2.17.** *Let $f_0, \ldots, f_{r-1}$ be vectorial boolean functions from $\{0,1\}^n$ to $\{0,1\}^n$. The sequence of linear approximations $(\alpha_0, \ldots, \alpha_r)$*

$$\alpha_0 \xrightarrow{f_0} \alpha_1 \ldots \xrightarrow{f_{r-1}} \alpha_r$$

*is called a linear trail (or path, or characteristic) for $f_{r-1} \circ \cdots \circ f_0$.*

Analogously to the differential case, given a linear trail $(\alpha_0, \ldots, \alpha_r)$ we call $(\alpha_0, \alpha_r)$ a *linear hull* [366]. And again, under suitable assumptions of independence between each component (i.e., the Markov assumption [287, 367, 454]), the correlation of the linear trail is approximately the product of each individual correlation. Note, however, that because correlations are signed, adding them up may cause cancellation. That is, if there are multiple trails belonging to the same linear hull, the correlation of the linear hull as a whole may not be higher than the correlation of the trail. There is some debate around the power, or even existence, of the linear hull effect [23, 349].

### Other Attacks

For completeness here we point out that there are numerous statistical attacks other than, or derived from, plain differential and linear cryptanalysis. Differential cryptanalysis itself has many variants:

**Truncated differentials [270]** Whereas differential cryptanalysis deals with full block values $(\alpha, \beta)$, truncated differentials may only cover subsets or equivalence classes of the input or output blocks.

**Impossible differentials [78–80, 269]** Instead of exploiting high-probability differentials, the attack exploits differentials with probability 0.

**Higher-order differentials [270, 285]** Higher-order differential attacks reinterpret a difference $f(x+\alpha)-f(x)$ as a partial derivative at a point $\alpha$, and naturally extend it to more points $\alpha_0, \ldots, \alpha_{d-1}$. They are particularly dangerous for ciphers with low algebraic degree.

**Differential-linear attacks [82, 83, 102, 290, 303]** combine a truncated (higher-order) differential with probability $p$ with a linear trail with correlation $q$ to mount a distinguisher of complexity approximately $p^{-1}q^{-2}$.

**Boomerang [462] and rectangle attacks [84]** Adaptive chosen-ciphertext attack of order 4; distinguishes 4-tuples of plaintexts and ciphertexts with prescribed differences.

**Rotational [261] and rotational-xor cryptanalysis [22]** Differential cryptanalysis with respect to the rotation, or rotation followed by an xor, operations.

**Cube [24, 167, 459], Integral [272], Square [146], Saturation [310] attacks** Slightly different ways to exploit higher-order differentials.

**Multiple differential and polytopic cryptanalysis [99, 100, 439]** When multiple high-probability differentials are used together in the same distinguisher.

Likewise, linear cryptanalysis has several variants:

**Partitioning [223] and $\chi^2$ [451] cryptanalysis** Given projection functions $\rho, \rho'$, not necessarily linear, from $\{0,1\}^n \to \{0,1\}^m$, distinguish $(\rho(x), \rho'(E_k(x)))$ from random using a statistical test like the $\chi^2$ method.

**Zero-correlation [107]** When a linear trail has zero correlation; analogous to impossible differential cryptanalysis.

**Multidimensional linear cryptanalysis [229]** Use linear subspaces of multiple linear approximations.

Given the many similarities between most of these attacks, Wagner [463], Phan [383, 384], and Vaudenay [452] have independently attempted to unify most blockcipher cryptanalysis methods into a common framework.

### 2.4.4 Algebraic Attacks

The basic idea of algebraic attacks goes back to Shannon [422]. Shannon observed that a block cipher taking $n$-bit inputs and $m$-bit keys can be written as a multivariate polynomial system of the form

$$\begin{aligned} y_0 &= f_0(x_0, \ldots, x_{n-1}, k_0, \ldots, k_{m-1}) \\ y_1 &= f_1(x_0, \ldots, x_{n-1}, k_0, \ldots, k_{m-1}) \\ &\ldots \\ y_{n-2} &= f_{n-2}(x_0, \ldots, x_{n-1}, k_0, \ldots, k_{m-1}) \\ y_{n-1} &= f_{n-1}(x_0, \ldots, x_{n-1}, k_0, \ldots, k_{m-1}), \end{aligned}$$

over the boolean polynomial ring, i.e., $\mathbb{F}_2[x_0, \ldots, x_{n-1}, k_0, \ldots, k_{m-1}]/(x_0^2 - x_0, \ldots, x_{n-1}^2 - x_{n-1}, k_0^2 - k_0, \ldots, k_{m-1}^2 - k_{m-1})$. Given some known plaintext-ciphertext pairs, the $x_i$ and $y_i$ variables can be replaced by their concrete values, and solving the resulting $m$-variable system would find the key. Shannon goes on to say that a good cipher should be as hard to solve as a random such system of equations, and should heavily involve the $k_i$ variables[11].

For relatively small systems of low degree, exhaustive search can be the most effective way to solve them [109, 110]. Another option is linearization: replace each variable product by a new variable, resulting in a linear system of $\sum_{i=0}^{d-1} \binom{m}{i+1}$ variables, $d$ being the algebraic degree of the system. This linear system can be solved in time $\left(\sum_{i=0}^{d-1} \binom{m}{i+1}\right)^{\omega} \approx O(m^{\omega d})$, where $\omega < 2.37286$ [9] is the complexity of matrix multiplication. More advanced attacks based on this principle are the relinearization [266], extended linearization [128, 136], and extended sparse linearization attacks [138].

[11] This heavy involvement of the $k_i$ variables is what is commonly called *confusion*.

Another way to approach the problem is by computing its Gröbner basis. Faugère's $F_4$ [187] and $F_5$ [187] are the best known algorithms for this. The average complexity of such algorithms is not very well understood, and is dependent on a property of the system called "degree of regularity," which is difficult to evaluate for concrete systems [166, 171]. For a degree of regularity $D_r$, $F_5$ has a complexity of $\binom{m+D_r}{D_r}^{\omega}$; but for a random system the complexity is worse than exhaustive search [39].

The system may also be converted into a satisfiability (SAT) problem, for which good off-the-shelf solvers are available [38].

Because it is fairly intuitive that a cipher with a low algebraic degree is not very strong, few ciphers have been broken by algebraic cryptanalysis. Exceptions exist, however:

- The Keeloq cipher was broken by algebraic means using a SAT solver [135];
- This Hitag2 stream cipher was also broken by SAT solving [137];
- Collisions on reduced-round SHA-1 were made slightly more efficient by resorting to the solution of multivariate polynomial systems [436].

# Chapter 3

# Tyche: a fast and small nonlinear pseudorandom number generator

Pseudorandom numbers are often used for testing, simulation and even aesthetic purposes. They are an integral part of Monte Carlo methods, genetic and evolutionary algorithms, and are extensively used in noise generation for computer graphics.

Monte Carlo methods were first used for computing purposes by Ulam and von Neumann, while attempting to solve hard problems in particle physics [335]. Monte Carlo methods consist of iterated random sampling of many inputs in some probability distribution, followed by later processing. Given enough inputs, it is possible to obtain an approximate solution with a reasonable degree of certainty. This is particularly useful for problems with many degrees of freedom, where analytical or exact methods would be far too inefficient. Monte Carlo methods are not, however, very computationally efficient—typically, to reduce the error margin by half, one has to quadruple the amount of sampled inputs [204]. Today, Monte Carlo methods are used in the most various fields, such as particle physics, weather forecasting, financial analysis, operations research, etc.

Current general-purpose processors typically have 2 or 4 cores. Graphics processing units have tens to hundreds [298]; future architectures are slated to scale up to hundreds and thousands of cores [449]. This development entails some consequences: silicon real estate is limited, and the increase in processing units decreases the total fast memory available on-chip. Thus, it becomes an interesting problem to design a random number generator that can take advantage of massively parallel architectures and still remain small, fast and of high quality. With these goals in mind, we introduce Tyche, a fast and small-state pseudorandom number generator especially suited for current parallel and SIMD architectures. The iteration function of Tyche, derived from the stream cipher ChaCha's *quarter-round* function [54], is nonlinear and fast; it uses a very small amount of state (4 32-bit registers) and, yet, has a very large expected period.

In Section 3.1 we review the state of the art in theory and practice of random number generation. In Section 3.2 we describe the Tyche function. In Section 3.3 we provide an analysis of several important features of the algorithm, such as the expected period, statistical quality and parallelism. We then introduce several variants of Tyche with different period-speed-parallelism tradeoffs Section 3.4. In Section 3.5, we experimentally evaluate and compare Tyche. We conclude with Section 3.6.

## 3.1 Related Work

There is an enormous body of work in the literature regarding pseudorandom number generation. One of the first and most popular methods to generate pseudorandom numbers in digital computers was Lehmer's linear congruential method, consisting of a linear recurrence modulo some integer $m$. Since then, researchers have proposed numerous other linear algorithms, most notably lagged Fibonacci generators, linear feedback shift registers, xorshift generators and the Mersenne Twister [275, 319, 326]. The statistical properties of linear generators are well known and widely studied; several empirical tests are described in [275, Chapter 3]. One of the drawbacks of such linear generators, in particular linear congruential generators, is their very regular lattice structure [275, Section 3.3.4]. This causes the usable amount of numbers in a simulation to be far less than the full period of the generator [381].

Nonlinear pseudorandom generators, like the *Inversive congruential generator* and *Blum Blum Shub* [103, 227], avoid the drawbacks of linearity. However, nonlinear generators often require more complex arithmetic than linear ones and have smaller periods, rendering them impractical for simulations.

The generation of pseudorandom numbers in parallel environments is also a well studied subject [111, 180, 414]. The main problem is to enable multiple concurrent threads of execution to get random numbers in parallel. Two main solutions exist for this problem: *parametrization* and *cycle splitting*. Parametrization consists in creating a slightly different full period generator for each instance; this can be done by, for example, changing the iteration function itself. Cycle splitting takes a full period sequence and splits it into a number of subsequences, each used within an instance. Cycle splitting is often used in linear congruential generators, since it is easy to leap to any arbitrary position in the stream. Other generators, such as the Mersenne Twister, rely on different initial parameters (i.e., parametrization) to differentiate between threads.

In the case of GPUs and other vector processors, we face additional restrictions, because the amount of fast memory per core is quite limited, thus restricting the internal state we can use. Furthermore, GPUs lack some important instructions, such as native integer multiplication and/or division, leading to a large slowdown for

some popular random number generators. Still, linear congruential generators have been studied in GPUs [289].

There have been some initial attempts to adapt cryptographic functions for fast GPU random number generation. Tzeng and Wei [446] used the MD5 hash function and reduced-round versions thereof to generate high-quality random numbers. Zafar and Olano [477] used the smaller and faster 8-round TEA block cipher to generate high-quality random numbers for Perlin noise generation.

## 3.2 Tyche Specification

This section will present Tyche. In the following, all values are assumed to be 32 bits long, unless otherwise noted, and $\boxplus$ (resp. $\boxminus$) represents addition (resp. subtraction) modulo $2^{32}$.

### 3.2.1 Initialization

The state of Tyche is composed of 4 32-bit words, which we will call $a$, $b$, $c$ and $d$. Tyche, when initialized, takes as inputs a 64-bit word $s$ and a 32-bit word $i$. Algorithm 3.1 describes the operations performed during initialization.

**Algorithm 3.1:** Tyche's initialization.

**Input:** $s \in \{0,1\}^{64}, i \in \{0,1\}^{32}$
$a \leftarrow \lfloor s/2^{32} \rfloor$ ;
$b \leftarrow s \bmod 2^{32}$;
$c \leftarrow 2654435769$;
$d \leftarrow 1367130551 \oplus i$;
**for** $r \leftarrow 0$ **to** 19 **do**
  $(a,b,c,d) = \mathsf{MIX}(a,b,c,d)$;
**end**
**return** $(a,b,c,d)$

The MIX function called in Algorithm 3.1 is used here to derive the initial state; it is described further in Algorithm 3.3. The constants used in the initialization, 2654435769 and 1367130551, were chosen to be $\lfloor 2^{32}/\varphi \rfloor$ and $\lfloor 2^{32}/\pi \rfloor$, where $\varphi$ is the golden ratio and $\pi$ is the well-known constant. Their purpose is to prevent a starting internal state of $(0,0,0,0)$.

### 3.2.2 The core algorithm

Once its internal state is initialized, Tyche is quite simple. It calls the MIX function once and returns the second word of the internal state, $b$, as shown in Algorithm 3.2 and Figure 3.1.

**Algorithm 3.2:** Tyche.

**Input:** $(a, b, c, d) \in \{0, 1\}^{32}$
$(a, b, c, d) = \mathsf{MIX}(a, b, c, d)$;
**return** $b$

### 3.2.3 The MIX function

The MIX function, used both in initialization and state update, is derived directly from the *quarter-round* function of the ChaCha stream cipher [54]. As described in Algorithm 3.3, it works on 4 32-bit words and uses only addition modulo $2^{32}$, xor and bitwise rotations.

**Algorithm 3.3:** MIX

**Input:** $(a, b, c, d) \in \{0, 1\}^{32}$
$a \leftarrow a \boxplus b$;
$d \leftarrow (d \oplus a) \lll 16$;
$c \leftarrow c \boxplus d$;
$b \leftarrow (b \oplus c) \lll 12$;
$a \leftarrow a \boxplus b$;
$d \leftarrow (d \oplus a) \lll 8$;
$c \leftarrow c \boxplus d$;
$b \leftarrow (b \oplus c) \lll 7$;
**return** $(a, b, c, d)$

This function is clearly invertible, as will be explored further in Section 3.4.1.

## 3.3 Analysis of Tyche

### 3.3.1 Design

We can find many different designs for random number generators. The design we propose here attempts to achieve high period, speed, and very low memory

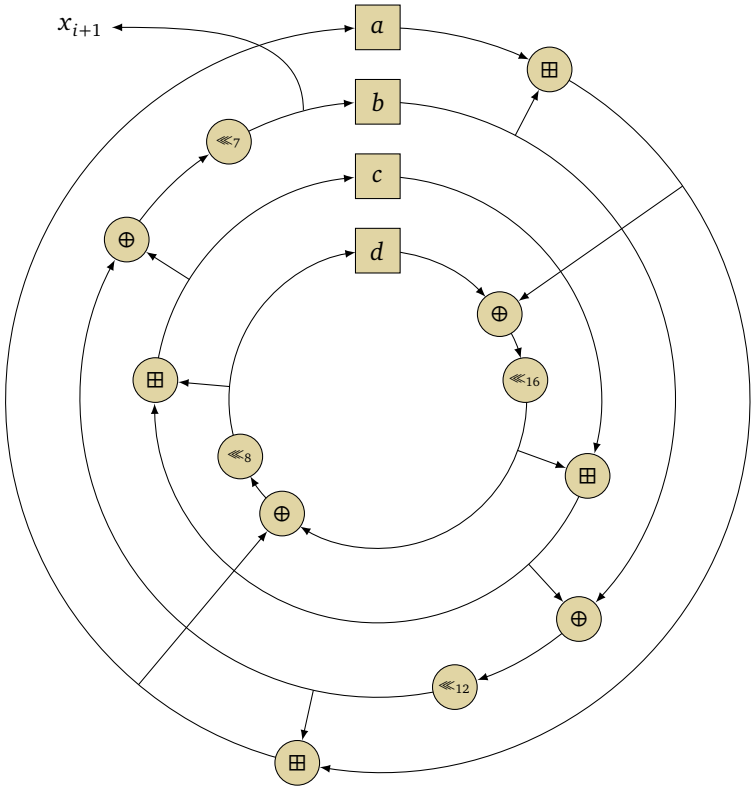



Figure 3.1: The Tyche core function MIX. Each Tyche iteration produces a 32-bit integer, $x_{i+1}$, extracted directly from the internal state.

consumption. One of the ways in which it achieves this is by using a very simple recursion:

$$x_{i+1} = f(x_i) \tag{3.1}$$

This requires no extra space other than the state's size and perhaps some overhead to execute $f$. One could use, e.g., a counter to ensure certain minimum period—this would evidently require more registers per state, which goes against our main objectives. A similar approach to ours has been used in the LEX [89] stream cipher, using the AES block cipher in Output Feedback mode (OFB) and extracting 4 bytes of the state per iteration.

Another crucial design choice concerns function $f$. Should it be linear? Most current random number generators are indeed linear: LCG, xorshift, LFSR constructions, etc. These functions have the advantage of being very simple and easily analyzed. However, linear random number generators tend to have highly regular outputs: their outputs lie on simple lattice structures of some dimension. This makes such generators unsuitable for some types of simulations and considerably reduces their

useful period [282]. Nonlinear generators generally do not have this problem [204]. Moreover, despite being very simple, linear generators may not be very fast. Linear congruential generators and their derivatives require multiplications and (possibly implicit) modular reductions. Unfortunately, these operations are not present in every instruction set and can be hard to implement otherwise.

One could then simply search for a good nonlinear random number generator. However, nonlinear generators for simulation purposes are hard to find, and generally much slower than their linear counterparts. Indeed, one could simply use a cryptographic stream cipher as a random number generator. That would, however, be several times slower and would require a much larger state. Indeed, even $\text{TEA}_8$ as described in [477] requires 136 instructions per 64 bits of random output, while MIX only requires 12 instructions per 32 bits of random output.

In light of these reasons, we chose our function to be nonlinear and to use exclusively instructions available in almost every chip—addition, xor, bit rotations[1]. The overlap of 32-bit addition and xor creates a high amount of nonlinearity and simultaneously allows for very fast implementations, owing to the simplicity of such instructions.

### 3.3.2 Period

The MIX function, used to update the internal state, is trivially invertible. Thus it is a permutation, with only one possible state before each other state. How does this affect the expected period? Were the MIX function irreversible, it would behave like a random mapping—in that case, the period would be about $2^{n/2}$ for an $n$-bit state [195]. In our case, the expected period is the average cycle length of a random element in a random permutation: $(2^n + 1)/2 \approx 2^{n-1}$ for an $n$-bit state [274, §1.3.3].

But even random permutations do have short cycles. Indeed, we can trivially find one cycle of length 1 in the MIX function: MIX (0,0,0,0) = (0,0,0,0). This is in fact its only fixed point [28]. However, if using the initialization described in Section 3.2.1, this state will never be reached. It is also extremely unlikely to reach a very short cycle—the probability of reaching a cycle of length $m$ is $1/2^n$; the probability of reaching a cycle of length $m$ *or less* is $\sum_i^m 1/2^n = m/2^n$ [121]. In our case, the chance of reaching a state with period less than or equal to $2^{32}$ is roughly $2^{-96}$.

### 3.3.3 Parallelization

Our algorithm is trivial to use in parallel environments. When initializing a state (using Algorithm 3.1 or Algorithm 3.4), each computing unit (e.g., thread, vector

[1] While many chips do not have bit rotations natively, they are trivially achievable with simple logical instructions such as shifts and xor.

element, core) uses the same 64-bit seed, but its own index in the computation (the `idx` argument of Algorithm 3.1). We chose a 64-bit seed to avoid collisions; since seeds are often chosen at random, it would only require about $2^{16}$ initializations for a better than 50% chance to rerun a simulation using the same seed if one used 32-bit seeds. This would be uncomfortably probable.

What about overlaps? Parallel streams will surely overlap eventually, given that the function is cyclic and reversible. This is as bad as a small period in a random number generator. To find out how fast streams overlap, consider a simple case: $s$ streams outputting a single value each. Given that each stream begins at an arbitrary state out of $n$ possible states, the probability of an overlap (i.e., a collision) would be given by the birthday paradox:

$$1 - \frac{(n)_s}{n^s} \tag{3.2}$$

This is, however, a simplified example; what we want to know is the likelihood that, given $s$ streams and a function $f$ that is a random permutation, no stream will meet the starting point of any other in fewer than $d$ calls to $f$. This can be seen as a generalization of the birthday problem, and was first solved by Naus [354]. The probability that at least one out of $s$ streams overlaps in less than $d$ steps in a cycle of length $m$ is given by

$$1 - \frac{(m - sd + s - 1)!}{(m - sd)! m^{s-1}} \tag{3.3}$$

It is possible to upper bound the probability of collision with a simpler expression. The basic idea is that stream $i$ cannot overlap with itself[2], but only with any of the values of the other $s - 1$ streams. In other words, there can only be overlap among any of the $\binom{s}{2}$ pairs of streams. If each stream has length at most $d$, we have that the probability of at least one overlap is at most

$$\frac{d\binom{s}{2}}{m} \leq \frac{ds^2}{m}. \tag{3.4}$$

In our particular case, $m$ is in average $2^{127}$; $s$ should be no more than $2^{16}$; $d$ should be a large enough distance to make the generator useful—we choose $2^{64}$ here as an example minimum requirement. Thus, $2^{16}$ parallel streams each producing $2^{64}$ numbers will overlap with a probability of roughly $2^{-32}$. Conversely, when running $2^{16}$ parallel streams, an overlap is not expected until after about $2^{64}/2^{-32} = 2^{96}$ iterations.

[2]Unless it enters a cycle, but that is not assumed to occur here.

**Remark.** *The above result has been rediscovered, sometimes in a less exact form, many times. Vigna [460] reported several of these less accurate rediscoveries, and rediscovered (3.3) himself.*

## 3.4 Variants

### 3.4.1 Tyche-i

One issue with the construction described in the previous section is that it is completely sequential. Each instruction of the MIX function directly depends on the immediately preceding one. This does not take any advantage of modern superscalar CPU machinery. Thus, we propose a variant of Tyche, which we call Tyche-i, able to take advantage of pipelined processors. Tyche-i is presented in Algorithm 3.4, Algorithm 3.5, and Algorithm 3.6.

**Algorithm 3.4:** Tyche-i initialization.

**Input:** $s \in \{0,1\}^{64}, i \in \{0,1\}^{32}$
$a \leftarrow \lfloor s/2^{32} \rfloor$ ;
$b \leftarrow s \bmod 2^{32}$;
$c \leftarrow 2654435769$;
$d \leftarrow 1367130551 \oplus i$;
**for** $r \leftarrow 0$ **to** 19 **do**
    $(a,b,c,d) = \mathsf{MIX\text{-}i}(a,b,c,d)$;
**end**
**return** $(a,b,c,d)$

**Algorithm 3.5:** Tyche-i

**Input:** $(a,b,c,d) \in \{0,1\}^{32}$
$(a,b,c,d) = \mathsf{MIX\text{-}i}(a,b,c,d)$;
**return** $a$

The main difference between Tyche and Tyche-i is the MIX-i function. The MIX-i function is simply the *inverse function* of Tyche's MIX. Unlike MIX, MIX-i allows for 2 simultaneous executing operations at any given time, which is a better suit to superscalar processors than MIX is. The downside, however, is that MIX-i diffuses bits slower than MIX does: for 1-bit differences in the internal state, 1 MIX call averages 26 bit flipped bits, while MIX-i averages 8.

**Algorithm 3.6:** MIX-i

**Input:** $(a, b, c, d) \in \{0, 1\}^{32}$
$b \leftarrow (b \ggg 7) \oplus c; c \leftarrow c \boxminus d;$
$d \leftarrow (d \ggg 8) \oplus a; a \leftarrow a \boxminus b;$
$b \leftarrow (b \ggg 12) \oplus c; c \leftarrow c \boxminus d;$
$d \leftarrow (d \ggg 16) \oplus a; a \leftarrow a \boxminus b;$
**return** $(a, b, c, d)$

### 3.4.2 Tweaking Tyche with a Counter

Although the Tyche and Tyche-i generators presented in the previous sections show great performance across many architectures, due to their use of simple 32-bit instructions, they have several drawbacks:

**No provable period** Treating the core permutation MIX as a random permutation allows us to estimate the expected period of a sequence to be roughly $2^{127}$. This, however, says nothing about the actual cycle structure of Tyche, and unlikely as it may be, there may be some hidden pitfalls in this generator.

**No random access** While Tyche provides some higher level parallelism support by defining different stream starting points, it is impossible to jump ahead inside a single stream. This may be inconvenient in some situations.

We propose a tweak, named Tyche-CTR-$R$, that changes the mode of operation of Tyche. Once the initial state is set up, the least significant 64 bits are used as a counter incremented by the odd constant 5871781008561895865[3], while the most significant 64 bits remain constant, serving as identifier (a *nonce*) of the current stream. Then, this state is processed $R$ times by the MIX function, and the least significant word is returned. Algorithm 3.7 describes Tyche-CTR-$R$.

**Algorithm 3.7:** Tyche-CTR-$R$

**Input:** $(a, b, c, d) \in \{0, 1\}^{32}$
$(b, a) \leftarrow (a + 2^{32}b) + 5871781008561895865;$
$(a', b', c', d') \leftarrow (a, b, c, d);$
**for** $i \leftarrow 0$ **to** $R - 1$ **do**
    $(a', b', c', d') \leftarrow \mathsf{MIX}(a', b', c', d');$
**end**
**return** $a'$

[3] Counters that add an odd constant different from 1 are often known as Weyl generators.

Our experiments suggest that 5 rounds, i.e., Tyche-CTR-5, are sufficient to achieve enough diffusion to pass known statistical tests. It is easy to see that the period of Tyche-CTR-$R$ is $2^{64}$ and it is easy to jump ahead arbitrarily within a stream, by adding an appropriate multiple of the constant used in Algorithm 3.7. One should notice that Tyche-CTR-5 provides $2^{64}$ distinct streams with a guaranteed period of $2^{64}$, and still enables further tweaks to the lengths of the counter and nonce.

### 3.4.3 Tyche as a counter-dependent generator

The tweak presented in the previous section was fairly aggressive: instead of one MIX call per iteration, we now require $R \geq 5$ calls to achieve the same effect. This is a massive slowdown, even though it does enable some desirable properties, and the higher latency may be hidden by computing several values in parallel.

We propose in this section a compromise: a $2^{32}$ guaranteed minimum period, $2^{159}$ average period, and 160 bits of state. The approach we follow is known as *counter-dependent* generators [420], and is pictured in Algorithm 3.8.

**Algorithm 3.8:** Tyche-CD-32

**Input:** $(a, b, c, d, e) \in \{0, 1\}^{32}$
$e \leftarrow e \boxplus (e^2 \vee 5) \bmod 2^{32}$;
$(a, b, c, d) \leftarrow \mathsf{MIX}(a, b, c, d)$;
**return** $b + e$

Since this tweak does not enable random stream access, we opted to use the T-function $x + (x^2 \vee 5) \pmod{2^n}$, proved by Klimov and Shamir to be invertible and single-cycle [268]. This function is executed in parallel with MIX, and is not expected to significantly slow down the generator. Additionally, the greater complexity of this function provides some more diffusion than a simpler counter.

## 3.5 Experimental Evaluation

### 3.5.1 Performance

We implemented and compared the performance of Tyche and its variants against two other similarly-sized generators: the XORWOW generator found in the CURAND library [143], and a 128-bit linear congruential generator optimized for speed. Table 3.1 shows the average number of cycles per iteration of the aforementioned generators for the generation of $2^{16}$ random integers. The timings were obtained on an Intel Core-i7 “Sandy Bridge” processor, with both Turbo Boost and hyperthreading

Table 3.1: Sandy Bridge timings for several Tyche variants, along with other competing generators.

| Generator | State bits | Cycles/Word | Avg. period | Min. period | Jump ahead |
|---|---|---|---|---|---|
| $TEA_8$ [476] | 128 | 49 | $2^{64}$ | $2^{64}$ | Yes |
| LCG-128 | 127 | 32 | $2^{127}-1$ | $2^{127}-1$ | Yes |
| MT19937 [326] | 19968 | 8 | $2^{19937}-1$ | $2^{19937}-1$ | Yes |
| XORWOW [143] | 192 | 7 | $2^{192}-2^{32}$ | $2^{192}-2^{32}$ | Yes |
| Tyche [357] | 128 | 12 | $\approx 2^{127}$ | 1 | No |
| Tyche-i [357] | 128 | 6 | $\approx 2^{127}$ | 1 | No |
| Tyche-CD-32 | 160 | 12 | $\approx 2^{159}$ | $2^{32}$ | No |
| Tyche-CTR-5 | 128 | 44 | $2^{64}$ | $2^{64}$ | Yes |

Table 3.2: Cycles per word on an Intel Skylake CPU for Tyche when running with various amounts of SIMD lanes.

| Lanes | Cycles per word |
|---|---|
| 1 | 12 |
| 4 | 3.93 |
| 8 | 1.97 |
| 16 | 1.21 |
| 24 | 0.98 |

disabled. As expected (cf. Section 3.4.1), Tyche-i is roughly twice as fast as Tyche on a processor with high instruction-level parallelism.

We also include a comparison with $TEA_8$, suggested by Zafar and Olano [477] for GPUs, which reveals to be markedly slower than any of the other choices for random generation. $TEA_8$ requires at least 136 instructions per 64-bit word, which is much higher than either Tyche, Tyche-i or XORWOW.

When taking advantage of SIMD, Tyche becomes much faster. Table 3.2 shows how performance changes when generation is parallelized using more than one instance, using the index feature, and implemented using AVX2 on an Intel Skylake CPU. Although AVX2 consists of 8-word vectors, it is possible to obtain an additional speedup by taking advantage of the roughly 3 independent vector instructions per cycle possible to execute on this microarchitecture. Ultimately, compared to the sequential implementation the vector code obtains a speedup of over 12.

### 3.5.2 Statistical quality tests

In order to assess the statistical quality of Tyche and its variants, we performed a rather exhaustive battery of tests. We employed the ENT and DIEHARD suites and the TestU01 "BigCrush" battery of tests [283, 317, 464]. Every test performed in both variants showed no statistical weaknesses.

Another aspect of Tyche is that it is based on the ChaCha stream cipher. ChaCha's "quarter-round" function is also employed, albeit slightly modified, in the BLAKE SHA-3 candidate [28] and BLAKE2 [32] (cf. Chapter 4). The "quarter-round" has been extensively analyzed in the context of these functions, but both functions are still regarded as secure [25, 26]. This increases our confidence in the quality of Tyche as a generator.

Finally, note that the XORWOW algorithm fails 3 tests in the "BigCrush" battery: CollisionOver ($t = 7$), SimpPoker ($r = 27$), and LinearComp ($r = 29$), the latter being a consequence of its (almost) linear nature.

## 3.6 Conclusion

In this chapter we presented and analyzed Tyche and several variants thereof, fast and small nonlinear pseudorandom generators based on ChaCha's quarter-round.

Tyche and Tyche-i use a very small amount of state that fits entirely into 4 32-bit registers. Our experiments show that Tyche and Tyche-i are much faster than the also nonlinear and cryptographic function-derived $TEA_8$, while exhibiting a large enough period for serious simulations with many parallel threads. On the other hand, when we compare Tyche and Tyche-i to the slightly faster (but almost linear) XORWOW algorithm, statistical tests (i.e., Big Crush) suggest that both Tyche and Tyche-i have better statistical properties.

We have also addressed some of Tyche's limitations with respect to provable period, by either transforming them into counter generators, or counter-dependent generators. These variants achieve guaranteed period at the cost of performance.

# Chapter 4

# BLAKE2: fast and parallel hashing

Soon after the groundbreaking attacks on MD5 and SHA-1 [465–467], there was some uncertainty about whether the existent SHA-2 standard would fall to the same attack strategies, seeing that they shared similar design principles. In response, the National Institute for Standards and Technology (NIST) held a competition to select a new hashing standard, that would run from 2008 to 2012. There were 64 initial submissions, and after 3 years there were 5 remaining finalists: BLAKE, Grøstl, JH, Keccak, and Skein. In late 2012, NIST announced the winner of the SHA-3 competition—the new hashing standard would be based on the Keccak hash function [125].

The SHA-3 competition succeeded in selecting a hash function that complements SHA-2 and is more efficient than SHA-2 in hardware [125]. But in software remained a demand for fast software hashing for applications such as integrity checking and deduplication in filesystems and cloud storage, host-based intrusion detection, version control systems, or secure boot schemes. These applications sometimes hash a few large messages, but more often a lot of short ones, and the performance of the hash directly affects the user experience.

Many systems use faster algorithms like MD5, SHA-1, or a custom function to meet their speed requirements, even though those functions may be insecure. MD5 is famously vulnerable to collision and length-extension attacks [175, 434], but it is 2.53 times as fast as SHA-256 on an Intel Ivy Bridge and 2.98 times as fast as SHA-256 on a Qualcomm Krait CPU.

Despite MD5's significant security flaws, it continues to be among the most widely-used algorithms for file identification and data integrity. To choose just a handful of examples, the OpenStack cloud storage system [427], the popular version control system Perforce, and the recent object storage system used internally in AOL [387] all rely on MD5 for data integrity. The venerable `md5sum` Unix tool remains one of the most widely-used tools for data integrity checking. The Sun/Oracle ZFS filesystem includes the option of using SHA-256 for data integrity, but the default configuration

is to instead use a non-cryptographic 256-bit checksum, for performance reasons.

Some SHA-3 finalists outperform SHA-2 in software: for example, on Ivy Bridge BLAKE-512 is 1.41 times as fast as SHA-512, and BLAKE-256 is 1.70 times as fast as SHA-256. BLAKE-512 reaches 5.76 cycles per byte, or approximately 579 MB per second, against 411 for SHA-512, on an Ivy Bridge CPU clocked at 3.5 GHz. Some other non-finalist SHA-3 submissions were competitive in speed with BLAKE and Skein, but these have been less analyzed and generally inspire less confidence (e.g., due to distinguishers on the compression function).

BLAKE thus appears to be a good candidate for fast software hashing. Its security was evaluated by NIST in the SHA-3 process as having a "very large security margin", and the cryptanalysis published on BLAKE was noted as having "a great deal of depth" (see Section 4.4).

But as observed by Preneel [389], its design "reflects the state of the art in October 2008"; since then, and after extensive cryptanalysis, we have a better understanding of what contributes to BLAKE's security and efficiency properties. We therefore introduce BLAKE2, an improved BLAKE with the following properties:

- Faster than MD5 on most 64-bit x86 and ARM platforms;
- 32% less RAM required than BLAKE;
- Direct support, with no additional overhead, of
  - Parallelism for many-times faster hashing on multicore or SIMD CPUs;
  - Tree hashing for incremental update or verification of large files;
  - Keying to instantiate a PRF or authenticator that is simpler and faster than HMAC;
  - Personalization strings for defining a unique hash function for each application;
- Minimal padding, faster and simpler to implement.

The rest of this chapter is structured as follows: Section 4.1 specifies BLAKE2, Section 4.2 delineates the rationale behind the changes to BLAKE, Section 4.3 discusses its efficiency on various platforms and reports preliminary benchmarks, and Section 4.4 discusses its security.

## 4.1 Specification of BLAKE2

The BLAKE2 family consists of two main algorithms:

- **BLAKE2b** is optimized for *64-bit platforms*—including NEON-enabled ARMs—and produces digests of any size between 1 and 64 bytes.
- **BLAKE2s** is optimized for *8- to 32-bit platforms*, and produces digests of any size between 1 and 32 bytes.

Both are designed to offer security similar to that of an ideal function producing digests of same length. Each one is portable to any CPU, but can be up to twice as fast when used on the CPU size for which it is optimized; for example, on a Tegra 2 (32-bit ARMv7-based SoC) BLAKE2s is expected to be about twice as fast as BLAKE2b, whereas on an AMD A10-5800K (64-bit, Piledriver microarchitecture), BLAKE2b is expected to be more than 1.5 times as fast as BLAKE2s.

### 4.1.1 BLAKE2b

BLAKE2b operates on 64-bit words and returns a hash value between 1 and 64 bytes. In the following we specify BLAKE2b's constants, keyed permutation, compression function, and iterative hashing mode.

#### Constants

BLAKE2b uses the following set of constants:

$$\begin{aligned} \mathrm{IV}_0 &= \texttt{6a09e667f3bcc908} & \mathrm{IV}_1 &= \texttt{bb67ae8584caa73b} \\ \mathrm{IV}_2 &= \texttt{3c6ef372fe94f82b} & \mathrm{IV}_3 &= \texttt{a54ff53a5f1d36f1} \\ \mathrm{IV}_4 &= \texttt{510e527fade682d1} & \mathrm{IV}_5 &= \texttt{9b05688c2b3e6c1f} \\ \mathrm{IV}_6 &= \texttt{1f83d9abfb41bd6b} & \mathrm{IV}_7 &= \texttt{5be0cd19137e2179} \end{aligned}$$

These constants are identical to the ones used in BLAKE-512 and SHA-512's IV. Their provenance is

$$\mathrm{IV}_i = \left\lfloor 2^{64} \sqrt{p_i} \right\rfloor \bmod 2^{64},$$

with $p_i$ being the $i$th prime number.

Furthermore, 10 permutations $\sigma_0, \ldots, \sigma_9$ are used in the keyed permutation and defined in Table 4.1.

#### Keyed Permutation

The most complex component of BLAKE2b is its internal keyed permutation, or blockcipher, E. It operates over a block of 16 64-bit words $v_0, \ldots, v_{15}$, and is also keyed by 16 64-bit words $m_0, \ldots, m_{15}$. It performs 12 rounds. At each round, the operation G is applied in parallel first to each column of $v$, viewed as a $4 \times 4$ matrix, and then to each diagonal. Figure 4.1 shows this operation visually, and Algorithm 4.1 specifies its mechanism.

Table 4.1: Permutations $\sigma_r$ of $\{0,\ldots,15\}$ used by the BLAKE2 functions.

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\sigma_0$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| $\sigma_1$ | 14 | 10 | 4 | 8 | 9 | 15 | 13 | 6 | 1 | 12 | 0 | 2 | 11 | 7 | 5 | 3 |
| $\sigma_2$ | 11 | 8 | 12 | 0 | 5 | 2 | 15 | 13 | 10 | 14 | 3 | 6 | 7 | 1 | 9 | 4 |
| $\sigma_3$ | 7 | 9 | 3 | 1 | 13 | 12 | 11 | 14 | 2 | 6 | 5 | 10 | 4 | 0 | 15 | 8 |
| $\sigma_4$ | 9 | 0 | 5 | 7 | 2 | 4 | 10 | 15 | 14 | 1 | 11 | 12 | 6 | 8 | 3 | 13 |
| $\sigma_5$ | 2 | 12 | 6 | 10 | 0 | 11 | 8 | 3 | 4 | 13 | 7 | 5 | 15 | 14 | 1 | 9 |
| $\sigma_6$ | 12 | 5 | 1 | 15 | 14 | 13 | 4 | 10 | 0 | 7 | 6 | 3 | 9 | 2 | 8 | 11 |
| $\sigma_7$ | 13 | 11 | 7 | 14 | 12 | 1 | 3 | 9 | 5 | 0 | 15 | 4 | 8 | 6 | 2 | 10 |
| $\sigma_8$ | 6 | 15 | 14 | 9 | 11 | 3 | 0 | 8 | 12 | 2 | 13 | 7 | 1 | 4 | 10 | 5 |
| $\sigma_9$ | 10 | 2 | 8 | 4 | 7 | 6 | 1 | 5 | 15 | 11 | 9 | 14 | 3 | 12 | 13 | 0 |

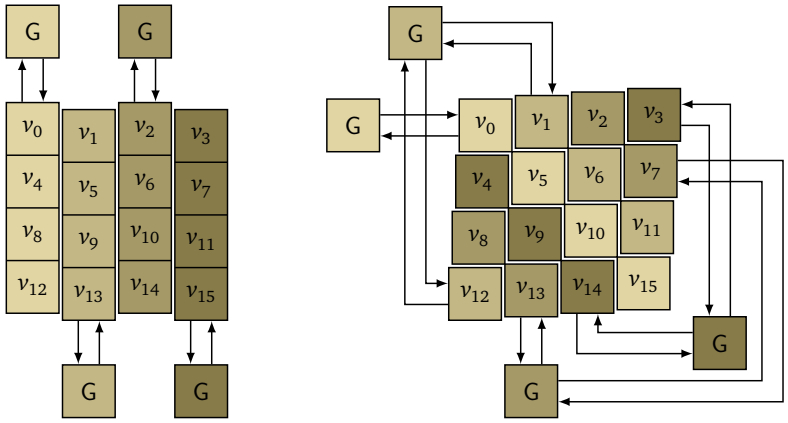



Figure 4.1: Column and diagonal steps of E.

**Algorithm 4.1:** BLAKE2b's E.

**Input:** $m \in \mathbb{Z}_{2^{64}}^{16}, v \in \mathbb{Z}_{2^{64}}^{16}$
**for** $r \leftarrow 0$ **to** 11 **do**
    $\mathsf{G}(v_0, v_4, v_8, v_{12}, m, r, 0)$;
    $\mathsf{G}(v_1, v_5, v_9, v_{13}, m, r, 1)$;
    $\mathsf{G}(v_2, v_6, v_{10}, v_{14}, m, r, 2)$;
    $\mathsf{G}(v_3, v_7, v_{11}, v_{15}, m, r, 3)$;
    $\mathsf{G}(v_0, v_5, v_{10}, v_{15}, m, r, 4)$;
    $\mathsf{G}(v_1, v_6, v_{11}, v_{12}, m, r, 5)$;
    $\mathsf{G}(v_2, v_7, v_8, v_{13}, m, r, 6)$;
    $\mathsf{G}(v_3, v_4, v_9, v_{14}, m, r, 7)$;
**end**
**return** $v$

The G function operates on 4 64-bit words and is keyed by 2 words of $m$ dependent on the current round (by the $\sigma$ permutation of Table 4.1) and column or diagonal. Algorithm 4.2 describes this function.

**Algorithm 4.2:** BLAKE2b's G.

**Input:** $a \in \mathbb{Z}_{2^{64}}, b \in \mathbb{Z}_{2^{64}}, c \in \mathbb{Z}_{2^{64}}, d \in \mathbb{Z}_{2^{64}}, m \in \mathbb{Z}_{2^{64}}^{16}, r, i$

$a \leftarrow a + b + m_{\sigma_{r \bmod 10}(2i)}$;
$d \leftarrow (d \oplus a) \ggg 32$;
$c \leftarrow c + d$;
$b \leftarrow (b \oplus c) \ggg 24$;
$a \leftarrow a + b + m_{\sigma_{r \bmod 10}(2i+1)}$;
$d \leftarrow (d \oplus a) \ggg 16$;
$c \leftarrow c + d$;
$b \leftarrow (b \oplus c) \ggg 63$;
**return** $a, b, c, d$

### Compression Function

The compression function F takes as input a chain value $h$ of 8 64-bit words $h_0, \ldots, h_7$, a message block $m$ of 16 64-bit words $m_0, \ldots, m_{15}$, a 128-bit counter $t$ of 2 64-bit words $t_0, t_1$, and two 64-bit flags $f_0$ and $f_1$.

First, F initializes the 16-word internal block $v_0, \ldots, v_{15}$ as

$$\begin{pmatrix} v_0 & v_1 & v_2 & v_3 \\ v_4 & v_5 & v_6 & v_7 \\ v_8 & v_9 & v_{10} & v_{11} \\ v_{12} & v_{13} & v_{14} & v_{15} \end{pmatrix} \leftarrow \begin{pmatrix} h_0 & h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 & h_7 \\ \mathrm{IV}_0 & \mathrm{IV}_1 & \mathrm{IV}_2 & \mathrm{IV}_3 \\ t_0 \oplus \mathrm{IV}_4 & t_1 \oplus \mathrm{IV}_5 & f_0 \oplus \mathrm{IV}_6 & f_1 \oplus \mathrm{IV}_7 \end{pmatrix}.$$

The compression function then simply encrypts $v$ using the message block $m$ as key, and returns the xor of its two resulting halves, along with the original chaining value $h$. This is depicted in Algorithm 4.3.

### Iterated Hashing

BLAKE2b can hash data of any byte length $0 \leq \ell < 2^{128}$. The input is first padded with zeros if necessary to form a sequence of $N = \lceil \ell / 128 \rceil$ 16-word blocks $m^0, m^1, \ldots, m^{N-1}$. The words are parsed from bytes in little endian order.

The initial chaining value $h$ is initialized as the xor of the IV array with the parameter block PB. The parameter block fields are specified in Section 4.1.3.

For each message block, the block length in bytes, excluding the zero padding, is added to the counter $t$. $t_0$ represents the least significant 64 bits of the sum, and $t_1$

**Algorithm 4.3:** BLAKE2b's F.

**Input:** $h \in \mathbb{Z}_{2^{64}}^8, m \in \mathbb{Z}_{2^{64}}^{16}, t \in \mathbb{Z}_{2^{64}}^2, f \in \mathbb{Z}_{2^{64}}^2$

$$\begin{pmatrix} v_0 & v_1 & v_2 & v_3 \\ v_4 & v_5 & v_6 & v_7 \\ v_8 & v_9 & v_{10} & v_{11} \\ v_{12} & v_{13} & v_{14} & v_{15} \end{pmatrix} \leftarrow \begin{pmatrix} h_0 & h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 & h_7 \\ \mathrm{IV}_0 & \mathrm{IV}_1 & \mathrm{IV}_2 & \mathrm{IV}_3 \\ t_0 \oplus \mathrm{IV}_4 & t_1 \oplus \mathrm{IV}_5 & f_0 \oplus \mathrm{IV}_6 & f_1 \oplus \mathrm{IV}_7 \end{pmatrix};$$

$v \leftarrow \mathsf{E}(m, v)$;
**for** $i \leftarrow 0$ **to** 7 **do**
    $h_i \leftarrow h_i \oplus v_i \oplus v_{i+8}$;
**end**
**return** $h$

the most significant ones. The algorithm then proceeds iteratively, at each message block obtaining a new chaining value from the existing chaining value, message block, counter and flags. This is shown in Algorithm 4.4.

**Algorithm 4.4:** BLAKE2b.

**Input:** $m \in \{0,1\}^{8\times *}$
$m^0, \ldots, m^{N-1} \leftarrow \mathsf{pad}(m)$;
**for** $i \leftarrow 0$ **to** 7 **do**
    $h_i = \mathrm{IV}_i \oplus \mathrm{PB}_i$;
**end**
$t \leftarrow 0$;
$f_0 \leftarrow 0$;
$f_1 \leftarrow 0$;
**for** $i \leftarrow 0$ **to** $N-2$ **do**
    $t \leftarrow t + 128$;
    $h \leftarrow \mathsf{F}(h, m^i, t, f)$;
**end**
$f_0 \leftarrow 2^{64} - 1$;
$t \leftarrow t + |m^{N-1}|$;
$h \leftarrow \mathsf{F}(h, m^{N-1}, t, f)$;
**return** $h$

In the last block, the finalization flags are used: $f_0$ is set to $2^{64}-1$, while $f_1$ remains 0. The resulting hash is obtained from the last chaining value $h$ by converting each word to bytes in little endian order.

### 4.1.2 BLAKE2s

BLAKE2s operates on 32-bit words and returns a hash value between 1 and 32 bytes. In the following we specify BLAKE2s's constants, keyed permutation, compression function, and iterative hashing mode.

#### Constants

BLAKE2s uses the following IV:

$$\begin{aligned} \mathrm{IV}_0 &= \texttt{6a09e667} & \mathrm{IV}_1 &= \texttt{bb67ae85} \\ \mathrm{IV}_2 &= \texttt{3c6ef372} & \mathrm{IV}_3 &= \texttt{a54ff53a} \\ \mathrm{IV}_4 &= \texttt{510e527f} & \mathrm{IV}_5 &= \texttt{9b05688c} \\ \mathrm{IV}_6 &= \texttt{1f83d9ab} & \mathrm{IV}_7 &= \texttt{5be0cd19} \end{aligned}$$

These constants are identical to the ones used in BLAKE-256 and SHA-256's IV. Their provenance is

$$\mathrm{IV}_i = \left\lfloor 2^{32} \sqrt{p_i} \right\rfloor \bmod 2^{32},$$

with $p_i$ being the $i$th prime number.

Furthermore, the same 10 permutations $\sigma_0, \ldots, \sigma_9$ are used in BLAKE2s keyed permutation and are defined in Table 4.1.

#### Keyed Permutation

BLAKE2s's keyed permutation operates over a block of 16 32-bit words $v_0, \ldots, v_{15}$, and is also keyed by 16 32-bit words $m_0, \ldots, m_{15}$. It performs 10 rounds. At each round, the operation G is applied in parallel first to each column of $v$, viewed as a $4 \times 4$ matrix, and then to each diagonal. This is identical to the mechanism described in Figure 4.1, and is specified in Algorithm 4.5.

The G function operates on 4 32-bit words and is keyed by 2 words of $m$ dependent on the current round (by the $\sigma$ permutation of Table 4.1) and column or diagonal. Algorithm 4.6 describes this function.

#### Compression Function

The compression function F takes as input a chain value $h$ of 8 32-bit words $h_0, \ldots, h_7$, a message block $m$ of 16 32-bit words $m_0, \ldots, m_{15}$, a 128-bit counter $t$ of 2 32-bit words $t_0, t_1$, and two 32-bit flags $f_0$ and $f_1$.

Similarly to BLAKE2b F initializes the 16-word internal block $v_0, \ldots, v_{15}$ as

$$\begin{pmatrix} v_0 & v_1 & v_2 & v_3 \\ v_4 & v_5 & v_6 & v_7 \\ v_8 & v_9 & v_{10} & v_{11} \\ v_{12} & v_{13} & v_{14} & v_{15} \end{pmatrix} \leftarrow \begin{pmatrix} h_0 & h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 & h_7 \\ \mathrm{IV}_0 & \mathrm{IV}_1 & \mathrm{IV}_2 & \mathrm{IV}_3 \\ t_0 \oplus \mathrm{IV}_4 & t_1 \oplus \mathrm{IV}_5 & f_0 \oplus \mathrm{IV}_6 & f_1 \oplus \mathrm{IV}_7 \end{pmatrix}.$$

**Algorithm 4.5:** BLAKE2s's E.

**Input:** $m \in \mathbb{Z}_{2^{32}}^{16}, v \in \mathbb{Z}_{2^{32}}^{16}$
**for** $r \leftarrow 0$ **to** 9 **do**
  $\mathsf{G}(v_0, v_4, v_8, v_{12}, m, r, 0)$;
  $\mathsf{G}(v_1, v_5, v_9, v_{13}, m, r, 1)$;
  $\mathsf{G}(v_2, v_6, v_{10}, v_{14}, m, r, 2)$;
  $\mathsf{G}(v_3, v_7, v_{11}, v_{15}, m, r, 3)$;
  $\mathsf{G}(v_0, v_5, v_{10}, v_{15}, m, r, 4)$;
  $\mathsf{G}(v_1, v_6, v_{11}, v_{12}, m, r, 5)$;
  $\mathsf{G}(v_2, v_7, v_8, v_{13}, m, r, 6)$;
  $\mathsf{G}(v_3, v_4, v_9, v_{14}, m, r, 7)$;
**end**
**return** $v$

**Algorithm 4.6:** BLAKE2s's G.

**Input:** $a \in \mathbb{Z}_{2^{32}}, b \in \mathbb{Z}_{2^{32}}, c \in \mathbb{Z}_{2^{32}}, d \in \mathbb{Z}_{2^{32}}, m \in \mathbb{Z}_{2^{32}}^{16}, r, i$
$a \leftarrow a + b + m_{\sigma_r(2i)}$;
$d \leftarrow (d \oplus a) \ggg 16$;
$c \leftarrow c + d$;
$b \leftarrow (b \oplus c) \ggg 12$;
$a \leftarrow a + b + m_{\sigma_r(2i+1)}$;
$d \leftarrow (d \oplus a) \ggg 8$;
$c \leftarrow c + d$;
$b \leftarrow (b \oplus c) \ggg 7$;
**return** $a, b, c, d$

The compression function then simply encrypts $v$ using the message block $m$ as key, and returns the xor of its two resulting halves, along with the original chaining value $h$. This is depicted in Algorithm 4.7.

### Iterated Hashing

BLAKE2s can hash data of any byte length $0 \leq \ell < 2^{64}$. The input is first padded with zeros if necessary to form a sequence of $N = \lceil \ell/64 \rceil$ 16-word blocks $m^0, m^1, \ldots, m^{N-1}$. The words are parsed from bytes in little endian order.

The initial chaining value $h$ is initialized as the xor of the IV array with the parameter block PB. The parameter block fields are specified in Section 4.1.3.

For each message block, the block length in bytes, excluding the zero padding, is added to the counter $t$. $t_0$ represents the least significant 32 bits of the sum, and $t_1$ the most significant ones. The algorithm then proceeds iteratively, at each message

**Algorithm 4.7:** BLAKE2s's F.

**Input:** $h \in \mathbb{Z}_{2^{32}}^8, m \in \mathbb{Z}_{2^{32}}^{16}, t \in \mathbb{Z}_{2^{32}}^2, f \in \mathbb{Z}_{2^{32}}^2$

$$\begin{pmatrix} v_0 & v_1 & v_2 & v_3 \\ v_4 & v_5 & v_6 & v_7 \\ v_8 & v_9 & v_{10} & v_{11} \\ v_{12} & v_{13} & v_{14} & v_{15} \end{pmatrix} \leftarrow \begin{pmatrix} h_0 & h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 & h_7 \\ \mathrm{IV}_0 & \mathrm{IV}_1 & \mathrm{IV}_2 & \mathrm{IV}_3 \\ t_0 \oplus \mathrm{IV}_4 & t_1 \oplus \mathrm{IV}_5 & f_0 \oplus \mathrm{IV}_6 & f_1 \oplus \mathrm{IV}_7 \end{pmatrix};$$

$v \leftarrow \mathsf{E}(m, v)$;
**for** $i \leftarrow 0$ **to** 7 **do**
    $h_i \leftarrow h_i \oplus v_i \oplus v_{i+8}$;
**end**
**return** $h$

block obtaining a new chaining value from the existing chaining value, message block, counter and flags. This is shown in Algorithm 4.8.

**Algorithm 4.8:** BLAKE2s.

**Input:** $m \in \{0,1\}^{8\times *}$
$m^0, \ldots, m^{N-1} \leftarrow \mathsf{pad}(m)$;
**for** $i \leftarrow 0$ **to** 7 **do**
    $h_i = \mathrm{IV}_i \oplus \mathrm{PB}_i$;
**end**
$t \leftarrow 0$;
$f_0 \leftarrow 0$;
$f_1 \leftarrow 0$;
**for** $i \leftarrow 0$ **to** $N-2$ **do**
    $t \leftarrow t + 64$;
    $h \leftarrow \mathsf{F}(h, m^i, t, f)$;
**end**
$f_0 \leftarrow 2^{32} - 1$;
$t \leftarrow t + |m^{N-1}|$;
$h \leftarrow \mathsf{F}(h, m^{N-1}, t, f)$;
**return** $h$

In the last block, the finalization flags are used: $f_0$ is set to $2^{32}-1$, while $f_1$ remains 0. The resulting hash is obtained from the last chaining value $h$ by converting each word to bytes in little endian order.

### 4.1.3 Parameter block

The parameter block of BLAKE2 is *xored with the IV* prior to the processing of the first data block. It encodes parameters for secure tree hashing, as well as key length (in keyed mode) and digest length.

The parameters are described below, and the block structure is shown in Tables 4.2 and 4.3:

- General parameters:

  **Digest byte length (1 byte)** an integer in $[1, 64]$ for BLAKE2b, in $[1, 32]$ for BLAKE2s

  **Key byte length (1 byte)** an integer in $[0, 64]$ for BLAKE2b, in $[0, 32]$ for BLAKE2s (set to 0 if no key is used)

  **Salt (16 or 8 bytes)** an arbitrary string of 16 bytes for BLAKE2b, and 8 bytes for BLAKE2s (set to all-zero by default)

  **Personalization (16 or 8 bytes)** an arbitrary string of 16 bytes for BLAKE2b, and 8 bytes for BLAKE2s (set to all-zero by default)

- Tree hashing parameters:

  **Fanout (1 byte)** an integer in $[0, 255]$ (set to 0 if unlimited, and to 1 only in sequential mode)

  **Maximal depth (1 byte)** an integer in $[1, 255]$ (set to 255 if unlimited, and to 1 only in sequential mode)

  **Leaf maximal byte length (4 bytes)** an integer in $[0, 2^{32} - 1]$, that is, up to 4 GB (set to 0 if unlimited, or in sequential mode)

  **Node offset (8 or 6 bytes)** an integer in $[0, 2^{64} - 1]$ for BLAKE2b, and in $[0, 2^{48} - 1]$ for BLAKE2s (set to 0 for the first, leftmost, leaf, or in sequential mode)

  **Node depth (1 byte)** an integer in $[0, 255]$ (set to 0 for the leaves, or in sequential mode)

  **Inner hash byte length (1 byte)** an integer in $[0, 64]$ for BLAKE2b, and in $[0, 32]$ for BLAKE2s (set to 0 in sequential mode)

This is 50 bytes in total for BLAKE2b, and 32 bytes for BLAKE2s. Any bytes left are reserved for future and/or application-specific use, and are zeroed. Values spanning more than one byte are written in *little-endian*. Note that tree hashing may be keyed, in which case leaf instances hash the key followed by a number of bytes equal to (at most) the maximal leaf length.

Table 4.2: BLAKE2b parameter block structure (offsets in bytes).

| Offset | **0** | **1** | **2** | **3** |
|---|---|---|---|---|
| **0** | Digest length | Key length | Fanout | Depth |
| **4** | Leaf length | | | |
| **8**<br>**12** | Node offset | | | |
| **16** | Node depth | Inner length | RFU | |
| **20**<br>**24**<br>**28** | RFU | | | |
| **32**<br>...<br>**44** | Salt | | | |
| **48**<br>...<br>**60** | Personalization | | | |

Table 4.3: BLAKE2s parameter block structure (offsets in bytes).

| Offset | **0** | **1** | **2** | **3** |
|---|---|---|---|---|
| **0** | Digest length | Key length | Fanout | Depth |
| **4** | Leaf length | | | |
| **8** | Node offset | | | |
| **12** | Node offset (cont.) | | Node depth | Inner length |
| **16**<br>**20** | Salt | | | |
| **24**<br>**28** | Personalization | | | |

### 4.1.4 Keyed hashing (MAC and PRF)

When keyed (that is, when the key length field of the parameter block is non-zero), BLAKE2 sets the first data block to the key padded with zeros, the second data block to the first block of the message, the third block to the second block of the message, etc. Note that the padded key is treated as arbitrary data, therefore:

- The counter $t$ includes the 64 (or 128) bytes of the key block, regardless of the key length.
- When hashing the empty message with a key, BLAKE2b and BLAKE2s make only one call to the compression function.

The main application of keyed BLAKE2 is as a message authentication code (MAC): BLAKE2 can be used securely in prefix-MAC mode due to its indifferentiability, cf. Section 4.4.4. Prefix-MAC is faster than HMAC, as it saves at least one call to the

compression function. Keyed BLAKE2 can also be used to instantiate PRFs, for example within the PBKDF2 password hashing scheme.

### 4.1.5 Tree hashing

The parameter block supports arbitrary tree hashing modes, be it binary or ternary trees, arbitrary-depth updatable tree hashing or fixed-depth parallel hashing, etc. Note that, unlike other functions, BLAKE2 does not restrict the leaf length and the fanout to be powers of 2.

**Basic mechanism** Informally, tree hashing processes chunks of data of "leaf length" bytes independently of each other, then combines the respective hashes using a tree structure wherein each node takes as input the concatenation of "fanout" hashes. The "node offset" and "node depth" parameters ensure that each invocation to the hash function (leaf or internal node) uses a different hash function. The finalization flag $f_1$ signals when a hash invocation is the last one at a given depth (where "last" is with respect to the node offset counter, for both leaves and intermediate nodes). The flag $f_1$ can only be non-zero for the last block compressed within a hash invocation, and the root node always has $f_1$ set to $-1$.

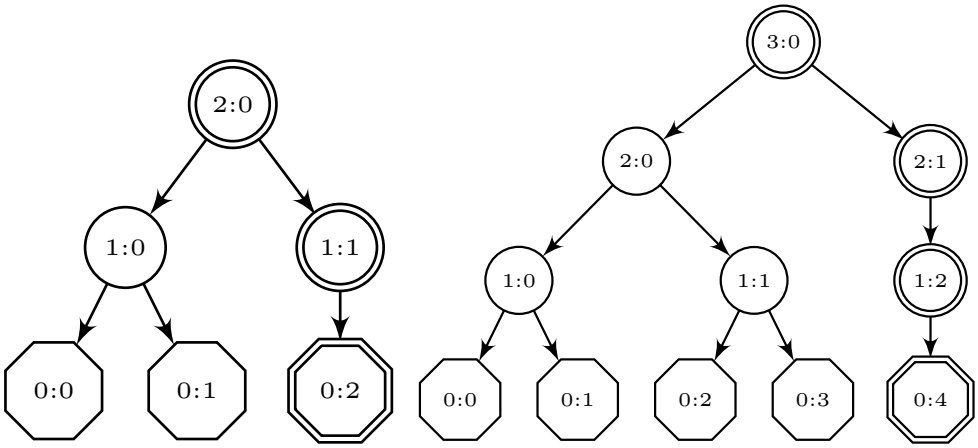
The tree hashing mechanism is illustrated on Figures 4.2 and 4.3, which show layout of trees given different parameters and different input lengths. On those figures, octagons represent leaves (i.e., instances of the hash function processing input data), double-lined nodes (including leaves) are the last nodes of a layer, and thus have the flag $f_1$ set). Labels "$i$:$j$" indicate a node's depth $i$ and offset $j$.

We refer to [69, 148, 169, 214] for a comprehensive overview of secure tree hashing constructions.

**Message parsing** Unless specified otherwise, we recommend that data be parsed as contiguous blocks: for example, if leaf length is 1024 bytes, then the first 1024-byte data block is processed by the leaf with offset 0, the subsequent 1024-byte data block is processed by the leaf with offset 1, etc.

**Special cases** We highlight some special cases of tree hashing:

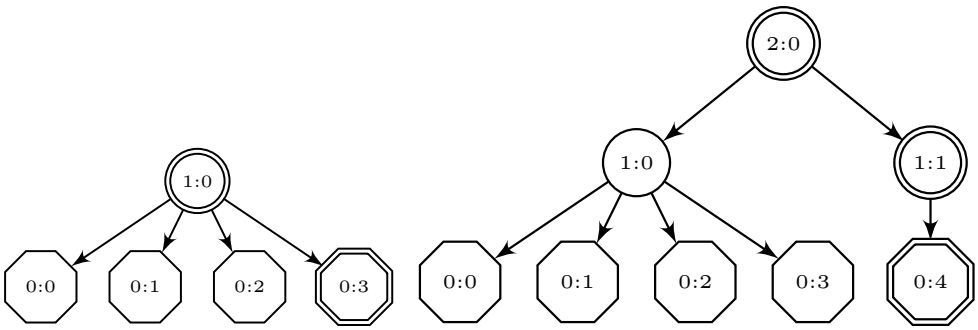
**Unlimited fanout** When the fanout is unlimited (parameter set to 0), then the root node hashes the concatenation of as many leaves are required to process the message. That is, the depth of the tree is always 2, regardless of the maximal depth parameter. Nevertheless, changing the maximal depth parameter changes the final hash value returned. We thus recommend to set the depth parameter to 2.




(a) Hashing 3 blocks: the tree has depth 3.

(b) Hashing 5 blocks: the tree has depth 4.

Figure 4.2: Layouts of tree hashing with fanout 2, and maximal depth at least 4.




(a) Hashing 4 blocks: the tree has depth 2.

(b) Hashing 5 blocks: the tree has depth 3.

Figure 4.3: Layouts of tree hashing with fanout 4, and maximal depth at least 3.

**Saturated trees** If a tree hashing instance has fanout $f \geq 2$, maximal depth $d \geq 2$, and leaf maximal length $\ell \geq 1$ bytes, then up to $f^{d-1} \cdot \ell$ can be processed within a single tree. If more bytes have to be hashed, the fanout of the root node is extended to hash as many digests as necessary to respect the depth limit. This mechanism is illustrated on Figure 4.4. Note that if the maximal depth is 2, then the value does not affect the layout of the tree, which is identical to that of a tree hash with unlimited fanout.

**Generic tree parameters** Tree parameters supported by the parameter block allow for a wide range of implementation trade-offs, for example to efficiently support

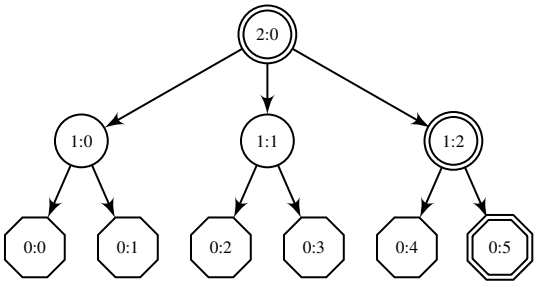



Figure 4.4: Tree hashing with maximal depth 3, fanout 2, but a root with larger fanout due to the reach of the maximal depth.

updatable hashing, which is typically an advantage when hashing many (small) chunks of data.

Although optimal performance will be reached by choosing the parameters specific to one's application, we specify the following parameters for a *generic tree mode*: binary tree (i.e., fanout 2), unlimited depth, and leaves of 4 KB (the typical size of a memory page).

**Example: updatable hashing** Assume one has to provide a digest of a 1 TB filesystem disk image that is updated every day. Instead of recomputing the digest by reading all the $2^{40}$ bytes, one can use our generic tree mode to implement an updatable hashing scheme:

1. Apply the generic tree mode, and store the $2^{40}/4096 = 2^{28}$ hashes from the leaves as well as the $2^{28} - 2$ intermediate hashes

2. When a leaf is changed, update the final digest by recomputing the 28 intermediate hashes

If BLAKE2b is used with intermediate hashes of 32 bytes, and that it hashes at a rate of 500 MB per second, then step 1 takes approximately 35 minutes and generates about 16 GB of intermediate data, whereas step 2 is instantaneous.

Note however that much less data may be stored: For many applications it is preferable to only store the intermediate hashes for larger pieces of data (without increasing the leaf size), which reduces memory requirement by only storing "higher" intermediate values. For example, storing intermediate values for 4 MB chunks instead of all 4 KB leaves reduces the storage to only 16 MB. Indeed, using 4 KB leaves allows applications with different piece sizes (as long as they are powers-of-two of at least 4 KB) to produce the same root hash, while allowing them to make different granularity vs. storage trade-offs.

### 4.1.6 Parallel hashing: BLAKE2sp and BLAKE2bp

We specify 2 parallel hash functions (that is, with depth 2 and unlimited leaf length):

- BLAKE2bp runs *4 instances* of BLAKE2b in parallel
- BLAKE2sp runs *8 instances* of BLAKE2s in parallel

These functions use a different *parsing rule* than the default one in Section 4.1.5: The first instance (node offset 0) hashes the message composed of the concatenation of all message blocks of index 0 modulo 4; the second instance (node offset 1) hashes blocks of index 1 modulo 4, and so on. Note that when the leaf length is unlimited, parsing the input as contiguous blocks would require the knowledge of the input length before any parallel operation, which is undesirable (e.g., when hashing a stream of data of undefined length, or a file received over a network).

When hashing one single large file, and when incrementability is not required, such parallel modes with unlimited leaf length seem appropriate, since

- They are simpler to implement than full fledged tree modes;
- They *minimize the computation overhead* by doing only one non-leaf call to the sequential hash function;
- They *maximize the usage of the CPU* by keeping multiple cores and instruction pipelines busy simultaneously;
- They require *realistic bandwidth and memory*.

Within a parallel hash, the same parameter block, except for the node offset, is used for all 4 or 8 instances of the sequential hash.

## 4.2 Design Rationale

We now describe the core differences between BLAKE and BLAKE2, along with their respective reasoning.

### 4.2.1 Fewer rounds

BLAKE2b and BLAKE2s perform *12 rounds* and *10 rounds* respectively, in contrast to 16 and 14 for BLAKE. Based on the security analysis performed so far and on reasonable assumptions on future progress, it appears unlikely that 16 and 14 rounds are meaningfully more secure than 12 and 10 rounds (cf. Section 4.4). Recall that the initial BLAKE submission [27] had 14 and 10 rounds, respectively, and that the

later increase [28] was motivated by the high speed of BLAKE (i.e., it could afford a few extra rounds for the sake of conservativeness), rather than by cryptanalytic results.

This change gives a direct speedup of about 25% and 29%, respectively, on long inputs. Speed on short inputs also significantly improves, though by a lower ratio, due to the overhead of initialization and finalization.

### 4.2.2 Rotations optimized for speed

The core function (G) of BLAKE-512 performs four 64-bit word rotations of respectively 32, 25, 16, and 11 bits. BLAKE2b replaces 25 with 24, and 11 with 63:

- Using a 24-bit rotation allows SSSE3-capable CPUs to perform two rotations in parallel with a single SIMD instruction (namely, `pshufb`), whereas two shifts plus a logical OR are required for a rotation of 25 bits. This reduces the arithmetic cost of the G function in recent Intel CPUs from 18 single-cycle instructions to 16 instructions, a 12% decrease.

- A 63-bit rotation can be implemented as an addition (doubling) and a shift followed by a logical OR. This provides a slight speedup on platforms where addition and shift can be realized in parallel but not two shifts (i.e., some recent Intel CPUs). Additionally, since a rotation right by 63 is equal to a rotation left by 1, this may be slightly faster in some architectures where 1 is treated as a special case.

No platform that we know of suffers a penalty from these changes. For an in-depth analysis of optimized implementations of rotations, we refer to previous work on SIMD implementations of BLAKE [358, 359].

### 4.2.3 Minimal padding and finalization flags

BLAKE2 zero pads the last data block *if and only if necessary*. In other words, if the data length is a multiple of the block length, no padding is added. The padding thus does not include the message length, as in BLAKE, MD5, or SHA-2.

To ensure the soundness of the hashing mode, BLAKE2 introduces the *finalization flags* $f_0$ and $f_1$ as auxiliary inputs to the compression function:

- The security functionality of the padding is transferred to a finalization flag $f_0$, a word set to $-1$ if the block processed is the last, and to 0 otherwise. The flag $f_0$ is 64-bit for BLAKE2b, and 32-bit for BLAKE2s.

- A second finalization flag $f_1$ is used to signal the last node of a layer in tree-hashing modes (see Section 4.1.5). When processing the last block—that is,

when $f_0$ is $-1$—the flag $f_1$ is also set to $-1$ if the node considered is the last, and to 0 otherwise.

The finalization flags are processed by the compression function as described in Sections 4.1.1 and 4.1.2.

BLAKE2s thus supports hashing of data of at most $2^{64}-1$ bytes, that is, almost 16 EB (the amount of memory addressable by 64-bit processors). BLAKE2b's upper limit of $2^{128}-1$ bytes ought to be enough for anybody.

### 4.2.4 Fewer constants

Whereas BLAKE used 8 word constants as IV plus 16 word constants for use in the compression function, BLAKE2 uses a total of *8 word constants, instead of 24*. This saves 128 ROM bytes and 128 RAM bytes in BLAKE2b implementations, and 64 ROM bytes and 64 RAM bytes in BLAKE2s implementations.

Omitting the constants in G gives an algorithm similar to the (unattacked) BLAZE toy version of BLAKE (cf. [26, 246, 458]). Constants in G initially aimed to guarantee early propagation of carries, but it turned out that the benefits (if any) are not worth the performance penalty, as observed by a number of cryptanalysts. This change saves two xors and two loads per G, that is, 192 fewer xors on BLAKE2b and 160 fewer xors on BLAKE2s, or 16% of the total arithmetic (addition and xor) instructions.

### 4.2.5 Little-endian

BLAKE, like SHA-1 and SHA-2, parses data blocks in the big-endian byte order. Like MD5, *BLAKE2 is little-endian*, because the large majority of target platforms is little-endian (AMD and Intel processors, most mainstream ARM systems). Switching to little-endian may provide a slight speedup, and often simplifies implementations.

Note that in BLAKE, the counter $t$ is composed of two words $t_0$ and $t_1$, where $t_0$ holds the least significant bits of the integer encoded. This little-endian convention is preserved in BLAKE2.

### 4.2.6 Counter in bytes

The counter $t$ counts *bytes rather than bits*. This simplifies implementations and reduces the risk of error, since target applications measure data volumes in bytes rather than bits.

Note that BLAKE supported messages of arbitrary bit size for the sole purpose of conforming to NIST's requirements[1]. However, as discussed on the SHA-3 mailing

[1] https://web.archive.org/web/20170220230333/http://csrc.nist.gov/groups/ST/hash/documents/SHA3-C-API.pdf

list, there is no evidence of an actual need to support this. As observed during the first months of the competition, the support of arbitrary bit sizes was the origin of several bugs in reference implementations—including that of BLAKE [345].

### 4.2.7 Salt processing

BLAKE's predecessor LAKE [31] introduced the built-in support for a salt, to simplify the use of randomized hashing within digital signature schemes (although the RMX transform [218] can be used with arbitrary hash functions).

In BLAKE2 the salt is processed as a one-time input to the hash function, through the IV, rather than as an input to each compression function. This both simplifies the compression function and saves a few instructions as well as a few bytes in RAM, since the salt does not have to be stored throughout the whole computation.

Randomized hashing was introduced by Halevi and Krawczyk [218] as a countermeasure against collision attacks. Given the high confidence in BLAKE and BLAKE2's collision resistance, cf. Section 4.4, salting every compression function to prevent attacks did not seem justifiable. But as part of the IV it still may be used to distinguish between separate uses of the hash function, for example.

## 4.3 Performance

BLAKE2 is much faster than BLAKE, mainly due to its reduced number of rounds. On long messages, the BLAKE2b and BLAKE2s versions are expected to be approximately 25% and 29% faster, ignoring any savings from the absence of constants, optimized rotations, or little-endian conversion. The parallel versions BLAKE2bp and BLAKE2sp are expected to be 4 and 8 times faster than BLAKE2b and BLAKE2s on long messages, when implemented in parallel. Parallel hashing also benefits from vector units in CPUs, as previously observed [359, §5.2].

### 4.3.1 Why BLAKE2 is fast in software

BLAKE2, along with its parallel variants, can take advantage of the following architectural features, or combinations thereof:

**Instruction-level parallelism** Most modern processors are superscalar, that is, able to run several instructions per cycle through pipelining, out-of-order execution, and other related techniques. BLAKE2 has a natural instruction parallelism of 4 instructions within the G function; processors that are able to handle more instruction-level parallelism can do so in BLAKE2bp or BLAKE2sp, by interleaving independent compression function calls. Examples of processors with notorious amount of instruction

Table 4.4: Speed, in cycles per byte, of BLAKE2 and MD5 in sequential mode. Measurements are presented for long, 1536, and 64-byte messages.

| Microarchitecture | BLAKE2b | | | BLAKE2s | | | MD5 | | |
|---|---|---|---|---|---|---|---|---|---|
| | Long | 1536 | 64 | Long | 1536 | 64 | Long | 1536 | 64 |
| Intel Sandy Bridge | 3.32 | 3.81 | 9.00 | 5.34 | 5.35 | 5.50 | 5.38 | 5.75 | 14.34 |
| Intel Haswell | 3.08 | 3.14 | 7.53 | 5.35 | 5.38 | 6.23 | 4.97 | 5.27 | 12.47 |
| Intel Skylake | 3.19 | 3.23 | 7.36 | 4.83 | 4.86 | 5.66 | 5.01 | 5.32 | 12.61 |
| Intel Icelake | 2.99 | 2.99 | 6.34 | 4.53 | 4.52 | 4.41 | 4.78 | 5.02 | 10.62 |
| AMD Bulldozer | 5.29 | 5.30 | 11.95 | 8.20 | 8.21 | 7.91 | 5.34 | 5.13 | 14.67 |
| AMD Zen 1 | 3.19 | 3.20 | 6.88 | 5.36 | 5.35 | 5.23 | 5.00 | 5.32 | 12.55 |
| AMD Zen 3 | 3.37 | 3.36 | 7.02 | 5.26 | 5.25 | 5.05 | 4.77 | 5.04 | 11.45 |
| Apple M1 | 3.07 | 3.21 | 6.25 | 5.70 | 5.99 | 6.25 | 5.98 | 6.51 | 12.50 |

parallelism are any recent Intel or AMD chip, as well as high-end ARM processors such as the Cortex-X1 or Apple M1.

**SIMD instructions** Many modern processors contain *vector units*, which enable SIMD processing of data. Again, BLAKE2 can take advantage of vector units not only in its G function, but also in tree modes (such as the mode proposed in Section 4.1.6), by running several compression instances within vector registers.

**Multiple cores** Limits in both semiconductor manufacturing processes, as well as instruction-level parallelism have driven CPU manufacturers towards yet another kind of coarse-grained parallelism, where multiple independent CPUs are placed inside the same die, and enable the programmer to get thread-level parallelism. While sequential BLAKE2 does not take advantage of this, the parallel mode described in Section 4.1.6, and other tree modes, can run each intermediate hashing in its own thread.

### 4.3.2 64-bit CPUs

We have submitted optimized BLAKE2 implementations to eBACS [62], that take advantage of the SSSE3, AVX2, and XOP instruction sets. Table 4.4 reports the timings obtained in several popular microarchitectures, as well as MD5 for comparison.

Compared to the best known timings for BLAKE [358, 359],

- On Sandy Bridge, BLAKE2b is 72% faster than BLAKE-512, and BLAKE2s is 40% faster than BLAKE-256,

- On Bulldozer, BLAKE2b is 30% faster than BLAKE-512, and BLAKE2s is 44% faster than BLAKE-256.

Due to the lack of native rotation instructions on SIMD registers, the speedup of BLAKE2b is greater on the Intel processors, which benefit not only from the round reduction, but also from the easier-to-implement rotations.

On short messages, the speed advantage of the improved padding on BLAKE2 is quite noticeable. On Sandy Bridge, no other cryptographic hash function measured in eBACS [62] (including MD5 and MD4) is faster than BLAKE2s on 64-byte messages, while BLAKE2b is roughly as fast as MD4. Furthermore, as apparent in Table 4.4, BLAKE2b handily outperforms MD5 in every 64-bit processor tested, sometimes by almost a factor of 2.

The speedup for the BLAKE2sp and BLAKE2bp modes is also significant, even when not using multiple CPU cores. We implemented both using AVX2[2], and obtained

- 1.37 cycles per byte on Haswell for BLAKE2bp;
- 1.39 cycles per byte on Haswell for BLAKE2sp.
- 1.29 cycles per byte on Skylake for BLAKE2bp;
- 1.30 cycles per byte on Skylake for BLAKE2sp.

Compared to Keccak's SHA-3 final submission, BLAKE2 does quite well on 64-bit hardware. On Sandy Bridge, the 512-bit Keccak$[r = 576, c = 1024]$ hashes at 20.46 cycles per byte, while the 256-bit Keccak$[r = 1088, c = 512]$ hashes at 10.87 cycles per byte.

Keccak is, however, a very versatile design. By lowering the capacity from $4n$ to $2n$, where $n$ is the output bit length, one achieves $n/2$-bit security for both collisions and second preimages [72], but also higher speed. We estimate that a 512-bit Keccak$[r = 1088, c = 512]$ would hash at about 10 cycles per byte on high-end Intel and AMD CPUs, and a 256-bit Keccak$[r = 1344, c = 256]$ would hash at roughly 8 cycles per byte. This parametrization would put Keccak at a performance level superior to SHA-2, but at a substantial cost in second-preimage resistance. BLAKE2 does not require such tradeoffs, and still offers much higher speed.

The Keccak variant KangarooTwelve [70] is a more recent member of the Keccak family, which adopts several of the same ideas that made BLAKE2 faster: reduced rounds (24 to 12!), lower capacity $c = 256$, and unlimited fanout tree hashing. Its performance is very competitive.

[2] https://github.com/sneves/blake2-avx2

### 4.3.3 Low-end platforms

A typical implementation of BLAKE-256 in *embedded software* stores in RAM at least the chaining value (32 bytes), the message (64 bytes), the constants (64 bytes), the permutation internal state (64 bytes), the counter (8 bytes), and the salt, if used (16 bytes); that is, 232 bytes, and 248 with a salt. BLAKE2s reduces these figures to *168 bytes*—recall that the salt doesn't have to be stored anymore—that is, a gain of respectively 28% and 32%. Similarly, BLAKE2b only requires 336 bytes of RAM, against 464 or 496 for BLAKE-512.

### 4.3.4 Hardware

Hardware directly benefit from the 29% and 25% speed-up in sequential mode, due to the round reduction, for any message length. Parallelism is straightforward to implement by replicating the architecture of the sequential hash. BLAKE2 enjoys the same degrees of freedom as BLAKE to implement various space-time tradeoffs (horizontal and vertical folding, pipelining, etc.). In addition, parallel hashing provides *another dimension for trade-offs* in hardware architectures: depending on the system properties (e.g. how many input bits can be read per cycle), one may choose between, for example, BLAKE2sp based on 8 high-latency compact cores, or BLAKE2s based on a single low-latency unrolled core.

## 4.4 Security

BLAKE2 takes advantage of the high confidence built by BLAKE in the SHA-3 competition. Although BLAKE2 performs fewer rounds than BLAKE, this does not imply lower security (it does imply a lower *security margin*), as explained below.

### 4.4.1 Implications of BLAKE2 tweaks

We argue that the reduced number of rounds and the optimized rotations are unlikely to meaningfully reduce the security of BLAKE2, compared to that of BLAKE. We summarize the security implications of other tweaks:

#### Salt-independent compressions

BLAKE2 salts the hash function in the IV, rather than each compression. This preserves the uniqueness of the hash function for any distinct salt, but facilitates multicollision attacks relying on offline precomputations (see [81, 248]). However, this leaves fewer attacker-controlled bits in the initial state of the compression function, which complicates the finding of fixed points.

#### Many valid IVs

Due to the high number of valid parameter blocks, BLAKE2 admits many valid initial chaining values. For example, if an attacker has an oracle that returns collisions for random chaining values and messages, they are more likely to succeed in attacking the hash function because they have many valid targets, rather than a valid one. However, such a scenario assumes that (free-start) collisions can be found efficiently, that is, that the hash function is already broken. Note that the best collision-like results on BLAKE are near-collisions for the compression function with 4 *reordered* rounds [216, 435].

Note also that the existence of many valid IVs does not compromise the soundness of the mode, since the counter and flags ensure that first and last compression function calls are adequately domain-separated (see also Section 4.4.4).

#### Simplified padding

The new padding does not include the message length of the message, unlike BLAKE. However, it is easy to see that the message length is encoded through the counter, and that this simpler zero padding preserves the unambiguous encoding of the initial padding. That is, the padding simplification does not affect the security of the hash function.

### 4.4.2 BLAKE Legacy

The security of BLAKE2 is closely related to that of BLAKE, since they rely on a similar core permutation originally used in Bernstein's ChaCha stream cipher [54] (itself a variant of Salsa20 [59], co-winner in the eSTREAM project [183]).

Since 2009, at least 16 research papers have described cryptanalysis results on reduced versions of BLAKE. The most advanced attacks on the BLAKE as hash function—as opposed to its building blocks—are *preimage attacks on 2.5 rounds* by Ji and Liangyu, with respective complexities $2^{241}$ and $2^{481}$ for BLAKE-256 and BLAKE-512 [297]. Most research actually considered reduced versions of the compression function or keyed permutation of BLAKE, regardless of the constraints imposed by the IV. The most recent results of this type are the following (see also Table 4.5)

- A distinguisher on 6 rounds of the permutation of BLAKE-256, with complexity $2^{456}$, by Dunkelman and Khovratovich [174];
- A boomerang distinguisher on 8 rounds of the core permutation of BLAKE-512, with complexity $2^{242}$, by Biryukov, Nikolic, and Roy [95] (recent work questions the correctness of this result [294, 295]).

Table 4.5: Summary of cryptanalysis on BLAKE and BLAKE2. Attacks on a keyed permutation are denoted by "k.p." and on the compression function by "c.f."

| Primitive | Type | Rounds | Complexity | Reference |
|---|---|---|---|---|
| BLAKE-256 k.p. | Boomerang | 7 | $2^{44}$ | [33, 221] |
| BLAKE-256 k.p. | Boomerang | 8 | $2^{198}$ | [221] |
| BLAKE-256 k.p. | Differential | 4 | $2^{192}$ | [174] |
| BLAKE-256 k.p. | Differential | 6 | $2^{456}$ | [174] |
| BLAKE-256 k.p. | Impossible Differential | 6.5 | – | [215] |
| BLAKE-256 c.f. | Boomerang | 7 | $2^{242}$ | [95] |
| BLAKE-256 c.f. | Near collision (152/256) | 4 | $2^{21}$ | [435] |
| BLAKE-256 c.f. | Near collision (232/256) | 4 | $2^{56}$ | [26] |
| BLAKE-256 c.f. | Preimage | 2.5 | $2^{241}$ | [297] |
| BLAKE-256 c.f. | Pseudo-preimage | 6.75 | $2^{253.9}$ | [182] |
| BLAKE2s k.p. | Boomerang | 7.5 | $2^{184}$ | [221] |
| BLAKE2s k.p. | Impossible Differential | 6.5 | – | [215] |
| BLAKE2s k.p. | Rotational | 4 | $2^{489}$ | [215, 262] |
| BLAKE2s c.f. | Boomerang | 5 | $\approx 2^{86}$ | [95, 215] |
| BLAKE2s c.f. | Pseudo-preimage | 6.75 | $2^{253.8}$ | [182] |

The exact attacks as described in research papers may not directly apply to BLAKE2, due to the changes of initialization and rotation counts. Nevertheless, attacks on reduced BLAKE with $r$ rounds are expected to adapt to BLAKE2 with $r$ rounds, though with slightly different complexities.

### 4.4.3 Third-party Analysis

Ever since its publication [32], BLAKE2 has seen numerous cryptanalytic efforts from third parties. Table 4.5 lists those efforts and compares them to similar cryptanalytic efforts on BLAKE.

In particular [215] conclude that, although some of the some of the changes individually appear to make BLAKE2 weaker in some attack models, taken as a whole the changes do not seem to weaken it. BLAKE2 still continues to have a very high security margin against all known attacks.

### 4.4.4 Indifferentiability

The BLAKE mode of operation was proved to be indifferentiable from a random oracle by Chang, Nandi, and Yung [124] and, independently, Andreeva, Luykx, and Mennink [15]. Their proof, however, required the description and analysis of a

simulator for the keyed permutation that interacted with the whole hash function, since BLAKE's compression function can be differentiated from random in $2^{n/4}$ queries, where $n$ is the compression function output size.

BLAKE2 makes the security reduction more modular, by making the compression function indifferentiable. Given an indifferentiable compression function, any tree hashing mode (which includes sequential hashing) can be proven indifferentiable from a random oracle as long as it follows three simple criteria [148, 214] (see also [69, 169]):

**Subtree-freeness** No valid tree can be a subtree of another.

**Radical-decodability** The position of the chaining value and message bits are always unambiguous. That is, the mode defines for a subtree a set of bits that corresponds to a chaining value to any tree it is part of.

**Message decodability** The whole message is processed by the hash. Or in other words, given all inputs to compression functions, the original message can be reconstructed.

These three criteria are met by BLAKE2: the parameter block and finalization flags ensure that no tree can be a subtree of another and there is no ambiguity with respect to the chaining value location.

Armed with these conditions, the indifferentiability of the hash function can now be proven in terms of the security of the compression function.

**Theorem 4.1** ([148, 214, Theorem 1])**.** *Consider a hashing mode* $\mathsf{H}$ *of a random arbitrary function* $\mathsf{F}$*, and assume that it is subtree-free, radical-decodable, and message-decodable. There exists a simulator* $\mathsf{Sim}$ *such that for any distinguisher* $\mathsf{D}$ *with total complexity at most* $q$*,*

$$\mathsf{Adv}^{\mathrm{diff}}_{\mathsf{H}^{\mathsf{F}},\mathsf{Sim}}(\mathsf{D}) \leq \frac{\binom{q}{2}}{2^n}\,.$$

*The simulator makes at most* $q$ *queries to* $\mathsf{RO}$*.*

Luykx et al. [313] showed that the compression function is indifferentiable up to the birthday bound.

**Theorem 4.2** ([313, Theorem 1])**.** *Let* $\mathsf{E} \leftarrow\!\!\$\ \mathsf{Block}^*(2n)$ *be a weakly ideal cipher, and consider the* BLAKE2 *compression function* $\mathsf{F}^{\mathsf{E}}$ *of (4.1.1) that internally uses* $\mathsf{E}$*. There exists a simulator* $\mathsf{Sim}$ *such that for any distinguisher* $\mathsf{D}$ *with total complexity* $q$*,*

$$\mathsf{Adv}^{\mathrm{diff}}_{\mathsf{F}^{\mathsf{E}},\mathsf{Sim}}(\mathsf{D}) \leq \frac{\binom{q}{2}}{2^{2n}} + \frac{\binom{q}{2}}{2^n} + \frac{q}{2^{n/2}}\,,$$

*where* $\mathsf{S}$ *makes at most* $q$ *queries to the random oracle.*

Combining Theorem 4.2 with Theorem 4.1 immediately leads to the following corollary.

**Corollary 4.3** (Indifferentiability of BLAKE2 Hashing Mode)**.** *Let* $\mathsf{E} \leftarrow\!\!\$\ \mathsf{Block}^*(2n)$ *be a weakly ideal cipher, and consider the BLAKE2 hash function* $\mathsf{H}^{\mathsf{E}}$ *that internally uses* $\mathsf{E}$*. There exists a simulator* $\mathsf{Sim}$ *such that for any distinguisher* $\mathsf{D}$ *with total complexity* $q$*,*

$$\mathsf{Adv}^{\mathrm{diff}}_{\mathsf{H}^{\mathsf{E}},\mathsf{Sim}}(\mathsf{D}) \leq \frac{\binom{q}{2}}{2^{2n}} + \frac{2\binom{q}{2}}{2^{n}} + \frac{q}{2^{n/2}}\,,$$

*where* $\mathsf{S}$ *makes at most* $q$ *queries to the random oracle.*

# Chapter 5

# NORX: Parallel Authenticated Encryption

Authenticated encryption [48] (AE) is the standard technology to protect data that needs to be sent over unsecured communication channels and is deployed in countless applications and protocols, such as (D)TLS, SSH and IPSec. In comparison to regular symmetric encryption schemes, authenticated encryption not only ensures *privacy* of the data but also guarantees *integrity* and *authenticity* (cf. Section 2.3.5). Unfortunately, failures in the design and implementation of authenticated encryption schemes are a common sight and there are numerous examples. To name just a few (see also [55]):

- Vaudenay's 2002 CBC *padding oracle attack* on MAC-then-encrypt authenticated encryption modes allows an active adversary to decrypt messages without access to the secret key [455]. This attack stemmed from the authenticity verification leaking whether the decrypted message was adequately padded. Over the years, this strategy has been used quite successfully against TLS [8, 117, 176, 340]. Indeed, the TLS failures lead to a situation where the broken RC4 cipher was the preferred encryption algorithm for a short period of time.

- In 2007, an attack [438] on the Wired Equivalent Privacy (WEP) standard, used in many 802.11 Wi-Fi networks, allowed to recover the secret key within minutes from a few thousand intercepted messages. The attack exploited weaknesses in RC4.

- In 2009, Albrecht, Paterson, and Watson [6] exploited a flaw in the SSH protocol and its OpenSSH implementation, when coupled with a block cipher in CBC mode. The attack allowed an adversary to recover 14 plaintext bits with probability $2^{-14}$ or 32 plaintext bits with probability $2^{-18}$.

- In 2012, a flaw was uncovered in EAXprime [19], an authenticated encryption block cipher mode derived from EAX [50], standardized as ANSI C12.22-2008 for Smart Grid applications, and also subject of a forthcoming NIST standard. The flaw facilitates forgery, distinguishing, and message-recovery attacks [337].

- In 2018, a critical flaw was uncovered in the security proof of the OCB2 authenticated encryption scheme, introduced in 2004 [404] and part of the ISO/IEC 19772:2009 standard. Like the case of EAXprime, the flaw resulted in both the authenticity and confidentiality of OCB2 being compromised [240, 241].

The *Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR)* [114], initiated in 2013, invited cryptographers to submit authenticated encryption schemes supporting associated data (AEAD) [403], that would offer advantages over the existing state-of-the-art and would be suitable for widespread adoption. Much like the NIST call leading to the SHA-3 competition was preceded by some concerning results regarding the state of hash functions, the CAESAR competition was preceded by some concerning failures in the state of authenticated encryption.

NORX[1], our submission to CAESAR, is a novel authenticated encryption scheme with associated data supporting an arbitrary parallelism degree, based on ARX primitives—namely ChaCha and BLAKE2—yet not using modular additions.

While modular addition, like xor or rotation, is a single-cycle operation in most processors, adder circuits have logarithmic depth [279], making ARX circuits larger and higher latency than equivalent circuits without addition. Furthermore, to protect against electromagnetic [200] and (differential) power analysis [278] side-channel attacks, secure implementations often use *masking* [208]—represent each value as the sum of one or more "shares", each of which randomized—but masking bitwise operations requires different (xor) masking than addition, which requires arithmetic masking. Converting between xor and arithmetic masking is costly [207], and this makes additions comparatively more costly in this setting. As such, while ARX constructions perform very well on software, once again they create a tradeoff between safety and performance when implemented in other scenarios, as described above.

NORX has a unique parallel architecture based on the monkeyDuplex construction [65, 71], where the parallelism degree and tag size can be tuned arbitrarily. An original domain separation scheme allows simple processing of header/payload/trailer data. NORX was optimized for efficiency in both software and hardware, with a SIMD-friendly core, almost byte-aligned rotations, no secret-dependent memory lookups, and only bitwise operations. The NORX core is inspired by the ARX primitive ChaCha [54], however it replaces integer addition with

[1]The name stems from "NO(T A)RX" and is pronounced like "norcks".

the approximation $a \oplus b \oplus (a \wedge b) \lll 1$[2]. This simplifies cryptanalysis and improves hardware efficiency. Furthermore, NORX specifies a dedicated datagram format to facilitate interoperability and avoid users the trouble of defining custom encoding and signaling.

NORX traces its heritage back to the following cryptographic algorithms and constructions:

- The basic layout relies on the duplex construction [71], which allows to construct an efficient and flexible AEAD scheme from a single large permutation. It supports variable-length authentication tags and an easy realization of trade-offs between performance and security.

- The core permutation is inspired by ChaCha [54] and BLAKE2 [32], which both have been intensively analyzed [25, 423] (Section 4.4) and offer excellent performance.

The remainder of this chapter is organized as follows: Section 5.1 describes our goals when designing NORX. Section 5.2 specifies the NORX family of authenticated ciphers, with the rationale for each component being elaborated in Section 5.3. Finally, Section 5.4 discusses the performance of NORX in various platforms, and Section 5.5 discusses the security of the NORX mode of operation.

## 5.1 Design Goals and Characteristics

NORX was designed with end users in mind, provides several features desirable for practical applications, and offers a number of advantages over AES-GCM [177]. In the following we list our design goals for NORX, along with a description of how they are achieved in the final product.

**High security** NORX supports 128- and 256-bit keys and authentication tags of arbitrary size, thanks to the duplex construction. The core permutation of NORX was designed and evaluated to be cryptographically strong. The minimal number of 8 rounds for initialization / finalization (i.e. 16 steps consisting of 8 column and 8 diagonal steps interleaved with each other) and of 4 rounds (i.e. 8 steps consisting of 4 column and 4 diagonal steps interleaved with each other) for the data processing part ensure a high security margin against cryptanalytic attacks. Large internal states of 512 and 1024 bits and the duplex construction offer protection against generic attacks.

[2]Derived from the well-known identity $a + b = (a \oplus b) + (a \wedge b) \lll 1$ [42, 276], already implicitly used in the IAS machine for parallel integer addition [113, page 17].

**Efficiency** NORX was designed with 64-bit processors in mind, but is also compatible with smaller architectures like 8- to 32-bit platforms. Software implementations of NORX are able to take advantage of multi-core processors, due to the parallel duplex construction, and specialized instruction sets like AVX / AVX2 or NEON. Moreover, state sizes of 512 and 1024 bits make NORX very cache-friendly. Hardware implementations benefit from hardware-friendly operations, next to the arbitrary parallelism degree for payload processing, which results in highly competitive hardware performance of NORX.

**Simplicity** The core algorithm iterates a simple round function and can be implemented by translating our pseudocode into a programming language of choice. NORX requires no S-boxes, no Galois field operations, and no integer arithmetic; AND, XOR, and shifts are the only instructions required. This simplifies cryptanalysis and the task of implementing the cipher.

**High key agility** NORX requires no key expansion when setting up a new key, in contrast to many blockcipher-based schemes, like AES-GCM. Switching the secret key is therefore very cheap. As an additional benefit, there are also no hidden costs of loading precomputed expanded keys from DRAM into L1 cache.

**Adjustable tag sizes** The NORX family uses a default tag size of $4w$ bits for our proposed instances. Thanks to the duplex construction, tag sizes can be easily adapted to the demands of any given application.

**Simple integration** NORX can be easily integrated into a protocol stack, as it supports flexible processing of arbitrary datagrams: any header and trailer are authenticated (and left in clear) and the payload is both encrypted and authenticated.

**Interoperability** Dedicated datagrams encode parameters of the cipher and encapsulate the protected data. This aims to increase interoperability across implementations.

**Single pass** Encryption and decryption of data is done in a single pass of the algorithm.

**Online** NORX supports encryption of data streams, i.e. the size of processed data needs not to be known in advance.

**High data processing volume** NORX allows to process very large data sizes from a single key-nonce pair. The usage exponent (see Section 5.5) theoretically limits the number of calls to the core permutation to values of $2^{64}$ (NORX32) and $2^{128}$ (NORX64). This translates to data volumes orders of magnitude beyond

everything relevant for current real-world applications. In particular, these values are a lot higher than the maximum of $2^{32}$ calls to the authenticated encryption function of AES-GCM, which could be easily reached already nowadays in practical applications.

**Minimal overhead** Payload encryption is non-expanding, i.e. the ciphertext has the same length as the plaintext. The authentication tag has a length of 16 or 32 bytes depending on the concrete instance of NORX.

**Robustness against side-channel attacks** By avoiding data-dependent table lookups, like S-boxes, and integer additions, the goal to harden both software and hardware implementations of NORX against timing and (differential) power attacks should be comparably easy to achieve.

**Moderate misuse resistance** NORX retains its security on nonce reuse as long as it can be guaranteed that header data is unique[3]. For comparison, nonce reuse in AES-GCM is a major security issue, allowing an attacker to recover the secret key [247].

**Autonomy** NORX is self-contained and requires no external primitive.

**Diversity** The cipher does not depend on AES instructions, thereby adding to the diversity among cryptographic algorithms.

**Extensibility** Thanks to the duplex construction and a simple, yet powerful domain separation scheme, NORX can be easily extended to support additional features, like secret message numbers, sessions, or forward secrecy without losing its security guarantees.

## 5.2 Specification

This section gives a complete specification of NORX and its proposed instances.

### 5.2.1 Parameters and Interface

A NORX instance is parameterized by

- a *word size* of $w \in \{32, 64\}$ bits,
- a *round number* $1 \leq l \leq 63$,

[3]Nevertheless, the designers discourage this approach, and recommend that nonce freshness should be ensured by all means.

- a *parallelism degree* $0 \leq p \leq 255$,
- a *tag size* of $t \leq 4w$ bits.

**Encryption Mode**

NORX encryption takes as input

- a *key* $K$ of $k = 4w$ bits,
- a *nonce* $N$ of $n = 2w$ bits,
- a *datagram* $(A, M, Z)$ where
  - $A$ is a *header*,
  - $M$ is a *message*,
  - $Z$ is a *trailer/footer*,

  and where any of $A$, $M$, $Z$ can be the empty string (that is, of length 0).

NORX encryption produces as output

- a *ciphertext* (or *encrypted payload*) $C$ of the same size as $M$,
- an *authentication tag* $T$ of $t$ bits.

In summary, NORX encryption $\mathsf{E}$ is specified as

$$\mathsf{E} : \{0,1\}^k \times \{0,1\}^n \times \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^t$$

with

$$\mathsf{E}(K, N, A, M, Z) = (C, T)$$

where $|M| = |C|$.

**Decryption Mode**

NORX decryption takes as input

- a key $K$ of $k = 4w$ bits,
- a nonce $N$ of $n = 2w$ bits,
- a datagram $(A, C, Z)$ where,
  - $A$ is a header,

Table 5.1: NORX instances

| $w$ | $l$ | $p$ | $t$ | $k$ | $n$ |
|---|---|---|---|---|---|
| 64 | 4 | 1 | 256 | 256 | 128 |
| 32 | 4 | 1 | 128 | 128 | 64 |
| 64 | 6 | 1 | 256 | 256 | 128 |
| 32 | 6 | 1 | 128 | 128 | 64 |
| 64 | 4 | 4 | 256 | 256 | 128 |

  - $C$ is a ciphertext,
  - $Z$ is a trailer,

  and where any of $A$, $M$, $Z$ can be the empty string (that is, of length 0).

- an authentication tag $T$ of $t$ bits.

NORX decryption either returns a failure $\perp$, upon failed verification of the tag, or produces a plaintext $M$ of the same size as $C$ if the tag verification succeeds.

In summary, NORX decryption D is specified as

$$\mathsf{D} : \{0,1\}^k \times \{0,1\}^n \times \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^t \to \{0,1\}^* \cup \{\perp\}$$

with

$$\mathsf{D}(K,N,A,C,Z,T) = \begin{cases} M & \text{if } T = T' \\ \perp & \text{if } T \neq T' \end{cases}$$

where $T$ denotes the received authentication tag, $T'$ the one computed on the recipient's side and $|M| = |C|$.

### 5.2.2 Instances

A NORX instance is a choice of values for the four parameters $w$, $l$, $p$, and $t$. Table 5.1 proposes five NORX instances for different use cases: 128- or 256-bit security, four or six rounds, and a version with 4-way parallelism. Table 5.1 also shows the corresponding nonce and key sizes $n$ and $k$.

We set the *default tag size* $t$ for a given word size $w$ to $t = 4w$, i.e. for $w = 32$ we get $t = 128$ and for $w = 64$ we get $t = 256$.

A NORX instance is denoted by NORX$w$-$l$-$p$-$t$, where $w$, $l$, $p$, and $t$ are the parameters of the instance, see Section 5.2.1. If the default tag size is used, i.e. if $t = 4w$, the notation for an instance is shortened to NORX$w$-$l$-$p$. So for example, NORX64-6-1 has $(w,l,p,t) = (64,6,1,256)$.

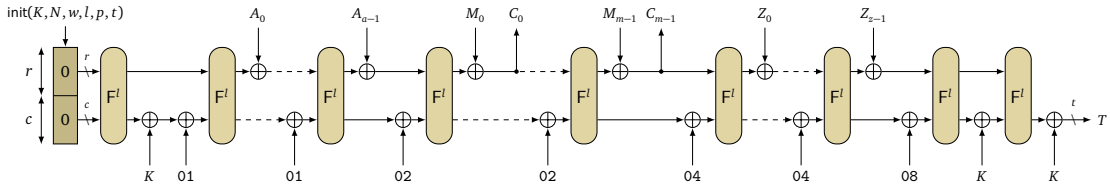



Figure 5.1: Layout of NORX for $p = 1$

We consider NORX32-4-1 and NORX64-4-1 as the standard instances for the respective word sizes of 32 and 64 bit. These configurations offer a good balance between performance and security. We recommend NORX32-4-1 for low resource applications on 8- to 32-bit platforms and NORX64-4-1 for software implementations on modern 64-bit CPUs or standard hardware implementations. Applications that require a higher security margin and where performance has less priority are advised to use the instances NORX32-6-1 and NORX64-6-1.

For use cases where very high data throughput is necessary, we recommend NORX64-4-4, which allows payload encryption on four parallel lanes, thus enabling very high data processing speeds. Finally, we advise hardware implementers not to realize multiple instances of NORX with different parameter combinations at the same time. This holds especially for different values of the parallelism degree $p$. An implementation should rather be optimized for one set of parameters to gain higher efficiency.

### 5.2.3 Layout Overview

NORX is based on the monkeyDuplex construction [65, 71] extended with the capability to process payloads in parallel. The number $i$ of parallel encryption lanes $L_i$ is controlled by the parameter $0 \leq p \leq 255$. For the value $p = 1$, the layout of NORX corresponds to a standard (sequential) duplex construction, as in Figure 5.1. For $p > 1$, the number of lanes $L_i$ is bounded by the latter value, e.g. for $p = 2$ see Figure 5.2. If $p = 0$, the number of lanes $L_i$ is bounded by the size of the payload. In that case, the layout of NORX is similar to that of the PPAE construction [92, 93].

The core algorithm F of NORX is a permutation of $b = r + c$ bits, where $b$ is called the *width*, $r$ the *rate* (or block length), and $c$ the *capacity*. We call F a *round* and $\mathsf{F}^l$ denotes its $l$-fold iteration. The organization of the internal state $S$ of NORX is as follows:

| $w$ | $b$ | $r$ | $c$ |
|---|---|---|---|
| 32 | 512 | 384 | 128 |
| 64 | 1024 | 768 | 256 |

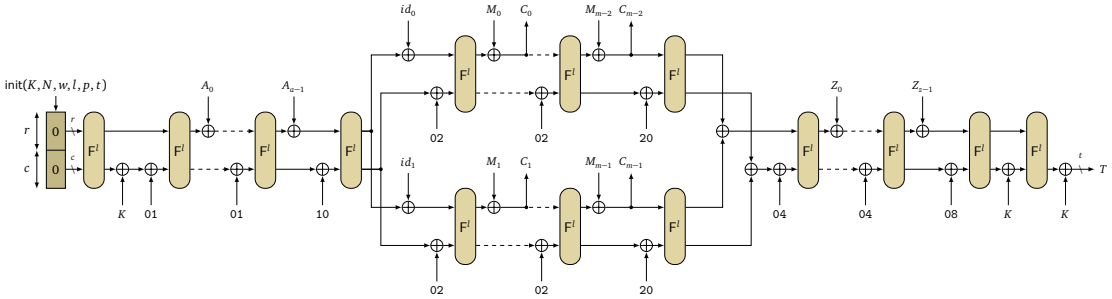



Figure 5.2: Layout of NORX for $p = 2$

The state is viewed as a concatenation of 16 words, i.e. $S = s_0 \parallel \cdots \parallel s_{15}$, where $s_0, \ldots, s_{11}$ are called the *rate words* (where data blocks are injected) $s_{12}, \ldots, s_{15}$ are called the *capacity words* (which remain out of attacker control). Conceptually, the 16 state words are arranged in a $4 \times 4$ matrix:

$$S = \left(\begin{array}{cccc} s_0 & s_1 & s_2 & s_3 \\ s_4 & s_5 & s_6 & s_7 \\ s_8 & s_9 & s_{10} & s_{11} \\ \hline s_{12} & s_{13} & s_{14} & s_{15} \end{array}\right)$$

### 5.2.4 The Permutation $\mathsf{F}^l$

The complete pseudocode for the NORX core permutation $\mathsf{F}^l$ is given in Figure 5.4. A single NORX round F processes the state $S$ by first transforming its columns with

$$\mathsf{G}(s_0, s_4, s_8, s_{12}) \quad \mathsf{G}(s_1, s_5, s_9, s_{13}) \quad \mathsf{G}(s_2, s_6, s_{10}, s_{14}) \quad \mathsf{G}(s_3, s_7, s_{11}, s_{15})$$

and then transforming its diagonals with

$$\mathsf{G}(s_0, s_5, s_{10}, s_{15}) \quad \mathsf{G}(s_1, s_6, s_{11}, s_{12}) \quad \mathsf{G}(s_2, s_7, s_8, s_{13}) \quad \mathsf{G}(s_3, s_4, s_9, s_{14})$$

Those two operations are called *column step* and *diagonal step*, as in BLAKE2, cf. the similarity with Figure 4.1, and will be denoted by col and diag, respectively. An illustration of these operations is shown in Figure 5.3.

The G function uses xor, *cyclic rotations* $\ggg$ and a *non-linear operation* H interchangeably to update its four input words $a$, $b$, $c$ and $d$. The rotation offsets $r_0$, $r_1$, $r_2$, and $r_3$ for the cyclic rotations of 32- and 64-bit NORX are specified in Table 5.2.

### 5.2.5 The NORX Mode

The NORX mode is divided into a high-level and a low-level API. The *high-level interface* consists of only two functions: AEADEnc and AEADDec. These provide

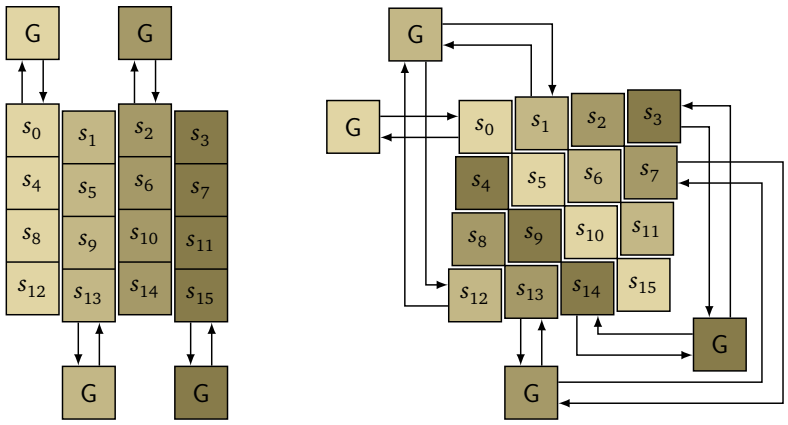



Figure 5.3: Column step and diagonal step of F

Table 5.2: Rotation offsets for 32- and 64-bit NORX

| $w$ | $r_0$ | $r_1$ | $r_2$ | $r_3$ |
|---|---|---|---|---|
| 32 | 8 | 11 | 16 | 31 |
| 64 | 8 | 19 | 40 | 63 |

**Algorithm:** $\mathsf{F}^l(S)$

**for** $i \in \{0, \ldots, l-1\}$ **do**
1 $\quad S \leftarrow \mathsf{diag}(\mathsf{col}(S))$;
**end**
2 **return** $S$;

**Algorithm:** $\mathsf{G}(a, b, c, d)$

1 $a \leftarrow \mathsf{H}(a, b)$;
2 $d \leftarrow (a \oplus d) \ggg r_0$;
3 $c \leftarrow \mathsf{H}(c, d)$;
4 $b \leftarrow (b \oplus c) \ggg r_1$;
5 $a \leftarrow \mathsf{H}(a, b)$;
6 $d \leftarrow (a \oplus d) \ggg r_2$;
7 $c \leftarrow \mathsf{H}(c, d)$;
8 $b \leftarrow (b \oplus c) \ggg r_3$;
9 **return** $a, b, c, d$;

**Algorithm:** $\mathsf{col}(S)$

1 $s_0, s_4, s_8, s_{12} \leftarrow \mathsf{G}(s_0, s_4, s_8, s_{12})$;
2 $s_1, s_5, s_9, s_{13} \leftarrow \mathsf{G}(s_1, s_5, s_9, s_{13})$;
3 $s_2, s_6, s_{10}, s_{14} \leftarrow \mathsf{G}(s_2, s_6, s_{10}, s_{14})$;
4 $s_3, s_7, s_{11}, s_{15} \leftarrow \mathsf{G}(s_3, s_7, s_{11}, s_{15})$;
5 **return** $S$;

**Algorithm:** $\mathsf{diag}(S)$

1 $s_0, s_5, s_{10}, s_{15} \leftarrow \mathsf{G}(s_0, s_5, s_{10}, s_{15})$;
2 $s_1, s_6, s_{11}, s_{12} \leftarrow \mathsf{G}(s_1, s_6, s_{11}, s_{12})$;
3 $s_2, s_7, s_8, s_{13} \leftarrow \mathsf{G}(s_2, s_7, s_8, s_{13})$;
4 $s_3, s_4, s_9, s_{14} \leftarrow \mathsf{G}(s_3, s_4, s_9, s_{14})$;
5 **return** $S$;

**Algorithm:** $\mathsf{H}(x, y)$

1 **return** $(x \oplus y) \oplus ((x \wedge y) \ll 1)$;

Figure 5.4: The NORX permutation $\mathsf{F}^l$

**Algorithm:** AEADEnc$(K, N, A, M, Z)$

1 $S \leftarrow \mathsf{initialize}(K, N)$;
2 $S \leftarrow \mathsf{absorb}(S, A, 01)$;
3 $\overline{S} \leftarrow \mathsf{branch}(S, |M|, 10)$;
4 $\overline{S}, C \leftarrow \mathsf{encrypt}(\overline{S}, M, 02)$;
5 $S \leftarrow \mathsf{merge}(\overline{S}, |M|, 20)$;
6 $S \leftarrow \mathsf{absorb}(S, Z, 04)$;
7 $S, T \leftarrow \mathsf{finalize}(S, 08)$;
8 **return** $C, T$;

**Algorithm:** AEADDec$(K, N, A, C, Z, T)$

1 $S \leftarrow \mathsf{initialize}(K, N)$;
2 $S \leftarrow \mathsf{absorb}(S, A, 01)$;
3 $\overline{S} \leftarrow \mathsf{branch}(S, |C|, 10)$;
4 $\overline{S}, M \leftarrow \mathsf{decrypt}(\overline{S}, C, 02)$;
5 $S \leftarrow \mathsf{merge}(\overline{S}, |C|, 20)$;
6 $S \leftarrow \mathsf{absorb}(S, Z, 04)$;
7 $S, T' \leftarrow \mathsf{finalize}(S, 08)$;
8 **if** $T = T'$ **then return** $M$ **else return** $\perp$;

Figure 5.5: High-level interface functions of the NORX mode

functionality for encryption and authentication of a message on the one hand and decryption and verification of an encrypted payload on the other. Both functions of course also support processing of associated data. The *low-level interface* defines the concrete implementation of padding, domain separation, absorption or encryption of data block sequences, tag generation, etc.

### High-level Structure

The two high-level interface functions AEADEnc and AEADDec are depicted in Figure 5.5.

### Low-level Structure

The low-level functions of NORX are depicted in Figure 5.6. Before going into the details of those methods, we first introduce the mechanisms for padding and domain separation which are required later on.

### Padding

NORX adopts the so-called *multi-rate padding* [71]. This padding rule is defined by the map

$$\mathsf{pad}_r : X \longmapsto X \parallel 10^u 1$$

where $X$ is a bitstring and $u = (-|X| - 2) \bmod r$. If $r$ and $|X|$ are divisible by 8 and $X$ is viewed as a sequence of bytes, then the multi-rate padding can be written as

$$\mathsf{pad}_r : \begin{cases} X \longmapsto X \parallel 01 \parallel 00^u \parallel 80 & \text{if } |X|_8 \neq -1 \pmod{r/8} \\ X \longmapsto X \parallel 81 & \text{otherwise} \end{cases}$$

**Algorithm:** initialize$(K, N)$

$k_0 \| k_1 \| k_2 \| k_3 \leftarrow K$, s.t. $|k_i| = w$;
$n_0 \| n_1 \| n_2 \| n_3 \leftarrow N$, s.t. $|n_i| = w$;
$S \leftarrow \begin{pmatrix} n_0 & n_1 & n_2 & n_3 \\ k_0 & k_1 & k_2 & k_3 \\ u_8 & u_9 & u_{10} & u_{11} \\ u_{12} & u_{13} & u_{14} & u_{15} \end{pmatrix}$;
$(s_{12}, s_{13}, s_{14}, s_{15}) \leftarrow (s_{12}, s_{13}, s_{14}, s_{15}) \oplus (w, l, p, t)$;
$S \leftarrow \mathsf{F}^l(S)$;
$(s_0, s_1, s_2, s_3) \leftarrow (s_0, s_1, s_2, s_3) \oplus (k_0, k_1, k_2, k_3)$;
**return** $S$;

**Algorithm:** encrypt$(\overline{S}, M, v)$

$C \leftarrow \varepsilon$;
$M_0 \| \cdots \| M_{m-1} \leftarrow M$, s.t. $|M_i| = r, 0 \leq |M_{m-1}| < r$;
**if** $|M| > 0$ **then**
  **for** $i \in \{0, \ldots, m-2\}$ **do**
    $j \leftarrow i \bmod |\overline{S}|_b$;
    $\overline{s}_{j,15} \leftarrow \overline{s}_{j,15} \oplus v$;
    $\overline{S}_j \leftarrow \mathsf{F}^l(\overline{S}_j)$;
    $C_i \leftarrow \mathsf{left}_r(\overline{S}_j) \oplus M_i$;
    $\overline{S}_j \leftarrow C_i \| \mathsf{right}_c(\overline{S}_j)$;
  **end**
  $j \leftarrow (m-1) \bmod |\overline{S}|_b$;
  $\overline{s}_{j,15} \leftarrow \overline{s}_{j,15} \oplus v$;
  $\overline{S}_j \leftarrow \mathsf{F}^l(\overline{S}_j)$;
  $C_{m-1} \leftarrow \mathsf{left}_{|M_{m-1}|}(\overline{S}_j) \oplus M_{m-1}$;
  $\overline{S}_j \leftarrow \overline{S}_j \oplus (\mathsf{pad}_r(M_{m-1}) \| 0^c)$;
  $C \leftarrow C_0 \| \cdots \| C_{m-1}$;
**end**
**return** $\overline{S}, C$;

**Algorithm:** decrypt$(\overline{S}, C, v)$

$M \leftarrow \varepsilon$;
$C_0 \| \cdots \| C_{m-1} \leftarrow C$ s.t. $|C_i| = r, 0 \leq |C_{m-1}| < r$;
**if** $|C| > 0$ **then**
  **for** $i \in \{0, \ldots, m-2\}$ **do**
    $j \leftarrow i \bmod |\overline{S}|_b$;
    $\overline{s}_{j,15} \leftarrow \overline{s}_{j,15} \oplus v$;
    $\overline{S}_j \leftarrow \mathsf{F}^l(\overline{S}_j)$;
    $M_i \leftarrow \mathsf{left}_r(\overline{S}_j) \oplus C_i$;
    $\overline{S}_j \leftarrow C_i \| \mathsf{right}_c(\overline{S}_j)$;
  **end**
  $j \leftarrow (m-1) \bmod |\overline{S}|_b$;
  $\overline{s}_{j,15} \leftarrow \overline{s}_{j,15} \oplus v$;
  $\overline{S}_j \leftarrow \mathsf{F}^l(\overline{S}_j)$;
  $M_{m-1} \leftarrow \mathsf{left}_{|C_{m-1}|}(\overline{S}_j) \oplus C_{m-1}$;
  $\overline{S}_j \leftarrow \overline{S}_j \oplus (\mathsf{pad}_r(M_{m-1}) \| 0^c)$;
  $M \leftarrow M_0 \| \cdots \| M_{m-1}$;
**end**
**return** $\overline{S}, M$;

**Algorithm:** absorb$(S, X, v)$

$X_0 \| \cdots \| X_{m-1} \leftarrow X$, s.t. $|X_i| = r, 0 \leq |X_{m-1}| < r$;
**if** $|X| > 0$ **then**
  **for** $i \in \{0, \ldots, m-2\}$ **do**
    $s_{15} \leftarrow s_{15} \oplus v$;
    $S \leftarrow \mathsf{F}^l(S)$;
    $S \leftarrow S \oplus (X_i \| 0^c)$;
  **end**
  $s_{15} \leftarrow s_{15} \oplus v$;
  $S \leftarrow \mathsf{F}^l(S)$;
  $S \leftarrow S \oplus (\mathsf{pad}_r(X_{m-1}) \| 0^c)$;
**end**
**return** $S$;

**Algorithm:** branch$(S, m, v)$

$\overline{S} \leftarrow 0^b$;
**if** $p \neq 1$ *and* $m > 0$ **then**
  $s \leftarrow p$;
  **if** $p = 0$ **then**
    $s \leftarrow \lceil m/r \rceil$;
  **end**
  $\overline{S} = (\overline{S}_0, \ldots, \overline{S}_{s-1}) \leftarrow (0^b, \ldots, 0^b)$;
  $s_{15} \leftarrow s_{15} \oplus v$;
  $S \leftarrow \mathsf{F}^l(S)$;
  **for** $i \in \{0, \ldots, s-1\}$ **do**
    $\overline{S}_i \leftarrow S \oplus (i, i, i, i, i, i, i, i, i, i, i, i, 0, 0, 0, 0)$;
  **end**
**else**
  $\overline{S} \leftarrow S$;
**end**
**return** $\overline{S}$;

**Algorithm:** merge$(\overline{S}, m, v)$

$S \leftarrow 0^b$;
**if** $p \neq 1$ *and* $m > 0$ **then**
  **for** $i \in \{0, \ldots, |\overline{S}|_b - 1\}$ **do**
    $\overline{s}_{i,15} \leftarrow \overline{s}_{i,15} \oplus v$;
    $\overline{S}_i \leftarrow \mathsf{F}^l(\overline{S}_i)$;
    $S \leftarrow S \oplus \overline{S}_i$;
  **end**
**else**
  $S \leftarrow \overline{S}$;
**end**
**return** $S$;

**Algorithm:** finalize$(S, K, v)$

$s_{15} \leftarrow s_{15} \oplus v$;
$S \leftarrow \mathsf{F}^l(S)$;
$(s_0, s_1, s_2, s_3) \leftarrow (s_0, s_1, s_2, s_3) \oplus (k_0, k_1, k_2, k_3)$;
$s_{15} \leftarrow s_{15} \oplus v$;
$S \leftarrow \mathsf{F}^l(S)$;
$(s_0, s_1, s_2, s_3) \leftarrow (s_0, s_1, s_2, s_3) \oplus (k_0, k_1, k_2, k_3)$;
$T \leftarrow \mathsf{left}_t(S)$;
**return** $S, T$;

Figure 5.6: Low-level interface functions of the NORX mode

Table 5.3: Domain separation constants

| header | payload | trailer | tag | branching | merging |
|---|---|---|---|---|---|
| 01 | 02 | 04 | 08 | 10 | 20 |

Table 5.4: Initialization constants

| $w$ | 32 | 64 | $w$ | 32 | 64 |
|---|---|---|---|---|---|
| $u_0$ | 0454EDAB | E4D324772B91DF79 | $u_8$ | A3D8D930 | B15E641748DE5E6B |
| $u_1$ | AC6851CC | 3AEC9ABAAEB02CCB | $u_9$ | 3FA8B72C | AA95E955E10F8410 |
| $u_2$ | B707322F | 9DFBA13DB4289311 | $u_{10}$ | ED84EB49 | 28D1034441A9DD40 |
| $u_3$ | A0C7C90D | EF9EB4BF5A97F2C8 | $u_{11}$ | EDCA4787 | 7F31BBF964E93BF5 |
| $u_4$ | 99AB09AC | 3F466E92C1532034 | $u_{12}$ | 335463EB | B5E9E22493DFFB96 |
| $u_5$ | A643466D | E6E986626CC405C1 | $u_{13}$ | F994220B | B980C852479FAFBD |
| $u_6$ | 21C22362 | ACE40F3B549184E1 | $u_{14}$ | BE0BF5C9 | DA24516BF55EAFD4 |
| $u_7$ | 1230C950 | D9CFD35762614477 | $u_{15}$ | D7C49104 | 86026AE8536F1501 |

where $u = (-|X|_8 - 2) \bmod (r/8)$.

### Domain Separation

NORX has a very simple and lightweight domain separation mechanism: it is performed by xoring a *domain separation constant* to the least significant byte of $s_{15}$ each time before the state $s$ is transformed by the permutation $\mathsf{F}^l$. Distinct constants are used for the different algorithm phases, i.e. for the three different message processing stages, for tag generation, and in case of $p \neq 1$, for branching and merging steps. Table 5.3 gives the specification of those constants and Figures 5.1 and 5.2 illustrate their integration into the state of NORX. Figures 5.5 and 5.6 show their concrete usage.

### Initialization

The method initialize sets up the $16w$-bit internal state $S = (s_0, \ldots, s_{15})$ of NORX by processing a $4w$-bit key $K = k_0 \parallel k_1 \parallel k_2 \parallel k_3$, a $2w$-bit nonce $N = n_0 \parallel n_1$, the instance parameters $w$, $l$, $p$, and $t$ and some initialization constants. These constants are given in Table 5.4 and can be derived by

$$(u_0, \ldots, u_{15}) = \mathsf{F}^2(0, \ldots, 15)$$

which allows on-the-fly computation if necessary. Note, however, that only $u_2, u_3, u_8, \ldots, u_{15}$ are actually used in initialize.

### Data Absorption

The method absorb takes an arbitrary long bitstring $X$ as input and absorbs it in blocks of $r$ bits into the internal state thereby ensuring authenticity of $X$. If the last block is smaller than $r$ bits, it is extended to the block size through $\mathsf{pad}_r$. For domain separation the constant $v$ is used. Data absorption is skipped entirely in case the input has length 0, i.e. if $X$ corresponds to the empty bitstring $\varepsilon$.

In NORX the function absorb is used for authenticating associated data in the form of header data $A$ using domain separation constant $v = 01$ and/or trailer data $Z$ using domain separation constant $v = 04$. Refer to the high-level interface in Figure 5.5 to see where and how absorb is used concretely in NORX.

### Branching

If the parallelism degree $p \neq 1$ then branch is used to prepare parallel payload processing. branch is skipped entirely if either $p = 1$ or $|P| = 0$. The state $S$ is extended to a multi-state vector $\overline{S}$ having either $p$ elements if $p > 1$ or $\lceil |M|_r \rceil$ elements if $p = 0$. Note that in order to ensure that each lane produces a unique bitstream for encryption, a *lane number* $i$ is integrated into state copy $\overline{S}_i$ (included into words $\overline{s}_{i,0}$ to $\overline{s}_{i,11}$) together with the domain separation constant $v = 10$.

### Data Encryption and Decryption

The method encrypt (decrypt) takes an arbitrary long bitstring $P$ ($C$) as input and encrypts (decrypts) it thereby producing the encrypted (decrypted) payload $C$ ($P$). Since $P$ is also absorbed into the state $S$, its authenticity is ensured as well. As in absorb, data is processed in $r$-bit blocks and the last block is padded using $\mathsf{pad}_r$. Note that in the latter case only a truncated data block of the same size as the unpadded input block is extracted such that $|P| = |C|$ holds. The constant $v = 02$ is used for domain separation.

The different cases for $p$ are handled as follows. For $p = 1$ the NORX mode corresponds to a regular sequential sponge construction and no special steps have to be taken for data encryption or decryption. For $p > 1$ a fixed number of $p$ parallel lanes is available for data processing. Data blocks are rotated in a round-robin fashion across the states by assigning the $i$-th data block to state $i \bmod p$. In the last case, if $p = 0$, each data block is processed on its own separate lane.

### Merging

The merge function is only executed if $p \neq 1$ and $|M| > 0$. After parallel-processing all payload data blocks, the states $\overline{S}_i$ are merged back into a single state $S$. The domain separation constant for merge is $v = 20$.

#### Finalization

The finalize function generates an authentication tag $T$ by first injecting the domain separation constant $v = 08$ then transforming $S$ *twice* with the permutation $\mathsf{F}^l$ and finally extracting the $t$ leftmost bits from $s_0 \| \cdots \| s_{11}$ which are returned as the tag $T$.

#### Tag Verification

Note that tag verification is not listed explicitly among the low-level interface functions in Figure 5.6 but rather in Figure 5.5, see the last step of AEADDec.

Tag verification consists of comparing the *received tag* $T$ to the *generated tag* $T'$. If $T = T'$, tag verification succeeds; otherwise tag verification fails, the decrypted payload is discarded and an error $\perp$ is returned.

Implementations of tag verification should satisfy the following requirements:

- Tag verification should not leak information on the (relative) values of the strings compared. In particular tag verification should be implemented in constant time, so that a comparison of identical strings take the same time as a comparison of distinct strings.
- The decrypted payload should not be returned to the user if tag verification fails. Ideally, extracted bytes should be securely erased from any temporary memory if tag verification fails.

### 5.2.6 Datagrams

Many issues with encryption interoperability are due to ad hoc ways to represent and transport cryptograms and the associated data. For example IVs are sometimes prepended to the ciphertext, sometimes appended, or sent separately. We thus specify datagrams that can be integrated in a protocol stack, encapsulating the ciphertext as a payload. Using a standardized encoding simplifies the transmission of NORX cryptograms across different APIs, and reduces the risk of insecure or suboptimal encodings. We specify two distinct types of datagrams, depending on whether the NORX parameters are fixed or need to be signaled in the datagram header.

#### Fixed Parameters

With *fixed parameters* shared by the parties (for example through the application using NORX), there is no need to include the parameters in the *header of the datagram*[4].

[4]The header referred to is that of the datagram specified, not that of the data processed by the NORX instance.

The datagram for fixed parameters thus only needs to contain $N$, $A$, $C$, $Z$, and $T$, as well as information to parse those elements.

We encode the byte length of $A$ and $Z$ on 16 bits, allowing for headers and trailers of up to 64 KiB, a large enough value for most real applications. The byte length of the encrypted payload is encoded on 32 bits for NORX32 and on 64 bits for NORX64, which translates to a maximum payload size of 4 GiB and 16 EiB, respectively[5]. Similarly to frame check sequences in data link protocols, the tag is added as a *trailer of the datagram* specified. The header, encrypted payload, and trailer of the underlying protocol are viewed as the *payload of the datagram*. The default tag length being a constant value of the NORX instance, it needs not be signaled.

Tables 5.5 and 5.6 show the fixed-parameters datagrams for NORX32 and NORX64. The length of the datagram header is 28 bytes for NORX64 and 16 bytes for NORX32.

Note that the CAESAR API (as per the final call, see [114]) receives the nonce and the associated data in two separate buffers, but the tag is included in the ciphertext buffer.

#### Variable Parameters

With *variable parameters*, the datagram needs to signal the values of $w$, $l$, and $p$. The header is thus extended to encode those values, as specified in Tables 5.7 and 5.8. To minimize bandwidth, $w$ is encoded on one bit, supporting the two choices 32-bit ($w = 0$) and 64-bit ($w = 1$), $l$ on 7 bits (with the MSB fixed at 0, i.e. supporting up to 63 rounds), and $p$ on 8 bits (supporting parallelization degree up to 255). The datagram header is thus only 2 bytes longer than the header for fixed parameters.

## 5.3 Design Rationale

In this section we motivate the design choices made in NORX. We pursue a top-down approach, starting with the general layout and going into the details of NORX's components in the later sections.

### 5.3.1 The Parallel Duplex Construction

The layout of NORX is based on the monkeyDuplex construction [65, 71], but enhanced by the capability of parallel payload processing on multiple lanes (cf. Figures 5.1 and 5.2). The *parallel duplex construction* is similar to the tree-hashing mode for sponge functions [66]. It allows NORX to take advantage of multicore processors and enables high-throughput hardware implementations. Associated data can be

[5]Note that NORX is capable of (safely) processing much larger data sizes, those are just the maximum values when our proposed datagrams are used.

Table 5.5: NORX32 datagram for fixed parameters (offsets are in bytes)

| Offset | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| **0**<br>**4** | Nonce $N$ | | | |
| **8** | Header byte length $\|A\|$ | | Trailer byte length $\|Z\|$ | |
| **12** | Encrypted payload byte length $\|C\|$ | | | |
| **16**<br>...<br>**??** | Header $A$ | | | |
| **??**<br>...<br>**??** | Encrypted payload $C$ | | | |
| **??**<br>...<br>**??** | Trailer $Z$ | | | |
| **??**<br>...<br>**??** | Tag $T$ | | | |

authenticated as header and/or trailer data but only on a single lane. We felt that it is not worth the effort to enable processing of $A$ and $Z$ in parallel, as they are usually rather short metadata. The number of encryption lanes is controlled by the parallelism degree $0 \leq p \leq 255$, which is a fixed instance parameter. Hence two instances with distinct $p$ values cannot decrypt data from each other. Obviously the same holds for differing $w$ and $l$ values.

To ensure that the payload blocks on parallel lanes are encrypted with distinct key streams, we use the branching phase to include an index into each of the parallel lanes. For NORX this index is a simple counter. Once the parallel payload processing is finished, the states are merged and NORX advances to the processing of the trailer (if present) or generation of the authentication tag.

### 5.3.2 The G Function

The G function of NORX is inspired by the quarter-round function of the stream cipher ChaCha [54], which itself is a variant of the quarter-round function of the eSTREAM finalist Salsa20 [59, 183]. Variants of ChaCha's quarter-round function can be found for example in the SHA-3 finalist BLAKE [28, 417] and BLAKE2 (Chapter 4).

Table 5.6: NORX64 datagram for fixed parameters (offsets are in bytes)

| Offset | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| **0**<br>**4**<br>**8**<br>**12** | Nonce $N$ | | | |
| **16** | Header byte length $\|A\|$ | | Trailer byte length $\|Z\|$ | |
| **20**<br>**24** | Encrypted payload byte length $\|C\|$ | | | |
| **28**<br>...<br>**??** | Header $A$ | | | |
| **??**<br>...<br>**??** | Encrypted payload $C$ | | | |
| **??**<br>...<br>**??** | Trailer $Z$ | | | |
| **??**<br>...<br>**??** | Tag $T$ | | | |

### Overview

One of the main goals for NORX was to design a core primitive that does not rely on integer addition to introduce nonlinearity. Instead it should use exclusively more hardware-friendly bitwise logic operations like NOT, AND, OR, or XOR and bit-shifts. Figure 5.7 shows how the G function of NORX transforms an input $(a, b, c, d)$ compared to the quarter-round function of ChaCha . The rotation offsets for NORX are specified in Table 5.2. The offsets of ChaCha are $(s_0, s_1, s_2, s_3) = (16, 12, 8, 7)$ for 32-bit and $(s_0, s_1, s_2, s_3) = (32, 24, 16, 63)$ for 64-bit.[6]

In NORX integer addition is replaced by the following expression

$$x \leftarrow (x \oplus y) \oplus ((x \wedge y) \ll 1)$$

which uses bitwise AND to introduce nonlinearity. It mimics integer addition of two bit strings $x$ and $y$ with a 1-bit carry propagation and thus provides, in addition to nonlinearity, also a slight diffusion of bits. In accordance with the main design

[6]The original ChaCha stream cipher is only defined for 32-bit words. For the 64-bit version we used the rotation offsets $(32, 24, 16, 63)$ from BLAKE2.

Table 5.7: NORX32 datagram for variable parameters (offsets are in bytes)

| Offset | **0** | **1** | **2** | **3** |
|---|---|---|---|---|
| **0**<br>**4** | Nonce $N$ | | | |
| **8** | Header byte length $\|A\|$ | | Trailer byte length $\|Z\|$ | |
| **12** | Encrypted payload byte length $\|C\|$ | | | |
| **16** | $w(1)\|\|l(7)$ | $p$ | | |
| **20**<br>...<br>**??** | Header $A$ | | | |
| **??**<br>...<br>**??** | Encrypted payload $C$ | | | |
| **??**<br>...<br>**??** | Trailer $Z$ | | | |
| **??**<br>...<br>**??** | Tag $T$ | | | |

$$
\begin{array}{rcl|rcl}
a & \leftarrow & (a \oplus b) \oplus \big((a \wedge b) \ll 1\big) & a & \leftarrow & a + b \\
d & \leftarrow & (a \oplus d) \ggg r_0 & d & \leftarrow & (a \oplus d) \ggg s_0 \\
c & \leftarrow & (c \oplus d) \oplus \big((c \wedge d) \ll 1\big) & c & \leftarrow & c + d \\
b & \leftarrow & (b \oplus c) \ggg r_1 & b & \leftarrow & (b \oplus c) \ggg s_1 \\
a & \leftarrow & (a \oplus b) \oplus \big((a \wedge b) \ll 1\big) & a & \leftarrow & a + b \\
d & \leftarrow & (a \oplus d) \ggg r_2 & d & \leftarrow & (a \oplus d) \ggg s_2 \\
c & \leftarrow & (c \oplus d) \oplus \big((c \wedge d) \ll 1\big) & c & \leftarrow & c + d \\
b & \leftarrow & (b \oplus c) \ggg r_3 & b & \leftarrow & (b \oplus c) \ggg s_3
\end{array}
$$

Figure 5.7: Comparison of NORX (left) and ChaCha (right) core functions

Table 5.8: NORX64 datagram for variable parameters (offsets are in bytes)

| Offset | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| **0**<br>**4**<br>**8**<br>**12** | Nonce $N$ | | | |
| **16** | Header byte length $\|A\|$ | | Trailer byte length $\|Z\|$ | |
| **20**<br>**24** | Encrypted payload byte length $\|C\|$ | | | |
| **28** | $w(1)\|\|l(7)$ | $p$ | | |
| **32**<br>...<br>**??** | Header $A$ | | | |
| **??**<br>...<br>**??** | Encrypted payload $C$ | | | |
| **??**<br>...<br>**??** | Trailer $Z$ | | | |
| **??**<br>...<br>**??** | Tag $T$ | | | |

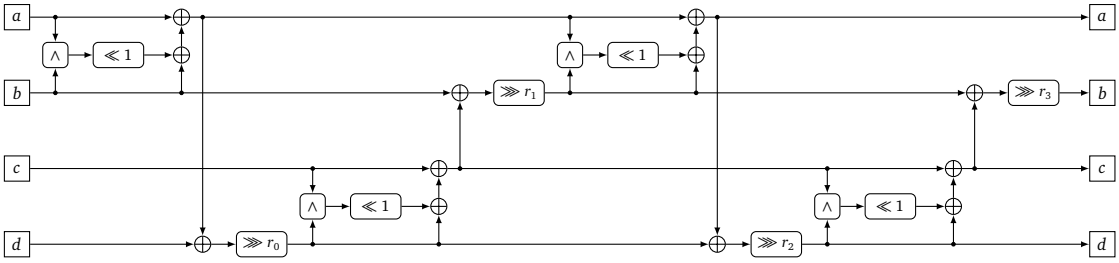



Figure 5.8: The G circuit

principle of NORX we tried to make the nonlinear operation as simple as possible in order to ease cryptanalysis and to reduce the risk of overlooking potential security weaknesses. Moving to simple bitwise logical operations also reduces the latency of hardware implementations. One way to instantiate G as a circuit is depicted in Figure 5.8.

### Bijectivity

The only expression in G which is not obviously invertible at a first glance, is the non-linear operation

$$z = (x \oplus y) \oplus ((x \wedge y) \ll 1)$$

with $n$-bit words $x$, $y$ and $z$. In order to proof bijectivity of the above expression we show how to invert it, under the assumption that one of its inputs is fixed. Therefore we write $x = \sum_{i=0}^{n-1} x_i \cdot 2^i$, $y = \sum_{i=0}^{n-1} y_i \cdot 2^i$ and $z = \sum_{i=0}^{n-1} z_i \cdot 2^i$ with $x_i$, $y_i$ and $z_i \in \{0, 1\}$ and assume that $y$ is fixed. Writing down the inverse nonlinear operation at bit level is straightforward:

$$\begin{aligned}
x_0 &= (z_0 \oplus y_0) \\
x_1 &= (z_1 \oplus y_1) \oplus (x_0 \wedge y_0) \\
&\vdots \\
x_i &= (z_i \oplus y_i) \oplus (x_{i-1} \wedge y_{i-1}) \\
&\vdots \\
x_{n-1} &= (z_{n-1} \oplus y_{n-1}) \oplus (x_{n-2} \wedge y_{n-2})
\end{aligned}$$

This proves that G is indeed a permutation. Further, it is a permutation when either of its input arguments is fixed, making it also a latin square.

### Features

The only operations required to define G are bitwise `XOR`, `AND` and logical bit shifts, which has several advantages: All of the mentioned instructions can be implemented in constant time regardless of the word size. Especially for hardware implementations there are no carry propagations to worry about, for example, as there would be for integer addition $\bmod 2^n$. This also makes masked implementations simpler and more efficient.

Moreover no table lookup instructions, like S-boxes, are required, where the table index is data-dependent. Those operations, if not handled with extreme care, are often the reason for implementations leaking side-channel information, making the affected algorithm vulnerable, e.g., to timing-attacks [53]. By avoiding them, the task of hardening the cipher against side-channel attacks gets obviously much easier. No specialized implementations are required, e.g., bitsliced S-boxes [213, 258, 330], for table lookups in constant time. Additionally, the absence of more complex instructions like integer addition, multiplication, Galois field arithmetic or other constructs based on linear algebra, has the effect that the algorithm is much

easier to implement (both in soft- and hardware) and thus reduces the threat of introducing accidental bugs.

### 5.3.3 The F Function

The layout of the round function F of NORX is the same as used in ChaCha [54].

#### Overview

Recall that F transforms a state $S = s_0 \| \cdots \| s_{15}$ in two phases. First a column step is applied

$$\mathsf{G}(s_0, s_4, s_8, s_{12}) \quad \mathsf{G}(s_1, s_5, s_9, s_{13}) \quad \mathsf{G}(s_2, s_6, s_{10}, s_{14}) \quad \mathsf{G}(s_3, s_7, s_{11}, s_{15})$$

followed by a diagonal step

$$\mathsf{G}(s_0, s_5, s_{10}, s_{15}) \quad \mathsf{G}(s_1, s_6, s_{11}, s_{12}) \quad \mathsf{G}(s_2, s_7, s_8, s_{13}) \quad \mathsf{G}(s_3, s_4, s_9, s_{14})$$

#### Bijectivity

As G is a permutation, F is obviously a permutation, too. This means that there exist no states $S$ and $S'$, with $S \neq S'$, which produce the same result, i.e. $\mathsf{F}^l(S) = \mathsf{F}^l(S')$, after any number of rounds $l$. This characteristic of F is important for the duplex construction [65, 71] in order to retain some desirable security properties.

#### Features

One great advantage of the ChaCha-related layout of F is that the modification of a single bit in the input has the chance of affecting all 16 output words after only one application of F. This greatly enhances diffusion. Another benefit of the layout is the ability to execute the four applications of G in a step completely in parallel, which improves performance.

### 5.3.4 Number of Rounds

For a higher protection of the key and authentication tag, e.g. against differential cryptanalysis, we chose twice the number of rounds for initialization and finalization, compared to the data processing phases. This measure was already proposed in [65] and while it has only minor effects on the overall performance, it increases the security margin of NORX. The minimal value of $l = 4$ is based on the following observations:

1. The best attacks on Salsa20 and ChaCha [25, 423, 444] break 8 and 7 rounds, respectively, which roughly corresponds to 4 and 3.5 rounds of the NORX core.

While the attack models of ChaCha and NORX are not the same, this gives us a starting point for further analysis.

2. The preliminary cryptanalysis of NORX as presented in Chapter 6. The best differentials we were able to find, belong to a class of high-probability truncated differentials over 1.5 rounds and a class of impossible differentials over 3.5 rounds. Despite the fact that those differentials cannot be used to mount an attack on NORX, it might be possible to find similar differentials, using more advanced cryptanalytic techniques, which could be used for an attack.

### 5.3.5 Selection of Constants

#### Initialization

The initialization constants are listed in Table 5.4 and are derived through

$$(u_0, \ldots, u_{15}) = \mathsf{F}^2(0, \ldots, 15)$$

as already mentioned in Section 5.2.5. This approach allows an on-the-fly computation, if necessary, and is meant to provide transparency in order to show that the values belong to the "nothing-up-my-sleeve" category, i.e. that they were selected in such a way that there is no possibility to hide a backdoor. The main purpose of the initialization constants is to provide some asymmetry during initialization.

#### Domain Separation

The NORX algorithm is separated into different data processing phases. Each phase uses its own domain separation constant to mark the end of certain events like the absorbing of data blocks or merging and branching steps in case of an instance with parallelism degree $p \neq 1$. A domain separation constant is always added to the least significant byte of the capacity word $s_{15}$. The constants are given in Table 5.3. The separation of the processing phase is important for the soundness of the mode of operation, preventing ambiguities between different data types. In addition they help to break the self-similarity of the round function and thus increase the complexity of certain kind of attacks on NORX, for example, like slide attacks.

#### Rotation Offsets

The rotation offsets $(r_0, r_1, r_2, r_3)$ used by NORX provide a good balance between security and efficiency. The values $r_i$, with $0 \leq i \leq 3$, were selected according to the following conditions:

1. At least two out of four offsets are multiples of 8.

2. The remaining offsets are odd and have the form $8n \pm 1$ or $8n \pm 3$, with a preference for the first shape.

The motivation behind those criteria has the following reasons: An offset which is a multiple of 8 preserves byte alignment and thus is much faster than an unaligned rotation on many non-64-bit architectures. Many 8-bit microcontrollers have only 1-bit shifts of bytes, so for example rotations by 5 bits are particularly expensive. Using aligned rotations, i.e. permutations of bytes, greatly increases the performance of the entire algorithm. Even 64-bit architectures benefit from such aligned rotations, for example when an instruction sequence of two shifts followed by `OR` can be replaced by SSSE3's byte shuffling instruction `pshufb`. Odd offsets break up the byte structure and therefore increase diffusion.

In order to find good rotation offsets and assess their diffusion properties, we used an automated search combined with a diffusion test. Therefore let $l$ denote a round number and let $L$ and $L_l$ be lists. For each offset tuple $(r_0, r_1, r_2, r_3)$ with $r_i \in \{1, \ldots, w-1\}$ satisfying the above criteria, the following steps are repeated $10^6$ times, after the offsets have been plugged into G:

1. Choose two $b$-bit sized states $S$ and $S'$ uniformly at random, such that $\mathsf{hw}(S \oplus S') = 1$.

2. Compute $X = \mathsf{F}^l(S) \oplus \mathsf{F}^l(S')$, where F denotes the round function of NORX.

3. Save $\mathsf{hw}(X)$ to $L_l$.

After the above loop is finished the test computes minimum, maximum, average and median values of the elements of $L_l$, saves the latter together with the offsets to $L$ and resets $L_l$. Then it proceeds to the analysis of the next rotation tuple. This test is repeated until all candidate offsets have been processed.

Finally, we chose the offsets $(8, 19, 40, 63)$ for NORX64 and $(8, 11, 16, 31)$ for NORX32, which belonged to those having very high values for average and median Hamming weight for $l = 1$, achieve full diffusion after $l = 2$, and additionally offer good performance.

Table 5.9 lists the results of the test for 32- and 64-bit core functions with $l \leq 4$ and rotation offsets as specified above. The test results show that the diffusion speed of NORX's round function F is almost as high as ChaCha's and that full diffusion is reached after two rounds. Unfortunately there seems to be no combination of rotation values with 3 offsets being a multiple of 8 and one being $w-1$, like BLAKE2's $(32, 24, 16, 63)$, where F achieves a comparably strong diffusion as illustrated in Table 5.9. The reason for this can be traced back to the replacement of integer addition by the nonlinear operation of NORX.

Table 5.9: Diffusion statistics for NORX and ChaCha round functions

| | NORX32 | | | | ChaCha (32-bit) | | | |
|---|---|---|---|---|---|---|---|---|
| $l$ | min | max | avg | med | min | max | avg | med |
| 1 | 83 | 280 | 179.22 | 181 | 73 | 294 | 182.19 | 185 |
| 2 | 194 | 307 | 256.02 | 256 | 199 | 312 | 255.99 | 256 |
| 3 | 198 | 312 | 255.99 | 256 | 204 | 313 | 255.98 | 256 |
| 4 | 201 | 307 | 255.99 | 256 | 200 | 314 | 255.98 | 256 |
| | NORX64 | | | | ChaCha (64-bit) | | | |
| $l$ | min | max | avg | med | min | max | avg | med |
| 1 | 95 | 429 | 230.13 | 222 | 73 | 506 | 248.84 | 246 |
| 2 | 440 | 589 | 511.98 | 512 | 430 | 591 | 512.01 | 512 |
| 3 | 434 | 589 | 512.00 | 512 | 439 | 589 | 511.97 | 512 |
| 4 | 428 | 589 | 511.98 | 512 | 435 | 585 | 512.00 | 512 |

### 5.3.6 The Padding Rule

The sponge (or duplex) construction offers protection against generic attacks if the padding rule is sponge-compliant, i.e. if it is injective and ensures that the last block is different from the all-zero block. In [66] it has been proven that the multi-rate padding satisfies those properties. Moreover it is simple to describe, easy to implement and very efficient. Thus it was a natural choice to be used in NORX. Additionally, the multi-rate padding increases the complexity to mount certain kind of attacks on NORX, like slide attacks.

## 5.4 Performance

NORX was designed to perform well across both software and hardware. This section details our implementations and performance results.

### 5.4.1 Generalities

In this part we analyze some general performance-relevant properties of NORX, like number of operations in G and $F^l$, parallelism degree, and upper bounds for the speed of NORX on different platforms.

#### Number of Operations

Table 5.10 shows the number of operations required for the NORX core functions. We omit the overhead of initialization, integration of parameters, domain separation

constants, padding messages, and so on, as those costs are negligible compared to that of the core permutation $\mathsf{F}^l$.

Table 5.10: Overview on the number of operations of the NORX functions

| function | #XOR | #AND | #shifts | #rotations | total |
|---|---|---|---|---|---|
| G | 12 | 4 | 4 | 4 | 24 |
| F | 96 | 32 | 32 | 32 | 192 |
| $\mathsf{F}^4$ | 384 | 128 | 128 | 128 | 768 |
| $\mathsf{F}^6$ | 576 | 192 | 192 | 192 | 1152 |
| $\mathsf{F}^8$ | 768 | 256 | 256 | 256 | 1536 |
| $\mathsf{F}^{12}$ | 1152 | 384 | 384 | 384 | 2304 |

#### Memory

NORX32 and NORX64 require at least 16 and 32 bytes to be stored in ROM for the initialization constants[7]. To store all initialization constants 40 and 80 bytes of ROM are necessary.

Processing a message in NORX requires enough RAM to store the internal state, i.e., 64 bytes in NORX32 and 128 bytes in NORX64. The data being processed need not be in memory for more than 1 byte at a time. In practice, however, it is preferable to process blocks of 48 (resp. 96) bytes at a time.

#### Parallelism

The core permutation F of NORX has a natural parallelism of 4 independent G applications. Additionally, NORX allows for greater parallelism levels using multiple lanes. Using the $p = 0$ mode, see Section 5.2.5, the internal parallelism level of NORX is effectively unbounded for long enough messages.

### 5.4.2 Software

NORX is easily implemented for 32-bit and 64-bit processors, as it works on 32- and 64-bit words and uses only word-based operations (XOR, AND, shifts and rotations). The specification can directly be translated to code and requires no specific technique such as look-up tables or bitslicing. The core of NORX essentially consists of repeated usage of the G function, which allows simple and compact implementations (e.g., by having only one copy of the G code).

[7] Note that the 10 constants can be generated on-the-fly from $0, \ldots, 15$, see Section 5.2.5.

Furthermore, constant-time implementations of NORX are straightforward to write, due to the absence of secret-dependent instructions or branches.

### Bit Interleaving

While NORX's lack of integer addition avoids dealing with carry chains, the implementer may still have to perform full-word rotations and shifts in words wider than the natural CPU word size. In 8-bit processors, some of this burden is alleviated by 2 out of 4 rotations being multiples of 8. However, this is only a half-measure.

Instead, the implementer can employ the *bit interleaving* technique presented in [67]. This technique consists of splitting an $n$-bit word $w$ into $s = n/m$ $m$-bit words $b_i$, with $b_{ij} = w_{i+jn/m}$. A rotation by $r$ in this representation can be performed by rotating each $b_i$ by $\lfloor r/w \rfloor + 1$ if $i + r \bmod m < r$, $\lfloor r/w \rfloor$ otherwise, and moving $b_i$ to $b_{i+r \bmod m}$. Rotations by 1 or $n-1$ are particularly attractive, since they result in a single $m$-bit rotation. For example, consider implementing NORX64 on a 32-bit CPU. Each state word $w$ will be split into the 2 words $b_0$ and $b_1$. To rotate by $r$:

- If $r \bmod 2 = 0$, rotate both $b_0$ and $b_1$ by $\lfloor r/2 \rfloor$;
- If $r \bmod 2 = 1$, rotate $b_1$ by $\lfloor r/2 \rfloor + 1$, $b_0$ by $\lfloor r/2 \rfloor$, and swap them.

Conversion between representations can be performed in logarithmic time using bit "zip" and "unzip" operations [21].

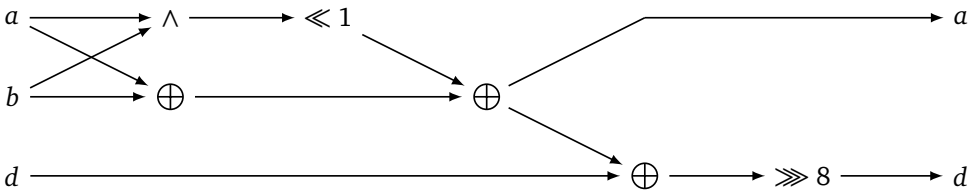
### Avoiding Latency

One drawback of G is that it has little instruction parallelism. In architectures where one is limited by the latency of the G function, an implementer can trade a few extra instructions by reduced latency:

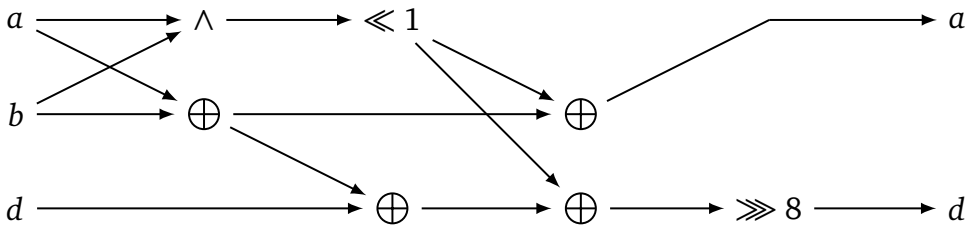
$$\begin{aligned} t_0 &\leftarrow a \oplus b \\ t_1 &\leftarrow a \wedge b \\ t_1 &\leftarrow t_1 \ll 1 \\ a &\leftarrow t_0 \oplus t_1 \\ d &\leftarrow d \oplus t_0 \\ d &\leftarrow d \oplus t_1 \\ d &\leftarrow d \ggg r_0 \end{aligned}$$

This tweak saves up to 1 cycle per instruction sequence, of which there are 4 per G, at the cost of 1 extra instruction (cf. Figure 5.9). In a sufficiently parallel architecture, this can save at least $4 \times 2 \times l$ cycles, which translates to $6.4l/w$ cycles per byte saved

overall. In our measurements, this translated to a performance improvement of NORX from 0.4 to 0.7 cycles per byte, depending on the target architecture, word size, and number of rounds.




(a) Naïve implementation of the G instruction sequence




(b) Latency-oriented version of the G instruction sequence

Figure 5.9: Improving the latency of G.

### Vectorization

NORX lends itself quite well to implementations taking advantage of SIMD extensions present in modern processors, such as AVX or NEON.

The typical vectorized implementation of NORX, when $p = 1$, works in full rows of the $4 \times 4$ state, and computes whole column and diagonal steps of F in parallel.

### Results

We wrote portable C reference implementations for both NORX64 and NORX32, as well as optimized versions for CPUs supporting AVX and AVX2 and for NEON-enabled ARMs. Table 5.11 shows speed measurements on various platforms for messages with varying lengths. The listed CPU frequencies are nominal ones, i.e. without dynamic overclocking features like Turbo Boost, to improve the accuracy of measurements. Furthermore we listed only those platform-compiler combinations that achieved the highest speeds.

The top speed of NORX (for $p = 1$), in terms of bytes per second, was achieved by an AVX2 implementation of NORX64-4-1 on a Haswell CPU, listed in Table 5.11.

It achieves a throughput of about 1.75 GB/s (1.99 cycles per byte at 3.5 GHz). The overhead for short messages ($\leq 64$ bytes) is mainly due to the additional initialization and finalization rounds (see Figure 5.1). However the cost per byte quickly decreases, and stabilizes for messages larger than about 1 KB.

Note that the speed between reference and optimized implementations differs by a factor of less than 2, suggesting that straightforward and portable implementations will provide reasonable performance in most applications. Such consistent performance reduces development costs and improves interoperability.

### 5.4.3 Hardware

Hardware architectures of NORX are efficient and easy to design from the specification: vertical and parallel folding are naturally derived from the iterated and parallel structure of NORX. The cipher benefits from the hardware-friendliness of the function G, which requires only bitwise logical AND, XOR, and bit shifts, and the iterated usage of G inside the core permutation of NORX.

Michael Muehlberghuber and Frank Gürkaynak [347] designed a hardware architecture supporting parameters $w \in \{32, 64\}$, $l \in \{2, \ldots, 16\}$ and $p = 1$. It was synthesized with the Synopsys Design Compiler for an ASIC using 65 nm UMC technology. The implementation was targeted at high data throughput. The requirements in area amounted to about 189 kGE. Simulations for NORX64-4-1 report a throughput of about 124 Gbps (15.5 GB/s), at a maximum frequency of 775 MHz. It achieves a thoughput per gate equivalent of 845 kbps/GE.

## 5.5 Security Goals

We expect NORX with $l \geq 4$ rounds to provide the target security level for any AEAD scheme with the same interface (input and output types and lengths). The following requirements should be satisfied in order to use NORX securely:

**Unique nonces** Each key and nonce pair should not be used to process more than one message.

**Abort on verification failure** If the tag verification fails, only an error is returned. In particular, the decrypted plaintext and the wrong authentication tag must not be given as an output and should be erased from memory in a safe way.

We do not make any claim regarding attackers using “related keys”, “known keys”, “chosen keys”, etc. We also exclude from the claims below models where information is “leaked” on the internal state or key.

Table 5.11: Software performance of NORX in cycles per byte

| data length [byte] | | long | 4096 | 1536 | 576 | 64 | 8 |
|---|---|---|---|---|---|---|---|
| Samsung Exynos 4412 Prime (Cortex-A9) at 1.7 GHz | | | | | | | |
| NORX32-4-1 | Ref | 16.72 | 18.03 | 20.52 | 27.92 | 109.48 | 771.88 |
| | NEON | 9.27 | 10.20 | 11.95 | 16.46 | 72.30 | 521.00 |
| NORX64-4-1 | Ref | 15.60 | 17.91 | 22.02 | 32.42 | 148.55 | 1177.12 |
| | NEON | 7.13 | 8.40 | 10.61 | 16.25 | 82.12 | 648.88 |
| BeagleBone Black Rev B (Cortex-A8) at 1.0 GHz | | | | | | | |
| NORX32-4-1 | Ref | 16.66 | 17.90 | 20.28 | 26.49 | 102.34 | 708.00 |
| | NEON | 9.49 | 10.52 | 12.36 | 17.92 | 75.62 | 550.12 |
| NORX64-4-1 | Ref | 17.24 | 19.81 | 24.34 | 35.73 | 164.86 | 1317.50 |
| | NEON | 7.00 | 8.35 | 10.67 | 16.44 | 85.66 | 680.00 |
| Intel Core i7-2630QM (Sandy Bridge) at 2.0 GHz | | | | | | | |
| NORX64-6-1 | Ref | 6.33 | 7.02 | 8.24 | 13.96 | 70.62 | 607.50 |
| | AVX | 4.02 | 4.42 | 5.14 | 6.90 | 63.75 | 204.00 |
| NORX64-4-1 | Ref | 4.83 | 5,35 | 6.30 | 8.66 | 50.00 | 400.62 |
| | AVX | 2.68 | 2.96 | 3.45 | 4.66 | 17.18 | 137.5 |
| Intel Core i7-3667U (Ivy Bridge) at 2.0 GHz | | | | | | | |
| NORX64-6-1 | Ref | 8.15 | 9.01 | 10.49 | 14.15 | 53.20 | 425.62 |
| | AVX | 5.04 | 5.56 | 6.45 | 8.65 | 32.19 | 255.00 |
| NORX64-4-1 | Ref | 5.58 | 6,17 | 7.22 | 9.82 | 38.05 | 303.75 |
| | AVX | 3.37 | 3.72 | 4.35 | 5.84 | 22.11 | 174.38 |
| Intel Core i7-4770K (Haswell) at 3.5 GHz | | | | | | | |
| NORX64-6-1 | Ref | 5.37 | 5.94 | 6.92 | 9.40 | 36.44 | 292.00 |
| | AVX2 | 2.98 | 3.29 | 3.84 | 5.17 | 19.00 | 153.00 |
| NORX64-4-1 | Ref | 3.98 | 4.39 | 5.11 | 6.97 | 27.19 | 217.00 |
| | AVX2 | 1.99 | 2.20 | 2.58 | 3.49 | 12.94 | 104.50 |

The security of NORX is mainly limited by the key length (128 or 256 bits) and by the tag length (128 or 256 bits). Plaintext confidentiality should thus have the order of 128 or 256 bits of security. The same level of security should hold for integrity of the plaintext or of associated data (based on the fact that an attacker trying $2^n$ tags will succeed with probability $2^{n-256}$, $n < 256$). In particular, recovery of a $k$-bit NORX key should require resources ("computations", energy, etc.) comparable to those required to recover the key of an ideal $k$-bit key cipher. Table 5.12 summarizes the security goals of NORX.

Table 5.12: Overview on the security levels (in bits)

| security goal | NORX32 | NORX64 |
|---|---|---|
| plaintext confidentiality | 128 | 256 |
| plaintext integrity | 128 | 256 |
| associated data integrity | 128 | 256 |
| public message number integrity | 128 | 256 |

Note that NORX restricts the number of messages processed with a given key: in [64] the *usage exponent* $e$ is defined as the value such that the implementation imposes an upper limit of $2^e$ uses to a given key. In NORX we set it to $e_{64} = 128$ for 64-bit and $e_{32} = 64$ for 32-bit, which corresponds in both cases to the size of the nonce.

### 5.5.1 Security Bounds for the Mode of Operation

Using the generic bounds for the duplex sponge [71] and the indifferentiability of the sponge construction [72], one would expect an attacker advantage of roughly $O\left(\frac{q^2}{2^c} + \frac{q}{2^k}\right)$, cf. (2.4), where $c$ is the capacity of the sponge and $q$ is the total number of calls to either NORX or $\mathsf{F}^l$. The first submission of NORX was guided by this analysis, and its parameters were slightly more conservative. This analysis is, however, overly pessimistic; Jovanovic, Luykx, and Mennink [250] showed that the generic security of the NORX mode can be tightened further. We state their result below.

Let $\Pi = (\mathsf{E}, \mathsf{D})$ denote the NORX mode of operation with encryption function $\mathsf{E}$, decryption function $\mathsf{D}$, and based on an ideal underlying permutation $p$. Then the

following privacy and authenticity security bounds are satisfied:

$$\mathbf{Adv}^{\mathsf{priv}}_{\Pi}(q_p, q_{\mathsf{E}}, \lambda_{\mathsf{E}}) \leq \frac{3(q_p + \sigma_{\mathsf{E}})^2}{2^{b+1}} + \left(\frac{8eq_p\sigma_{\mathsf{E}}}{2^b}\right)^{1/2} + \frac{rq_p}{2^c} + \frac{q_p + \sigma_{\mathsf{E}}}{2^k}$$

$$\begin{aligned}\mathbf{Adv}^{\mathsf{auth}}_{\Pi}(q_p, q_{\mathsf{E}}, \lambda_{\mathsf{E}}, q_{\mathsf{D}}, \lambda_{\mathsf{D}}) \leq{}& \frac{(q_p + \sigma_{\mathsf{E}} + \sigma_{\mathsf{D}})^2}{2^b} + \left(\frac{8eq_p\sigma_{\mathsf{E}}}{2^b}\right)^{1/2} + \frac{rq_p}{2^c} \\ &+ \frac{q_p + \sigma_{\mathsf{E}} + \sigma_{\mathsf{D}}}{2^k} + \frac{(q_p + \sigma_{\mathsf{E}} + \sigma_{\mathsf{D}})\sigma_{\mathsf{D}}}{2^c} + \frac{q_{\mathsf{D}}}{2^t}\end{aligned}$$

where $r$, $c$, $b$, $k$ and $t$ are rate, capacity, state, key and tag sizes, $e \approx 2.718$ is Euler's number, $q_p$ are the number of permutation queries, $q_{\mathsf{E}}$ are the number of encryption queries of total length $\lambda_{\mathsf{E}}$ and $\sigma_{\mathsf{E}}$ is specified as follows:

$$\sigma_{\mathsf{E}} := \sum_{j=1}^{q_{\mathsf{E}}} \sigma_{\mathsf{E},j} \leq \begin{cases} 2\lambda_{\mathsf{E}} + 4q_{\mathsf{E}}, & \text{if } p = 0 \\ \lambda_{\mathsf{E}} + 3q_{\mathsf{E}}, & \text{if } p = 1 \\ \lambda_{\mathsf{E}} + (p+4)q_{\mathsf{E}}, & \text{if } p > 1 \end{cases}$$

The values $q_{\mathsf{D}}$, $\lambda_{\mathsf{D}}$ and $\sigma_{\mathsf{D}}$ for decryption D are specified analogously.

In summary, the NORX mode of operation achieves security levels of $\min\{2^{b/2}, 2^c, 2^k\}$ assuming an ideal underlying permutation $p$ and, intuitively speaking, offers authenticity as long as it offers privacy and the $\frac{(q_p+\sigma_{\mathsf{E}}+\sigma_{\mathsf{D}})\sigma_{\mathsf{D}}}{2^c}$ term—quadratic on the number of forgery attempts—is negligible. For more information on NORX and sponge authenticated encryption security proofs see [14, 147, 250, 251].

# Chapter 6

# Analysis of NORX

This chapter deals with the analysis of the NORX core permutation from a cryptanalytic perspective. In particular, we investigate its symmetry properties (Section 6.1), the propagation of differential and linear trails (Sections 6.2, 6.3 and 6.5), and rotational properties (Section 6.4).

To assist in identifying good trails, we automate the process of finding them by reducing the search to a SMT problem, which can then be fed to a SMT solver such as Z3 [159], Boolector [112, 361, 362], or STP [201]. This was inspired by the innovative work of Mouha et al. [344, 346]. In Section 6.6 we apply automated search to other primitives, to great effect.

We refer to Chapter 5 for the background and specification of NORX's mode and components.

## 6.1 Symmetries

The original NORX specification contained a discussion about the all-zero state, which is mapped to itself by F, and why it is not a problem for the security of the scheme. However, due to the structure of F, there is another, larger, class of weak states. These are of the form

$$\begin{pmatrix} a & a & a & a \\ b & b & b & b \\ c & c & c & c \\ d & d & d & d \end{pmatrix}$$

with $a$, $b$, $c$, and $d$ being arbitrary $W$-bit sized words. States of this form are preserved by F. Later, Chaigneau et al. [120] and independently Biryukov, Udovenko, and Velichkov [96] identified a larger class of symmetries on the F permutation: rotation

of columns. Specifically,

$$\mathsf{F}\left(\begin{pmatrix} a & e & i & m \\ b & f & j & n \\ c & g & k & o \\ d & h & l & p \end{pmatrix}\right) \lll 1 = \mathsf{F}\left(\begin{pmatrix} e & i & m & a \\ f & j & n & b \\ g & k & o & c \\ h & l & p & d \end{pmatrix}\right),$$

and more generally

$$\mathsf{F}(S \lll r) = \mathsf{F}(S) \lll r\,,$$

where $\lll$ here refers to rotation of columns of the state, not of individual words. This leads to a set of $2^{8w}$ weak states, namely those of the form

$$\begin{pmatrix} a & e & a & e \\ b & f & b & f \\ c & g & c & g \\ d & h & d & h \end{pmatrix}$$

that remain invariant under rotation by 2 columns. Chaigneau et al. [120] turned this property into a forgery and key recovery attack for an earlier version of NORX. The version of NORX presented in Chapter 5, by adding the key at the finalization stage, prevents this property from being exploitable.

## 6.2 Differential Cryptanalysis

This section is dedicated to the differential cryptanalysis of NORX. First, we introduce the required mathematical formulas to describe differential propagation through $\mathsf{F}^R$ of NORX. Then we describe a search framework for automatic differential trail discovery and apply it to NORX.

### 6.2.1 Simple Differential Analysis

We started by manually tracking how differentials propagated through G. We noticed that putting input differences in the most significant bits of each word would result in such differences bypassing the first H (since $a \ll 1$ and $b \ll 1$ would erase such differences going into the AND), giving them higher probability.

With this simple strategy and after some effort, we found the following high-probability differentials for the 64-bit version of G:

$$\begin{pmatrix} 8000000000000000 \\ 8000000000000000 \\ 8000000000000000 \\ 0000000000000000 \end{pmatrix} \xrightarrow[1]{\mathsf{G}} \begin{pmatrix} 0000000000000000 \\ 0000000000000001 \\ 8000000000000000 \\ 0000000000000000 \end{pmatrix}$$

$$\begin{pmatrix} 8000000000000000 \\ 8000000000000000 \\ 8000000000000000 \\ 0000000000000000 \end{pmatrix} \xrightarrow[2^{-1}]{\mathsf{G}} \begin{pmatrix} 8000000000000000 \\ 0000000001000001 \\ 8000000000800000 \\ 0000000000800000 \end{pmatrix}$$

$$\begin{pmatrix} 0000000000000000 \\ 8000000000000000 \\ 8000000000000000 \\ 8000000000000000 \end{pmatrix} \xrightarrow[2^{-1}]{\mathsf{G}} \begin{pmatrix} 8000000000000000 \\ 0000000003000001 \\ 8000000001800000 \\ 0000000000800000 \end{pmatrix}$$

Applying those differentials to F has the effect that the diffusion of the state is delayed by one step. Note that input differences with other combinations of active most significant bits lead to similar output differences, but none with a lower or equal Hamming weight as the above. Using the first of the above differentials, we were able to easily derive a truncated differential over 3 steps (i.e., $\mathsf{F}^{1.5}$), with probability 1.

Of course, continuing with this strategy, manually tracking such differences into the whole round, or multiple rounds, does not quite scale and it is immensely laborious. As such, our efforts focused on automating this process.

### 6.2.2 Propagation Properties of H

Because H is the only nonlinear operation in the entirety of NORX, this is the only operation we need to be concerned with. Differences propagate deterministically through all other operations, namely cyclic rotation and xor.

Let $n$ denote the word size, let $x$ and $y$ denote bit strings of size $n$ and let $\alpha$, $\beta$ and $\gamma$ denote differences of size $n$. We identify by $\alpha_i$, $\beta_i$, $\gamma_i$, $x_i$ and $y_i$ the individual bits of $\alpha$, $\beta$, $\gamma$, $x$ and $y$, with $0 \leq i \leq n-1$. We recall the definition of H.

**Definition 6.1.** *The nonlinear operation* H *of NORX is the vectorial Boolean function defined by*

$$\begin{aligned} \mathsf{H} : \{0,1\}^n \times \{0,1\}^n &\to \{0,1\}^n \\ (a,b) &\mapsto (a \oplus b) \oplus ((a \wedge b) \ll 1)\,. \end{aligned}$$

Since bitwise AND is the only nonlinear operation within H, we begin by investigating its differential propagation.

**Theorem 6.2.** *Let* $\wedge : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ *be the bitwise* AND *operation on n-bit words. A differential* $(\alpha, \beta) \xrightarrow{\wedge} \gamma$ *has nonzero probability if and only if* $\neg(\alpha \vee \beta) \wedge \gamma = 0$. *Furthermore, such a differential has probability* $2^{-\mathrm{hw}(\alpha \vee \beta)}$.

*Proof.* We can interpret a bitwise $\wedge$ operation as the parallel application of $n$ independent $\wedge$ bit operations. For each individual bit operation, we can exhaustively compute the differential probability for every possible difference pattern $(\alpha_i, \beta_i) \to \gamma_i$, which follows in the table below.

| | | $\gamma_i$ | |
|---|---|---|---|
| $\beta_i$ | $\alpha_i$ | 0 | 1 |
| 0 | 0 | $2^0$ | 0 |
| 0 | 1 | $2^{-1}$ | $2^{-1}$ |
| 1 | 0 | $2^{-1}$ | $2^{-1}$ |
| 1 | 1 | $2^{-1}$ | $2^{-1}$ |

The condition $\neg(\alpha \vee \beta) \wedge \gamma = 0$ is the boolean encoding of the above table for the impossible differential $(0,0) \to 1$.

Furthermore we see from the table that a viable differential either has probability $2^0$ when the input difference is all-zero, or probability $2^{-1}$ when at least one of the input differences is 1. Since there are $n$ parallel applications of $\wedge$, all independent, the overall probability amounts to $2^{-(\alpha_0 \vee \beta_0 + \cdots + \alpha_{n-1} \vee \beta_{n-1})} = 2^{-\mathrm{hw}(\alpha \vee \beta)}$. □

Theorem 6.2 gives us an easy way to check how a differential propagates through bitwise AND. This gives us a way to easily reason about the propagation of differences through H.

**Theorem 6.3.** *Let* $\mathsf{H} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ *be the operation defined in Definition 6.1. A differential* $(\alpha, \beta) \xrightarrow{\mathsf{H}} \gamma$ *has nonzero probability if and only if* $\neg((\alpha \vee \beta) \ll 1) \wedge (\alpha \oplus \beta \oplus \gamma) = 0$. *Furthermore, such a differential has probability* $2^{-\mathrm{hw}((\alpha \vee \beta) \ll 1)}$.

*Proof.* Differences propagate deterministically through linear operations. In particular, input differences $(\alpha, \beta)$ become $(\alpha \ll 1, \beta \ll 1)$ after being shifted by 1, and the output difference of xor is the xor of the input differences. As such, the input differences to the $\wedge$ operation are $(\alpha \ll 1, \beta \ll 1)$. If the output difference of H is $\gamma$, the difference at the output of $\wedge$ must be $\gamma \oplus \alpha \oplus \beta$, by the linearity of xor. Applying Theorem 6.2 to $(\alpha \ll 1, \beta \ll 1) \xrightarrow{\wedge} \gamma \oplus \alpha \oplus \beta$ completes the proof. □

It is interesting to compare and contrast Theorem 6.3 to Lipmaa and Moriai's treatment of the differential probability of integer addition modulo $2^n$ [299].

**Theorem 6.4** ([299]). *Let $\boxplus : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be integer addition modulo $2^n$. Then $(\alpha,\beta) \xrightarrow{\boxplus} \gamma$ is a differential with nonzero probability if and only if*

$$\mathsf{eq}(\alpha \ll 1, \beta \ll 1, \gamma \ll 1) \wedge \alpha \oplus \beta \oplus \gamma \oplus (\beta \ll 1) = 0\,,$$

*where*

$$\mathsf{eq}(\alpha,\beta,\gamma) = (\neg\alpha \oplus \beta) \wedge ((\neg\alpha) \oplus \gamma)$$

*Furthermore the probability of such a differential is given by $2^{-\mathsf{hw}((\neg\mathsf{eq}(\alpha,\beta,\gamma)) \ll 1)}$.*

Instead of looking at differences with respect to xor, one could alternatively also investigate differences with respect to H, which is done in the following. This is somewhat analogous to the study of the propagation of xor through integer additions [300].

**Definition 6.5.** *Let $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be a vectorial Boolean function and let $(\alpha,\beta) \to \gamma$ be differences with respect to the $f$ operation. We call $(\alpha,\beta) \xrightarrow{\oplus} \gamma$ an $f$-*differential *of xor if there exist $n$-bit strings $x$ and $y$ such that the following equation holds:*

$$f(x,\alpha) \oplus f(y,\beta) = f(x \oplus y, \gamma)$$

*Otherwise, if no such $n$-bit strings $x$ and $y$ exist, we call $(\alpha,\beta) \xrightarrow{\oplus} \gamma$ an* impossible *$f$-*differential.

Plugging the nonlinear operation H into the formula of Definition 6.5 we obtain the following equation

$$\alpha \oplus \beta \oplus \gamma = ((x \wedge (\alpha \oplus \gamma)) \oplus (y \wedge (\beta \oplus \gamma))) \ll 1 \tag{6.1}$$

which can be expressed at the bit level as

$$\begin{aligned} 0 &= \alpha_0 \oplus \beta_0 \oplus \gamma_0 \\ 0 &= (\alpha_i \oplus \beta_i \oplus \gamma_i) \oplus (x_{i-1} \wedge (\alpha_{i-1} \oplus \gamma_{i-1})) \oplus (y_{i-1} \wedge (\beta_{i-1} \oplus \gamma_{i-1})), \quad i > 0\,. \end{aligned}$$

**Lemma 6.6.** *Let* H *denote the nonlinear operation of* NORX*. For each* H*-differential in terms of Definition 6.5 the following equation is satisfied:*

$$(\alpha \oplus \beta \oplus \gamma) \wedge (\neg(\gamma \ll 1) \oplus (\alpha \ll 1)) \wedge (\neg(\beta \ll 1) \oplus (\gamma \ll 1)) = 0 \tag{6.2}$$

*Proof.* It is easy to see that the least significant bits (i.e. $i = 0$) of (6.1) and (6.2) are the same. Therefore, we will consider them no longer. Looking at the bit level representation of (6.1) (for $i > 0$) we consider two cases:

- $\alpha_i \oplus \beta_i \oplus \gamma_i = 0$: Here, (6.1) has always the solution $x_{i-1} = y_{i-1} = 0$.
- $\alpha_i \oplus \beta_i \oplus \gamma_i = 1$: In this case, the bit level representation of (6.1) is only solvable if either $\alpha_{i-1} \neq \gamma_{i-1}$ or $\beta_{i-1} \neq \gamma_{i-1}$. Furthermore, the bit level representation of (6.2) is given by

$$(\alpha_i \oplus \beta_i \oplus \gamma_i) \wedge (\alpha_{i-1} \oplus \gamma_{i-1} \oplus 1) \wedge (\beta_{i-1} \oplus \gamma_{i-1} \oplus 1) = 0, \qquad i > 0$$

  It is evident that the latter equation only holds if $(\alpha_i \oplus \beta_i \oplus \gamma_i) = 0$, $\alpha_{i-1} \neq \gamma_{i-1}$, or $\beta_{i-1} \neq \gamma_{i-1}$. As seen above, these are the very same conditions that define a H-differential.

□

**Lemma 6.7.** *Let* H *denote the non-linear operation of NORX and let* $(\alpha, \beta) \xrightarrow{\mathsf{H}} \gamma$ *be an* H*-differential in terms of Definition 6.5. Its probability is given by* $2^{-\mathsf{hw}(((\alpha \oplus \gamma) \vee (\beta \oplus \gamma)) \ll 1)}$.

*Proof.* The claim can be proven analogously to Theorem 6.3. It follows from the fact that in the bit level representation of (6.1) the expression

$$(x_{i-1} \wedge (\alpha_{i-1} \oplus \gamma_{i-1})) \oplus (y_{i-1} \wedge (\beta_{i-1} \oplus \gamma_{i-1}))$$

is balanced if $\alpha_{i-1} \oplus \gamma_{i-1} = 1$ or $\beta_{i-1} \oplus \gamma_{i-1} = 1$. □

While we exclusively consider differentials with respect to xor in the rest of this section, H-differentials may yet prove to be of future interest.

### 6.2.3 Automated Trail Search with Satisfiability

Armed with the results from Section 6.2.2, in particular Theorem 6.3, we can now begin to automate the process of finding differential trails for NORX's F. We follow the blueprint of Mouha and Preneel [344], with some simplifications.

For modeling the differential propagation through a sequence of operations, we use a technique well-known from algebraic cryptanalysis—for every output of an operation a new set of variables is introduced. The set of values of those output variables are then constrained as a function of the input variables. Moreover, the former are used as input to the next operation. This is repeated until all required operations have been integrated into the problem description. Before we show how the differential propagation in $\mathsf{F}^R$ is modeled concretely, we introduce the required variables.

Let $s$ denote the number of (column and diagonal) steps to be analysed and let $0 \leq i \leq 15$ and $0 \leq j \leq 2(s-1)$. For example, if we analyse $\mathsf{F}^2$, we have $s = 4$. Let $x_i$, $y_{i,j}$ and $z_i$ be $w$-bit sized variables, which model the input, internal and output

differences of a differential trail. Recall that $w \in \{32, 64\}$ denotes the word size of NORX. Moreover, let $d_{i,k}$, with $0 \leq k \leq s-1$, be $w$-bit sized helper variables which are used for differential weight computations or equivalently to determine the probability of a differential characteristic. We use the common assumption, flawed as it may be, that the probability of a differential trail is reasonably approximated by the sum of weights of each non-linear operation H. Furthermore, let $d$ denote a $w$-bit sized variable which fixes the total weight of the characteristic we plan to search for. The description of the search problem is generated through the following steps:

1. Every time the function G applies the nonlinear operation H we add two expressions to our description:

   a) Add the constraint $0 = (\alpha \oplus \beta \oplus \gamma) \wedge (\neg((\alpha \vee \beta) \ll 1))$ from Theorem 6.3, with $\alpha$, $\beta$ and $\gamma$ each substituted by one of the variables $x_i$, $y_{i,j}$ or $z_i$. This ensures that only viable trails are considered.

   b) Add the expression $d_{i,k} = (\alpha \vee \beta) \ll 1$ from Theorem 6.3, with $\alpha$ and $\beta$ substituted by the same variables $x_i$, $y_{i,j}$ or $z_i$ as in step (a). This expression keeps track of the weight of the trail.

2. Every time the function G applies a rotation we apply the same rotation to the corresponding xor difference, i.e. we add $\gamma = (\alpha \oplus \beta) \ggg r$ to the problem description, with $\alpha$, $\beta$ and $\gamma$ substituted appropriately. Note that the rotation is a linear operation and thus does not change the differential probability.

3. Add an expression corresponding to the following equation:

$$d = \sum_{k=0}^{s-1} \sum_{i=0}^{15} \mathsf{hw}(d_{i,k}) \tag{6.3}$$

   This equation ensures that indeed a characteristic of weight $d$ is found. Depending on the technique how Hamming weights are computed, additional variables might be necessary.

4. Constrain the variable $d$ to be equal to the target differential weight and append it to the problem description.

5. Exclude the trivial characteristic mapping an all-zero input difference to an all-zero output difference. To do so, it is sufficient to exclude the all-zero input difference. Therefore, append an expression equivalent to $((x_0 \neq 0) \vee \cdots \vee (x_{15} \neq 0))$ to the problem description.

Doing this using existing SMT solvers is quite simple. Appendix A.1 demonstrates how to find the best differential from Section 6.2.1 instantly. We model the problem as

described above, print it out in the standard SMTLIB [41] format, and then use an off-the-shelf solver to obtain a solution. For example, using the script from Appendix A.1 and Boolector [112, 360, 362] we instantly obtain

```
$ python norxex.py | boolector -x -m --output-format=btor | sort -k3 | cut -d' ' -f2-
sat
8000000000000000 x0
8000000000000000 x1
8000000000000000 x2
0000000000000000 x3
0000000000000000 y0
0000000000000000 y1
0000000000000000 y2
0000000000000000 y3
0000000000000000 z0
0000000000000001 z1
8000000000000000 z2
0000000000000000 z3
```

### 6.2.4 Applications of Automated Search to NORX

In this part we describe the application of the search framework to the permutation $\mathsf{F}^l$ of NORX. Depending on the concrete attack model, there are different ways an attacker could inject differences into the NORX state. During initialization an adversary is allowed to modify either the nonce words $s_1$ and $s_2$ ($\mathsf{init}_N$) or nonce and key words $s_1, s_2, s_4, \ldots, s_7$ ($\mathsf{init}_{N,K}$). During data processing an attacker can inject differences into the words of the rate $s_0, \ldots, s_9$ (rate). Last but not least, we also investigate the case where an attacker can manipulate the whole state $s_0, \ldots, s_{15}$ (full).

While an attacker is not able to influence the entire state at any point directly due to the duplex construction, the full scenario is nevertheless useful to estimate the general strength of $\mathsf{F}^l$, because all of the other settings described above are special cases of the latter. Additionally, it could be useful for the chaining of trails: For example, an attacker could start with a search in the data processing part (i.e. under the rate setting) over a couple of steps, say $\mathsf{F}^{l_1}$, and continue afterwards with a second search, starting from the full state for another couple of steps, say $\mathsf{F}^{l_2}$, so that differentials from the second search connect to those from the first, resulting in differentials for $\mathsf{F}^{l_1+l_2}$. We will explore this divide and conquer strategy in more detail below.

We denote a differential trail as a tuple of differences $(\delta_0, \ldots, \delta_n)$, where $\delta_0$ is the *input difference* and $\delta_n$ is the *output difference*. The values $\delta_i$ for $0 < i < n$ are called *internal differences*. The weight of the probability that difference $\delta_i$ results in the difference $\delta_{i+1}$ by the $r_i$th iteration of $\mathsf{F}$ is denoted by $d_i$ for $0 \leq i \leq n-1$. Recall that we assume that the probability of the entire trail is equal to the multiplication of probabilities of the intermediate differentials, and thus we have $d = \sum_{i=0}^{n-1} d_i$ for the total weight of the trail. The notation $\mathsf{F}^{l+0.5}$ describes that we do $l$ full rounds

followed by one more column step, e.g., $\mathsf{F}^{1.5}$ corresponds to one full round plus one additional column step.

### Experimental Verification of the Search Framework

The goal of the experimental verification is to show that the framework indeed does what it is supposed to do, namely find differentials of a predetermined weight $d$ in $\mathsf{F}^l$. Therefore, we generated differentials for $\mathsf{F}^{1.5}$ (full) and verified them against a C reference implementation of $\mathsf{F}^{1.5}$. Under these prerequisites our framework found the first differentials at a weight of 12, for both $w = 32$ and $w = 64$, which thus should have a probability of about $2^{-12}$. To get a better coverage of our verification test, we did not use only differentials of that particular weight, but other discovered differential trails of weights $d \in \{12, \ldots, 18\}$, which are listed in Appendix A.2.1 for both 32- and 64-bit. Then we applied them to the C implementation of $\mathsf{F}^{1.5}$ for $2^{d+16}$ pairs of randomly chosen input states having the input difference of the characteristic. In each case, we checked if the output difference had the predicted pattern. The number of pairs matching the trail should be around $2^{16}$. The results are illustrated in the first table of Appendix A.2.1 and show that the trails found by the search framework do have the expected properties.

### Lower Bounds for Differential Weights of $\mathsf{F}^l$

We made an extensive analysis on the weight bounds of differential paths in $\mathsf{F}^l$, where we investigated $1 \leq s \leq 4$ steps for our four different scenarios $\mathsf{init}_N$, $\mathsf{init}_{N,K}$, rate and full. We tried to find the lowest weights where differentials appear for the first time. These cases are listed in Table 6.1 as entries without brackets. For example, in case of NORX32 under the setting full, there are no differentials in $\mathsf{F}^{1.5}$ with a weight smaller than 12. Entries in brackets are the maximal weights we were capable of examining without finding any differentials. Due to memory constraints, our methods failed for differential weights higher than those presented in Table 6.1. For example, our search routine did not find any characteristics of weight smaller than 40 (i.e. of probability higher than $2^{-40}$) for the scenario $\mathsf{F}^{1.5}$, $\mathsf{init}_{N,K}$ and $w = 32$. The required amount of RAM, to execute this check, was approximately 49 GiB (using CryptoMiniSat with 16 threads) with a running time of 8 hours. A longer computation using Boolector found the best trails for $\mathsf{F}^2$ in the full setting have both weight 39.

The security of NORX depends heavily on the security of the initialization, which transforms the initial state by $\mathsf{F}^{2l}$. As $\mathsf{init}_N$ is the most realistic attack scenario, we conducted a search over all possible 1- and 2-bit differences in the nonce words. Our search revealed that the best characteristics have weights of 67 (32-bit) and 76 (64-bit) under those prerequisites. Obviously, these weights are not too far away from

Table 6.1: Lower bounds for differential trail weights. Parenthesis indicate the minimal weight was not reached.

| | NORX32 | | | | NORX64 | | | |
|---|---|---|---|---|---|---|---|---|
| | $\text{init}_N$ | $\text{init}_{N,K}$ | rate | full | $\text{init}_N$ | $\text{init}_{N,K}$ | rate | full |
| $\mathsf{F}^{0.5}$ | 6 | 2 | 2 | 0 | 6 | 2 | 2 | 0 |
| $\mathsf{F}^{1.0}$ | (60) | 22 | 10 | 2 | (53) | 22 | 12 | 2 |
| $\mathsf{F}^{1.5}$ | (60) | (40) | (31) | 12 | (53) | (35) | (27) | 12 |
| $\mathsf{F}^{2.0}$ | (61) | (45) | (34) | 39 | (51) | (37) | (30) | 39 |

the computationally verified values of 60 (32-bit) and 53 (64-bit) from Table 6.1, showing that the bounds for F ($\text{init}_N$) are quite tight.

Extrapolating the above results to $F^8$ (i.e. $l = 4$), we get lower weights of $61+3\cdot 27 = 142$ ($\text{init}_N$) or $45+3\cdot 27 = 126$ ($\text{init}_{N,K}$) for NORX32 and $51+3\cdot 23 = 132$ ($\text{init}_N$) or $37+3\cdot 23 = 106$ ($\text{init}_{N,K}$) for NORX64. However, these are only loose bounds and we expect the real ones to be considerably higher.

### Search for Differential Characteristics in $\mathsf{F}^4$

This part shows how we constructed differential characteristics in $\mathsf{F}^4$ under the setting full for both versions of the permutation, i.e. 32- and 64-bit. Unsurprisingly, a direct approach to find such characteristics turned out to be infeasible, hence we decomposed the search into multiple parts and constructed the entire path step by step.

At first we made searches that only stretched over $l \leq 2$ rounds. After tens of thousands of iterations using many different search parameter combinations we found differentials having internal differences of Hamming weight 1 and 2 after one application of F. We also used a probability-1 differential in G, which is listed as the first entry in the table of Appendix A.2.2, as a starting place. We expanded all those characteristics for both word sizes, in forward and backward direction one column or diagonal step at a time, until their paths stretched the entire 4 rounds. The best differential paths we found this way have weights of 584 (32-bit) and 836 (64-bit), respectively. Both are depicted in Appendix A.2.3.

### Iterative Differentials

We also performed extensive searches for iterative differentials in F for the setting full. Using our framework, we could show that there are no such differentials up to a weight of 29 (32-bit) and 27 (64-bit), before our methods failed due to computational constraints. Extrapolating these results to $\mathsf{F}^8$ and $\mathsf{F}^{12}$, i.e. the number of initialization rounds for $l = 4$ and $l = 6$, we get lower weight bounds of 232 and 348, for 32-bit, or of 216 and 324 for 64-bit. The best iterative differentials we could find for F, have

weights of 512 (32-bit) and 843 (64-bit) and are depicted in Appendix A.2.4. These weights are obviously much higher than our guaranteed lower bounds, and hence we expect that the latter are much better compared to the values we were able to verify computationally.

#### Differentials with Equal Columns

The class of weak states from Section 6.1 can be obviously transformed into differentials having four equal columns. The best differentials we could find for F have weight 44 for both 32-bit and 64-bit. They exploit an already well known probability-1 differential in G, see Appendix A.2.2. The 64-bit variant was also used in the construction of the characteristics with weight 836 in $\mathsf{F}^4$ above. Concrete representations of these differentials can be found in Appendix A.2.5.

## 6.3 Linear Cryptanalysis

As noted in Section 2.4.3 linear cryptanalysis is in many ways similar to differential cryptanalysis. So much so that it is logical to also try and study the propagation of linear approximations through F. Since much of the background is shared with Section 6.2, we present only the main results and refer to that section for more background.

Matsui [323, §3] observed that the propagation of linear approximations and differentials through linear operations are dual:

- Two input differences $\alpha, \beta$ going through a xor become the output difference $\gamma = \alpha \oplus \beta$ with probability 1; two linear masks going into a xor can only have correlation 1 if $\alpha = \beta = \gamma$, 0 otherwise.
- A differential $\alpha$ that is used as input into two separate expressions, say $\alpha'$ and $\alpha''$, must preserve its value, i.e., $\alpha = \alpha' = \alpha''$; a linear mask $\alpha$ in the same situation instead must preserve the relation $\alpha \oplus \alpha' \oplus \alpha'' = 0$.

As with differential cryptanalysis, we begin with the analysis of the simpler AND primitive.

**Theorem 6.8.** *Let $\wedge : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be the bitwise AND operation on n-bit words. A linear approximation $(\alpha, \beta) \xrightarrow{\wedge} \gamma$ has nonzero correlation if and only if $(\alpha \vee \beta) \wedge \neg\gamma = 0$. Furthermore, such an approximation has correlation $(-1)^{(\alpha\wedge\beta)\cdot\gamma} 2^{-\mathsf{hw}(\gamma)}$.*

*Proof.* Like in Theorem 6.2, we observe that a bitwise AND can be treated as the parallel application of $n$ bit AND operations. The linear correlation table for every input and output mask is given below.

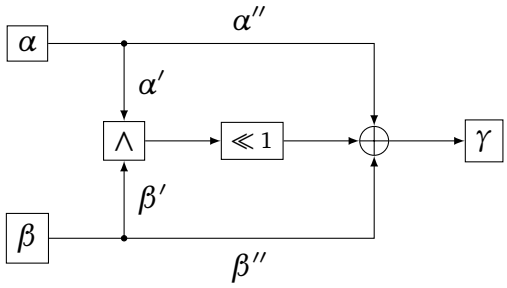



Figure 6.1: Propagation of linear masks through H

| $\beta_i$ | $\alpha_i$ | $\gamma_i$ 0 | $\gamma_i$ 1 |
|---|---|---|---|
| 0 | 0 | $2^0$ | $2^{-1}$ |
| 0 | 1 | 0 | $2^{-1}$ |
| 1 | 0 | 0 | $2^{-1}$ |
| 1 | 1 | 0 | $-2^{-1}$ |

We see that a mask is impossible if $\gamma_i = 0$ and $\alpha_i \vee \beta_i = 1$. The negation of this condition is given by $(\alpha \vee \beta) \wedge \neg\gamma = 0$, which covers all linear masks with nonzero correlation.

From the table we also see that the only case that contributes to the weight of the linear approximation is when the output mask is nonzero. As such, the Hamming weight of the output mask yields its weight. The sign of the correlation is $-1$ if $\alpha_i \wedge \beta_i \wedge \gamma_i = 1$ and 1 otherwise. Multiplying every (independent) correlation together we obtain the final result $(-1)^{(\alpha\wedge\beta)\cdot\gamma}$. □

**Theorem 6.9.** *Let* $\mathsf{H} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ *be the operation defined in Definition 6.1. A linear approximation* $(\alpha, \beta) \xrightarrow{\mathsf{H}} \gamma$ *has nonzero correlation if and only if* $((\alpha \oplus \gamma) \vee (\beta \oplus \gamma)) \wedge \neg(\gamma \gg 1) = 0$. *Furthermore, such a linear approximation has correlation* $(-1)^{((\alpha\oplus\gamma)\wedge(\beta\oplus\gamma))\cdot(\gamma\gg 1)} 2^{-\mathsf{hw}(\gamma\gg 1)}$.

*Proof.* Recall that linear masks going through "three-forked branches" induce a xor relation, and xor induces equality of the input and output masks. Figure 6.1 can be used as a visual aid for this process. Supposing the mask at the output of H is $\gamma$, then at the output of $\wedge$ must be $\gamma \gg 1$, reversing the shift.

Furthermore, let $\alpha', \alpha''$ be the values of $a$ going into the input of $\wedge$ and the final xor, respectively $\beta', \beta''$ for $\beta$. The three-forked branch and final xor impose the following conditions:

- $\gamma = \alpha'' = \beta''$;

- $\alpha' = \alpha \oplus \alpha'' = \alpha \oplus \gamma$;
- $\beta' = \beta \oplus \beta'' = \beta \oplus \gamma$.

Having the input and output masks into the $\wedge$ operation, and no other nonlinear operation being present, we can reduce the correlation of H to the correlation of the masks $\alpha \oplus \gamma$, $\beta \oplus \gamma$, $\gamma \gg 1$ going into and out of the $\wedge$. Applying Theorem 6.8 yields the result. □

We may also compare Theorem 6.9 with Schulte-Geers's analogous result [415] for integer addition.

**Theorem 6.10** ([415, Theorem 4]). *Let $\boxplus : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be integer addition modulo $2^n$. Then $(\alpha, \beta) \xrightarrow{\boxplus} \gamma)$ is a linear approximation with nonzero probability if and only if*

$$((\gamma \oplus \alpha) \vee (\gamma \oplus \beta)) \wedge \neg\tau = 0\,,$$

*where $\tau$ satisfies*

$$\tau \oplus (\tau \gg 1) \oplus ((\alpha \oplus \beta \oplus \gamma) \gg 1) = 0\,.$$

*Furthermore the correlation of such an approximation is given by $(-1)^{(\gamma\oplus\alpha)\cdot(\gamma\oplus\beta)} 2^{-\mathsf{hw}(\tau \ll 1)}$.*

### 6.3.1 Application to NORX

As with differential search, we initiated a search for linear trails using Theorem 6.9 to discover the trails with minimum weight. We ignored the sign of the correlations, since our only interest was in measuring the strength of the correlations, not their exact value. Using the framework from Section 6.2 and Theorem 6.9 we quickly find trails with correlation 1 for the 64-bit version of H, for example

$$\begin{pmatrix} 0000000000000001 \\ 0000000000000001 \\ 0000000000000000 \\ 0000000000000001 \end{pmatrix} \xrightarrow[1]{\mathsf{G}} \begin{pmatrix} 0100000000000000 \\ 0000000000000000 \\ 0000000000000000 \\ 0000000000010000 \end{pmatrix}.$$

Because finding linear trails is not as useful for the cryptanalysis of NORX as differential search, we were concerned only with determining whether the weights of linear trails grew sufficiently quickly. Table 6.2 contains the weights for the best linear trails found for NORX. From there, and comparing with Table 6.1 in the full setting, we can see that the squared correlations are comparable in magnitude with the the best differential trails.

Table 6.2: Lower bounds for linear trail weights in the NORX F function.

| | NORX32 | NORX64 |
|---|---|---|
| $F^{0.5}$ | 0 | 0 |
| $F^{1.0}$ | 1 | 1 |
| $F^{1.5}$ | 6 | 6 |
| $F^{2.0}$ | 20 | 20 |

## 6.4 Rotational Cryptanalysis

**Definition 6.11.** *Let $f$ be a vectorial Boolean function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ and let $a$, $b$ be $n$-bit strings. We call $(a, b)$ a* rotational pair *with respect to $f$ if the following equation holds:*

$$f(a,b) \ggg r = f(a \ggg r, b \ggg r)$$

**Lemma 6.12.** *Let $\mathsf{H}$ be the nonlinear function of NORX, and let $a$, $b$ be $n$-bit strings. The probability of $(a, b)$ being a rotational pair is:*

$$\Pr(\mathsf{H}(a,b) \ggg r = \mathsf{H}(a \ggg r, b \ggg r)) = \frac{9}{16} \ (\approx 2^{-0.83})$$

*Proof.* After evaluating and simplifying the equation $\mathsf{H}(a,b) \ggg r = \mathsf{H}(a \ggg r, b \ggg r)$ we get $((a \wedge b) \ll 1) \ggg r = ((a \ggg r) \wedge (b \ggg r)) \ll 1$. Translating this equation to bit vectors results in

$$\begin{aligned}&(a_{r-1} \wedge b_{r-1}, \ldots, a_0 \wedge b_0, 0, a_{n-2} \wedge b_{n-2}, \ldots, a_r \wedge b_r)\\ &= (a_{r-1} \wedge b_{r-1}, \ldots, a_0 \wedge b_0, a_{n-1} \wedge b_{n-1}, a_{n-2} \wedge b_{n-2}, \ldots, 0)\end{aligned}$$

The probability that those two vectors match is $(3/4)^2 = 9/16$, as $a \wedge b = 0$ with probability $3/4$ for bits $a$ and $b$ chosen uniformly at random.

□

Now we can use Lemma 6.12 and Theorem 1 from [261] (under the assumption that the latter holds for H, too) to compute the probability of $\Pr(\mathsf{F}^l(S) \ggg r = \mathsf{F}^l(S \ggg r))$ for a state $S$ and a number of rounds $l$. It is given by:

$$\Pr(\mathsf{F}^l(S) \ggg r = \mathsf{F}^l(S \ggg r) = (9/16)^{4 \cdot 4 \cdot 2 \cdot l}$$

Table 6.3 summarizes the (approximate) weights (i.e., the negative logarithms of the probabilities) for different values of $l$, which are relevant for NORX.

As a consequence, the permutation $F^l$ on a $16W$ state is indistinguishable from a random permutation for $l \geq 20$ if $w = 32$ and for $l \geq 39$ if $w = 64$ with probabilities of $\leq 2^{-531}$ and $\leq 2^{-1035}$ respectively.

Table 6.3: Weights for rotational distinguishers of $\mathsf{F}^l$

| $l$ | 4 | 6 | 8 | 12 |
|---|---|---|---|---|
| $w$ | 106 | 159 | 212 | 318 |

**Remark.** *Khovratovich et al. [262, Appendix B] showed that our estimates were overly pessimistic, and the assumption of independence does not hold. Because there are some* H *operations feeding into each other, the values going into the second* H *in a chain are constrained and decrease the rotational probability.*

**Definition 6.13.** *Let $f$ be a vectorial Boolean function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ and let $a$, $b$ be $n$-bit strings. We call $(a,b)$ a* rotational fixed point *with respect to $f$ if the following equation holds:*

$$f(a,b) \ggg r = f(a,b)$$

**Lemma 6.14.** *Let $f$ be a vectorial Boolean function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$, $(a,b) \mapsto f(a,b)$, which is a permutation on $\{0,1\}^n$, if either $a$ or $b$ is fixed. The probability that $(a,b)$ is a rotational fixed point is:*

$$\Pr[f(a,b) \ggg r = f(a,b)] = 2^{\gcd(r,n)-n}$$

*Proof.* The first important observation is that the statement of this lemma is independent of the function $f$, as it only makes a claim on the image of $f$. Thus it is sufficient to prove the lemma for $z \ggg r = z$, where $z = f(a,b)$ and $a$ or $b$ was fixed.

We identify the indices of an $n$-bit string by the elements in $G := \mathbb{Z}/n\mathbb{Z}$. Let $\tau : G \to G$, $i \bmod n \mapsto (i+1) \bmod n$. Then $\tau$ obviously generates the cyclic group $G$, i.e. $\mathrm{ord}(\tau) = n$. Moreover, for an arbitrary $r \in \mathbb{Z}$ we have $\mathrm{ord}(\tau^r) = n/\gcd(r,n)$, see [425, §6.2]. In other words, the subgroup $H := \langle \tau^r \rangle$ of $G$ has order $n/\gcd(r,n)$. By Lagrange's theorem we have $\mathrm{ord}(G) = [G:H] \cdot \mathrm{ord}(H)$ and it follows for the group index $[G:H] = \gcd(r,n)$, which corresponds to the number of (left) cosets of $H$ in $G$. These cosets contain the indices of a bit string which are mapped onto each other by a rotation $\ggg r$. This means that there are $2^{\gcd(r,n)}$ $n$-bit strings $z$ which satisfy $z \ggg r = z$. Thus the probability, that an $n$-bit string $z$, chosen uniformly at random among all $n$-bit strings, satisfies $z \ggg r = z$ is $2^{\gcd(r,n)-n}$. This proves the lemma. □

A direct consequence of Lemma 6.14 is that for $n$ even and $r = n/2$ the probability that $(a,b)$ is a rotational fixed point is $2^{-n/2}$. The rotation $r = n/2$, which swaps the two halves of a bit string, is especially interesting for cryptanalysis as it results in the highest probability among all $0 < r < n$.

The nonlinear function H of NORX satisfies the requirement of being a permutation on $\{0,1\}^n$ when either of its inputs is fixed. Therefore we get probabilities of $2^{-16}$ (32-bit, $r = 16$) and $2^{-32}$ (64-bit, $r = 32$), that randomly chosen $(a,b)$ is a rotational fixed point of H.

## 6.5 Truncated Differentials

Given a randomly chosen state $S$ and an input difference $\delta$ we define *the (truncated) differential bias* of an output state bit $b_i(S)$ as

$$\mathbf{Pr}[b_i(\mathsf{F}^r(S) \oplus \mathsf{F}^r(S \oplus \delta)) = 0] = \frac{1}{2}(1 + \epsilon_d),$$

where $b_i$ indicates the $i$th bit of $S$. The quantity $\epsilon_d$ may also be interpreted as the correlation between $b_i(\mathsf{F}^r(S))$ and $b_i(\mathsf{F}^r(S \oplus \delta))$, as in linear cryptanalysis (cf. Definition 2.16).

Das, Maitra, and Meier [155, §5.1] reported the following truncated differential, which is injected into the first column of the $4 \times 4$ state of the NORX32 permutation:

$$\texttt{00000C00}, \texttt{80000400}, \texttt{80000000}, \texttt{80000000}.$$

They claimed a noticeable bias of $> 0.01$ on over 20 bits after $2^{27}$ measurements on $\mathsf{F}^2$. The differential in question is strikingly similar to a column differential also present in low-weight differentials for NORX32's $\mathsf{F}^{1.5}$ (cf. Appendix A.2.1). We also observed that placing the same difference in any other column of the state also yields similar biases. This suggests a simple semi-automated approach to find truncated differentials over 4 steps of ChaCha-like permutations:

1. Using automated methods, enumerate low-weight differentials for $\mathsf{F}^{1.5}$;
2. For every 4-word column of the differential trails found in the first step, verify if differential biases are present for the output bits of $\mathsf{F}^2$. If so, save that column.

Using this approach we have found stronger biases than [155], and also other previously unknown truncated differentials for BLAKE2[1] and ChaCha20. Additionally, these differentials are very strong in both directions of the permutation. Table 6.4 lists the truncated differential biases identified for $\mathsf{F}^2$ and its inverse. Some facts are immediately apparent:

- Our differential bias for NORX32 is twice as strong as that of Das et al. [155, §5.1];
- The 64-bit permutations are somewhat weaker than their 32-bit counterparts when it comes to resistance to differential cryptanalysis;
- The lack of integer addition in NORX results in slightly stronger biases, but no significantly higher weaknesses when compared to its ARX counterparts;

[1] The BLAKE2 core primitive is a blockcipher; the BLAKE2 permutation alluded to here refer to its blockcipher with a fixed all-zero key.

Table 6.4: Truncated differential biases for NORX, BLAKE2, and ChaCha. The "Input $\delta$" column indicates state bits corresponding to the input difference; "$\max \log_2 \epsilon_d$" indicates the logarithm of the largest bias found for the corresponding difference. Measurements were performed over $2^{32}$ samples. In the case of ChaCha, the biases apply to 2 double-rounds.

| Permutation | Input $\delta$ | $\max \log_2 \epsilon_d$ for $\mathsf{F}^2$ | $\max \log_2 \epsilon_d$ for $\mathsf{F}^{-2}$ |
|---|---|---|---|
| NORX32 [155] | 10, 11, 138, 159, 287, 415 | −3.46 | −0.80 |
| NORX32 | 10, 138, 159, 287, 415 | −2.46 | −0.77 |
| NORX64 | 63, 575, 775, 831 | −0.00 | −0.05 |
| BLAKE2b | 63, 575, 799, 831 | −1.03 | −0.04 |
| BLAKE2s | 11, 139, 159, 287, 415 | −3.08 | −1.58 |
| ChaCha20 | 19, 147, 159, 287, 415 | −2.59 | −0.68 |

- The inverse permutation is significantly weaker than the forward direction in all cases. This is in stark contrast with the KECCAK permutation, which has a stronger inverse.

Most surprisingly, there are 4 probability-1 truncated differentials for NORX64's $\mathsf{F}^2$. In particular, an input of

8000000000000000, 0000000000000000, 8000000000000000, 8000000000000080

to column $i$ of the state $S$ will result in a difference of 0 at output bit 2 of $S_{i+2 \bmod 4}$.

We can extend these biases further by searching for differential trails that end with the input differentials from Table 6.4. This is accomplished, once again, by the automated search methods of Section 6.2, with the added constraint that the output difference is predefined to a specific value. Table 6.5 presents the best practical truncated differential biases we found. In the case of NORX64, for example, we are able to connect the input difference shown to the output difference in Table 6.4 over $\mathsf{F}^{0.5}$ with a trail of probability $2^{-6}$, yielding a truncated differential for $\mathsf{F}^{2.5}$ with the same probability. Extending the difference further showed to be implausible for practical verification—the best connecting trail over $\mathsf{F}^1$ has probability $2^{-48}$, which we could not verify due to insufficient computational resources. Similar remarks apply to the other primitives shown in Table 6.5.

On the reverse direction, we were able to find low-probability trails for one more round, culminating in very practical differential biases for $\mathsf{F}^{-3.5}$ of NORX64 and BLAKE2b.

Table 6.5: Extended truncated differential biases for NORX, BLAKE2, and ChaCha. Measurements were performed over $2^{32}$ samples.

| Permutation | Function | Input $\Delta$ | $\max \log_2 \epsilon_d$ |
|---|---|---|---|
| NORX32 | $\mathsf{F}^{2.5}$ | 10, 52, 62, 63, 74, 127, 169, 180, 202, 223, 255, 266, 351, 367, 383, 394, 402, 425, 446, 447, 479, 503 | −11.32 |
| NORX32 | $\mathsf{F}^{-3}$ | 84, 224, 229, 230, 245, 260, 261, 287, 420 | −2.89 |
| NORX64 | $\mathsf{F}^{2.5}$ | 63, 146, 199, 255, 402, 447, 455, 511, 575, 703, 711, 743, 751, 767, 775, 831, 959, 1007, 1015 | −6.00 |
| NORX64 | $\mathsf{F}^{-3.5}$ | 36, 63, 336, 337, 344, 345, 357, 376, 377, 381, 382, 655, 656, 663, 664, 695, 696, 700, 701, 975, 983, 1020 | −11.64 |
| BLAKE2b | $\mathsf{F}^{2.5}$ | 63, 151, 223, 255, 407, 447, 479, 511, 575, 703, 719, 735, 751, 767, 799, 831, 959, 975, 1007 | −7.03 |
| BLAKE2b | $\mathsf{F}^{-3.5}$ | 7, 63, 328, 336, 337, 352, 353, 368, 369, 376, 377, 655, 656, 671, 672, 687, 688, 695, 696, 975, 1007, 1015 | −11.63 |
| BLAKE2s | $\mathsf{F}^{2.5}$ | 11, 38, 62, 75, 127, 178, 190, 203, 223, 255, 267, 351, 359, 383, 395, 411, 422, 434, 479, 503 | −13.06 |
| BLAKE2s | $\mathsf{F}^{-3}$ | 83, 228, 229, 236, 248, 267, 268, 287, 427 | −4.96 |
| ChaCha | $\mathsf{F}^{2.5}$ | 31, 83, 111, 127, 211, 223, 239, 255, 287, 351, 359, 367, 375, 383, 399, 415, 479, 487, 503 | −9.94 |
| ChaCha | $\mathsf{F}^{-3}$ | 27, 31, 162, 170, 171, 174, 175, 182, 183, 190, 191, 323, 324, 327, 328, 335, 336, 343, 344, 483, 487, 503 | −11.00 |

## 6.6 Further Applications

### 6.6.1 Tyche

The initialization step of Tyche (Section 3.2) performs 20 iterations of MIX after setting the seed and index. Using the techniques of Section 6.2, we found that the best differential for 4 iterations of MIX has probability $2^{-32}$ and is

$$\begin{pmatrix} 08801889 \\ 88080809 \\ 00080080 \\ 80089088 \end{pmatrix} \xrightarrow[2^{-32}]{\text{MIX}^4} \begin{pmatrix} 88000000 \\ 40404404 \\ 00808088 \\ 00800088 \end{pmatrix}.$$

This upper-bounds the probability of a single 20-iteration differential trail to $\left(2^{-32}\right)^5 = 2^{-160}$, well below the targeted security level. Therefore it appears unlikely that streams with related seeds or indices will appear correlated.

This analysis also strengthens the claim (cf. Section 3.4.2) that Tyche-CTR requires 5 rounds to be statistically indistinguishable —Tyche-CTR-4's small differences between adjacent blocks are detectable with TestU01's $2^{36}$ outputs generated during testing.

### 6.6.2 McMambo

McMambo [284] is an authenticated encryption scheme submitted to CAESAR, which essentially consists of a Salsa20-based tweakable block cipher in the McOE mode [196] and, like NORX, only uses bitwise logical operations.

Using the techniques of Section 6.2, we quickly found an iterative differential trail on the "double-round" D of the block cipher, which invalidates McMambo's security claims and allows an attack to easily forge messages:

$$\begin{pmatrix} 00000000 & 00000000 & 00000000 & 00000000 \\ 00000000 & 00000000 & 00000000 & 00000000 \\ 00000000 & 00000000 & 00000000 & 00000000 \\ 00000000 & 00000000 & 00000000 & \delta \end{pmatrix} \xrightarrow[2^{-2}]{\text{D}} \begin{pmatrix} 00000000 & 00000000 & 00000000 & 00000000 \\ 00000000 & 00000000 & 00000000 & 00000000 \\ 00000000 & 00000000 & 00000000 & 00000000 \\ 00000000 & 00000000 & 00000000 & \delta \end{pmatrix},$$

for any 32-bit $\delta$ of Hamming weight 1. Seeing that the McMambo tweakable block-cipher is composed of 12 double-rounds, the probability of this iterated differential throughout the whole cipher is expected to be $2^{-24}$. In reality there appears to be a strong clustering effect, as experimental results suggest the differential probability is closer to $2^{-17}$.

Such a high probability differential can be exploited many ways; one of them is to easily forge messages by taking a message block $m_i$ and replacing it by $m_i \oplus \delta$, $\delta$

being one of the above input differences. In response to our results, McMambo was withdrawn from the CAESAR competition[2].

### 6.6.3 Wheesht

Wheesht [329] was another CAESAR candidate submitted by Peter Maxwell. The authentication portion of Wheesht consists of the application of function $\theta^3$ to each block of 256-bit ciphertext, as

$$\sum_i \theta^3(c_i \oplus k_i) \oplus k_i \,,$$

with $c_i$ being the $i$th 256-bit block of ciphertext, $k_i$ being a block key, and the sum being performed word-wise on 64-bit words.

Once again, using automated methods we found the probability-1 iterated differential

$$\begin{pmatrix} 8000000000000000 \\ 0000000000000000 \\ 8000000000000000 \\ 8000000000000000 \end{pmatrix} \xrightarrow[1]{\theta} \begin{pmatrix} 8000000000000000 \\ 0000000000000000 \\ 8000000000000000 \\ 8000000000000000 \end{pmatrix},$$

which coupled with the word-wise sum modulo $2^{64}$ of every block value, for which this differential also has probability 1, results in trivial deterministic forgeries by using this difference in any even number of blocks of any message. Our observation was acknowledged by the author[3], and Wheesht did not make it past round 1 of the competition.

### 6.6.4 Cypress

Cypress is a lightweight block cipher family proposed by Rodinko and Oliynykov in 2017 [16, 400]. Like some of the primitives in the previous chapters, Cypress uses a ChaCha-inspired core.

Cypress is a Feistel network, as depicted in Figure 6.2, whose round function P consists of two branches of 4 words each. It uses the ChaCha quarter-round, iterated twice, as the Feistel function, which is keyed at each round by xoring its input by a round key, the derivation of which we omit here. Cypress has two variants, Cypress-256 and Cypress-512, parameterized on the word size:

[2] https://groups.google.com/g/crypto-competitions/c/ysiDA5Qqfrs/m/vUnBeBzkctsJ

[3] https://groups.google.com/g/crypto-competitions/c/16pjT_05NtE/m/9z9mS6aSb1MJ

- For 32-bit words, the ChaCha quarter-round is used verbatim;
- For 64-bit words, the ChaCha quarter-round uses rotation constants 32, 24, 16, 15.

Finally, Cypress-256 performs 10 rounds, while Cypress-512 performs 14.

Using automated search we found the following differential for 4 rounds of Cypress-256:

$$\begin{pmatrix} 81181000 \\ 80081000 \\ 00000000 \\ 01008000 \\ 00000220 \\ 00000260 \\ e0202020 \\ 20002020 \end{pmatrix} \xrightarrow[2^{-40}]{P^4} \begin{pmatrix} 81181000 \\ 80081000 \\ 00000000 \\ 01008000 \\ 00000220 \\ 00000260 \\ e0202020 \\ 20002020 \end{pmatrix}.$$

This differential trail has probability $2^{-40}$, and also conveniently happens to be iterative. However, experimental verification shows that its measured probability after $2^{47}$ trials is approximately $2^{-38}$. This might be due to clustering or data dependencies between Feistel rounds [116]. Combined with the 2-round differential

$$\begin{pmatrix} 81181000 \\ 80081000 \\ 00000000 \\ 01008000 \\ 00000220 \\ 00000260 \\ e0202020 \\ 20002020 \end{pmatrix} \xrightarrow[2^{-22}]{P^4} \begin{pmatrix} 81381000 \\ 80081000 \\ 00000000 \\ 01008000 \\ 00000220 \\ 00000220 \\ 60202020 \\ 20002020 \end{pmatrix},$$

results in a distinguisher for the full blockcipher with complexity $2^{38+38+22} = 2^{98}$.

We found similar results for Cypress-512. In Cypress-512 the best differential trail found for 4 rounds—with probability $2^{-41}$—is not iterative. Nevertheless we did find a good iterative differential

$$\begin{pmatrix} 80013f8001000000 \\ 8000008001000000 \\ 0000000000000000 \\ 0001000080000000 \\ 0000000000202000 \\ 0000000000202000 \\ e000200020002000 \\ 2000000020002000 \end{pmatrix} \xrightarrow[2^{-48}]{P^4} \begin{pmatrix} 80013f8001000000 \\ 8000008001000000 \\ 0000000000000000 \\ 0001000080000000 \\ 0000000000202000 \\ 0000000000202000 \\ e000200020002000 \\ 2000000020002000 \end{pmatrix},$$

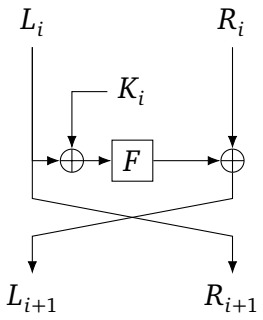



Figure 6.2: Round function P of Cypress.

which, again, combined with a $2^{-21}$ probability 2-round trail results in a distinguisher for the full cipher with complexity $2^{165}$.

While these distinguishers have relatively impractical complexities, they significantly improve the ones in the literature [399, 401, 402] and also show that Cypress does not live up to its security claims. In particular, distinguishers for fewer rounds than the full cipher can be used for key recovery, and those will have lower complexity than the ones presented above. In conclusion, for Cypress-256 and Cypress-512 to be resistant against differential cryptanalysis up to $2^{256}$ (resp $2^{512}$) complexity, they would need at least 28 and 44 rounds, respectively.

# Chapter 7

# Practical Cryptanalysis of the Open Smart Grid Protocol

The Open Smart Grid Protocol (OSGP) [184] is an application layer communication protocol for smart grids built on top of the ISO/IEC 14908-1 protocol stack [242], developed by the Energy Service Network Association (ESNA), and is a standard of the European Telecommunications Standards Institute (ETSI) since 2012 [20]. According to estimates, OSGP-based smart meters and devices are deployed in over 4 million devices worldwide as of 2015, making OSGP one of the most widely used network protocols for smart grid applications.

In this chapter, we investigate the authenticated encryption scheme specified in OSGP. Its authenticated encryption scheme is described in Section 7.1, with special focus on the authentication component of the scheme, the *OMA digest*. In Section 7.2 we present a thorough analysis of the OMA digest. This function has been found to be extremely weak, and cannot be assumed to provide any authenticity guarantee whatsoever. We describe multiple attacks having different levels of applicability in the context of OSGP. The forgery attacks presented in Section 7.2.4 belong to the most powerful and practical, and allow to retrieve the 96-bit secret key in a mere 144 and 168 chosen-plaintext queries to a tag-verification oracle exploiting the very slow propagation of additive and xor-differences in the OMA digest. We also describe how this variant can work as a ciphertext-only attack, making it even more devastating. For easier verifiability, we implemented the attacks of Section 7.2 in the Python programming language; the code is listed in Appendix B.

Table 7.1 summarizes our different attacks on the authenticated encryption scheme of OSGP and also lists their corresponding sections. While the attacks have various tradeoffs between the number of oracle queries and the computational complexity, each constitutes a complete break of the OSGP authenticated encryption scheme.

In summary, the work at hand is another entry in the long list of examples of flawed authenticated encryption schemes, and shows once more how easily a determined

Table 7.1: Required number of queries and expected complexity for the attacks of Section 7.2, with varying time-query tradeoff parameter $B$. The abbreviation KP+ means *known-plaintext with common prefix*, CP denotes *chosen-plaintext*, CC stands for *chosen-ciphertext*, and TG and TV denote *tag-generation* and *tag-verification* oracles, respectively.

| Attack | $B$ | Queries | Complexity | Type | Oracle |
|---|---|---|---|---|---|
| §7.2.1 | 1 | 13 | $2^{3.58}$ | CP | TG |
| | 2 | 7 | $2^{10.58}$ | | |
| | 3 | 5 | $2^{18.00}$ | | |
| | 4 | 4 | $2^{25.58}$ | | |
| | 5 | 4 | $2^{33.58}$ | | |
| | 6 | 3 | $2^{41.00}$ | | |
| §7.2.2 | 1 | 24 / 13 | $2^{10.58}$ | KP+ / CP | TG |
| | 2 | 12 / 7 | $2^{17.58}$ | | |
| | 3 | 8 / 5 | $2^{25.00}$ | | |
| | 4 | 6 / 4 | $2^{32.58}$ | | |
| | 5 | 6 / 4 | $2^{40.32}$ | | |
| | 6 | 4 / 3 | $2^{48.58}$ | | |
| §7.2.4 (xor) | — | $\approx 168$ | $\approx 168$ | CP / CC | TV |
| §7.2.4 (Additive) | — | $\approx 144$ | $\approx 144$ | CP | |

attacker can break the security of protocols based on weak cryptography.

### Related Work

Feiten and Sauer [191] showed that the RC4 encryption key used in the OSGP AE scheme can be recovered given $\approx 90000$ captured messages, thus compromising its confidentiality. In late 2013, Kursawe and Peters independently analysed OSGP and identified several security flaws, some of which overlap with our own findings [281]. Their work gives a good overview on the various security flaws and shows how they can be exploited to mount some basic attacks on OSGP's cryptographic infrastructure. We, on the other hand, focus on the digest function in more detail and, as a consequence, are able to further move the attacks into practicality. We note that our analysis has been performed solely against the OSGP specification [184] and not against any deployed devices.

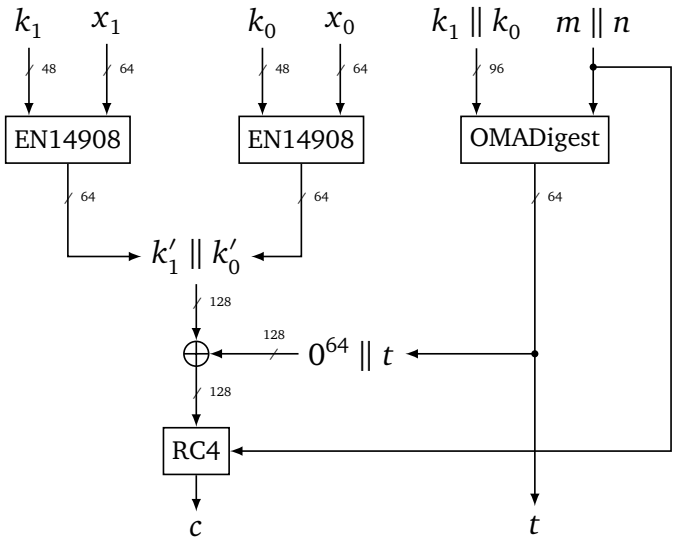



Figure 7.1: The OSGP AE scheme. Notation: $x_0 = \{81, 3F, 52, 9A, 7B, E3, 89, BA\}$, $x_1 = \{72, B0, 91, 8D, 44, 05, AA, 57\}$, $k = k_1 \parallel k_0$ : Open Media Access Key (OMAK), $m$ : message, $n$ : sequence number, $t$ : authentication tag, $k' = k'_1 \parallel k'_0$ : Base Encryption Key (BEK), $c$ : ciphertext.

## 7.1 Preliminaries

We focus solely on OSGP's cryptographic infrastructure, and not on the protocol itself. The high-level structure of OSGP's authenticated encryption scheme is depicted in Figure 7.1.

The OSGP scheme is based on three algorithms: the EN 14908 algorithm[1], the stream cipher RC4 and the so-called OMA digest, a message authentication code (MAC). These three algorithms are combined in a mixture of the generic composition [48] approaches *MAC-and-encrypt* and *MAC-then-encrypt* to form an authenticated encryption scheme, see again Figure 7.1. We note that, while the OMA digest is described in the OSGP specification [184], public information on the EN 14908 algorithm, specified in ISO/IEC 14908-1 [242], is hard to come by. All information on the latter was retrieved from the OSGP specification [184] and the related standard ISO/IEC CD 14543-6-1 [243, p.232] which, like ISO/IEC 14908-1 and a few other standards [18, 239, 430], is also a direct descendant of LonTalk [179].

The security of OSGP's AE scheme depends on the 96-bit *Open Media Access Key* (OMAK) $k = k_1 \parallel k_0$ from which all other key material is derived. The OMAK is usually unique to a device but not hardcoded and can be changed, often to be

[1]The OSGP specification describes EN 14908 as an encryption algorithm, but it is clearly nothing of the sort. We therefore only talk about the *EN 14908 algorithm* in this work.

shared with other devices under the same concentrator [184, §7.1]. Two things are derived from the OMAK: firstly, a so-called *Base Encryption Key* (BEK) $k' = k'_1 \parallel k'_0$ is computed [184, §7.3] which is a 128-bit key forming the basis for the RC4 encryption key. The BEK is constructed[2] using the EN 14908 algorithm which appears to have been the basis for the OMA digest but uses smaller 48-bit keys and processes message bytes in reversed order. The EN 14908 algorithm is applied to each of the halves $k_0$ and $k_1$ of the OMAK and the two constants $x_0 = \{81, 3F, 52, 9A, 7B, E3, 89, BA\}$ and $x_1 = \{72, B0, 91, 8D, 44, 05, AA, 57\}$. The two 64-bit results are then concatenated to form $k'$, see Figure 7.1. Note that the BEK only depends on the OMAK and is thus fixed as long as $k$ remains unchanged.

Secondly, an authentication tag $t$ is produced using the OMA digest on the message $m$ concatenated with a sequence number $n$ and the OMAK $k$. Let $l$ denote the size of $m \parallel n$ in bytes. The OMA digest starts with its 8-byte internal state $a = (a_0, \ldots, a_7)$ set to zero. First, $m \parallel n$ is zero-padded to a multiple of 144 bytes, meaning

$$m' = m \parallel n \parallel 0^{-l \bmod 144} .$$

Let $m' = m'_0 \parallel \cdots \parallel m'_{143}$ denote the first, and possibly only, 144-byte block of the message. The internal state is updated continuously using a nonlinear function $f_{b,c}$ where $b = k_{i \bmod 12, 7-j}$ is a key bit and $c = j$ is the current position in the state. Its specification is as follows:

$$f_{b,c}(x,y,z) = \begin{cases} y + z + (\neg(x + c)) \lll 1 & \text{if } b = 1 \\ y + z - (\neg(x + c)) \ggg 1 & \text{otherwise.} \end{cases}$$

In order to update state element $a_j$, the function $f$ takes, for $0 \le i \le 17$ and $7 \ge j \ge 0$, two adjacent state elements $a_j$ and $a_{j+1 \bmod 8}$ and a message-byte $m'_{8i+7-j}$ as input, i.e., $a_j = f_{k_{i \bmod 12, 7-j}, j}(a_j, a_{j+1 \bmod 8}, m_{8i+7-j})$, and depending on the value of the key bit $k_{i \bmod 12, 7-j}$ one of the two branches depicted above is evaluated. The next 144-byte message block is processed similarly, with the initial internal state carried over from the previous block. The complete pseudocode of the OMA digest is shown in Algorithm 7.1 and a visualization of its innermost loop, where the message bytes are processed, is given in Figure 7.2. For the reference implementation we refer to [184, Annex E].

After the tag generation, $t$ is xored into the lower half of the BEK $k'$ which then produces the final 128-bit RC4 encryption key $k'' = k'_1 \parallel (k'_0 \oplus t)$, see again Figure 7.1. This measure is intended to provide RC4 with ever-changing key material,

[2] The OSGP specification is rather unclear on how the BEK is derived. The presented description is based on our investigations also involving other standards [243, p.232]. The key observation here is that the BEK derived from the OMAK. The concrete realization is not too important, though, and is only described for the sake of completeness.

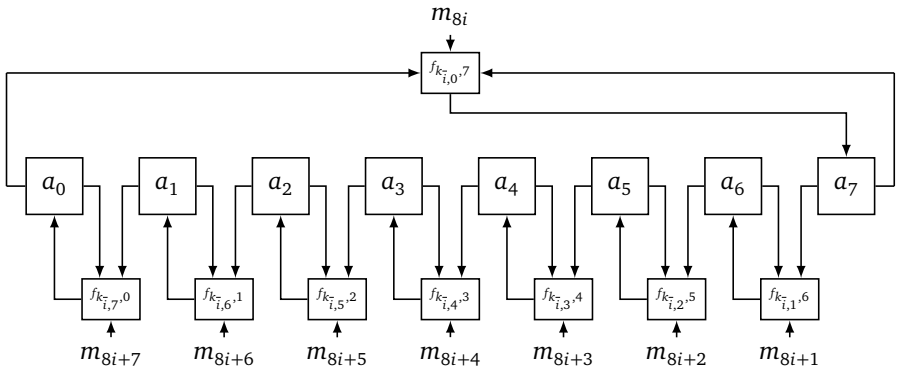



Figure 7.2: Data processing (right-to-left) in the OMA digest, with $\bar{i} = i \bmod 12$.

**Algorithm 7.1:** The OSGP OMA digest.

**Function** `OMADigest(`$m$`,`$k$`)`
  $a \leftarrow (0,0,0,0,0,0,0,0)$;
  $m \leftarrow m \,\|\, 0^{-|m| \bmod 144}$;
  **foreach** 144-*byte block* $b$ *of* $m$ **do**
    **for** $i \leftarrow 0$ **to** 17 **do**
      **for** $j \leftarrow 7$ **to** 0 **do**
        **if** $k_{i \bmod 12, 7-j} = 1$ **then**
          $a_j \leftarrow a_{(j+1) \bmod 8} + b_{8i+(7-j)} + (\neg(a_j + j)) \lll 1$;
        **else**
          $a_j \leftarrow a_{(j+1) \bmod 8} + b_{8i+(7-j)} - (\neg(a_j + j)) \ggg 1$;
        **end**
      **end**
    **end**
  **end**
  **return** $a$;

thus producing a fresh keystream with every new message, since, according to the OSGP specification, the sequence number $n$, which is appended to $m$, is continuously increased.

Sequence numbers are shared between sender and receiver in OSGP. The receiver of a message verifies that the correct sequence number was appended to the latter. Messages with sequence numbers in the range $\{n, \ldots, n+8\}$ are accepted as valid requests. If a message with sequence number $n-1$ is received, then the recipient does not execute the request but instead re-sends the answer of the (previously executed) request of number $n-1$. Sequence numbers outside of this range trigger an error and the OSGP device replies with a failure code and the correct sequence number. More details on the handling of sequence numbers can be found in [184, §9.7].

After the setup phase is finished, $k''$ is used to encrypt $m \,\|\, n$ via RC4 to obtain

the ciphertext $c$. Finally, $c \parallel t$ is transmitted. Messages $m \parallel n$ processed in OSGP are allowed to have a maximum size of 114 bytes [184, §9.2]. This complicates some attacks that require up to 136-byte messages. Nevertheless, we will also describe scenarios that respect this message size limit.

## 7.2 Analysis

OSGP uses RC4 for encryption without discarding any initial bytes. RC4 has known statistical key- and plaintext-recovery attacks, and these have been shown to be practical [7, 197, 198, 217, 416, 438, 456]. However, in this work we do not focus on RC4, but instead on the OMA digest, see Algorithm 7.1.

The OMA digest algorithm presents multiple flaws. Firstly, it uses a simple zero byte message padding, which results in messages with any number of trailing zeroes sharing the same tag. Secondly, given a tuple $(a, m, k)$ where $a$ is the OMA digest's state or authentication tag, $m$ a message and $k$ the OMAK, the function is fully reversible (see Algorithm 7.2) which is a very useful property for the attacks presented in Sections 7.2.1 and 7.2.2. Likewise, it is also possible to take an arbitrary internal state, and continue to process it as if to resume a partially digested message. This is depicted in Algorithm 7.3.

**Algorithm 7.2:** The "backward" OSGP OMA digest, reverting the internal state back by $n$ message bytes.

**Function** `OMABackward(a,m,k,n)`
  `// Assumes` $|m| \leq 144$.
  $m \leftarrow m \parallel 0^{-|m| \bmod 144}$;
  **for** $l \leftarrow 0$ **to** $n-1$ **do**
    $i, j \leftarrow \lfloor l/8 \rfloor, l \bmod 8$;
    **if** $k_{(17-i) \bmod 12, 7-j} = 1$ **then**
      $x \leftarrow (a_j - a_{(j+1) \bmod 8} - m_{143-8i-j}) \ggg 1$;
    **else**
      $x \leftarrow (a_{(j+1) \bmod 8} + m_{143-8i-j} - a_j) \lll 1$;
    **end**
    $a_j \leftarrow \neg x - j$;
  **end**
  **return** $a$;

**Algorithm 7.3:** The "forward" OSGP OMA digest, starting with a known initial state and processing message bytes starting at position $n$.

**Function** `OMAForward(a,m,k,n)`
`/* Essentially Algorithm 7.1, but start at byte` $m_n$ `with a known state` $a$`, and assume` $|m| \leq 144$`. */`
$m \leftarrow m \parallel 0^{-|m| \bmod 144}$;
**for** $l \leftarrow n$ **to** 143 **do**
  $i, j \leftarrow \lfloor l/8 \rfloor, 7 - l \bmod 8$;
  **if** $k_{i \bmod 12, 7-j} = 1$ **then**
    $a_j \leftarrow a_{(j+1) \bmod 8} + m_{8i+7-j} + (\neg(a_j + j)) \lll 1$;
  **else**
    $a_j \leftarrow a_{(j+1) \bmod 8} + m_{8i+7-j} - (\neg(a_j + j)) \ggg 1$;
  **end**
**end**
**return** $a$;

### 7.2.1 Chosen-Plaintext Key Recovery Attacks

Let $a = (a_0, \ldots, a_7)$ denote the 8-byte internal state of the OMA digest. The attacks discussed below use chosen 144-byte messages $m = m_0 \parallel \cdots \parallel m_{143}$[3], and exploit differential weaknesses in the OMA digest.

#### Bitwise Key Recovery

The first attack recovers the key one bit at a time by differential cryptanalysis. Specifically, we exploit the xor-differential $(\Delta m_i, \Delta a_j) = (80, 80)$, where $\Delta m_i$ and $\Delta a_j$ denote input and output differences, respectively, for $j = 7 - i \bmod 8$. The output difference is obtained immediately after processing message byte $m_i$ (see Algorithm 7.1) and can be written as

$$\begin{aligned} & f_{k,j}(a_j, a_{j+1 \bmod 8}, m_i \oplus 80) \\ &= a_{j+1 \bmod 8} + (m_i \oplus 80) \pm (\mathsf{FF} \oplus (a_j + j) \lll r) \\ &= (a_{j+1 \bmod 8} + m_i \pm (\mathsf{FF} \oplus (a_j + j) \lll r)) \oplus 80 \\ &= f_{k,j}(a_j, a_{j+1 \bmod 8}, m_i) \oplus 80 \end{aligned}$$

where the rotation offset $r \in \{1, 7\}$ and the $\pm$ operation depend on the value of the key bit $k \in \{0, 1\}$. This differential has probability 1, by well-known differential properties of addition modulo $2^n$ [299], and propagates cleanly through the state $a$

[3] For simplicity, we use 144-byte messages throughout this section. Note, however, that the presented attacks use messages which are never longer than 136 bytes.

for the next 8 iterations, resulting in the following difference over the state:

$$\Delta a = (80, 80, 80, 80, 80, 80, 80, 80)\ .$$

The next iteration reveals one key bit. By xor-linearizing the state update function $f$, the new output difference $\Delta a'_j$ is of the form

$$\begin{aligned}\Delta a'_j = ((a_{j+1 \bmod 8} \oplus 80) \oplus m_i \oplus (\mathsf{FF} \oplus ((a_j \oplus 80) \oplus j) \lll r)) \oplus \\ (a_{j+1 \bmod 8} \oplus m_i \oplus (\mathsf{FF} \oplus (a_j \oplus j) \lll r))\end{aligned}$$

where $r \in \{1, 7\}$. As a consequence, we have $\Delta a'_j = 81$, if bit $7-i \bmod 8$ of $k_{\lfloor i/8 \rfloor \bmod 12}$ is 1, and $\Delta a'_j = \mathsf{C0}$, if the same key bit is 0. While integer addition and xor behave differently with respect to the propagation of xor-differences, the *least significant bit* of integer addition and xor behave identically in this case and can be used to recover the key bit with probability 1.

The above leak, combined with Algorithm 7.2, can be turned into a chosen-plaintext key-recovery attack retrieving the OMAK $k$ bitwise in at most $96+1$ queries. Algorithm 7.4 describes this attack in full detail. Looking at Figure 7.1, we see immediately that the reconstruction of $k$ breaks the complete OSGP AE scheme. In the following, we will explore how the attack can be further improved.

**Algorithm 7.4:** Bit-by-bit chosen-plaintext key-recovery attack.

**Function** `RecoverKey(`$\mathcal{O}$`)`
  `//` $\mathcal{O}$ `is an oracle returning a message's OMADigest under key` $k$.
  $k \leftarrow \{0\}^{12}$;
  $m \xleftarrow{\$} \{0..255\}^{144}$;
  $a \leftarrow \mathcal{O}(m)$;
  **for** $i \leftarrow 0$ **to** 11 **do**
    **for** $j \leftarrow 0$ **to** 7 **do**
      $m' \leftarrow m$;
      $m'_{136-8i-1-j} \leftarrow m'_{136-8i-1-j} \oplus 80$;
      $a' \leftarrow \mathcal{O}(m')$;
      $b \leftarrow \mathsf{OMABackward}(a, m, k, 8i)$ `// Algorithm 7.2;`
      $b' \leftarrow \mathsf{OMABackward}(a', m', k, 8i)$ `// Algorithm 7.2;`
      $k_{(17-i) \bmod 12, 7-j} \leftarrow (b_{j,0} \oplus b'_{j,0})$;
    **end**
  **end**
  **return** $k$;

**Bytewise Key Recovery**

Analysing the above attack more thoroughly, we noticed that we can recover one key byte at a time by injecting the input difference 80 into the message a couple of steps earlier. This reduces the number of queries and the work load of the attack drastically. In other words, we will show how to reconstruct the entire OMAK with only $12+1$ chosen-plaintext queries.

Let $k_{i \bmod 12,j}$ denote the $j$th bit of key byte $i \bmod 12$, for $i = 17, 16, \ldots, 6$ and $j = 0, \ldots, 7$. When injecting the message difference $\Delta m_{8i-8} = 80$ and thereupon processing 16 message bytes, we obtain an xor-difference of the internal state of the form $\Delta a = (\Delta a_0, \ldots, \Delta a_7) = (\Delta x_0, \ldots, \Delta x_7)$ where $\Delta x_l$ are arbitrary values for $l = 0, \ldots, 7$. The evolution of the difference propagation in the internal state can be visualized as follows:

| $i = 17, \ldots, 6$ | $\Delta a_0$ | $\Delta a_1$ | $\Delta a_2$ | $\Delta a_3$ | $\Delta a_4$ | $\Delta a_5$ | $\Delta a_6$ | $\Delta a_7$ |
|---|---|---|---|---|---|---|---|---|
| $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ |
| $m_{8i-9}$ | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| $m_{8i-8}$ | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 80 |
| $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ |
| $m_{8i-1}$ | 80 | 80 | 80 | 80 | 80 | 80 | 80 | 80 |
| $m_{8i}$ | 80 | 80 | 80 | 80 | 80 | 80 | 80 | $\Delta x_7$ |
| $m_{8i+1}$ | 80 | 80 | 80 | 80 | 80 | 80 | $\Delta x_6$ | $\Delta x_7$ |
| $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ |
| $m_{8i+7}$ | $\Delta x_0$ | $\Delta x_1$ | $\Delta x_2$ | $\Delta x_3$ | $\Delta x_4$ | $\Delta x_5$ | $\Delta x_6$ | $\Delta x_7$ |

By analyzing again the xor-linearization of the state update function $f$, one realizes that a key byte can be recovered in its entirety by exploiting, as in the case of the bitwise key recovery attack, the information on the key bits stored in the least significant bit of the output differences $\Delta x_0, \ldots, \Delta x_7$. More precisely, key byte $k_{i \bmod 12}$ can be reconstructed as follows:

1. $k_{i \bmod 12,0} = \mathtt{lsb}(\Delta x_7) \oplus \mathtt{lsb}(80)$
2. $k_{i \bmod 12,1} = \mathtt{lsb}(\Delta x_6) \oplus \mathtt{lsb}(\Delta x_7)$
3. $k_{i \bmod 12,2} = \mathtt{lsb}(\Delta x_5) \oplus \mathtt{lsb}(\Delta x_6)$
4. $k_{i \bmod 12,3} = \mathtt{lsb}(\Delta x_4) \oplus \mathtt{lsb}(\Delta x_5)$
5. $k_{i \bmod 12,4} = \mathtt{lsb}(\Delta x_3) \oplus \mathtt{lsb}(\Delta x_4)$
6. $k_{i \bmod 12,5} = \mathtt{lsb}(\Delta x_2) \oplus \mathtt{lsb}(\Delta x_3)$
7. $k_{i \bmod 12,6} = \mathtt{lsb}(\Delta x_1) \oplus \mathtt{lsb}(\Delta x_2)$
8. $k_{i \bmod 12,7} = \mathtt{lsb}(\Delta x_0) \oplus \mathtt{lsb}(\Delta x_1)$

In order to verify that the above key recovery indeed works, consider the following steps. As we have already seen in the bitwise key recovery attack, the value of $k_{i \bmod 12,0}$ can be read off right away from $\Delta x_7$, see step 1 above. The remaining key bits $k_{i \bmod 12,j+1}$, for $j = 0, \ldots, 6$, can be recovered from the xor-linearization of $f$ which gives us the relation

$$\Delta x_{6-j} = \Delta x_{7-j} \oplus (\Delta x'_{6-j} \lll r) = \Delta x_{7-j} \oplus (80 \lll r)$$

where $\Delta x_{7-j}$ and $\Delta x_{6-j}$ denote output differences and $\Delta x'_{6-j}$ corresponds to the difference before $a_{6-j}$ is updated in the $j$th step. The latter simply has the value 80 as can be seen in the table on the difference propagation. The above equation can be re-written as

$$\mathtt{lsb}(80 \lll r) = \mathtt{lsb}(\Delta x_{6-j}) \oplus \mathtt{lsb}(\Delta x_{7-j})$$

and since the rotation offset $r \in \{1, 7\}$ depends on $k_{i \bmod 12, j+1}$, the formula above gives us the value of the latter key bit.

**Algorithm 7.5:** Byte-by-byte chosen-plaintext key-recovery attack.

**Function** `RecoverKey(`$\mathcal{O}$`)`
  `//` $\mathcal{O}$ `is an oracle returning a message's OMADigest under key` $k$`.`
  $k \leftarrow \{0\}^{12}$;
  $m \xleftarrow{\$} \{0..255\}^{144}$;
  $a \leftarrow \mathcal{O}(m)$;
  **for** $i \leftarrow 0$ **to** 11 **do**
    $m' \leftarrow m$;
    $m'_{136-8i-8} \leftarrow m'_{136-8i-8} \oplus 80$;
    $a' \leftarrow \mathcal{O}(m')$;
    $b \leftarrow \mathtt{OMABackward}(a, m, k, 8i)$ `// Algorithm 7.2;`
    $b' \leftarrow \mathtt{OMABackward}(a', m', k, 8i)$ `// Algorithm 7.2;`
    $k_{(17-i) \bmod 12} \leftarrow \mathtt{RecoverByte}(b, b')$;
  **end**
  **return** $k$;
**Function** `RecoverByte(`$a, a'$`)`
  $x \leftarrow 0$;
  $x_0 \leftarrow a_{7,0} \oplus a'_{7,0}$;
  **for** $i \leftarrow 0$ **to** 6 **do** $x_{i+1} \leftarrow a_{6-i,0} \oplus a'_{6-i,0} \oplus a_{7-i,0} \oplus a'_{7-i,0}$ ;
  **return** $x$;

### 7.2.2 Known-Plaintext Key Recovery Attack

The second attack is not differential in nature and requires a weaker attacker. We only assume in the following that the attacker is able to capture plaintexts with a *common prefix* of various lengths. This may be feasible by, e.g., capturing repeated messages with different sequence numbers.

This attack relies uniquely on the OMA digest's invertibility, as seen in Algorithm 7.2. The basic idea here is to have two messages, $m$ and $m'$ that are equal except in the last $r$ bytes; partially reversing the final state of $m$ by $r$ iterations, then using that state to process the final bytes of $m'$ should only happen when the

(guessed) key bits used in those iterations are correct. This does not always happen, but it reduces the keyspace to virtually one or two guesses per key byte. The concrete realization of the attack is also described in Algorithm 7.6.

However, due to the slow diffusion of differences already described in Section 7.2.1, to recover $r$ bits of the key one needs more than $r$ iterations back; this is not a problem, though, as long as the key bits corresponding to the common prefix bytes of the message are the same for the forwards and backwards processing of the message. In practice, we have found that $r + 8$ iterations suffice to recover the key with overwhelming probability.

**Algorithm 7.6:** Byte-by-byte known-plaintext key-recovery attack.

**Function** `RecoverKey(`$\mathcal{O}$`)`
  `//` $\mathcal{O}$ `is an oracle returning a message's OMADigest under key` $k$`.`
  $k \leftarrow \{0\}^{12}$;
  $m \xleftarrow{\$} \{0..255\}^{144}$;
  $a \leftarrow \mathcal{O}(m)$;
  **for** $i \leftarrow 0$ **to** 11 **do**
    $m' \leftarrow m$;
    $m'_{128-8i..|m'|-1} \xleftarrow{\$} \{0..255\}^{|m|-128-8i}$;
    $a' \leftarrow \mathcal{O}(m')$;
    **for** $x \leftarrow 0$ **to** 255 **do**
      $k_{(17-i) \bmod 12} \leftarrow x$;
      $b \leftarrow$ `OMABackward`$(a, m, k, 8i + 16)$ `// Algorithm 7.2`;
      $b' \leftarrow$ `OMAForward`$(b, m', k, 128 - 8i)$ `// Algorithm 7.3`;
      **if** $a' = b'$ **then**
        **break** `// May be a false positive; handling omitted.`
      **end**
    **end**
  **end**
  **return** $k$;

### 7.2.3 Optimizing the Attacks

The attacks of Sections 7.2.1 and 7.2.2 have an obvious generalization that trades queries for computation time. This is also a consequence of the OMA digest's reversibility.

Let $B \geq 1$ be the number of key bytes to recover per query; the attack from Section 7.2.2 generalizes trivially to any $B$, by guessing $B$ adjacent key bytes per

query, at an average cost of $\lceil \frac{12}{B} \rceil + 1$ queries and $\lceil \frac{12}{B} \rceil 2^{8B-1}$ operations[4].

The method from Section 7.2.1 also generalizes well to any $B$, by guessing the last $B-1$ bytes and recovering the first one by injecting a difference. Its average cost is $\lceil \frac{12}{B} \rceil + 1$ queries and $\lceil \frac{12}{B} \rceil 2^{8(B-1)-1}$ operations. We note that for $B \geq 2$ the messages used in either case need not be longer than 113 bytes, bypassing OSGP's restriction on message sizes.

### 7.2.4 Forgeries and a Third Key-Recovery Attack

Forgeries in the OMA digest are possible by exploiting the differential properties described in Section 7.2.1. To this end, we first explore xor differentials and afterwards describe attacks using additive differentials.

#### Forgeries using xor-Differentials

For this attack, we consider input xor-differences of the shape

$$(\Delta m_{8i+j}, \Delta m_{8i+j+1}, \Delta m_{8i+j+8}) = (80, 80, \Delta x)$$

for $i = 0, \ldots, 17$ and $j = 0, \ldots, 7$. After processing message bytes $m_{8i+j}, m_{8i+j+1}, \ldots, m_{8i+j+7}$, the xor-differences in the internal state are, up to a rotation, of the form $\Delta a = (80, 00, 00, 00, 00, 00, 00, 00)$. More precisely, after injecting $\Delta m_{8i+j} = 80$, the difference $\Delta m_{8i+j+1} = 80$ is used to prevent the difference of $\Delta m_{8i+j}$ from spreading to the rest of the state. Creating this stationary difference can be achieved with probability 1. Finally, the difference $\Delta m_{8i+j+8} = \Delta x$ is used to cancel the stationary difference from above thereby creating a forgery. The success of the forgery hinges on whether the formula

$$(m_{8i+j+8} \oplus \Delta x) \pm (\mathtt{FF} \oplus ((a_j \oplus 80) + j) \lll r) = m_{8i+j+8} \pm (\mathtt{FF} \oplus (a_j + j) \lll r)$$

is satisfied. Note that the above formula again includes both possible cases which depend on the value of the key bit $k \in \{0, 1\}$. Using the formulas of Lipmaa and Moriai [299], we can determine the optimal value for $\Delta x$ with respect to its probability $p$ and the value of the key bit $k_{i+1 \bmod 12, j}$:

| $k_{i+1 \bmod 12, j}$ | 0 | | 1 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $\Delta x$ | C0 | 40 | 01 | 03 | 07 | 0F | 1F | 3F | 7F | FF |
| $-\log_2 p$ | 1 | 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 7 |

Thus, choosing $\Delta x \in \{\mathtt{C0}, 40, 01\}$ has a probability of about $1/4$ of creating a valid forgery, assuming a uniformly random key bit.

[4] An "operation" here is taken to mean at most the cost of an OMA digest evaluation over a message.

### Forgeries using Additive Differentials

Injecting additive differences is also useful to get a wider range of possible high-probability differences, since every operation in the OMA digest, with the exception of the cyclic rotation, has additive differential probability 1[5].

Using a similar approach as above, one can inject the additive difference $(\Delta^{\boxminus}x, -\Delta^{\boxminus}x, -\Delta^{\boxminus}y)$ at $(m_i, m_{i+1}, m_{i+8})$. The success of the forgery here depends on the quality of the approximations

$$\Delta^{\boxminus}y = ((-a_j - j - 1) \lll 1) - ((-a_j - \Delta^{\boxminus}x - j - 1) \lll 1)$$
$$\Delta^{\boxminus}y = -((-a_j - j - 1) \ggg 1) + ((-a_j - \Delta^{\boxminus}x - j - 1) \ggg 1)$$

for $a_j$ chosen uniformly at random. Since cyclic rotation is not a deterministic operation with respect to additive differences, one cannot obtain $\Delta^{\boxminus}y$ that works with probability 1. By replacing $((-a_j - \Delta^{\boxminus}x - j - 1) \lll 1)$ by $((-a_j - j - 1) \lll 1) + (-\Delta^{\boxminus}x \lll 1)$, and taking advantage of Daum's results on the interaction of integer addition and rotation [157], we have $\Delta^{\boxminus}y = -((-\Delta^{\boxminus}x \lll 1) - 2\alpha + \beta)$, where $(\alpha, \beta)$ has, as a function of $\Delta^{\boxminus}x_R = \lfloor(-\Delta^{\boxminus}x)/2\rfloor$ and $\Delta^{\boxminus}x_L = (-\Delta^{\boxminus}x) \bmod 2^7$, one of the following values of probability $p$:

| $(\alpha, \beta)$ | $p$ |
|---|---|
| $(0,0)$ | $2^{-8}(2^7 - \Delta^{\boxminus}x_R)(2 + \Delta^{\boxminus}x_L)$ |
| $(0,1)$ | $2^{-8}\Delta^{\boxminus}x_R(2 - \Delta^{\boxminus}x_L - 1)$ |
| $(1,0)$ | $2^{-8}(2^7 - \Delta^{\boxminus}x_R)\Delta^{\boxminus}x_L$ |
| $(1,1)$ | $2^{-8}\Delta^{\boxminus}x_R(\Delta^{\boxminus}x_L + 1)$ |

Similar remarks apply to the rotation by 7 case. By choosing $\Delta^{\boxminus}x$ carefully, one can maximize the probability of $\Delta^{\boxminus}y$ as well, as also previously exploited by Daum [157]. For instance, choosing the difference $\Delta^{\boxminus}x = \mathtt{02}$, one obtains $\Delta^{\boxminus}y \in \{\mathtt{01}, \mathtt{FC}, \mathtt{81}, \mathtt{FB}, \mathtt{FD}\}$, with respective probabilities $\{127/256, 126/256, 1/256, 1/256, 1/256\}$. Therefore, one can expect 2 queries to be sufficient in over $\approx 98\%$ of the time with this method.

### Using Forgeries for Key Recovery

Such a high-probability forgery attack, dependent on the value of key bits, gives us yet another attack vector for key recovery. This attack is much simpler than the previous ones, and unlike those it does not need to work "right to left" on the message bytes: given a known plaintext, inject $(\mathtt{02}, -\mathtt{02}, -\Delta^{\boxminus}y)$ and query a verification oracle. If the forged message is validated, recover the key bit corresponding to $m_{i+8}$ by looking

[5] Note that $\neg x = x \oplus \mathtt{FF} = -x - 1$.

up which $\Delta^{\boxminus} y$ corresponds to which key bit. This process can be repeated 96 times to recover the entire key.

Additionally, this attack can work even over ciphertext, by using the xor-differences $(80, 80, \Delta x)$ with $\Delta x \in \{40, C0, 01\}$. The approach here is the same, albeit requiring a few more queries, but it can be applied over unknown ciphertext encrypted with RC4, as is the case with OSGP. The attack thus completely breaks not only the OMA digest, but also the entire cryptographic security of OSGP.

The average number of queries can be reduced by using the following trick: instead of picking a difference at random from the possible set of differences, pick C0 and 40 in order. If none of them results in a forgery, the key bit can only be 1; this results in key recovery in an average of 168 queries. Algorithm 7.7 illustrates the xor key-recovery attack on OSGP using this trick, only taking as input a valid ciphertext-tag pair and an oracle that verifies *ciphertexts*.

**Algorithm 7.7:** Bit-by-bit chosen-ciphertext key-recovery attack, in the context of the OSGP protocol.

**Function** RecoverKey($\mathcal{O}, c, a$)
  // $\mathcal{O}$ is an oracle that returns 1 if $(c,a)$ is a valid OSGP ciphertext-tag pair, 0 otherwise.
  // $c,a$ is a valid OSGP ciphertext-tag pair, i.e., $\mathcal{O}(c,a) = 1$.
  $k \leftarrow \{0\}^{12}$;
  **for** $i \leftarrow 0$ **to** 95 **do**
    $c' \leftarrow c$;
    $c'_i \leftarrow c_i \oplus 80$;
    $c'_{i+1} \leftarrow c_{i+1} \oplus 80$;
    $c'_{i+8} \leftarrow c_{i+8} \oplus C0$;
    **if** $\mathcal{O}(c', a) = 1$ **then**
      $k_{\lfloor (i+8)/8 \rfloor \bmod 12, (i+8) \bmod 8} \leftarrow 0$;
      **continue**;
    **end**
    $c'_{i+8} \leftarrow c_{i+8} \oplus 40$;
    $k_{\lfloor (i+8)/8 \rfloor \bmod 12, (i+8) \bmod 8} \leftarrow 1 - \mathcal{O}(c', a)$;
  **end**
  **return** $k$;

### 7.2.5 Extension of the OSGP Analysis to Other Standards

The EN 14908 algorithm, used in OSGP for key derivation and quite similar to the OMA digest, is also used in other LonTalk-derived standards for authentication [18, 179, 239, 242, 243, 430]. We found evidence that the foundations of the technology

(presumably also including the EN 14908 algorithm) were laid in 1988 [302, p.3]. LonTalk was estimated to be implemented in over 90 million devices as of 2010 [178]. Given that the EN 14908 algorithm has a 48-bit key, it is already broken by design. That said, the attacks described in the previous sections can be adapted to key recovery attacks on the EN 14908 algorithm—likely present in every other LonTalk-derived standard—in much less than $2^{48}$ work.

# Chapter 8

# Conclusion

Over the course of this thesis I have presented several related designs, all sharing the same purpose—to reduce unnecessary choices between performance and safety in current and upcoming hardware.

The first way I did this was with Tyche. This is a random number generator, not aimed at cryptographic applications, whose focus is to be fast and keep a very small state. The small state requirement is important for many-core applications, where it is not practical to synchronize a single generator, and the combined size of all generators should not be a significant part of the computation. My goal here was that users should not have to sacrifice random number quality to accomplish this, and performance should be good across the board. In particular Tyche does not use wide integer multiplications, which have varying performance from platform to platform. Tyche has excellent performance even in desktop machines, especially when vectorized, and does not suffer from statistical defects to get there.

The second way was with BLAKE2. Interestingly, while most design challenges are about what to put in a primitive, one of the main challenges with BLAKE2 was what to take away from its predecessor, BLAKE. While BLAKE was the fastest SHA-3 candidate on desktop processors, it was still quite slower than the ever-present MD5 and SHA-1, which have long resisted their obsolescence. With the introduction of BLAKE2 that was no longer the case. BLAKE2 improves over MD5 in nearly every way, so much so that it has been adopted by numerous other cryptographic constructions as a building block, e.g., [10, 90, 91, 122, 199, 232, 235, 426], and also in various real-world products, e.g., RFC 7693 [410], the RAR file format [408], Zcash [236], Noise [382], Wireguard [170], librsync, the Linux kernel's random number generator, and many others. Overall I believe that the BLAKE2 design contributed in a tangible, even if small, way to a more secure Internet.

The third contribution was in the field of authenticated encryption. NORX, my submitted design, was one of the fastest functions in the CAESAR competition that did not rely on hardware accelerated AES. While ultimately NORX was not selected

as a member of the final CAESAR portfolio—with 4 out of the 6 CAESAR winners relying on AES hardware acceleration to be efficient—it did pave the way for later designs inspired by it such as Gimli [61]. In view of this I believe that while NORX may not have been a real-world adoption success like BLAKE2, it did prove to be a successful research cipher.

The fourth way I contributed was less tangible, and it consisted mainly of cryptanalysis. First, my cryptanalytic results on NORX solidified the confidence on the design of the NORX core permutation. Secondly my results on McMambo and Wheesht directly helped the CAESAR committee to select the candidates for the next round of the competition. My work on the OSGP authenticated encryption algorithm sped up the adoption of an adequate replacement in its updated standard, the OSGP-AES-128-PSK cipher suite. And more indirectly, my OSGP work also demonstrated why new schemes should not be adopted or standardized without public cryptanalysis; novel designs, even by experienced cryptographers, often get broken.

I can therefore conclude that security need not come at the cost of performance or complexity. Indeed, if the design process takes into account the target hardware and ease of implementation, safer primitives may also be faster and simpler than existing primitives that do not take implementation aspects into account.

# Bibliography


[1] Josh Aas, Richard Barnes, Benton Case, Zakir Durumeric, Peter Eckersley, Alan Flores-López, J. Alex Halderman, Jacob Hoffman-Andrews, James Kasten, Eric Rescorla, Seth D. Schoen, and Brad Warren. "Let's Encrypt: An Automated Certificate Authority to Encrypt the Entire Web". In: *ACM CCS 2019: 26th Conference on Computer and Communications Security*. Ed. by Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz. ACM Press, Nov. 2019, pp. 2473–2487. DOI: 10.1145/3319535.3363192 (cit. on p. 1).

[2] Harold Abelson, Ross J. Anderson, Steven M. Bellovin, Josh Benaloh, Matt Blaze, Whitfield Diffie, John Gilmore, Matthew Green, Susan Landau, Peter G. Neumann, Ronald L. Rivest, Jeffrey I. Schiller, Bruce Schneier, Michael A. Specter, and Daniel J. Weitzner. "Keys under doormats: mandating insecurity by requiring government access to all data and communications". In: *J. Cybersecur.* 1.1 (2015), pp. 69–79. URL: https://doi.org/10.1093/cybsec/tyv009 (cit. on p. 2).

[3] Ruba Abu-Salma, M. Angela Sasse, Joseph Bonneau, Anastasia Danilova, Alena Naiakshina, and Matthew Smith. "Obstacles to the Adoption of Secure Communication Tools". In: *2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2017, pp. 137–153. DOI: 10.1109/SP.2017.65 (cit. on p. 2).

[4] *Advanced Encryption Standard (AES)*. National Institute of Standards and Technology, NIST FIPS PUB 197, U.S. Department of Commerce. Nov. 2001 (cit. on p. 5).

[5] William Aiello and Ramarathnam Venkatesan. "Foiling Birthday Attacks in Length-Doubling Transformations - Benes: A Non-Reversible Alternative to Feistel". In: *Advances in Cryptology – EUROCRYPT'96*. Ed. by Ueli M. Maurer. Vol. 1070. Lecture Notes in Computer Science. Springer, Heidelberg, May 1996, pp. 307–320. DOI: 10.1007/3-540-68339-9_27 (cit. on p. 17).

[6] Martin R. Albrecht, Kenneth G. Paterson, and Gaven J. Watson. “Plaintext Recovery Attacks against SSH”. In: *2009 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2009, pp. 16–26. DOI: 10.1109/SP.2009.5 (cit. on pp. 22, 77).

[7] Nadhem J. AlFardan, Daniel J. Bernstein, Kenneth G. Paterson, Bertram Poettering, and Jacob C. N. Schuldt. “On the Security of RC4 in TLS”. In: *USENIX Security 2013: 22nd USENIX Security Symposium*. Ed. by Samuel T. King. USENIX Association, Aug. 2013, pp. 305–320 (cit. on pp. 2, 136).

[8] Nadhem J. AlFardan and Kenneth G. Paterson. “Lucky Thirteen: Breaking the TLS and DTLS Record Protocols”. In: *2013 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2013, pp. 526–540. DOI: 10.1109/SP.2013.42 (cit. on p. 77).

[9] Josh Alman and Virginia Vassilevska Williams. “A Refined Laser Method and Faster Matrix Multiplication”. In: *32nd Annual ACM-SIAM Symposium on Discrete Algorithms*. Ed. by Dániel Marx. ACM-SIAM, Jan. 2021, pp. 522–539. DOI: 10.1137/1.9781611976465.32 (cit. on p. 37).

[10] Leonardo C. Almeida, Ewerton R. Andrade, Paulo S. L. M. Barreto, and Marcos A. Simplício Jr. “Lyra: password-based key derivation with tunable memory and processing costs”. In: *J. Cryptographic Engineering* 4.2 (2014), pp. 75–89. URL: http://dx.doi.org/10.1007/s13389-013-0063-5 (cit. on p. 147).

[11] Elena Andreeva, Andrey Bogdanov, and Bart Mennink. “Towards Understanding the Known-Key Security of Block Ciphers”. In: *Fast Software Encryption – FSE 2013*. Ed. by Shiho Moriai. Vol. 8424. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2014, pp. 348–366. DOI: 10.1007/978-3-662-43933-3_18 (cit. on p. 27).

[12] Elena Andreeva, Charles Bouillaguet, Orr Dunkelman, Pierre-Alain Fouque, Jonathan J. Hoch, John Kelsey, Adi Shamir, and Sébastien Zimmer. “New Second-Preimage Attacks on Hash Functions”. In: *Journal of Cryptology* 29.4 (Oct. 2016), pp. 657–696. DOI: 10.1007/s00145-015-9206-4 (cit. on p. 25).

[13] Elena Andreeva, Charles Bouillaguet, Orr Dunkelman, and John Kelsey. “Herding, Second Preimage and Trojan Message Attacks beyond Merkle-Damgård”. In: *SAC 2009: 16th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini. Vol. 5867. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2009, pp. 393–414. DOI: 10.1007/978-3-642-05445-7_25 (cit. on p. 25).

[14] Elena Andreeva, Joan Daemen, Bart Mennink, and Gilles Van Assche. "Security of Keyed Sponge Constructions Using a Modular Proof Approach". In: *Fast Software Encryption – FSE 2015*. Ed. by Gregor Leander. Vol. 9054. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2015, pp. 364–384. DOI: 10.1007/978-3-662-48116-5_18 (cit. on p. 108).

[15] Elena Andreeva, Atul Luykx, and Bart Mennink. "Provable Security of BLAKE with Non-ideal Compression Function". In: *SAC 2012: 19th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Lars R. Knudsen and Huapeng Wu. Vol. 7707. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2013, pp. 321–338. DOI: 10.1007/978-3-642-35999-6_21 (cit. on p. 73).

[16] Alina Andrushkevych, Yurii Gorbenko, Olexandr Kuznetsov, Roman Oliynykov, and Mariia Rodinko. "A Prospective Lightweight Block Cipher for Green IT Engineering". In: *Green IT Engineering: Social, Business and Industrial Applications*. Ed. by Vyacheslav Kharchenko, Yuriy Kondratenko, and Janusz Kacprzyk. Cham: Springer International Publishing, 2019, pp. 95–112. ISBN: 978-3-030-00253-4. URL: https://doi.org/10.1007/978-3-030-00253-4_5 (cit. on p. 128).

[17] Ralph Ankele and Stefan Kölbl. "Mind the Gap - A Closer Look at the Security of Block Ciphers against Differential Cryptanalysis". In: *SAC 2018: 25th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Carlos Cid and Michael J. Jacobson Jr: vol. 11349. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2019, pp. 163–190. DOI: 10.1007/978-3-030-10970-7_8 (cit. on p. 33).

[18] ANSI. *Control Network Protocol Specification*. ANSI/CEA-709.1-C. American National Standards Institute, Dec. 2010 (cit. on pp. 133, 144).

[19] ANSI. *Protocol Specification For Interfacing to Data Communication Networks*. ANSI C12.22-2008. American National Standards Institute, Jan. 2009 (cit. on p. 78).

[20] *Approval of OSGP as an ETSI Standard*. http://www.etsi.org/news-events/news/382-news-release-18-january-2012. 2012 (cit. on p. 131).

[21] Jörg Arndt. *Matters Computational: Ideas, Algorithms, Source Code*. 1st ed. New York, NY, USA: Springer-Verlag New York, Inc., 2010. ISBN: 9783642147630. URL: http://jjj.de/fxt/fxtpage.html#fxtbook (cit. on p. 103).

[22] Tomer Ashur and Yunwen Liu. “Rotational Cryptanalysis in the Presence of Constants”. In: *IACR Transactions on Symmetric Cryptology* 2016.1 (2016). https://tosc.iacr.org/index.php/ToSC/article/view/535, pp. 57–70. ISSN: 2519-173X. DOI: 10.13154/tosc.v2016.i1.57-70 (cit. on p. 36).

[23] Tomer Ashur and Vincent Rijmen. “On Linear Hulls and Trails”. In: *Progress in Cryptology - INDOCRYPT 2016: 17th International Conference in Cryptology in India*. Ed. by Orr Dunkelman and Somitra Kumar Sanadhya. Vol. 10095. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2016, pp. 269–286. DOI: 10.1007/978-3-319-49890-4_15 (cit. on p. 36).

[24] Jean-Philippe Aumasson, Itai Dinur, Willi Meier, and Adi Shamir. “Cube Testers and Key Recovery Attacks on Reduced-Round MD6 and Trivium”. In: *Fast Software Encryption – FSE 2009*. Ed. by Orr Dunkelman. Vol. 5665. Lecture Notes in Computer Science. Springer, Heidelberg, Feb. 2009, pp. 1–22. DOI: 10.1007/978-3-642-03317-9_1 (cit. on p. 36).

[25] Jean-Philippe Aumasson, Simon Fischer, Shahram Khazaei, Willi Meier, and Christian Rechberger. “New Features of Latin Dances: Analysis of Salsa, ChaCha, and Rumba”. In: *Fast Software Encryption – FSE 2008*. Ed. by Kaisa Nyberg. Vol. 5086. Lecture Notes in Computer Science. Springer, Heidelberg, Feb. 2008, pp. 470–488. DOI: 10.1007/978-3-540-71039-4_30 (cit. on pp. 50, 79, 98).

[26] Jean-Philippe Aumasson, Jian Guo, Simon Knellwolf, Krystian Matusiewicz, and Willi Meier. “Differential and Invertibility Properties of BLAKE”. In: *Fast Software Encryption – FSE 2010*. Ed. by Seokhie Hong and Tetsu Iwata. Vol. 6147. Lecture Notes in Computer Science. Springer, Heidelberg, Feb. 2010, pp. 318–332. DOI: 10.1007/978-3-642-13858-4_18 (cit. on pp. 50, 67, 73).

[27] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C.-W. Phan. *SHA-3 proposal BLAKE*. Submission to NIST (Round 1/2). 2008. URL: http://ehash.iaik.tugraz.at/uploads/0/06/Blake.pdf (cit. on p. 65).

[28] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C.-W. Phan. *SHA-3 proposal BLAKE*. Submission to NIST (Round 3). 2010. URL: http://131002.net/blake/blake.pdf (cit. on pp. 6, 44, 50, 66, 93).

[29] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. “Analysis of NORX: Investigating Differential and Rotational Properties”. In: *Progress in Cryptology - LATINCRYPT 2014: 3rd International Conference on Cryptology and Information Security in Latin America*. Ed. by Diego F. Aranha and Alfred Menezes. Vol. 8895. Lecture Notes in Computer Science. Springer, Heidelberg,

Sept. 2015, pp. 306–324. DOI: 10.1007/978-3-319-16295-9_17 (cit. on p. 7).

[30] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. "NORX: Parallel and Scalable AEAD". In: *ESORICS 2014: 19th European Symposium on Research in Computer Security, Part II*. Ed. by Miroslaw Kutylowski and Jaideep Vaidya. Vol. 8713. Lecture Notes in Computer Science. Springer, Heidelberg, Sept. 2014, pp. 19–36. DOI: 10.1007/978-3-319-11212-1_2 (cit. on p. 7).

[31] Jean-Philippe Aumasson, Willi Meier, and Raphael C.-W. Phan. "The Hash Function Family LAKE". In: *Fast Software Encryption – FSE 2008*. Ed. by Kaisa Nyberg. Vol. 5086. Lecture Notes in Computer Science. Springer, Heidelberg, Feb. 2008, pp. 36–53. DOI: 10.1007/978-3-540-71039-4_3 (cit. on p. 68).

[32] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. "BLAKE2: Simpler, Smaller, Fast as MD5". In: *ACNS 13: 11th International Conference on Applied Cryptography and Network Security*. Ed. by Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini. Vol. 7954. Lecture Notes in Computer Science. Springer, Heidelberg, June 2013, pp. 119–135. DOI: 10.1007/978-3-642-38980-1_8 (cit. on pp. 6, 50, 73, 79).

[33] Dongxia Bai, Hongbo Yu, Gaoli Wang, and Xiaoyun Wang. "Improved boomerang attacks on round-reduced SM3 and keyed permutation of BLAKE-256". In: *IET Inf. Secur.* 9.3 (2015), pp. 167–178. URL: https://doi.org/10.1049/iet-ifs.2013.0380 (cit. on p. 73).

[34] Thomas Baignères, Jacques Stern, and Serge Vaudenay. "Linear Cryptanalysis of Non Binary Ciphers". In: *SAC 2007: 14th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Carlisle M. Adams, Ali Miri, and Michael J. Wiener. Vol. 4876. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2007, pp. 184–211. DOI: 10.1007/978-3-540-77360-3_13 (cit. on p. 35).

[35] Thomas Baignères and Serge Vaudenay. "Proving the Security of AES Substitution-Permutation Network". In: *SAC 2005: 12th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Bart Preneel and Stafford Tavares. Vol. 3897. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2006, pp. 65–81. DOI: 10.1007/11693383_5 (cit. on p. 18).

[36] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. "GIFT: A Small Present - Towards Reaching the Limit of Lightweight Encryption". In: *Cryptographic Hardware and Embedded Systems – CHES 2017*. Ed. by Wieland Fischer and Naofumi Homma.

Vol. 10529. Lecture Notes in Computer Science. Springer, Heidelberg, Sept. 2017, pp. 321–345. DOI: 10.1007/978-3-319-66787-4_16 (cit. on p. 19).

[37] Gregory V. Bard. *The Vulnerability of SSL to Chosen Plaintext Attack*. Cryptology ePrint Archive, Report 2004/111. https://eprint.iacr.org/2004/111. 2004 (cit. on p. 20).

[38] Gregory V. Bard, Nicolas T. Courtois, and Chris Jefferson. *Efficient Methods for Conversion and Solution of Sparse Systems of Low-Degree Multivariate Polynomials over GF(2) via SAT-Solvers*. Cryptology ePrint Archive, Report 2007/024. https://eprint.iacr.org/2007/024. 2007 (cit. on p. 38).

[39] Magali Bardet, Jean-Charles Faugère, and Bruno Salvy. "On the complexity of the F5 Gröbner basis algorithm". In: *J. Symb. Comput.* 70 (2015), pp. 49–70. URL: https://doi.org/10.1016/j.jsc.2014.09.025 (cit. on p. 38).

[40] William Barker, William Polk, and Murugiah Souppaya. *Getting Ready for Post-Quantum Cryptography: Exploring Challenges Associated with Adopting and Using Post-Quantum Cryptographic Algorithms*. White Paper. National Institute of Standards and Technology, Apr. 2021. URL: https://doi.org/10.6028/NIST.CSWP.04282021 (cit. on p. 2).

[41] Clark Barrett, Aaron Stump, and Cesare Tinelli. "The SMT-LIB Standard: Version 2.0". In: *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK)*. Ed. by A. Gupta and D. Kroening. 2010 (cit. on p. 116).

[42] Michael Beeler, R. William Gosper, and Richard Schroeppel. *HAKMEM*. Artificial Intelligence Memo 239. Massachusetts Institute of Technology, Feb. 1972. URL: http://dspace.mit.edu/handle/1721.1/6086 (cit. on p. 79).

[43] Mihir Bellare. "Practice-Oriented Provable Security". In: *Lectures on Data Security, Modern Cryptology in Theory and Practice, Summer School, Aarhus, Denmark, July 1998*. Ed. by Ivan Damgård. Vol. 1561. Lecture Notes in Computer Science. Springer, 1998, pp. 1–15. ISBN: 3-540-65757-6. URL: http://dx.doi.org/10.1007/3-540-48969-X_1 (cit. on pp. 14, 27).

[44] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. "Keying Hash Functions for Message Authentication". In: *Advances in Cryptology – CRYPTO'96*. Ed. by Neal Koblitz. Vol. 1109. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 1996, pp. 1–15. DOI: 10.1007/3-540-68697-5_1 (cit. on p. 21).

[45] Mihir Bellare, Anand Desai, Eric Jokipii, and Phillip Rogaway. "A Concrete Security Treatment of Symmetric Encryption". In: *38th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society Press, Oct. 1997, pp. 394–403. DOI: 10.1109/SFCS.1997.646128 (cit. on p. 20).

[46] Mihir Bellare and Tadayoshi Kohno. "A Theoretical Treatment of Related-Key Attacks: RKA-PRPs, RKA-PRFs, and Applications". In: *Advances in Cryptology – EUROCRYPT 2003*. Ed. by Eli Biham. Vol. 2656. Lecture Notes in Computer Science. Springer, Heidelberg, May 2003, pp. 491–506. DOI: 10.1007/3-540-39200-9_31 (cit. on p. 27).

[47] Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. "Authenticated Encryption in SSH: Provably Fixing The SSH Binary Packet Protocol". In: *ACM CCS 2002: 9th Conference on Computer and Communications Security*. Ed. by Vijayalakshmi Atluri. ACM Press, Nov. 2002, pp. 1–11. DOI: 10.1145/586110.586112 (cit. on p. 22).

[48] Mihir Bellare and Chanathip Namprempre. "Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm". In: *Advances in Cryptology – ASIACRYPT 2000*. Ed. by Tatsuaki Okamoto. Vol. 1976. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2000, pp. 531–545. DOI: 10.1007/3-540-44448-3_41 (cit. on pp. 22, 77, 133).

[49] Mihir Bellare and Phillip Rogaway. "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols". In: *ACM CCS 93: 1st Conference on Computer and Communications Security*. Ed. by Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby. ACM Press, Nov. 1993, pp. 62–73. DOI: 10.1145/168588.168596 (cit. on p. 23).

[50] Mihir Bellare, Phillip Rogaway, and David Wagner. "The EAX Mode of Operation". In: *Fast Software Encryption – FSE 2004*. Ed. by Bimal K. Roy and Willi Meier. Vol. 3017. Lecture Notes in Computer Science. Springer, Heidelberg, Feb. 2004, pp. 389–407. DOI: 10.1007/978-3-540-25937-4_25 (cit. on pp. 22, 78).

[51] Steven M. Bellovin. "Frank Miller: Inventor of the One-Time Pad". In: *Cryptologia* 35.3 (2011), pp. 203–222. URL: http://dx.doi.org/10.1080/01611194.2011.583711 (cit. on p. 13).

[52] Ishai Ben-Aroya and Eli Biham. "Differtial Cryptanalysis of Lucifer". In: *Advances in Cryptology – CRYPTO'93*. Ed. by Douglas R. Stinson. Vol. 773. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 1994, pp. 187–199. DOI: 10.1007/3-540-48329-2_17 (cit. on p. 32).

[53] Daniel J. Bernstein. *Cache-Timing Attacks on AES*. http://cr.yp.to/antiforgery/cachetiming-20050414.pdf. 2005 (cit. on p. 97).

[54] Daniel J. Bernstein. "ChaCha, a Variant of Salsa20". In: *Workshop Record of SASC 2008: The State of the Art of Stream Ciphers*. 2008. URL: http://cr.yp.to/chacha.html (cit. on pp. 6, 39, 42, 72, 78, 79, 93, 98).

[55] Daniel J. Bernstein. *Cryptographic competitions — Disasters*. http://competitions.cr.yp.to/disasters.html. 2014 (cit. on p. 77).

[56] Daniel J. Bernstein. "How to Stretch Random Functions: The Security of Protected Counter Sums". In: *Journal of Cryptology* 12.3 (June 1999), pp. 185–192. DOI: 10.1007/s001459900051 (cit. on p. 21).

[57] Daniel J. Bernstein. *Speed, speed, speed*. Symmetric cryptography. Schloss Dagstuhl. Jan. 2020. URL: https://cr.yp.to/talks/2020.01.20/slides-djb-20200120-a4.pdf (cit. on p. 2).

[58] Daniel J. Bernstein. "The Poly1305-AES Message-Authentication Code". In: *Fast Software Encryption – FSE 2005*. Ed. by Henri Gilbert and Helena Handschuh. Vol. 3557. Lecture Notes in Computer Science. Springer, Heidelberg, Feb. 2005, pp. 32–49. DOI: 10.1007/11502760_3 (cit. on p. 21).

[59] Daniel J. Bernstein. "The Salsa20 Family of Stream Ciphers". In: *New Stream Cipher Designs*. Ed. by Matthew Robshaw and Olivier Billet. Vol. 4986. LNCS. Springer, 2008, pp. 84–97 (cit. on pp. 6, 72, 93).

[60] Daniel J. Bernstein. "Understanding brute force". In: *ECRYPT STVL Workshop on Symmetric Key Encryption*. Apr. 2005. URL: https://cr.yp.to/papers.html#bruteforce (cit. on p. 28).

[61] Daniel J. Bernstein, Stefan Kölbl, Stefan Lucks, Pedro Maat Costa Massolino, Florian Mendel, Kashif Nawaz, Tobias Schneider, Peter Schwabe, François-Xavier Standaert, Yosuke Todo, and Benoît Viguier. "Gimli : A Cross-Platform Permutation". In: *Cryptographic Hardware and Embedded Systems – CHES 2017*. Ed. by Wieland Fischer and Naofumi Homma. Vol. 10529. Lecture Notes in Computer Science. Springer, Heidelberg, Sept. 2017, pp. 299–320. DOI: 10.1007/978-3-319-66787-4_15 (cit. on p. 148).

[62] Daniel J. Bernstein and Tanja Lange, eds. *eBACS: ECRYPT Benchmarking of Cryptographic Systems*. accessed 1 November 2022. URL: http://bench.cr.yp.to (cit. on pp. 69, 70).

[63] Thomas A. Berson. "Differential Cryptanalysis Mod $2^{32}$ with Applications to MD5". In: *Advances in Cryptology – EUROCRYPT'92*. Ed. by Rainer A. Rueppel. Vol. 658. Lecture Notes in Computer Science. Springer, Heidelberg, May 1993, pp. 71–80. DOI: 10.1007/3-540-47555-9_6 (cit. on p. 32).

[64] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. *On the security of the keyed sponge construction*. Symmetric Key Encryption Workshop (SKEW). Feb. 2011 (cit. on p. 107).

[65] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. *Permutation-based encryption, authentication and authenticated encryption*. Directions in Authenticated Ciphers. July 2012 (cit. on pp. 26, 78, 84, 92, 98).

[66] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. *Sponge functions*. ECRYPT Hash Workshop 2007. May 2007 (cit. on pp. 25, 92, 101).

[67] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer. KECCAK *implementation overview*. http://keccak.noekeon.org/. May 2012 (cit. on p. 103).

[68] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "Keccak". In: *Advances in Cryptology – EUROCRYPT 2013*. Ed. by Thomas Johansson and Phong Q. Nguyen. Vol. 7881. Lecture Notes in Computer Science. Springer, Heidelberg, May 2013, pp. 313–314. DOI: 10.1007/978-3-642-38348-9_19 (cit. on p. 25).

[69] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "Sufficient conditions for sound tree and sequential hashing modes". In: *Int. J. Inf. Sec.* 13.4 (2014), pp. 335–353. URL: http://dx.doi.org/10.1007/s10207-013-0220-y (cit. on pp. 62, 74).

[70] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, Ronny Van Keer, and Benoît Viguier. "KangarooTwelve: Fast Hashing Based on Keccak-p". In: *ACNS 18: 16th International Conference on Applied Cryptography and Network Security*. Ed. by Bart Preneel and Frederik Vercauteren. Vol. 10892. Lecture Notes in Computer Science. Springer, Heidelberg, July 2018, pp. 400–418. DOI: 10.1007/978-3-319-93387-0_21 (cit. on p. 70).

[71] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications". In: *SAC 2011: 18th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Ali Miri and Serge Vaudenay. Vol. 7118. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2012, pp. 320–337. DOI: 10.1007/978-3-642-28496-0_19 (cit. on pp. 7, 26, 78, 79, 84, 87, 92, 98, 107).

[72] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "On the Indifferentiability of the Sponge Construction". In: *Advances in Cryptology – EUROCRYPT 2008*. Ed. by Nigel P. Smart. Vol. 4965. Lecture Notes in Computer Science. Springer, Heidelberg, Apr. 2008, pp. 181–197. DOI: 10.1007/978-3-540-78967-3_11 (cit. on pp. 26, 70, 107).

[73] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "Sponge-Based Pseudo-Random Number Generators". In: *Cryptographic Hardware and Embedded Systems – CHES 2010*. Ed. by Stefan Mangard and François-Xavier Standaert. Vol. 6225. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2010, pp. 33–47. DOI: 10.1007/978-3-642-15031-9_3 (cit. on p. 26).

[74] Eli Biham. "How to decrypt or even substitute DES-encrypted messages in $2^{28}$ steps". In: *Inf. Process. Lett.* 84.3 (2002), pp. 117–124. URL: http://dx.doi.org/10.1016/S0020-0190(02)00269-7 (cit. on p. 27).

[75] Eli Biham. *How to Forge DES-Encrypted Messages in $2^{28}$ Steps*. Tech. rep. 884. Haifa 32000, Israel: Technion - Israel Institute of Technology, 1996. URL: http://www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-get.cgi/1996/CS/CS0884.pdf (cit. on p. 27).

[76] Eli Biham. "New Types of Cryptanalytic Attacks Using Related Keys". In: *Journal of Cryptology* 7.4 (Dec. 1994), pp. 229–246. DOI: 10.1007/BF00203965 (cit. on p. 27).

[77] Eli Biham. "New Types of Cryptanalytic Attacks Using related Keys (Extended Abstract)". In: *Advances in Cryptology – EUROCRYPT'93*. Ed. by Tor Helleseth. Vol. 765. Lecture Notes in Computer Science. Springer, Heidelberg, May 1994, pp. 398–409. DOI: 10.1007/3-540-48285-7_34 (cit. on p. 27).

[78] Eli Biham, Alex Biryukov, and Adi Shamir. "Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials". In: *Advances in Cryptology – EUROCRYPT'99*. Ed. by Jacques Stern. Vol. 1592. Lecture Notes in Computer Science. Springer, Heidelberg, May 1999, pp. 12–23. DOI: 10.1007/3-540-48910-X_2 (cit. on p. 36).

[79] Eli Biham, Alex Biryukov, and Adi Shamir. "Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials". In: *Journal of Cryptology* 18.4 (Sept. 2005), pp. 291–311. DOI: 10.1007/s00145-005-0129-3 (cit. on p. 36).

[80] Eli Biham, Alex Biryukov, and Adi Shamir. "Miss in the Middle Attacks on IDEA and Khufu". In: *Fast Software Encryption – FSE'99*. Ed. by Lars R. Knudsen. Vol. 1636. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 1999, pp. 124–138. DOI: 10.1007/3-540-48519-8_10 (cit. on p. 36).

[81] Eli Biham and Orr Dunkelman. *A Framework for Iterative Hash Functions - HAIFA*. Cryptology ePrint Archive, Report 2007/278. https://eprint.iacr.org/2007/278. 2007 (cit. on pp. 25, 71).

[82] Eli Biham, Orr Dunkelman, and Nathan Keller. "Differential-Linear Cryptanalysis of Serpent". In: *Fast Software Encryption – FSE 2003*. Ed. by Thomas Johansson. Vol. 2887. Lecture Notes in Computer Science. Springer, Heidelberg, Feb. 2003, pp. 9–21. DOI: 10.1007/978-3-540-39887-5_2 (cit. on p. 36).

[83] Eli Biham, Orr Dunkelman, and Nathan Keller. "Enhancing Differential-Linear Cryptanalysis". In: *Advances in Cryptology – ASIACRYPT 2002*. Ed. by Yuliang Zheng. Vol. 2501. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2002, pp. 254–266. DOI: 10.1007/3-540-36178-2_16 (cit. on p. 36).

[84] Eli Biham, Orr Dunkelman, and Nathan Keller. "The Rectangle Attack - Rectangling the Serpent". In: *Advances in Cryptology – EUROCRYPT 2001*. Ed. by Birgit Pfitzmann. Vol. 2045. Lecture Notes in Computer Science. Springer, Heidelberg, May 2001, pp. 340–357. DOI: 10.1007/3-540-44987-6_21 (cit. on p. 36).

[85] Eli Biham and Adi Shamir. "Differential Cryptanalysis of DES-like Cryptosystems". In: *Journal of Cryptology* 4.1 (Jan. 1991), pp. 3–72. DOI: 10.1007/BF00630563 (cit. on p. 32).

[86] Eli Biham and Adi Shamir. "Differential Cryptanalysis of DES-like Cryptosystems". In: *Advances in Cryptology – CRYPTO'90*. Ed. by Alfred J. Menezes and Scott A. Vanstone. Vol. 537. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 1991, pp. 2–21. DOI: 10.1007/3-540-38424-3_1 (cit. on p. 32).

[87] Eli Biham and Adi Shamir. "Differential Cryptanalysis of Snefru, Khafre, REDOC-II, LOKI and Lucifer". In: *Advances in Cryptology – CRYPTO'91*. Ed. by Joan Feigenbaum. Vol. 576. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 1992, pp. 156–171. DOI: 10.1007/3-540-46766-1_11 (cit. on p. 32).

[88] Eli Biham and Adi Shamir. "Differential Cryptanalysis of the Full 16-Round DES". In: *Advances in Cryptology – CRYPTO'92*. Ed. by Ernest F. Brickell. Vol. 740. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 1993, pp. 487–496. DOI: 10.1007/3-540-48071-4_34 (cit. on p. 32).

[89] Alex Biryukov. "The Design of a Stream Cipher LEX". In: *SAC 2006: 13th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Eli Biham and Amr M. Youssef. Vol. 4356. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2007, pp. 67–75. DOI: 10.1007/978-3-540-74462-7_6 (cit. on p. 43).

[90] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. "Argon2: New Generation of Memory-Hard Functions for Password Hashing and Other Applications". In: *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*. IEEE, 2016, pp. 292–302. ISBN: 978-1-5090-1751-5. URL: http://dx.doi.org/10.1109/EuroSP.2016.31 (cit. on p. 147).

[91] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. *Fast and Tradeoff-Resilient Memory-Hard Functions for Cryptocurrencies and Password Hashing*. Cryptology ePrint Archive, Report 2015/430. https://eprint.iacr.org/2015/430. 2015 (cit. on p. 147).

[92] Alex Biryukov and Dmitry Khovratovich. "PAEQ: Parallelizable Permutation-Based Authenticated Encryption". In: *ISC 2014: 17th International Conference on Information Security*. Ed. by Sherman S. M. Chow, Jan Camenisch, Lucas Chi Kwong Hui, and Siu-Ming Yiu. Vol. 8783. Lecture Notes in Computer Science. Springer, Heidelberg, Oct. 2014, pp. 72–89. DOI: 10.1007/978-3-319-13257-0_5 (cit. on p. 84).

[93] Alex Biryukov and Dmitry Khovratovich. "PPAE: Parallelizable Permutation-based Authenticated Encryption". Presented at DIAC 2013, 11–13 August 2013, Chicago, USA, http://2013.diac.cr.yp.to/slides/khovratovich.pdf. 2013 (cit. on p. 84).

[94] Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolic. "Distinguisher and Related-Key Attack on the Full AES-256". In: *Advances in Cryptology – CRYPTO 2009*. Ed. by Shai Halevi. Vol. 5677. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2009, pp. 231–249. DOI: 10.1007/978-3-642-03356-8_14 (cit. on p. 27).

[95] Alex Biryukov, Ivica Nikolic, and Arnab Roy. "Boomerang Attacks on BLAKE-32". In: *Fast Software Encryption – FSE 2011*. Ed. by Antoine Joux. Vol. 6733. Lecture Notes in Computer Science. Springer, Heidelberg, Feb. 2011, pp. 218–237. DOI: 10.1007/978-3-642-21702-9_13 (cit. on pp. 72, 73).

[96] Alex Biryukov, Aleksei Udovenko, and Vesselin Velichkov. *Analysis of the NORX Core Permutation*. Cryptology ePrint Archive, Report 2017/034. https://eprint.iacr.org/2017/034. 2017 (cit. on p. 109).

[97] John Black, Shai Halevi, Hugo Krawczyk, Ted Krovetz, and Phillip Rogaway. "UMAC: Fast and Secure Message Authentication". In: *Advances in Cryptology – CRYPTO'99*. Ed. by Michael J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 1999, pp. 216–233. DOI: 10.1007/3-540-48405-1_14 (cit. on p. 21).

[98] John Black and Phillip Rogaway. "A Block-Cipher Mode of Operation for Parallelizable Message Authentication". In: *Advances in Cryptology – EUROCRYPT 2002*. Ed. by Lars R. Knudsen. Vol. 2332. Lecture Notes in Computer Science. Springer, Heidelberg, Apr. 2002, pp. 384–397. DOI: 10.1007/3-540-46035-7_25 (cit. on p. 21).

[99] Céline Blondeau and Benoît Gérard. "Multiple Differential Cryptanalysis: Theory and Practice". In: *Fast Software Encryption – FSE 2011*. Ed. by Antoine Joux. Vol. 6733. Lecture Notes in Computer Science. Springer, Heidelberg, Feb. 2011, pp. 35–54. DOI: 10.1007/978-3-642-21702-9_3 (cit. on p. 36).

[100] Céline Blondeau, Benoît Gérard, and Kaisa Nyberg. "Multiple Differential Cryptanalysis Using LLR and $\chi^2$ Statistics". In: *SCN 12: 8th International Conference on Security in Communication Networks*. Ed. by Ivan Visconti and Roberto De Prisco. Vol. 7485. Lecture Notes in Computer Science. Springer, Heidelberg, Sept. 2012, pp. 343–360. DOI: 10.1007/978-3-642-32928-9_19 (cit. on p. 36).

[101] Céline Blondeau, Benoît Gérard, and Jean-Pierre Tillich. "Accurate estimates of the data complexity and success probability for various cryptanalyses". In: *Des. Codes Cryptogr.* 59.1-3 (2011), pp. 3–34. URL: https://doi.org/10.1007/s10623-010-9452-2 (cit. on p. 32).

[102] Céline Blondeau, Gregor Leander, and Kaisa Nyberg. "Differential-Linear Cryptanalysis Revisited". In: *Fast Software Encryption – FSE 2014*. Ed. by Carlos Cid and Christian Rechberger. Vol. 8540. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2015, pp. 411–430. DOI: 10.1007/978-3-662-46706-0_21 (cit. on p. 36).

[103] Lenore Blum, Manuel Blum, and Mike Shub. "A Simple Unpredictable Pseudo-Random Number Generator". In: *SIAM Journal on Computing* 15.2 (May 1986), pp. 364–383 (cit. on p. 40).

[104] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. "Biclique Cryptanalysis of the Full AES". In: *Advances in Cryptology – ASIACRYPT 2011*. Ed. by Dong Hoon Lee and Xiaoyun Wang. Vol. 7073. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2011, pp. 344–371. DOI: 10.1007/978-3-642-25385-0_19 (cit. on p. 28).

[105] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. "PRESENT: An Ultra-Lightweight Block Cipher". In: *Cryptographic Hardware and Embedded Systems – CHES 2007*. Ed. by Pascal Paillier and Ingrid Verbauwhede. Vol. 4727. Lecture Notes in Computer Science. Springer, Heidelberg, Sept. 2007, pp. 450–466. DOI: 10.1007/978-3-540-74735-2_31 (cit. on p. 19).

[106] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, François-Xavier Standaert, John P. Steinberger, and Elmar Tischhauser. "Key-Alternating Ciphers in a Provable Setting: Encryption Using a Small Number of Public Permutations - (Extended Abstract)". In: *Advances in Cryptology – EUROCRYPT 2012*.

Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. Lecture Notes in Computer Science. Springer, Heidelberg, Apr. 2012, pp. 45–62. DOI: 10.1007/978-3-642-29011-4_5 (cit. on p. 19).

[107] Andrey Bogdanov and Vincent Rijmen. "Linear hulls with correlation zero and linear cryptanalysis of block ciphers". In: *Des. Codes Cryptogr.* 70.3 (2014), pp. 369–383. URL: https://doi.org/10.1007/s10623-012-9697-z (cit. on p. 37).

[108] Nikita Borisov, Monica Chew, Robert Johnson, and David Wagner. "Multiplicative Differentials". In: *Fast Software Encryption – FSE 2002*. Ed. by Joan Daemen and Vincent Rijmen. Vol. 2365. Lecture Notes in Computer Science. Springer, Heidelberg, Feb. 2002, pp. 17–33. DOI: 10.1007/3-540-45661-9_2 (cit. on p. 32).

[109] Charles Bouillaguet, Hsieh-Chung Chen, Chen-Mou Cheng, Tung Chou, Ruben Niederhagen, Adi Shamir, and Bo-Yin Yang. "Fast Exhaustive Search for Polynomial Systems in $\mathbb{F}_2$". In: *Cryptographic Hardware and Embedded Systems – CHES 2010*. Ed. by Stefan Mangard and François-Xavier Standaert. Vol. 6225. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2010, pp. 203–218. DOI: 10.1007/978-3-642-15031-9_14 (cit. on p. 37).

[110] Charles Bouillaguet, Chen-Mou Cheng, Tung Chou, Ruben Niederhagen, and Bo-Yin Yang. "Fast Exhaustive Search for Quadratic Systems in $\mathbb{F}_2$ on FPGAs". In: *SAC 2013: 20th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Tanja Lange, Kristin Lauter, and Petr Lisonek. Vol. 8282. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2014, pp. 205–222. DOI: 10.1007/978-3-662-43414-7_11 (cit. on p. 37).

[111] Richard Brent. "Uniform random number generators for supercomputers". In: *Proc. Fifth Australian Supercomputer Conference*. Melbourne, Dec. 1992, pp. 95–104 (cit. on p. 40).

[112] Robert Brummayer and Armin Biere. "Boolector: An Efficient SMT Solver for Bit-Vectors and Arrays". In: *Tools and Algorithms for the Construction and Analysis of Systems, 15th International Conference, TACAS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*. Ed. by Stefan Kowalewski and Anna Philippou. Vol. 5505. Lecture Notes in Computer Science. Springer, 2009, pp. 174–177. ISBN: 978-3-642-00767-5. URL: https://doi.org/10.1007/978-3-642-00768-2%5C_16 (cit. on pp. 109, 116).

[113] Arthur Walter Burks, Herman Heine Goldstine, and John von Neumann. *Preliminary discussion of the logical design of an electronic computer instrument*. Tech. rep. June 1946. URL: https://www.ias.edu/library/ecp (cit. on p. 79).

[114] *CAESAR – Competition for Authenticated Encryption: Security, Applicability, and Robustness*. http://competitions.cr.yp.to/caesar.html. 2014 (cit. on pp. 7, 78, 92).

[115] Ran Canetti, Oded Goldreich, and Shai Halevi. "The Random Oracle Methodology, Revisited (Preliminary Version)". In: *30th Annual ACM Symposium on Theory of Computing*. ACM Press, May 1998, pp. 209–218. DOI: 10.1145/276698.276741 (cit. on p. 23).

[116] Anne Canteaut, Eran Lambooij, Samuel Neves, Shahram Rasoolzadeh, Yu Sasaki, and Marc Stevens. "Refined Probability of Differential Characteristics Including Dependency Between Multiple Rounds". In: *IACR Transactions on Symmetric Cryptology* 2017.2 (2017), pp. 203–227. ISSN: 2519-173X. DOI: 10.13154/tosc.v2017.i2.203-227 (cit. on p. 129).

[117] Brice Canvel, Alain P. Hiltgen, Serge Vaudenay, and Martin Vuagnoux. "Password Interception in a SSL/TLS Channel". In: *Advances in Cryptology – CRYPTO 2003*. Ed. by Dan Boneh. Vol. 2729. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2003, pp. 583–599. DOI: 10.1007/978-3-540-45146-4_34 (cit. on pp. 22, 77).

[118] Claude Carlet. "Handling Vectorial Functions by Means of Their Graph Indicators". In: *IEEE Trans. Inf. Theory* 66.10 (2020), pp. 6324–6339. URL: https://doi.org/10.1109/TIT.2020.2981524 (cit. on p. 35).

[119] Florent Chabaud and Serge Vaudenay. "Links Between Differential and Linear Cryptanalysis". In: *Advances in Cryptology – EUROCRYPT'94*. Ed. by Alfredo De Santis. Vol. 950. Lecture Notes in Computer Science. Springer, Heidelberg, May 1995, pp. 356–365. DOI: 10.1007/BFb0053450 (cit. on p. 35).

[120] Colin Chaigneau, Thomas Fuhr, Henri Gilbert, Jérémy Jean, and Jean-René Reinhard. "Cryptanalysis of NORX v2.0". In: *IACR Transactions on Symmetric Cryptology* 2017.1 (2017), pp. 156–174. ISSN: 2519-173X. DOI: 10.13154/tosc.v2017.i1.156-174 (cit. on pp. 109, 110).

[121] William G. Chambers. "On Random Mappings and Random Permutations". In: *Fast Software Encryption – FSE'94*. Ed. by Bart Preneel. Vol. 1008. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 1995, pp. 22–28. DOI: 10.1007/3-540-60590-8_3 (cit. on p. 44).

[122] Donghoon Chang, Arpan Jati, Sweta Mishra, and Somitra Kumar Sanadhya. "Rig: A Simple, Secure and Flexible Design for Password Hashing". In: *Information Security and Cryptology - 10th International Conference, Inscrypt 2014, Beijing, China, December 13-15, 2014, Revised Selected Papers*. Ed. by Dongdai Lin, Moti Yung, and Jianying Zhou. Vol. 8957. Lecture Notes in Computer Science. Springer, 2014, pp. 361–381. ISBN: 978-3-319-16744-2.

URL: http://dx.doi.org/10.1007/978-3-319-16745-9_20 (cit. on p. 147).

[123] Donghoon Chang and Mridul Nandi. "Improved Indifferentiability Security Analysis of chopMD Hash Function". In: *Fast Software Encryption – FSE 2008*. Ed. by Kaisa Nyberg. Vol. 5086. Lecture Notes in Computer Science. Springer, Heidelberg, Feb. 2008, pp. 429–443. DOI: 10.1007/978-3-540-71039-4_27 (cit. on p. 25).

[124] Donghoon Chang, Mridul Nandi, and Moti Yung. *Indifferentiability of the Hash Algorithm BLAKE*. Cryptology ePrint Archive, Report 2011/623. https://eprint.iacr.org/2011/623. 2011 (cit. on p. 73).

[125] Shu-jen Chang, Ray Perlner, William E. Burr, Meltem Sönmez Turan, John M. Kelsey, Souradyuti Paul, and Lawrence E. Bassham. *Third-Round Report of the SHA-3 Cryptographic Hash Algorithm Competition*. NISTIR 7896. National Institute for Standards and Technology, Nov. 2012 (cit. on p. 51).

[126] Shan Chen, Rodolphe Lampe, Jooyoung Lee, Yannick Seurin, and John P. Steinberger. "Minimizing the Two-Round Even-Mansour Cipher". In: *Advances in Cryptology – CRYPTO 2014, Part I*. Ed. by Juan A. Garay and Rosario Gennaro. Vol. 8616. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2014, pp. 39–56. DOI: 10.1007/978-3-662-44371-2_3 (cit. on p. 19).

[127] Shan Chen and John P. Steinberger. "Tight Security Bounds for Key-Alternating Ciphers". In: *Advances in Cryptology – EUROCRYPT 2014*. Ed. by Phong Q. Nguyen and Elisabeth Oswald. Vol. 8441. Lecture Notes in Computer Science. Springer, Heidelberg, May 2014, pp. 327–350. DOI: 10.1007/978-3-642-55220-5_19 (cit. on p. 19).

[128] Chen-Mou Cheng, Tung Chou, Ruben Niederhagen, and Bo-Yin Yang. "Solving Quadratic Equations with XL on Parallel Architectures". In: *Cryptographic Hardware and Embedded Systems – CHES 2012*. Ed. by Emmanuel Prouff and Patrick Schaumont. Vol. 7428. Lecture Notes in Computer Science. Springer, Heidelberg, Sept. 2012, pp. 356–373. DOI: 10.1007/978-3-642-33027-8_21 (cit. on p. 37).

[129] Timothy H. Click, Aibing Liu, and George A. Kaminski. "Quality of random number generators significantly affects results of Monte Carlo simulations for organic and biological systems". In: *J. Comput. Chem.* 32.3 (2011), pp. 513–524. URL: https://doi.org/10.1002/jcc.21638 (cit. on p. 15).

[130] Benoît Cogliati, Yevgeniy Dodis, Jonathan Katz, Jooyoung Lee, John P. Steinberger, Aishwarya Thiruvengadam, and Zhe Zhang. "Provable Security of (Tweakable) Block Ciphers Based on Substitution-Permutation Networks". In: *Advances in Cryptology – CRYPTO 2018, Part I*. Ed. by Hovav Shacham

and Alexandra Boldyreva. Vol. 10991. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2018, pp. 722–753. DOI: 10.1007/978-3-319-96884-1_24 (cit. on p. 18).

[131] Benoît Cogliati and Yannick Seurin. "Strengthening the Known-Key Security Notion for Block Ciphers". In: *Fast Software Encryption – FSE 2016*. Ed. by Thomas Peyrin. Vol. 9783. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2016, pp. 494–513. DOI: 10.1007/978-3-662-52993-5_25 (cit. on p. 27).

[132] Don Coppersmith. "The Data Encryption Standard (DES) and its strength against attacks". In: *IBM Journal of Research and Development* 38.3 (1994), pp. 243–250. URL: http://dx.doi.org/10.1147/rd.383.0243 (cit. on p. 32).

[133] Don Coppersmith, Chris L. Holloway, Stephen M. Matyas, and Nevenko Zunic. "The data encryption standard". In: *Inf. Sec. Techn. Report* 2.2 (1997), pp. 22–24. URL: http://dx.doi.org/10.1016/S1363-4127(97)81325-8 (cit. on p. 32).

[134] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. "Merkle-Damgård Revisited: How to Construct a Hash Function". In: *Advances in Cryptology – CRYPTO 2005*. Ed. by Victor Shoup. Vol. 3621. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2005, pp. 430–448. DOI: 10.1007/11535218_26 (cit. on p. 24).

[135] Nicolas Courtois, Gregory V. Bard, and David Wagner. "Algebraic and Slide Attacks on KeeLoq". In: *Fast Software Encryption – FSE 2008*. Ed. by Kaisa Nyberg. Vol. 5086. Lecture Notes in Computer Science. Springer, Heidelberg, Feb. 2008, pp. 97–115. DOI: 10.1007/978-3-540-71039-4_6 (cit. on p. 38).

[136] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. "Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations". In: *Advances in Cryptology – EUROCRYPT 2000*. Ed. by Bart Preneel. Vol. 1807. Lecture Notes in Computer Science. Springer, Heidelberg, May 2000, pp. 392–407. DOI: 10.1007/3-540-45539-6_27 (cit. on p. 37).

[137] Nicolas Courtois, Sean O'Neil, and Jean-Jacques Quisquater. "Practical Algebraic Attacks on the Hitag2 Stream Cipher". In: *ISC 2009: 12th International Conference on Information Security*. Ed. by Pierangela Samarati, Moti Yung, Fabio Martinelli, and Claudio Agostino Ardagna. Vol. 5735. Lecture Notes in Computer Science. Springer, Heidelberg, Sept. 2009, pp. 167–176 (cit. on p. 38).

[138] Nicolas Courtois and Josef Pieprzyk. "Cryptanalysis of Block Ciphers with Overdefined Systems of Equations". In: *Advances in Cryptology – ASIACRYPT 2002*. Ed. by Yuliang Zheng. Vol. 2501. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2002, pp. 267–287. DOI: 10.1007/3-540-36178-2_17 (cit. on p. 37).

[139] Nicolas T. Courtois, Jörg Drobick, and Klaus Schmeh. "Feistel ciphers in East Germany in the communist era". In: *Cryptologia* 42.5 (2018), pp. 427–444. URL: https://doi.org/10.1080/01611194.2018.1428835 (cit. on pp. 16, 34).

[140] Nicolas T. Courtois, Maria-Bristena Oprisanu, and Klaus Schmeh. "Linear cryptanalysis and block cipher design in East Germany in the 1970s". In: *Cryptologia* 43.1 (2019), pp. 2–22. URL: https://doi.org/10.1080/01611194.2018.1483981 (cit. on p. 34).

[141] Paul Crowley and Eric Biggers. "Adiantum: length-preserving encryption for entry-level processors". In: *IACR Transactions on Symmetric Cryptology* 2018.4 (2018), pp. 39–61. ISSN: 2519-173X. DOI: 10.13154/tosc.v2018.i4.39-61 (cit. on p. 2).

[142] Paul Crowley and Eric Biggers. *Introducing Adiantum: Encryption for the Next Billion Users*. Feb. 2019. URL: https://security.googleblog.com/2019/02/introducing-adiantum-encryption-for.html (cit. on p. 2).

[143] *CUDA Toolkit 4.0 CURAND Guide*. NVIDIA. Jan. 2011. URL: http://developer.download.nvidia.com/compute/cuda/4_0_rc2/toolkit/docs/CURAND_Library.pdf (cit. on pp. 48, 49).

[144] Joan Daemen. "Limitations of the Even-Mansour Construction (Rump Session)". In: *Advances in Cryptology – ASIACRYPT'91*. Ed. by Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto. Vol. 739. Lecture Notes in Computer Science. Springer, Heidelberg, Nov. 1993, pp. 495–498. DOI: 10.1007/3-540-57332-1_46 (cit. on p. 19).

[145] Joan Daemen, René Govaerts, and Joos Vandewalle. "Correlation Matrices". In: *Fast Software Encryption – FSE'94*. Ed. by Bart Preneel. Vol. 1008. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 1995, pp. 275–285. DOI: 10.1007/3-540-60590-8_21 (cit. on p. 35).

[146] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. "The Block Cipher Square". In: *Fast Software Encryption – FSE'97*. Ed. by Eli Biham. Vol. 1267. Lecture Notes in Computer Science. Springer, Heidelberg, Jan. 1997, pp. 149–165. DOI: 10.1007/BFb0052343 (cit. on p. 36).

[147] Joan Daemen, Bart Mennink, and Gilles Van Assche. "Full-State Keyed Duplex with Built-In Multi-user Support". In: *Advances in Cryptology – ASIACRYPT 2017, Part II*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10625. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2017, pp. 606–637. DOI: 10.1007/978-3-319-70697-9_21 (cit. on p. 108).

[148] Joan Daemen, Bart Mennink, and Gilles Van Assche. "Sound Hashing Modes of Arbitrary Functions, Permutations, and Block Ciphers". In: *IACR Transactions on Symmetric Cryptology* 2018.4 (2018), pp. 197–228. ISSN: 2519-173X. DOI: 10.13154/tosc.v2018.i4.197-228 (cit. on pp. 62, 74).

[149] Joan Daemen and Vincent Rijmen. "Probability distributions of correlation and differentials in block ciphers". In: *J. Math. Cryptol.* 1.3 (2007), pp. 221–242. URL: https://doi.org/10.1515/JMC.2007.011 (cit. on pp. 31, 32).

[150] Joan Daemen and Vincent Rijmen. *The Advanced Encryption Standard*. 2001. URL: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf (cit. on p. 18).

[151] Joan Daemen and Vincent Rijmen. "Understanding Two-Round Differentials in AES". In: *SCN 06: 5th International Conference on Security in Communication Networks*. Ed. by Roberto De Prisco and Moti Yung. Vol. 4116. Lecture Notes in Computer Science. Springer, Heidelberg, Sept. 2006, pp. 78–94. DOI: 10.1007/11832072_6 (cit. on p. 31).

[152] Joan Daemen, Luc van Linden, René Govaerts, and Joos Vandewalle. "Propagation Properties of Multiplication Modulo $2^n - 1$". In: *Thirteenth Symposium on Information Theory in the Benelux*. Ed. by G. Heideman, F. Hoeksema, and H. Tattje. 1992, pp. 111–118. URL: https://www.esat.kuleuven.be/cosic/publications/article-136.pdf (cit. on p. 32).

[153] Ivan Damgård. "A Design Principle for Hash Functions". In: *Advances in Cryptology – CRYPTO'89*. Ed. by Gilles Brassard. Vol. 435. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 1990, pp. 416–427. DOI: 10.1007/0-387-34805-0_39 (cit. on p. 25).

[154] Andrew Danowitz, Kyle Kelley, James Mao, John P. Stevenson, and Mark Horowitz. "CPU DB: recording microprocessor history". In: *Commun. ACM* 55.4 (2012), pp. 55–63. URL: https://doi.org/10.1145/2133806.2133822 (cit. on p. 3).

[155] Sourav Das, Subhamoy Maitra, and Willi Meier. *Higher Order Differential Analysis of NORX*. Cryptology ePrint Archive, Report 2015/186. https://eprint.iacr.org/2015/186. 2015 (cit. on pp. 124, 125).

[156] Nilanjan Datta and Kan Yasuda. “Generalizing PMAC Under Weaker Assumptions”. In: *ACISP 15: 20th Australasian Conference on Information Security and Privacy*. Ed. by Ernest Foo and Douglas Stebila. Vol. 9144. Lecture Notes in Computer Science. Springer, Heidelberg, June 2015, pp. 433–450. DOI: 10.1007/978-3-319-19962-7_25 (cit. on p. 21).

[157] Magnus Daum. “Cryptanalysis of Hash functions of the MD4-family”. PhD thesis. Ruhr University Bochum, 2005. URL: http://www-brs.ub.ruhr-uni-bochum.de/netahtml/HSS/Diss/DaumMagnus/ (cit. on p. 143).

[158] Christophe De Cannière. “Trivium: A Stream Cipher Construction Inspired by Block Cipher Design Principles”. In: *ISC 2006: 9th International Conference on Information Security*. Ed. by Sokratis K. Katsikas, Javier Lopez, Michael Backes, Stefanos Gritzalis, and Bart Preneel. Vol. 4176. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2006, pp. 171–186 (cit. on p. 15).

[159] Leonardo De Moura and Nikolaj Bjørner. “Z3: An Efficient SMT Solver”. In: *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. TACAS’08/ETAPS’08. Budapest, Hungary: Springer-Verlag, 2008, pp. 337–340. ISBN: 978-3-540-78799-0. URL: http://dl.acm.org/citation.cfm?id=1792734.1792766 (cit. on p. 109).

[160] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc. “Design of ion-implanted MOSFET’s with very small physical dimensions”. In: *IEEE Journal of Solid-State Circuits* 9.5 (Oct. 1974), pp. 256–268. ISSN: 0018-9200. DOI: 10.1109/JSSC.1974.1050511 (cit. on p. 2).

[161] R. H. Dennard, F. H. Gaensslen, Hwa-Nien Yu, V. L. Rideout, E. Bassous, and A. R. Leblanc. “Design Of Ion-implanted MOSFET’s with Very Small Physical Dimensions”. In: *Proceedings of the IEEE* 87.4 (Apr. 1999), pp. 668–678. ISSN: 0018-9219. DOI: 10.1109/JPROC.1999.752522 (cit. on p. 2).

[162] *DES Modes of Operation*. Federal Information Processing Standard (FIPS) 81. US National Bureau of Standards, Dec. 1980. URL: http://csrc.nist.gov/publications/fips/fips81/fips81.htm (cit. on p. 20).

[163] *Data Encryption Standard*. National Bureau of Standards, NBS FIPS PUB 46, U.S. Department of Commerce. Jan. 1977 (cit. on pp. 1, 5, 16).

[164] Whitfield Diffie and Martin E. Hellman. “New Directions in Cryptography”. In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654 (cit. on p. 12).

[165] Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener. "Authentication and Authenticated Key Exchanges". In: *Designs, Codes and Cryptography* 2.2 (June 1992), pp. 107–125 (cit. on p. 12).

[166] Jintai Ding and Bo-Yin Yang. "Degree of Regularity for HFEv and HFEv-". In: *Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013*. Ed. by Philippe Gaborit. Springer, Heidelberg, June 2013, pp. 52–66. DOI: 10.1007/978-3-642-38616-9_4 (cit. on p. 38).

[167] Itai Dinur and Adi Shamir. "Cube Attacks on Tweakable Black Box Polynomials". In: *Advances in Cryptology – EUROCRYPT 2009*. Ed. by Antoine Joux. Vol. 5479. Lecture Notes in Computer Science. Springer, Heidelberg, Apr. 2009, pp. 278–299. DOI: 10.1007/978-3-642-01001-9_16 (cit. on p. 36).

[168] Yevgeniy Dodis, Harish Karthikeyan, and Daniel Wichs. "Small-Box Cryptography". In: *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*. Ed. by Mark Braverman. Vol. 215. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 56:1–56:25. ISBN: 978-3-95977-217-4. URL: https://doi.org/10.4230/LIPIcs.ITCS.2022.56 (cit. on p. 18).

[169] Yevgeniy Dodis, Leonid Reyzin, Ronald L. Rivest, and Emily Shen. "Indifferentiability of Permutation-Based Compression Functions and Tree-Based Modes of Operation, with Applications to MD6". In: *Fast Software Encryption – FSE 2009*. Ed. by Orr Dunkelman. Vol. 5665. Lecture Notes in Computer Science. Springer, Heidelberg, Feb. 2009, pp. 104–121. DOI: 10.1007/978-3-642-03317-9_7 (cit. on pp. 62, 74).

[170] Jason A. Donenfeld. "WireGuard: Next Generation Kernel Network Tunnel". In: *ISOC Network and Distributed System Security Symposium – NDSS 2017*. The Internet Society, Feb. 2017 (cit. on p. 147).

[171] Vivien Dubois and Nicolas Gama. "The Degree of Regularity of HFE Systems". In: *Advances in Cryptology – ASIACRYPT 2010*. Ed. by Masayuki Abe. Vol. 6477. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2010, pp. 557–576. DOI: 10.1007/978-3-642-17373-8_32 (cit. on p. 38).

[172] Orr Dunkelman, Nathan Keller, and Adi Shamir. "Minimalism in Cryptography: The Even-Mansour Scheme Revisited". In: *Advances in Cryptology – EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. Lecture Notes in Computer Science. Springer, Heidelberg, Apr. 2012, pp. 336–354. DOI: 10.1007/978-3-642-29011-4_21 (cit. on p. 19).

[173] Orr Dunkelman, Nathan Keller, and Adi Shamir. "Slidex Attacks on the Even-Mansour Encryption Scheme". In: *Journal of Cryptology* 28.1 (Jan. 2015), pp. 1–28. DOI: 10.1007/s00145-013-9164-7 (cit. on p. 19).

[174] Orr Dunkelman and Dmitry Khovratovich. “Iterative differentials, symmetries, and message modification in BLAKE-256”. In: *ECRYPT2 Hash Workshop*. 2011 (cit. on pp. 72, 73).

[175] Thai Duong and Juliano Rizzo. *Flickr’s API Signature Forgery Vulnerability*. http://netifera.com/research/. Sept. 2009 (cit. on p. 51).

[176] Thai Duong and Juliano Rizzo. *Here Come The ⊕ Ninjas*. Unpublished. May 2011 (cit. on pp. 20, 77).

[177] Morris J. Dworkin. *SP 800-38D. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. Tech. rep. Gaithersburg, MD, United States: National Institute of Standards & Technology, 2007 (cit. on p. 79).

[178] Echelon Corporation. *90 Million Energy-Aware LonWorks Devices Worldwide*. http://www.businesswire.com/news/home/20100412005544/en/90-Million-Energy-Aware-LonWorks-Devices-Worldwide. 2010 (cit. on p. 145).

[179] Echelon Corporation. *LonTalk Protocol Specification*. Version 3.0. 1994 (cit. on pp. 133, 144).

[180] W. F. Eddy. “Random Number Generators for Parallel Processors”. In: *Journal of Computational and Applied Mathematics* 31 (1990), pp. 63–71 (cit. on p. 40).

[181] Jürgen Eichenauer and Jürgen Lehn. “A non-linear congruential pseudo random number generator”. In: *Statistische Hefte* 27 (Dec. 1986), pp. 315–326. URL: https://doi.org/10.1007/BF02932576 (cit. on p. 5).

[182] Thomas Espitau, Pierre-Alain Fouque, and Pierre Karpman. “Higher-Order Differential Meet-in-the-middle Preimage Attacks on SHA-1 and BLAKE”. In: *Advances in Cryptology – CRYPTO 2015, Part I*. Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9215. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2015, pp. 683–701. DOI: 10.1007/978-3-662-47989-6_33 (cit. on p. 73).

[183] *eSTREAM - the ECRYPT Stream Cipher Project*. 2004. URL: http://www.ecrypt.eu.org/stream (cit. on pp. 72, 93).

[184] ETSI. *Open Smart Grid Protocol (OSGP)*. Reference DGS/OSG-001. http://www.osgp.org/. Sophia Antipolis Cedex – France: European Telecommunications Standards Institute, Jan. 2012 (cit. on pp. 131–136).

[185] Shimon Even and Yishay Mansour. "A Construction of a Cipher From a Single Pseudorandom Permutation". In: *Advances in Cryptology – ASIACRYPT'91*. Ed. by Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto. Vol. 739. Lecture Notes in Computer Science. Springer, Heidelberg, Nov. 1993, pp. 210–224. DOI: 10.1007/3-540-57332-1_17 (cit. on p. 19).

[186] Shimon Even and Yishay Mansour. "A Construction of a Cipher from a Single Pseudorandom Permutation". In: *Journal of Cryptology* 10.3 (June 1997), pp. 151–162. DOI: 10.1007/s001459900025 (cit. on p. 19).

[187] Jean Charles Faugère. "A new efficient algorithm for computing (Gröbner) bases (F4)". In: *Journal of Pure and Applied Algebra* 139.1 (1999), pp. 61–88. ISSN: 0022-4049. URL: https://www.sciencedirect.com/science/article/pii/S0022404999000055 (cit. on p. 38).

[188] H. Feistel, W.A. Notz, and J.L. Smith. "Some cryptographic techniques for machine-to-machine data communications". In: *Proceedings of the IEEE* 63.11 (Nov. 1975), pp. 1545–1554. ISSN: 0018-9219. DOI: 10.1109/PROC.1975.10005 (cit. on p. 16).

[189] Horst Feistel. *Cryptographic Coding for Data-Bank Privacy*. RC 2827. IBM Research, 1970 (cit. on p. 16).

[190] Horst Feistel. "Cryptography and Computer Privacy". In: *Scientific American* 228 (May 1973), pp. 15–23 (cit. on pp. 16, 32).

[191] Linus Feiten and Matthias Sauer. *Extracting the RC4 secret key of the Open Smart Grid Protocol*. Cryptology ePrint Archive, Report 2016/455. https://eprint.iacr.org/2016/455. 2016 (cit. on p. 132).

[192] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, and Jesse Walker. *The Skein Hash Function Family*. Submitted to NIST for their cryptographic hash algorithm competition [364]. Oct. 2010. URL: https://www.schneier.com/cryptography/skein/ (cit. on p. 6).

[193] Alan M. Ferrenberg, D. P. Landau, and Y. Joanna Wong. "Monte Carlo simulations: Hidden errors from "good" random number generators". In: *Phys. Rev. Lett.* 69 (23 Dec. 1992), pp. 3382–3384. URL: http://link.aps.org/doi/10.1103/PhysRevLett.69.3382 (cit. on p. 15).

[194] Max Fillinger and Marc Stevens. "Reverse-Engineering of the Cryptanalytic Attack Used in the Flame Super-Malware". In: *Advances in Cryptology – ASIACRYPT 2015, Part II*. Ed. by Tetsu Iwata and Jung Hee Cheon. Vol. 9453. Lecture Notes in Computer Science. Springer, Heidelberg, Nov. 2015, pp. 586–611. DOI: 10.1007/978-3-662-48800-3_24 (cit. on p. 2).

[195] Philippe Flajolet and Andrew M. Odlyzko. “Random Mapping Statistics”. In: *Advances in Cryptology – EUROCRYPT’89*. Ed. by Jean-Jacques Quisquater and Joos Vandewalle. Vol. 434. Lecture Notes in Computer Science. Springer, Heidelberg, Apr. 1990, pp. 329–354. DOI: 10.1007/3-540-46885-4_34 (cit. on p. 44).

[196] Ewan Fleischmann, Christian Forler, and Stefan Lucks. “McOE: A Family of Almost Foolproof On-Line Authenticated Encryption Schemes”. In: *Fast Software Encryption – FSE 2012*. Ed. by Anne Canteaut. Vol. 7549. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2012, pp. 196–215. DOI: 10.1007/978-3-642-34047-5_12 (cit. on p. 127).

[197] Scott R. Fluhrer, Itsik Mantin, and Adi Shamir. “Weaknesses in the Key Scheduling Algorithm of RC4”. In: *SAC 2001: 8th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Serge Vaudenay and Amr M. Youssef. Vol. 2259. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2001, pp. 1–24. DOI: 10.1007/3-540-45537-X_1 (cit. on p. 136).

[198] Scott R. Fluhrer and David A. McGrew. “Statistical Analysis of the Alleged RC4 Keystream Generator”. In: *Fast Software Encryption – FSE 2000*. Ed. by Bruce Schneier. Vol. 1978. Lecture Notes in Computer Science. Springer, Heidelberg, Apr. 2001, pp. 19–30. DOI: 10.1007/3-540-44706-7_2 (cit. on pp. 2, 136).

[199] Christian Forler, Stefan Lucks, and Jakob Wenzel. “Memory-Demanding Password Scrambling”. In: *Advances in Cryptology – ASIACRYPT 2014, Part II*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8874. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2014, pp. 289–305. DOI: 10.1007/978-3-662-45608-8_16 (cit. on p. 147).

[200] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. “Electromagnetic Analysis: Concrete Results”. In: *Cryptographic Hardware and Embedded Systems – CHES 2001*. Ed. by Çetin Kaya Koç, David Naccache, and Christof Paar. Vol. 2162. Lecture Notes in Computer Science. Springer, Heidelberg, May 2001, pp. 251–261. DOI: 10.1007/3-540-44709-1_21 (cit. on p. 78).

[201] Vijay Ganesh and David L. Dill. “A Decision Procedure for Bit-Vectors and Arrays”. In: *Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings*. Ed. by Werner Damm and Holger Hermanns. Vol. 4590. Lecture Notes in Computer Science. Springer, 2007, pp. 519–531. ISBN: 978-3-540-73367-6. URL: https://doi.org/10.1007/978-3-540-73368-3%5C_52 (cit. on p. 109).

[202] Christina Garman, Kenneth G. Paterson, and Thyla van der Merwe. "Attacks Only Get Better: Password Recovery Attacks Against RC4 in TLS". In: *USENIX Security 2015: 24th USENIX Security Symposium*. Ed. by Jaeyeon Jung and Thorsten Holz. USENIX Association, Aug. 2015, pp. 113–128 (cit. on p. 2).

[203] Peter Gazi and Stefano Tessaro. "Provably Robust Sponge-Based PRNGs and KDFs". In: *Advances in Cryptology – EUROCRYPT 2016, Part I*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9665. Lecture Notes in Computer Science. Springer, Heidelberg, May 2016, pp. 87–116. DOI: 10.1007/978-3-662-49890-3_4 (cit. on p. 26).

[204] James E. Gentle. *Random Number Generation and Monte Carlo Methods*. 2nd ed. Springer-Verlag, 2003. ISBN: 978-0-387-00178-4 (cit. on pp. 39, 44).

[205] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. "How to Construct Random Functions". In: *Journal of the ACM* 33.4 (Oct. 1986), pp. 792–807 (cit. on p. 14).

[206] Shafi Goldwasser and Silvio Micali. "Probabilistic Encryption". In: *Journal of Computer and System Sciences* 28.2 (1984), pp. 270–299 (cit. on p. 14).

[207] Louis Goubin. "A Sound Method for Switching between Boolean and Arithmetic Masking". In: *Cryptographic Hardware and Embedded Systems – CHES 2001*. Ed. by Çetin Kaya Koç, David Naccache, and Christof Paar. Vol. 2162. Lecture Notes in Computer Science. Springer, Heidelberg, May 2001, pp. 3–15. DOI: 10.1007/3-540-44709-1_2 (cit. on p. 78).

[208] Louis Goubin and Jacques Patarin. "DES and Differential Power Analysis (The "Duplication" Method)". In: *Cryptographic Hardware and Embedded Systems – CHES'99*. Ed. by Çetin Kaya Koç and Christof Paar. Vol. 1717. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 1999, pp. 158–172. DOI: 10.1007/3-540-48059-5_15 (cit. on p. 78).

[209] Peter Grassberger. "On correlations in "good" random number generators". In: *Physics Letters A* 181.1 (1993), pp. 43–46. ISSN: 0375-9601. URL: http://www.sciencedirect.com/science/article/pii/037596019391122L (cit. on p. 15).

[210] Bert F. Green Jr., J. E. Keith Smith, and Laura Klem. "Empirical Tests of an Additive Random Number Generator". In: *J. ACM* 6.4 (Oct. 1959), pp. 527–537. ISSN: 0004-5411. URL: http://doi.acm.org/10.1145/320998.321006 (cit. on p. 4).

[211] Martin Greenberger. "Method in Randomness". In: *Communications of the ACM* 8.3 (Mar. 1965), pp. 177–179. URL: http://doi.acm.org/10.1145/363791.363827 (cit. on pp. 4, 15).

[212] Glenn Greenwald. *No place to hide: Edward Snowden, the NSA and the surveillance state*. 1st ed. Picador, 2015. ISBN: 9781250062581 (cit. on p. 1).

[213] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varici. "LS-Designs: Bitslice Encryption for Efficient Masked Software Implementations". In: *Fast Software Encryption – FSE 2014*. Ed. by Carlos Cid and Christian Rechberger. Vol. 8540. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2015, pp. 18–37. DOI: 10.1007/978-3-662-46706-0_2 (cit. on p. 97).

[214] Aldo Gunsing, Joan Daemen, and Bart Mennink. "Errata to Sound Hashing Modes of Arbitrary Functions, Permutations, and BCs". In: *IACR Transactions on Symmetric Cryptology* 2020.3 (2020), pp. 362–366. ISSN: 2519-173X. DOI: 10.13154/tosc.v2020.i3.362-366 (cit. on pp. 62, 74).

[215] Jian Guo, Pierre Karpman, Ivica Nikolic, Lei Wang, and Shuang Wu. "Analysis of BLAKE2". In: *Topics in Cryptology – CT-RSA 2014*. Ed. by Josh Benaloh. Vol. 8366. Lecture Notes in Computer Science. Springer, Heidelberg, Feb. 2014, pp. 402–423. DOI: 10.1007/978-3-319-04852-9_21 (cit. on p. 73).

[216] Jian Guo and Krystian Matusiewicz. *Round-Reduced Near-Collisions of BLAKE-32*. Accepted for presentation at WEWoRC 2009. 2009. URL: http://www.jguo.org/blake-col.pdf (cit. on p. 72).

[217] Sourav Sen Gupta, Subhamoy Maitra, Goutam Paul, and Santanu Sarkar. "(Non-)Random Sequences from (Non-)Random Permutations - Analysis of RC4 Stream Cipher". In: *Journal of Cryptology* 27.1 (Jan. 2014), pp. 67–108. DOI: 10.1007/s00145-012-9138-1 (cit. on p. 136).

[218] Shai Halevi and Hugo Krawczyk. "Strengthening Digital Signatures Via Randomized Hashing". In: *Advances in Cryptology – CRYPTO 2006*. Ed. by Cynthia Dwork. Vol. 4117. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2006, pp. 41–59. DOI: 10.1007/11818175_3 (cit. on p. 68).

[219] Mathias Hall-Andersen and Philip S. Vejre. "Generating Graphs Packed with Paths". In: *IACR Transactions on Symmetric Cryptology* 2018.3 (2018), pp. 265–289. ISSN: 2519-173X. DOI: 10.13154/tosc.v2018.i3.265-289 (cit. on p. 33).

[220] Mike Hamburg. *The STROBE protocol framework*. Cryptology ePrint Archive, Report 2017/003. https://eprint.iacr.org/2017/003. 2017 (cit. on p. 26).

[221] Yonglin Hao. "The Boomerang Attacks on BLAKE and BLAKE2". In: *Information Security and Cryptology - 10th International Conference, Inscrypt 2014, Beijing, China, December 13-15, 2014, Revised Selected Papers*. Ed. by Dongdai Lin, Moti Yung, and Jianying Zhou. Vol. 8957. Lecture Notes in Computer Science. Springer, 2014, pp. 286–310. ISBN: 978-3-319-16744-2. URL: https://doi.org/10.1007/978-3-319-16745-9%5C_16 (cit. on p. 73).

[222] Carlo Harpes, Gerhard G. Kramer, and James L. Massey. "A Generalization of Linear Cryptanalysis and the Applicability of Matsui's Piling-Up Lemma". In: *Advances in Cryptology – EUROCRYPT'95*. Ed. by Louis C. Guillou and Jean-Jacques Quisquater. Vol. 921. Lecture Notes in Computer Science. Springer, Heidelberg, May 1995, pp. 24–38. DOI: 10.1007/3-540-49264-X_3 (cit. on p. 31).

[223] Carlo Harpes and James L. Massey. "Partitioning Cryptanalysis". In: *Fast Software Encryption – FSE'97*. Ed. by Eli Biham. Vol. 1267. Lecture Notes in Computer Science. Springer, Heidelberg, Jan. 1997, pp. 13–27. DOI: 10.1007/BFb0052331 (cit. on p. 37).

[224] David Harvey and Joris van der Hoeven. "Integer multiplication in time $O(n \log n)$". In: *Annals of Mathematics* 193.2 (2021), pp. 563–617. URL: https://doi.org/10.4007/annals.2021.193.2.4 (cit. on p. 4).

[225] John Earl Haynes and Harvey Klehr. *Venona: Decoding Soviet Espionage in America*. Yale Nota Bene. Yale University Press, Aug. 2000, p. 512. ISBN: 978-0300084627. URL: http://www.jstor.org/stable/j.ctt1npk87 (cit. on p. 13).

[226] Martin Hell, Thomas Johansson, and Willi Meier. "Grain: a stream cipher for constrained environments". In: *IJWMC* 2.1 (2007), pp. 86–93. URL: http://dx.doi.org/10.1504/IJWMC.2007.013798 (cit. on p. 15).

[227] P. Hellekalek. "Inversive Pseudorandom Number Generators: Concepts, Results, and Links". In: *Proceedings of the 1995 Winter Simulation Conference*. Ed. by C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman. IEEE Press, 1995, pp. 255–262 (cit. on p. 40).

[228] Martin E. Hellman. "A cryptanalytic time-memory trade-off". In: *IEEE Trans. Information Theory* 26.4 (1980), pp. 401–406. URL: http://dx.doi.org/10.1109/TIT.1980.1056220 (cit. on p. 28).

[229] Miia Hermelin, Joo Yeon Cho, and Kaisa Nyberg. "Multidimensional Linear Cryptanalysis". In: *Journal of Cryptology* 32.1 (Jan. 2019), pp. 1–34. DOI: 10.1007/s00145-018-9308-x (cit. on p. 37).

[230] Shoichi Hirose, Je Hong Park, and Aaram Yun. "A Simple Variant of the Merkle-Damgård Scheme with a Permutation". In: *Advances in Cryptology – ASIACRYPT 2007*. Ed. by Kaoru Kurosawa. Vol. 4833. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2007, pp. 113–129. DOI: 10.1007/978-3-540-76900-2_7 (cit. on p. 25).

[231] Shoichi Hirose, Je Hong Park, and Aaram Yun. "A Simple Variant of the Merkle-Damgård Scheme with a Permutation". In: *Journal of Cryptology* 25.2 (Apr. 2012), pp. 271–309. DOI: 10.1007/s00145-010-9095-5 (cit. on p. 25).

[232] Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway. "Robust Authenticated-Encryption AEZ and the Problem That It Solves". In: *Advances in Cryptology – EUROCRYPT 2015, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. Lecture Notes in Computer Science. Springer, Heidelberg, Apr. 2015, pp. 15–44. DOI: 10.1007/978-3-662-46800-5_2 (cit. on p. 147).

[233] Viet Tung Hoang and Phillip Rogaway. "On Generalized Feistel Networks". In: *Advances in Cryptology – CRYPTO 2010*. Ed. by Tal Rabin. Vol. 6223. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2010, pp. 613–630. DOI: 10.1007/978-3-642-14623-7_33 (cit. on p. 17).

[234] Viet Tung Hoang and Stefano Tessaro. "Key-Alternating Ciphers and Key-Length Extension: Exact Bounds and Multi-user Security". In: *Advances in Cryptology – CRYPTO 2016, Part I*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2016, pp. 3–32. DOI: 10.1007/978-3-662-53018-4_1 (cit. on p. 19).

[235] Viet Tung Hoang, Cong Wu, and Xin Yuan. "Faster Yet Safer: Logging System Via Fixed-Key Blockcipher". In: *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*. Ed. by Kevin R. B. Butler and Kurt Thomas. USENIX Association, 2022, pp. 2389–2406. ISBN: 978-1-939133-31-1. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/hoang (cit. on p. 147).

[236] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. *Zcash Protocol Specification*. 2016. URL: https://github.com/zcash/zips/blob/master/protocol/protocol.pdf (cit. on p. 147).

[237] JiaLin Huang and XueJia Lai. "What is the effective key length for a block cipher: an attack on every practical block cipher". In: *Science China Information Sciences* 57.7 (July 2014), pp. 1–11. URL: https://doi.org/10.1007/s11432-014-5096-6 (cit. on p. 28).

[238] Daniel Hutchinson. "A Robust and Sponge-Like PRNG with Improved Efficiency". In: *SAC 2016: 23rd Annual International Workshop on Selected Areas in Cryptography*. Ed. by Roberto Avanzi and Howard M. Heys. Vol. 10532. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2016, pp. 381–398. DOI: 10.1007/978-3-319-69453-5_21 (cit. on p. 26).

[239] IEEE. *Draft Standard for Communications Protocol Aboard Passenger Trains*. IEEE P1473/D8. http://ieeexplore.ieee.org/servlet/opac?punumber=5511471. July 2010 (cit. on pp. 133, 144).

[240] Akiko Inoue, Tetsu Iwata, Kazuhiko Minematsu, and Bertram Poettering. "Cryptanalysis of OCB2: Attacks on Authenticity and Confidentiality". In: *Advances in Cryptology – CRYPTO 2019, Part I*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11692. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2019, pp. 3–31. DOI: 10.1007/978-3-030-26948-7_1 (cit. on p. 78).

[241] Akiko Inoue, Tetsu Iwata, Kazuhiko Minematsu, and Bertram Poettering. "Cryptanalysis of OCB2: Attacks on Authenticity and Confidentiality". In: *Journal of Cryptology* 33.4 (Oct. 2020), pp. 1871–1913. DOI: 10.1007/s00145-020-09359-8 (cit. on p. 78).

[242] ISO. *Information Technology – Control Network Protocol – Part 1: Protocol Stack*. ISO/IEC 14908-1:2012. Geneva, Switzerland: International Organization for Standardization, 2012 (cit. on pp. 131, 133, 144).

[243] ISO. *Information Technology — Interconnection of Information Technology Equipment — Home Electronic System (HES) Architecture — Medium-independent Protocol Based on ANSI/CEA-709.1-B*. ISO/IEC CD 14543-6-1:2006. http://hes-standards.org/doc/SC25_WG1_N1229.pdf. International Organization for Standardization, 2006 (cit. on pp. 133, 134, 144).

[244] Tetsu Iwata, Tohru Yagi, and Kaoru Kurosawa. "On the Pseudorandomness of KASUMI Type Permutations". In: *ACISP 03: 8th Australasian Conference on Information Security and Privacy*. Ed. by Reihaneh Safavi-Naini and Jennifer Seberry. Vol. 2727. Lecture Notes in Computer Science. Springer, Heidelberg, July 2003, pp. 130–141. DOI: 10.1007/3-540-45067-X_12 (cit. on p. 17).

[245] Tetsu Iwata, Tomonobu Yoshino, Tomohiro Yuasa, and Kaoru Kurosawa. "Round Security and Super-Pseudorandomness of MISTY Type Structure". In: *Fast Software Encryption – FSE 2001*. Ed. by Mitsuru Matsui. Vol. 2355. Lecture Notes in Computer Science. Springer, Heidelberg, Apr. 2002, pp. 233–247. DOI: 10.1007/3-540-45473-X_20 (cit. on p. 17).

[246] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C.-W. Phan. *Toy versions of BLAKE*. 2009. URL: https://www.aumasson.jp/blake/toyblake.pdf (cit. on p. 67).

[247] Antoine Joux. *Authentication Failures in NIST Version of GCM*. http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/Joux_comments.pdf. 2006 (cit. on p. 81).

[248] Antoine Joux. "Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions". In: *Advances in Cryptology – CRYPTO 2004*. Ed. by Matthew Franklin. Vol. 3152. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2004, pp. 306–316. DOI: 10.1007/978-3-540-28628-8_19 (cit. on pp. 25, 71).

[249] Antoine Joux, Gwenaëlle Martinet, and Frédéric Valette. "Blockwise-Adaptive Attackers: Revisiting the (In)Security of Some Provably Secure Encryption Models: CBC, GEM, IACBC". In: *Advances in Cryptology – CRYPTO 2002*. Ed. by Moti Yung. Vol. 2442. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2002, pp. 17–30. DOI: 10.1007/3-540-45708-9_2 (cit. on p. 20).

[250] Philipp Jovanovic, Atul Luykx, and Bart Mennink. "Beyond $2^{c/2}$ Security in Sponge-Based Authenticated Encryption Modes". In: *Advances in Cryptology – ASIACRYPT 2014, Part I*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8873. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2014, pp. 85–104. DOI: 10.1007/978-3-662-45611-8_5 (cit. on pp. 107, 108).

[251] Philipp Jovanovic, Atul Luykx, Bart Mennink, Yu Sasaki, and Kan Yasuda. "Beyond Conventional Security in Sponge-Based Authenticated Encryption Modes". In: *Journal of Cryptology* 32.3 (July 2019), pp. 895–940. DOI: 10.1007/s00145-018-9299-7 (cit. on p. 108).

[252] Philipp Jovanovic and Samuel Neves. "Practical Cryptanalysis of the Open Smart Grid Protocol". In: *Fast Software Encryption – FSE 2015*. Ed. by Gregor Leander. Vol. 9054. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2015, pp. 297–316. DOI: 10.1007/978-3-662-48116-5_15 (cit. on p. 7).

[253] David Kahn. *The codebreakers: the story of secret writing*. New York: Scribner, 1996. ISBN: 0-684-83130-9 (cit. on p. 1).

[254] Burton S. Kaliski Jr. "The MD4 Message Digest Algorithm (Abstract) (Rump Session)". In: *Advances in Cryptology – EUROCRYPT'90*. Ed. by Ivan Damgård. Vol. 473. Lecture Notes in Computer Science. Springer, Heidelberg, May 1991, p. 492. DOI: 10.1007/3-540-46877-3_46 (cit. on p. 25).

[255] John B. Kam and George I. Davida. “Structured Design of Substitution-Permutation Encryption Networks”. In: *IEEE Trans. Computers* 28.10 (1979), pp. 747–753. URL: https://doi.org/10.1109/TC.1979.1675242 (cit. on p. 19).

[256] Ju-Sung Kang, Sang Uk Shin, Dowon Hong, and Okyeon Yi. “Provable Security of KASUMI and 3GPP Encryption Mode f8”. In: *Advances in Cryptology – ASIACRYPT 2001*. Ed. by Colin Boyd. Vol. 2248. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2001, pp. 255–271. DOI: 10.1007/3-540-45682-1_16 (cit. on p. 17).

[257] Ju-Sung Kang, Okyeon Yi, Dowon Hong, and Hyun-Sook Cho. “Pseudorandomness of MISTY-Type Transformations and the Block Cipher KASUMI”. In: *ACISP 01: 6th Australasian Conference on Information Security and Privacy*. Ed. by Vijay Varadharajan and Yi Mu. Vol. 2119. Lecture Notes in Computer Science. Springer, Heidelberg, July 2001, pp. 60–73. DOI: 10.1007/3-540-47719-5_7 (cit. on p. 17).

[258] Emilia Käsper and Peter Schwabe. “Faster and Timing-Attack Resistant AES-GCM”. In: *Cryptographic Hardware and Embedded Systems – CHES 2009*. Ed. by Christophe Clavier and Kris Gaj. Vol. 5747. Lecture Notes in Computer Science. Springer, Heidelberg, Sept. 2009, pp. 1–17. DOI: 10.1007/978-3-642-04138-9_1 (cit. on p. 97).

[259] John Kelsey and Tadayoshi Kohno. “Herding Hash Functions and the Nostradamus Attack”. In: *Advances in Cryptology – EUROCRYPT 2006*. Ed. by Serge Vaudenay. Vol. 4004. Lecture Notes in Computer Science. Springer, Heidelberg, May 2006, pp. 183–200. DOI: 10.1007/11761679_12 (cit. on p. 25).

[260] John Kelsey and Bruce Schneier. “Second Preimages on n-Bit Hash Functions for Much Less than 2n Work”. In: *Advances in Cryptology – EUROCRYPT 2005*. Ed. by Ronald Cramer. Vol. 3494. Lecture Notes in Computer Science. Springer, Heidelberg, May 2005, pp. 474–490. DOI: 10.1007/11426639_28 (cit. on p. 25).

[261] Dmitry Khovratovich and Ivica Nikolic. “Rotational Cryptanalysis of ARX”. In: *Fast Software Encryption – FSE 2010*. Ed. by Seokhie Hong and Tetsu Iwata. Vol. 6147. Lecture Notes in Computer Science. Springer, Heidelberg, Feb. 2010, pp. 333–346. DOI: 10.1007/978-3-642-13858-4_19 (cit. on pp. 6, 32, 36, 122).

[262] Dmitry Khovratovich, Ivica Nikolic, Josef Pieprzyk, Przemyslaw Sokolowski, and Ron Steinfeld. “Rotational Cryptanalysis of ARX Revisited”. In: *Fast Software Encryption – FSE 2015*. Ed. by Gregor Leander. Vol. 9054. Lecture

Notes in Computer Science. Springer, Heidelberg, Mar. 2015, pp. 519–536. DOI: 10.1007/978-3-662-48116-5_25 (cit. on pp. 73, 123).

[263] Dmitry Khovratovich, Christian Rechberger, and Alexandra Savelieva. "Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 Family". In: *Fast Software Encryption – FSE 2012*. Ed. by Anne Canteaut. Vol. 7549. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2012, pp. 244–263. DOI: 10.1007/978-3-642-34047-5_15 (cit. on p. 28).

[264] Joe Kilian and Phillip Rogaway. "How to Protect DES Against Exhaustive Key Search". In: *Advances in Cryptology – CRYPTO'96*. Ed. by Neal Koblitz. Vol. 1109. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 1996, pp. 252–267. DOI: 10.1007/3-540-68697-5_20 (cit. on p. 19).

[265] Joe Kilian and Phillip Rogaway. "How to Protect DES Against Exhaustive Key Search (an Analysis of DESX)". In: *Journal of Cryptology* 14.1 (Jan. 2001), pp. 17–35. DOI: 10.1007/s001450010015 (cit. on p. 19).

[266] Aviad Kipnis and Adi Shamir. "Cryptanalysis of the HFE Public Key Cryptosystem by Relinearization". In: *Advances in Cryptology – CRYPTO'99*. Ed. by Michael J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 1999, pp. 19–30. DOI: 10.1007/3-540-48405-1_2 (cit. on p. 37).

[267] Vlastimil Klima. *Tunnels in Hash Functions: MD5 Collisions Within a Minute*. Cryptology ePrint Archive, Report 2006/105. https://eprint.iacr.org/2006/105. 2006 (cit. on p. 2).

[268] Alexander Klimov and Adi Shamir. "Cryptographic Applications of T-Functions". In: *SAC 2003: 10th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Mitsuru Matsui and Robert J. Zuccherato. Vol. 3006. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2004, pp. 248–261. DOI: 10.1007/978-3-540-24654-1_18 (cit. on p. 48).

[269] Lars R. Knudsen. "DEAL — A 128-bit Block Cipher". In: *NIST AES Proposal*. 1998 (cit. on p. 36).

[270] Lars R. Knudsen. "Truncated and Higher Order Differentials". In: *Fast Software Encryption – FSE'94*. Ed. by Bart Preneel. Vol. 1008. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 1995, pp. 196–211. DOI: 10.1007/3-540-60590-8_16 (cit. on p. 36).

[271] Lars R. Knudsen and Vincent Rijmen. "Known-Key Distinguishers for Some Block Ciphers". In: *Advances in Cryptology – ASIACRYPT 2007*. Ed. by Kaoru Kurosawa. Vol. 4833. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2007, pp. 315–324. DOI: 10.1007/978-3-540-76900-2_19 (cit. on p. 27).

[272] Lars R. Knudsen and David Wagner. "Integral Cryptanalysis". In: *Fast Software Encryption – FSE 2002*. Ed. by Joan Daemen and Vincent Rijmen. Vol. 2365. Lecture Notes in Computer Science. Springer, Heidelberg, Feb. 2002, pp. 112–127. DOI: 10.1007/3-540-45661-9_9 (cit. on p. 36).

[273] D. Knuth. "Deciphering a Linear Congruential Encryption". In: *IEEE Trans. Inf. Theor.* 31.1 (Jan. 1985), pp. 49–52. ISSN: 0018-9448. URL: http://dx.doi.org/10.1109/TIT.1985.1056997 (cit. on p. 15).

[274] Donald E. Knuth. *Art of Computer Programming. Volume 1: Fundamental Algorithms*. Addison-Wesley, July 2002. ISBN: 0201896834 (cit. on p. 44).

[275] Donald E. Knuth. *Art of Computer Programming. Volume 2: Seminumerical Algorithms*. 3rd ed. Addison-Wesley Professional, Nov. 1997. ISBN: 0201896842 (cit. on pp. 4, 5, 29, 40).

[276] Donald E. Knuth. *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1*. Vol. 4A. Upper Saddle River, New Jersey: Addison-Wesley, 2011, xvi+883pp. ISBN: 0-201-03804-8. URL: http://www-cs-faculty.stanford.edu/~uno/taocp.html (cit. on p. 79).

[277] Neal Koblitz and Alfred J. Menezes. "The random oracle model: a twenty-year retrospective". In: *Des. Codes Cryptogr.* 77.2-3 (2015), pp. 587–610. URL: https://doi.org/10.1007/s10623-015-0094-2 (cit. on p. 23).

[278] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. "Differential Power Analysis". In: *Advances in Cryptology – CRYPTO'99*. Ed. by Michael J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 1999, pp. 388–397. DOI: 10.1007/3-540-48405-1_25 (cit. on p. 78).

[279] Peter M. Kogge and Harold S. Stone. "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations". In: *IEEE Trans. Computers* 22.8 (1973), pp. 786–793. URL: https://doi.org/10.1109/TC.1973.5009159 (cit. on p. 78).

[280] Leslie Kohn, Guillermo Maturana, Marc Tremblay, A. Prabhu, and Gregory B. Zyner. "The Visual Instruction Set (VIS) in UltraSPARC". In: *COMPCON*. 1995, pp. 462–469 (cit. on p. 3).

[281] Klaus Kursawe and Christiane Peters. "Structural Weaknesses in the Open Smart Grid Protocol". In: *10th International Conference on Availability, Reliability and Security, ARES 2015, Toulouse, France, August 24-27, 2015*. IEEE Computer Society, 2015, pp. 1–10. ISBN: 978-1-4673-6590-1. URL: https://doi.org/10.1109/ARES.2015.67 (cit. on p. 132).

[282] Pierre L'Ecuyer and Richard Simard. "On the performance of birthday spacings tests with certain families of random number generators". In: *Math. Comput. Simul.* 55.1-3 (2001), pp. 131–137. ISSN: 0378-4754. DOI: http://dx.doi.org/10.1016/S0378-4754(00)00253-6 (cit. on p. 44).

[283] Pierre L'Ecuyer and Richard Simard. "TestU01: A C library for empirical testing of random number generators". In: *ACM Trans. Math. Softw.* 33.4 (2007), p. 22. ISSN: 0098-3500. DOI: http://doi.acm.org/10.1145/1268776.1268777 (cit. on pp. 15, 50).

[284] Watson Ladd. *McMambo v1: A New Kind of Latin Dance*. Submitted to round 1 of CAESAR. Mar. 2014. URL: http://competitions.cr.yp.to/round1/mcmambov1.pdf (cit. on pp. 7, 127).

[285] Xuejia Lai. "Higher Order Derivatives and Differential Cryptanalysis". In: *Communications and Cryptography: Two Sides of One Tapestry*. Ed. by Richard E. Blahut, Daniel J. Costello, Ueli Maurer, and Thomas Mittelholzer. Boston, MA: Springer US, 1994, pp. 227–233. ISBN: 978-1-4615-2694-0. URL: http://dx.doi.org/10.1007/978-1-4615-2694-0_23 (cit. on p. 36).

[286] Xuejia Lai and James L. Massey. "A Proposal for a New Block Encryption Standard". In: *Advances in Cryptology – EUROCRYPT'90*. Ed. by Ivan Damgård. Vol. 473. Lecture Notes in Computer Science. Springer, Heidelberg, May 1991, pp. 389–404. DOI: 10.1007/3-540-46877-3_35 (cit. on p. 5).

[287] Xuejia Lai, James L. Massey, and Sean Murphy. "Markov Ciphers and Differential Cryptanalysis". In: *Advances in Cryptology – EUROCRYPT'91*. Ed. by Donald W. Davies. Vol. 547. Lecture Notes in Computer Science. Springer, Heidelberg, Apr. 1991, pp. 17–38. DOI: 10.1007/3-540-46416-6_2 (cit. on pp. 31, 33, 36).

[288] Rodolphe Lampe, Jacques Patarin, and Yannick Seurin. "An Asymptotically Tight Security Analysis of the Iterated Even-Mansour Cipher". In: *Advances in Cryptology – ASIACRYPT 2012*. Ed. by Xiaoyun Wang and Kazue Sako. Vol. 7658. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2012, pp. 278–295. DOI: 10.1007/978-3-642-34961-4_18 (cit. on p. 19).

[289] W. B. Langdon. "A fast high quality pseudo random number generator for nVidia CUDA". In: *GECCO '09: Proceedings of the 11th annual conference companion on Genetic and evolutionary computation conference*. Montreal, Québec, Canada: ACM, 2009, pp. 2511–2514. ISBN: 978-1-60558-505-5. DOI: http://doi.acm.org/10.1145/1570256.1570353 (cit. on p. 41).

[290] Susan K. Langford and Martin E. Hellman. "Differential-Linear Cryptanalysis". In: *Advances in Cryptology – CRYPTO'94*. Ed. by Yvo Desmedt. Vol. 839. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 1994, pp. 17–25. DOI: 10.1007/3-540-48658-5_3 (cit. on p. 36).

[291] Adam Langley. *Overclocking SSL*. June 2010. URL: https://www.imperialviolet.org/2010/06/25/overclocking-ssl.html (cit. on p. 2).

[292] Wonil Lee, Kouichi Sakurai, Seokhie Hong, and Sangjin Lee. "On the Pseudorandomness of a Modification of KASUMI Type Permutations". In: *ICISC 04: 7th International Conference on Information Security and Cryptology*. Ed. by Choonsik Park and Seongtaek Chee. Vol. 3506. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2005, pp. 313–329 (cit. on p. 17).

[293] D. H. Lehmer. "Mathematical Methods in Large Scale Computing Units". In: *Annals Comp. Laboratory Harvard University* 26 (1951), pp. 141–146 (cit. on p. 4).

[294] Gaëtan Leurent. "Analysis of Differential Attacks in ARX Constructions". In: *Advances in Cryptology – ASIACRYPT 2012*. Ed. by Xiaoyun Wang and Kazue Sako. Vol. 7658. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2012, pp. 226–243. DOI: 10.1007/978-3-642-34961-4_15 (cit. on p. 72).

[295] Gaëtan Leurent. "ARXtools: A toolkit for ARX analysis". In: *The Third SHA-3 Candidate Conference*. Mar. 2012 (cit. on p. 72).

[296] Gaëtan Leurent and Thomas Peyrin. "SHA-1 is a Shambles: First Chosen-Prefix Collision on SHA-1 and Application to the PGP Web of Trust". In: *USENIX Security 2020: 29th USENIX Security Symposium*. Ed. by Srdjan Capkun and Franziska Roesner. USENIX Association, Aug. 2020, pp. 1839–1856 (cit. on p. 2).

[297] Ji Li and Liangyu Xu. *Attacks on Round-Reduced BLAKE*. Cryptology ePrint Archive, Report 2009/238. https://eprint.iacr.org/2009/238. 2009 (cit. on pp. 72, 73).

[298] Erik Lindholm, John Nickolls, Stuart Oberman, and John Montrym. "NVIDIA Tesla: A Unified Graphics and Computing Architecture". In: *IEEE Micro* 28.2 (2008), pp. 39–55. ISSN: 0272-1732. DOI: http://dx.doi.org/10.1109/MM.2008.31 (cit. on p. 39).

[299] Helger Lipmaa and Shiho Moriai. "Efficient Algorithms for Computing Differential Properties of Addition". In: *Fast Software Encryption – FSE 2001*. Ed. by Mitsuru Matsui. Vol. 2355. Lecture Notes in Computer Science. Springer, Heidelberg, Apr. 2002, pp. 336–350. DOI: 10.1007/3-540-45473-X_28 (cit. on pp. 112, 113, 137, 142).

[300] Helger Lipmaa, Johan Wallén, and Philippe Dumas. “On the Additive Differential Probability of Exclusive-Or”. In: *Fast Software Encryption – FSE 2004*. Ed. by Bimal K. Roy and Willi Meier. Vol. 3017. Lecture Notes in Computer Science. Springer, Heidelberg, Feb. 2004, pp. 317–331. DOI: 10.1007/978-3-540-25937-4_20 (cit. on p. 113).

[301] J. S. Liptay. “Structural Aspects of the System/360 Model 85: II the Cache”. In: 7.1 (Mar. 1968), pp. 15–21. ISSN: 0018-8670. URL: http://dx.doi.org/10.1147/sj.71.0015 (cit. on p. 3).

[302] LonMark International. *LON and BACnet: History and Approach*. http://www.lonmark.org/connection/presentations/2012/Q2/Light-Building/06+LON+and+BACnet-+History+and--Newron+System.pdf (cit. on p. 145).

[303] Jiqiang Lu. “A Methodology for Differential-Linear Cryptanalysis and Its Applications - (Extended Abstract)”. In: *Fast Software Encryption – FSE 2012*. Ed. by Anne Canteaut. Vol. 7549. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2012, pp. 69–89. DOI: 10.1007/978-3-642-34047-5_5 (cit. on p. 36).

[304] Michael Luby. *Pseudorandomness and cryptographic applications*. Princeton computer science notes. Princeton, NJ, USA: Princeton University Press, 1996, pp. xvi + 234.

[305] Michael Luby and Charles Rackoff. “How to Construct Pseudo-Random Permutations from Pseudo-Random Functions (Abstract)”. In: *Advances in Cryptology – CRYPTO’85*. Ed. by Hugh C. Williams. Vol. 218. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 1986, p. 447. DOI: 10.1007/3-540-39799-X_34 (cit. on p. 17).

[306] Michael Luby and Charles Rackoff. “How to construct pseudorandom permutations from pseudorandom functions”. In: *SIAM Journal on Computing* 17.2 (1988) (cit. on p. 17).

[307] Stefan Lucks. “A Failure-Friendly Design Principle for Hash Functions”. In: *Advances in Cryptology – ASIACRYPT 2005*. Ed. by Bimal K. Roy. Vol. 3788. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2005, pp. 474–494. DOI: 10.1007/11593447_26 (cit. on p. 25).

[308] Stefan Lucks. *Design Principles for Iterated Hash Functions*. Cryptology ePrint Archive, Report 2004/253. https://eprint.iacr.org/2004/253. 2004 (cit. on p. 25).

[309] Stefan Lucks. "Faster Luby-Rackoff Ciphers". In: *Fast Software Encryption – FSE'96*. Ed. by Dieter Gollmann. Vol. 1039. Lecture Notes in Computer Science. Springer, Heidelberg, Feb. 1996, pp. 189–203. DOI: 10.1007/3-540-60865-6_53 (cit. on p. 17).

[310] Stefan Lucks. "The Saturation Attack - A Bait for Twofish". In: *Fast Software Encryption – FSE 2001*. Ed. by Mitsuru Matsui. Vol. 2355. Lecture Notes in Computer Science. Springer, Heidelberg, Apr. 2002, pp. 1–15. DOI: 10.1007/3-540-45473-X_1 (cit. on p. 36).

[311] Yiyuan Luo, Xuejia Lai, and Zheng Gong. "Pseudorandomness analysis of the (extended) Lai-Massey scheme". In: *Inf. Process. Lett.* 111.2 (2010), pp. 90–96. URL: http://dx.doi.org/10.1016/j.ipl.2010.10.012 (cit. on p. 17).

[312] Yiyuan Luo, Xuejia Lai, and Jing Hu. "The Pseudorandomness of Many-Round Lai-Massey Scheme". In: *J. Inf. Sci. Eng.* 31.3 (2015), pp. 1085–1096. URL: http://www.iis.sinica.edu.tw/page/jise/2015/201505_17.html (cit. on p. 17).

[313] Atul Luykx, Bart Mennink, and Samuel Neves. "Security Analysis of BLAKE2's Modes of Operation". In: *IACR Transactions on Symmetric Cryptology* 2016.1 (2016). https://tosc.iacr.org/index.php/ToSC/article/view/540, pp. 158–176. ISSN: 2519-173X. DOI: 10.13154/tosc.v2016.i1.158-176 (cit. on p. 74).

[314] Atul Luykx, Bart Preneel, Elmar Tischhauser, and Kan Yasuda. "A MAC Mode for Lightweight Block Ciphers". In: *Fast Software Encryption – FSE 2016*. Ed. by Thomas Peyrin. Vol. 9783. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2016, pp. 43–59. DOI: 10.1007/978-3-662-52993-5_3 (cit. on p. 21).

[315] W. E. Madryga. "A High Performance Encryption Algorithm". In: *Proceedings of the 2nd IFIP International Conference on Computer Security: A Global Challenge*. Toronto, Ontario, Canada: North-Holland Publishing Co., 1984, pp. 557–569. ISBN: 0-444-87618-9 (cit. on p. 5).

[316] G.A. Maranon, D. Martinez, F.M. Vitini, and A.P. Dominguez. "Akelarre: A New Block Cipher Algorithm". In: *Selected Areas in Cryptography '98, SAC'98, Kingston, Ontario, Canada, August 17-18, 1998, Proceedings*. Ed. by H. Meijer and S. Tavares. Lecture Notes in Computer Science. 1996 (cit. on p. 5).

[317] G. Marsaglia. "The Marsaglia Random Number CDROM including the DIEHARD Battery of Tests of Randomness". See http://stat.fsu.edu/pub/diehard. 1996 (cit. on pp. 15, 50).

[318] George Marsaglia. "Random numbers fall mainly in the planes". In: *PNAS* 61 (1 1968), pp. 25–28 (cit. on p. 4).

[319] George Marsaglia. “Xorshift RNGs”. In: *Journal of Statistical Software* 8.14 (July 2003). URL: http://www.jstatsoft.org/v08/i14 (cit. on pp. 4, 40).

[320] George Marsaglia and Liang-Huei Tsay. “Matrices and the structure of random number sequences”. In: *Linear Algebra and its Applications* 67 (1985), pp. 147–156. ISSN: 0024-3795. URL: https://www.sciencedirect.com/science/article/pii/0024379585901922 (cit. on p. 4).

[321] J.L. Massey. “Shift-register synthesis and BCH decoding”. In: *Information Theory, IEEE Transactions on* 15.1 (Jan. 1969), pp. 122–127. ISSN: 0018-9448 (cit. on p. 15).

[322] Mitsuru Matsui. “Linear Cryptanalysis Method for DES Cipher”. In: *Advances in Cryptology – EUROCRYPT’93*. Ed. by Tor Helleseth. Vol. 765. Lecture Notes in Computer Science. Springer, Heidelberg, May 1994, pp. 386–397. DOI: 10.1007/3-540-48285-7_33 (cit. on p. 34).

[323] Mitsuru Matsui. “On Correlation Between the Order of S-boxes and the Strength of DES”. In: *Advances in Cryptology – EUROCRYPT’94*. Ed. by Alfredo De Santis. Vol. 950. Lecture Notes in Computer Science. Springer, Heidelberg, May 1995, pp. 366–375. DOI: 10.1007/BFb0053451 (cit. on p. 119).

[324] Mitsuru Matsui. “The First Experimental Cryptanalysis of the Data Encryption Standard”. In: *Advances in Cryptology – CRYPTO’94*. Ed. by Yvo Desmedt. Vol. 839. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 1994, pp. 1–11. DOI: 10.1007/3-540-48658-5_1 (cit. on p. 34).

[325] Mitsuru Matsui and Atsuhiro Yamagishi. “A New Method for Known Plaintext Attack of FEAL Cipher”. In: *Advances in Cryptology – EUROCRYPT’92*. Ed. by Rainer A. Rueppel. Vol. 658. Lecture Notes in Computer Science. Springer, Heidelberg, May 1993, pp. 81–91. DOI: 10.1007/3-540-47555-9_7 (cit. on p. 34).

[326] Makoto Matsumoto and Takuji Nishimura. “Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator”. In: *ACM Trans. Model. Comput. Simul.* 8.1 (1998), pp. 3–30. ISSN: 1049-3301. DOI: http://doi.acm.org/10.1145/272991.272995 (cit. on pp. 4, 40, 49).

[327] Ueli M. Maurer. “A Simplified and Generalized Treatment of Luby-Rackoff Pseudorandom Permutation Generator”. In: *Advances in Cryptology – EUROCRYPT’92*. Ed. by Rainer A. Rueppel. Vol. 658. Lecture Notes in Computer Science. Springer, Heidelberg, May 1993, pp. 239–255. DOI: 10.1007/3-540-47555-9_21 (cit. on p. 17).

[328] Ueli M. Maurer, Renato Renner, and Clemens Holenstein. "Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology". In: *TCC 2004: 1st Theory of Cryptography Conference*. Ed. by Moni Naor. Vol. 2951. Lecture Notes in Computer Science. Springer, Heidelberg, Feb. 2004, pp. 21–39. DOI: 10.1007/978-3-540-24638-1_2 (cit. on p. 24).

[329] Peter Maxwell. *Wheesht: an AEAD stream cipher*. Submitted to round 1 of CAESAR. Mar. 2014. URL: http://competitions.cr.yp.to/round1/wheeshtv03.pdf (cit. on pp. 7, 128).

[330] Lauren May, Lyta Penna, and Andrew J. Clark. "An Implementation of Bitsliced DES on the Pentium MMX$^{TM}$ Processor". In: *ACISP 00: 5th Australasian Conference on Information Security and Privacy*. Ed. by Ed Dawson, Andrew Clark, and Colin Boyd. Vol. 1841. Lecture Notes in Computer Science. Springer, Heidelberg, July 2000, pp. 112–122. DOI: 10.1007/10718964_10 (cit. on p. 97).

[331] David A. McGrew and John Viega. "The Security and Performance of the Galois/Counter Mode (GCM) of Operation". In: *Progress in Cryptology - INDOCRYPT 2004: 5th International Conference in Cryptology in India*. Ed. by Anne Canteaut and Kapalee Viswanathan. Vol. 3348. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2004, pp. 343–355 (cit. on pp. 21, 22).

[332] Bart Mennink, Reza Reyhanitabar, and Damian Vizár. "Security of Full-State Keyed Sponge and Duplex: Applications to Authenticated Encryption". In: *Advances in Cryptology – ASIACRYPT 2015, Part II*. Ed. by Tetsu Iwata and Jung Hee Cheon. Vol. 9453. Lecture Notes in Computer Science. Springer, Heidelberg, Nov. 2015, pp. 465–489. DOI: 10.1007/978-3-662-48800-3_19 (cit. on p. 26).

[333] Ralph C. Merkle. "A Fast Software One-Way Hash Function". In: *Journal of Cryptology* 3.1 (Jan. 1990), pp. 43–58. DOI: 10.1007/BF00203968 (cit. on p. 26).

[334] Ralph C. Merkle. "One Way Hash Functions and DES". In: *Advances in Cryptology – CRYPTO'89*. Ed. by Gilles Brassard. Vol. 435. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 1990, pp. 428–446. DOI: 10.1007/0-387-34805-0_40 (cit. on p. 25).

[335] Nicholas Metropolis and S. Ulam. "The Monte Carlo Method". In: *Journal of the American Statistical Association* 44.247 (1949), pp. 335–341. URL: http://dx.doi.org/10.2307/2280232 (cit. on p. 39).

[336] C. Meyer and W. Tuchman. “Pseudorandom codes can be cracked”. In: *Electronic Design* (23 Nov. 1972), pp. 74–76 (cit. on p. 15).

[337] Kazuhiko Minematsu, Stefan Lucks, Hiraku Morita, and Tetsu Iwata. “Attacks and Security Proofs of EAX-Prime”. In: *Fast Software Encryption – FSE 2013*. Ed. by Shiho Moriai. Vol. 8424. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2014, pp. 327–347. DOI: 10.1007/978-3-662-43933-3_17 (cit. on p. 78).

[338] Shoji Miyaguchi. “The FEAL Cipher Family (Impromptu Talk)”. In: *Advances in Cryptology – CRYPTO’90*. Ed. by Alfred J. Menezes and Scott A. Vanstone. Vol. 537. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 1991, pp. 627–638. DOI: 10.1007/3-540-38424-3_46 (cit. on p. 34).

[339] Shoji Miyaguchi. “The FEAL-8 Cryptosystem and a Call for Attack (Rump Session)”. In: *Advances in Cryptology – CRYPTO’89*. Ed. by Gilles Brassard. Vol. 435. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 1990, pp. 624–627. DOI: 10.1007/0-387-34805-0_59 (cit. on p. 34).

[340] Bodo Möller, Thai Duong, and Krzysztof Kotowicz. *This POODLE Bites: Exploiting The SSL 3.0 Fallback*. https://www.openssl.org/~bodo/ssl-poodle.pdf. Oct. 2014 (cit. on p. 77).

[341] Gordon E. Moore. “Cramming More Components onto Integrated Circuits”. In: *Electronics* 38.8 (Apr. 1965), pp. 114–117. ISSN: 0018-9219 (cit. on p. 2).

[342] Gordon E. Moore. “Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff.” In: *IEEE Solid-State Circuits Society Newsletter* 11.5 (Sept. 2006), pp. 33–35. ISSN: 1098-4232. DOI: 10.1109/N-SSC.2006.4785860 (cit. on p. 2).

[343] Nicky Mouha and Atul Luykx. “Multi-key Security: The Even-Mansour Construction Revisited”. In: *Advances in Cryptology – CRYPTO 2015, Part I*. Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9215. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2015, pp. 209–223. DOI: 10.1007/978-3-662-47989-6_10 (cit. on p. 27).

[344] Nicky Mouha and Bart Preneel. *Towards Finding Optimal Differential Characteristics for ARX: Application to Salsa20*. Cryptology ePrint Archive, Report 2013/328. https://eprint.iacr.org/2013/328. 2013 (cit. on pp. 109, 114).

[345] Nicky Mouha, Mohammad S. Raunak, D. Richard Kuhn, and Raghu Kacker. “Finding Bugs in Cryptographic Hash Function Implementations”. In: *IEEE Trans. Reliab.* 67.3 (2018), pp. 870–884. URL: https://doi.org/10.1109/TR.2018.2847247 (cit. on p. 68).

[346] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. "Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming". In: *Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers*. Ed. by Chuankun Wu, Moti Yung, and Dongdai Lin. Vol. 7537. Lecture Notes in Computer Science. Springer, 2011, pp. 57–76. ISBN: 978-3-642-34703-0. URL: https://doi.org/10.1007/978-3-642-34704-7%5C_5 (cit. on p. 109).

[347] Michael Muehlberghuber and Frank K. Gürkaynak. *Towards Evaluating High-Speed ASIC Implementations of CAESAR Candidates for Data at Rest and Data in Motion*. Presented at DIAC 2015. Sept. 2015. URL: https://web.archive.org/web/20180508151135/http://www1.spms.ntu.edu.sg/~diac2015/slides/diac2015_08_caesar_asic.pdf (cit. on p. 105).

[348] Sean Murphy. "The Cryptanalysis of FEAL-4 with 20 Chosen Plaintexts". In: *Journal of Cryptology* 2.3 (Jan. 1990), pp. 145–154. DOI: 10.1007/BF00190801 (cit. on p. 32).

[349] Sean Murphy. "The effectiveness of the linear hull effect". In: *J. Math. Cryptol.* 6.2 (2012), pp. 137–147. URL: https://doi.org/10.1515/jmc-2011-0025 (cit. on p. 36).

[350] Valerie Nachef, Jacques Patarin, and Emmanuel Volte. *Feistel Ciphers: Security Proofs and Cryptanalysis*. Springer, Feb. 2017. ISBN: 978-3-319-49530-9 (cit. on p. 17).

[351] Mridul Nandi. "The Characterization of Luby-Rackoff and Its Optimum Single-Key Variants". In: *Progress in Cryptology - INDOCRYPT 2010: 11th International Conference in Cryptology in India*. Ed. by Guang Gong and Kishan Chand Gupta. Vol. 6498. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2010, pp. 82–97 (cit. on p. 17).

[352] Moni Naor and Omer Reingold. "On the Construction of Pseudo-Random Permutations: Luby-Rackoff Revisited (Extended Abstract)". In: *29th Annual ACM Symposium on Theory of Computing*. ACM Press, May 1997, pp. 189–199. DOI: 10.1145/258533.258581 (cit. on p. 17).

[353] Moni Naor and Omer Reingold. "On the Construction of Pseudorandom Permutations: Luby-Rackoff Revisited". In: *Journal of Cryptology* 12.1 (Jan. 1999), pp. 29–66. DOI: 10.1007/PL00003817 (cit. on p. 17).

[354] Joseph I. Naus. "An extension of the birthday problem". In: *The American Statistician* 22.1 (Feb. 1968). http://www.jstor.org/stable/2681879, pp. 27–29 (cit. on p. 45).

[355] Samuel Neves and Filipe Araujo. “An observation on NORX, BLAKE2, and ChaCha”. In: *Information Processing Letters* 149 (2019), pp. 1–5. ISSN: 0020-0190. DOI: https://doi.org/10.1016/j.ipl.2019.05.001 (cit. on p. 7).

[356] Samuel Neves and Filipe Araujo. “Engineering Nonlinear Pseudorandom Number Generators”. In: *Parallel Processing and Applied Mathematics - 10th International Conference, PPAM 2013, Warsaw, Poland, September 8-11, 2013, Revised Selected Papers, Part I*. Ed. by Roman Wyrzykowski, Jack Dongarra, Konrad Karczewski, and Jerzy Wasniewski. Vol. 8384. Lecture Notes in Computer Science. Springer, 2013, pp. 96–105. URL: http://dx.doi.org/10.1007/978-3-642-55224-3_10 (cit. on p. 6).

[357] Samuel Neves and Filipe Araujo. “Fast and Small Nonlinear Pseudorandom Number Generators for Computer Simulation”. In: *Parallel Processing and Applied Mathematics - 9th International Conference, PPAM 2011, Torun, Poland, September 11-14, 2011. Revised Selected Papers, Part I*. Ed. by Roman Wyrzykowski, Jack Dongarra, Konrad Karczewski, and Jerzy Wasniewski. Vol. 7203. Lecture Notes in Computer Science. Springer, 2011, pp. 92–101. URL: http://dx.doi.org/10.1007/978-3-642-31464-3_10 (cit. on pp. 6, 49).

[358] Samuel Neves and Jean-Philippe Aumasson. “BLAKE and 256-bit advanced vector extensions”. In: *The Third SHA-3 Candidate Conference*. Mar. 2012. URL: https://eden.dei.uc.pt/~sneves/pubs/2012-snjpa1.pdf (cit. on pp. 66, 69).

[359] Samuel Neves and Jean-Philippe Aumasson. *Implementing BLAKE with AVX, AVX2, and XOP*. Cryptology ePrint Archive, Report 2012/275. 2012. URL: http://eprint.iacr.org/2012/275 (cit. on pp. 66, 68, 69).

[360] Aina Niemetz, Mathias Preiner, and Armin Biere. “Boolector 2.0”. In: *JSAT* 9 (2014), pp. 53–58. URL: https://satassociation.org/jsat/index.php/jsat/article/view/120 (cit. on p. 116).

[361] Aina Niemetz, Mathias Preiner, and Armin Biere. “Boolector 2.0”. In: *JSAT* 9 (2015), pp. 53–58. URL: https://satassociation.org/jsat/index.php/jsat/article/view/120 (cit. on p. 109).

[362] Aina Niemetz, Mathias Preiner, Clifford Wolf, and Armin Biere. “Btor2 , BtorMC and Boolector 3.0”. In: *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*. Ed. by Hana Chockler and Georg Weissenbacher. Vol. 10981. Lecture Notes in Computer Science. Springer, 2018, pp. 587–595. ISBN: 978-3-319-96144-6. URL: https://doi.org/10.1007/978-3-319-96145-3%5C_32 (cit. on pp. 109, 116).

[363] Auguste Kerckhoffs (von Nieuwenhof). “La Cryptographie Militaire”. French. In: *Journal des Sciences Militaires* IX (Jan. 1883) (cit. on p. 27).

[364] NIST. *SHA-3 Competition*. http://csrc.nist.gov/groups/ST/hash/sha-3/Round3/index.html. 2007 (cit. on p. 171).

[365] Kaisa Nyberg. “Generalized Feistel Networks”. In: *Advances in Cryptology – ASIACRYPT’96*. Ed. by Kwangjo Kim and Tsutomu Matsumoto. Vol. 1163. Lecture Notes in Computer Science. Springer, Heidelberg, Nov. 1996, pp. 91–104. DOI: 10.1007/BFb0034838 (cit. on p. 17).

[366] Kaisa Nyberg. “Linear Approximation of Block Ciphers (Rump Session)”. In: *Advances in Cryptology – EUROCRYPT’94*. Ed. by Alfredo De Santis. Vol. 950. Lecture Notes in Computer Science. Springer, Heidelberg, May 1995, pp. 439–444. DOI: 10.1007/BFb0053460 (cit. on p. 36).

[367] Luke O’Connor and Jovan Dj. Golic. “A Unified Markow Approach to Differential and Linear Cryptanalysis”. In: *Advances in Cryptology – ASIACRYPT’94*. Ed. by Josef Pieprzyk and Reihaneh Safavi-Naini. Vol. 917. Lecture Notes in Computer Science. Springer, Heidelberg, Nov. 1995, pp. 387–397. DOI: 10.1007/BFb0000450 (cit. on pp. 33, 36).

[368] Jacques Patarin. “How to Construct Pseudorandom and Super Pseudorandom Permutations from one Single Pseudorandom Function”. In: *Advances in Cryptology – EUROCRYPT’92*. Ed. by Rainer A. Rueppel. Vol. 658. Lecture Notes in Computer Science. Springer, Heidelberg, May 1993, pp. 256–266. DOI: 10.1007/3-540-47555-9_22 (cit. on p. 17).

[369] Jacques Patarin. “Luby-Rackoff: 7 Rounds Are Enough for 2n(1-epsilon)Security”. In: *Advances in Cryptology – CRYPTO 2003*. Ed. by Dan Boneh. Vol. 2729. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2003, pp. 513–529. DOI: 10.1007/978-3-540-45146-4_30 (cit. on p. 17).

[370] Jacques Patarin. “New Results on Pseudorandom Permutation Generators Based on the DES Scheme”. In: *Advances in Cryptology – CRYPTO’91*. Ed. by Joan Feigenbaum. Vol. 576. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 1992, pp. 301–312. DOI: 10.1007/3-540-46766-1_25 (cit. on p. 17).

[371] Jacques Patarin. “On Linear Systems of Equations with Distinct Variables and Small Block Size”. In: *ICISC 05: 8th International Conference on Information Security and Cryptology*. Ed. by Dongho Won and Seungjoo Kim. Vol. 3935. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2006, pp. 299–321 (cit. on p. 17).

[372] Jacques Patarin. “Pseudorandom Permutations Based on the D.E.S. Scheme”. In: *ESORICS'90: 1st European Symposium on Research in Computer Security*. Lecture Notes in Computer Science. AFCET, Oct. 1990, pp. 185–187 (cit. on p. 17).

[373] Jacques Patarin. *Security of balanced and unbalanced Feistel Schemes with Linear Non Equalities*. Cryptology ePrint Archive, Report 2010/293. https://eprint.iacr.org/2010/293. 2010 (cit. on p. 17).

[374] Jacques Patarin. “Security of Random Feistel Schemes with 5 or More Rounds”. In: *Advances in Cryptology – CRYPTO 2004*. Ed. by Matthew Franklin. Vol. 3152. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2004, pp. 106–122. DOI: 10.1007/978-3-540-28628-8_7 (cit. on p. 17).

[375] Sarvar Patel, Zulfikar Ramzan, and Ganapathy S. Sundaram. “Towards Making Luby-Rackoff Ciphers Optimal and Practical”. In: *Fast Software Encryption – FSE'99*. Ed. by Lars R. Knudsen. Vol. 1636. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 1999, pp. 171–185. DOI: 10.1007/3-540-48519-8_13 (cit. on p. 17).

[376] Kenneth G. Paterson and Nadhem J. AlFardan. “Plaintext-Recovery Attacks Against Datagram TLS”. In: *ISOC Network and Distributed System Security Symposium – NDSS 2012*. The Internet Society, Feb. 2012 (cit. on p. 22).

[377] Kenneth G. Paterson, Thomas Ristenpart, and Thomas Shrimpton. “Tag Size Does Matter: Attacks and Proofs for the TLS Record Protocol”. In: *Advances in Cryptology – ASIACRYPT 2011*. Ed. by Dong Hoon Lee and Xiaoyun Wang. Vol. 7073. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2011, pp. 372–389. DOI: 10.1007/978-3-642-25385-0_20 (cit. on p. 22).

[378] Kenneth G. Paterson and Mario Strefler. “A Practical Attack Against the Use of RC4 in the HIVE Hidden Volume Encryption System”. In: *ASIACCS 15: 10th ACM Symposium on Information, Computer and Communications Security*. Ed. by Feng Bao, Steven Miller, Jianying Zhou, and Gail-Joon Ahn. ACM Press, Apr. 2015, pp. 475–482 (cit. on p. 2).

[379] Y. N. Patt, W. M. Hwu, and M. Shebanow. “HPS, a New Microarchitecture: Rationale and Introduction”. In: *ACM SIGMICRO Newsletter* 16.4 (Dec. 1985), pp. 103–108. ISSN: 1050-916X. URL: http://doi.acm.org/10.1145/18906.18916 (cit. on p. 3).

[380] Goutam Paul and Subhamoy Maitra. *RC4 Stream Cipher and Its Variants*. 1st ed. USA: CRC Press, Inc., 2011. ISBN: 1439831351 (cit. on p. 15).

[381] K. Pawlikowski, H.-D.J. Jeong, and J.-S. Ruth Lee. "On Credibility of Simulation Studies of Telecommunication Networks". In: *IEEE Communications Magazine* (Jan. 2002), pp. 132–139. URL: http://www.cosc.canterbury.ac.nz/krys.pawlikowski/publications/credibility.0101.pdf (cit. on pp. 15, 40).

[382] Trevor Perrin. *The Noise Protocol Framework*. 2016. URL: https://noiseprotocol.org/noise.html (cit. on p. 147).

[383] Raphael C.-W. Phan. "Extending commutative diagram cryptanalysis to slide, boomerang, rectangle and square attacks". In: *Comput. Stand. Interfaces* 29.4 (2007), pp. 444–448. URL: https://doi.org/10.1016/j.csi.2006.08.001 (cit. on p. 37).

[384] Raphael C.-W. Phan and Mohammad Umar Siddiqi. "A Framework for Describing Block Cipher Cryptanalysis". In: *IEEE Trans. Computers* 55.11 (2006), pp. 1402–1409. URL: https://doi.org/10.1109/TC.2006.169 (cit. on p. 37).

[385] Josef Pieprzyk. "How to Construct Pseudorandom Permutations from Single Pseudorandom Functions". In: *Advances in Cryptology – EUROCRYPT'90*. Ed. by Ivan Damgård. Vol. 473. Lecture Notes in Computer Science. Springer, Heidelberg, May 1991, pp. 140–150. DOI: 10.1007/3-540-46877-3_12 (cit. on p. 17).

[386] Josef Pieprzyk and Leonid Tombak. *Soviet Encryption Algorithm*. Tech. rep. 94-10. Department of Computer Science, The University of Wollongong, June 1994. URL: http://ftp.arnes.si/packages2/crypto-papers/GOST/tr-94-10.ps.gz (cit. on p. 16).

[387] Daniel Pollack. "HSS: A Simple File Storage System for Web Applications". In: *26th Large Installation System Administration Conference (LISA '12)*. 2012 (cit. on p. 51).

[388] J.B. Porta. *De furtivis literarum notis, vulgo de ziferis, libri IV.* Joh. Wolphius, 1591. URL: http://books.google.com/books?id=DcI9AAAAcAAJ (cit. on p. 1).

[389] Bart Preneel. "The First 30 Years of Cryptographic Hash Functions and the NIST SHA-3 Competition (Invited Talk)". In: *Topics in Cryptology – CT-RSA 2010*. Ed. by Josef Pieprzyk. Vol. 5985. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2010, pp. 1–14. DOI: 10.1007/978-3-642-11925-5_1 (cit. on p. 52).

[390] Bart Preneel, René Govaerts, and Joos Vandewalle. "Differential Cryptanalysis of Hash Functions Based on Block Ciphers". In: *ACM CCS 93: 1st Conference on Computer and Communications Security*. Ed. by Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby. ACM Press, Nov. 1993, pp. 183–188. DOI: 10.1145/168588.168611 (cit. on p. 32).

[391] Michael O. Rabin. "Digitalized signatures". In: *Foundations of secure computation*. Ed. by Richard A. DeMillo, David P. Dobkin, Anita K. Jones, and Richard J. Lipton. New York: Academic Press, 1978, pp. 155–168. ISBN: 0–12–210350–5. URL: http://hdl.handle.net/1853/40598 (cit. on p. 25).

[392] James Reeds. ""Cracking" a Random Number Generator". In: *Cryptologia* 1.1 (1977), pp. 20–26. URL: http://alumni.cs.ucr.edu/~jsun/random-number.pdf (cit. on p. 15).

[393] Charles T. Retter. "A Key-search Attack on Maclaren-Marsaglia Systems". In: *Cryptologia* 9.2 (1985), pp. 114–130. URL: http://dx.doi.org/10.1080/0161-118591859834 (cit. on p. 15).

[394] Charles T. Retter. "Cryptanalysis of a Maclaren-Marsaglia System". In: *Cryptologia* 8.2 (May 1984), pp. 97–103. URL: http://dx.doi.org/10.1080/0161-118491858854 (cit. on p. 15).

[395] Ronald L. Rivest. *RFC 1321: The MD5 Message-Digest Algorithm*. Internet Activities Board. Apr. 1992 (cit. on p. 25).

[396] Ronald L. Rivest. "The MD4 Message Digest Algorithm". In: *Advances in Cryptology – CRYPTO'90*. Ed. by Alfred J. Menezes and Scott A. Vanstone. Vol. 537. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 1991, pp. 303–311. DOI: 10.1007/3-540-38424-3_22 (cit. on p. 25).

[397] Ronald L. Rivest. "The RC5 Encryption Algorithm". In: *Fast Software Encryption – FSE'94*. Ed. by Bart Preneel. Vol. 1008. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 1995, pp. 86–96. DOI: 10.1007/3-540-60590-8_7 (cit. on p. 5).

[398] Ronald L. Rivest, M. J. B. Robshaw, R. Sidney, and Y. L. Yin. *The RC6 Block Cipher*. Posted on the RC6 site of RSA Laboratories. Aug. 1998. URL: https://people.csail.mit.edu/rivest/pubs/RRSY98.pdf (cit. on p. 5).

[399] Mariia Rodinko and Roman Oliynykov. "An Approach to Search for Multi-Round Differential Characteristics of Cypress-256". In: *2018 International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T)*. 2018, pp. 659–662. DOI: 10.1109/INFOCOMMST.2018.8631904 (cit. on p. 130).

[400] Mariia Rodinko and Roman Oliynykov. “Post-quantum lightweight symmetric block cipher “Cypress””. In: *Radiotekhnika* 2.189 (June 2017), pp. 100–107. URL: http://rt.nure.ua/article/view/183814 (cit. on p. 128).

[401] Mariia Rodinko and Roman Oliynykov. “The Method of Searching for Differential Trails of ARX-based Block Cipher Cypress”. In: *11th IEEE International Conference on Dependable Systems, Services and Technologies, DESSERT 2020, Kyiv, Ukraine, May 14-18, 2020*. IEEE, 2020, pp. 157–160. ISBN: 978-1-7281-9957-3. URL: https://doi.org/10.1109/DESSERT50317.2020.9125071 (cit. on p. 130).

[402] Mariia Rodinko, Roman Oliynykov, and Roman Eliseev. “Search for one-round differential characteristics of lighweight block cipher Cypress-256”. In: *2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT)*. 2018, pp. 312–315. DOI: 10.1109/DESSERT.2018.8409150 (cit. on p. 130).

[403] Phillip Rogaway. “Authenticated-Encryption With Associated-Data”. In: *ACM CCS 2002: 9th Conference on Computer and Communications Security*. Ed. by Vijayalakshmi Atluri. ACM Press, Nov. 2002, pp. 98–107. DOI: 10.1145/586110.586125 (cit. on pp. 22, 78).

[404] Phillip Rogaway. “Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC”. In: *Advances in Cryptology – ASIACRYPT 2004*. Ed. by Pil Joong Lee. Vol. 3329. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2004, pp. 16–31. DOI: 10.1007/978-3-540-30539-2_2 (cit. on p. 78).

[405] Phillip Rogaway. “Practice-Oriented Provable Security and the Social Construction of Cryptography”. In: *IEEE Security & Privacy* 14.6 (2016), pp. 10–17. URL: http://dx.doi.org/10.1109/MSP.2016.122 (cit. on pp. 14, 27).

[406] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. “OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption”. In: *ACM CCS 2001: 8th Conference on Computer and Communications Security*. Ed. by Michael K. Reiter and Pierangela Samarati. ACM Press, Nov. 2001, pp. 196–205. DOI: 10.1145/501983.502011 (cit. on p. 22).

[407] Phillip Rogaway and Thomas Shrimpton. “Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance”. In: *Fast Software Encryption – FSE 2004*. Ed. by Bimal K. Roy and Willi Meier. Vol. 3017. Lecture Notes in Computer Science. Springer, Heidelberg, Feb. 2004, pp. 371–388. DOI: 10.1007/978-3-540-25937-4_24 (cit. on p. 23).

[408] Alexander Roshal. *RAR 5.0 archive format*. Apr. 2013. URL: http://www.rarlab.com/technote.htm (cit. on p. 147).

[409] Karl Rupp. *Microprocessor Trend Data*. 2022. URL: https://github.com/karlrupp/microprocessor-trend-data (cit. on p. 3).

[410] M-J. Saarinen and J-P. Aumasson. *The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)*. RFC 7693 (Informational). Internet Engineering Task Force, Nov. 2015. URL: http://www.ietf.org/rfc/rfc7693.txt (cit. on p. 147).

[411] Markku-Juhani O. Saarinen. "Beyond Modes: Building a Secure Record Protocol from a Cryptographic Sponge Permutation". In: *Topics in Cryptology – CT-RSA 2014*. Ed. by Josh Benaloh. Vol. 8366. Lecture Notes in Computer Science. Springer, Heidelberg, Feb. 2014, pp. 270–285. DOI: 10.1007/978-3-319-04852-9_14 (cit. on p. 26).

[412] Babak Sadeghiyan and Josef Pieprzyk. "A Construction for Super Pseudorandom Permutations from A Single Pseudorandom Function". In: *Advances in Cryptology – EUROCRYPT'92*. Ed. by Rainer A. Rueppel. Vol. 658. Lecture Notes in Computer Science. Springer, Heidelberg, May 1993, pp. 267–284. DOI: 10.1007/3-540-47555-9_23 (cit. on p. 17).

[413] Bruce Schneier and John Kelsey. "Unbalanced Feistel Networks and Block Cipher Design". In: *Fast Software Encryption – FSE'96*. Ed. by Dieter Gollmann. Vol. 1039. Lecture Notes in Computer Science. Springer, Heidelberg, Feb. 1996, pp. 121–144. DOI: 10.1007/3-540-60865-6_49 (cit. on p. 17).

[414] M. Schoo, K. Pawlikowski, and D.C. McNickle. *A Survey and Empirical Comparison of Modern Pseudo-Random Number Generators for Distributed Stochastic Simulations*. Tech. rep. Department of Computer Science and Software Development, University of Canterbury, 2005 (cit. on p. 40).

[415] Ernst Schulte-Geers. "On CCZ-equivalence of addition mod 2 n". In: *Des. Codes Cryptogr.* 66.1-3 (2013), pp. 111–127. URL: https://doi.org/10.1007/s10623-012-9668-4 (cit. on p. 121).

[416] Pouyan Sepehrdad, Serge Vaudenay, and Martin Vuagnoux. "Discovery and Exploitation of New Biases in RC4". In: *SAC 2010: 17th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Alex Biryukov, Guang Gong, and Douglas R. Stinson. Vol. 6544. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2011, pp. 74–91. DOI: 10.1007/978-3-642-19574-7_5 (cit. on p. 136).

[417] *SHA-3 Competition*. http://csrc.nist.gov/groups/ST/hash/sha-3/Round3/index.html. 2007 (cit. on p. 93).

[418] *Secure Hash Standard*. National Institute of Standards and Technology, NIST FIPS PUB 180-1, U.S. Department of Commerce. Apr. 1995 (cit. on p. 25).

[419] Adi Shamir. "On the Security of DES". In: *Advances in Cryptology – CRYPTO'85*. Ed. by Hugh C. Williams. Vol. 218. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 1986, pp. 280–281. DOI: 10.1007/3-540-39799-X_22 (cit. on p. 34).

[420] Adi Shamir and Boaz Tsaban. "Guaranteeing the diversity of number generators". In: *Information and Computation* 171.2 (Jan. 2002), pp. 350–363. ISSN: 0890-5401. URL: http://dx.doi.org/10.1006/inco.2001.3045 (cit. on p. 48).

[421] Claude Shannon. *A Mathematical Theory of Cryptography*. Memorandum MM 45-110-02. Classified report. Superseded by [422]. Murray Hill, NJ, USA: Bell Laboratories, Sept. 1945, pp. 114 + 25 (cit. on pp. 1, 12, 15, 197).

[422] Claude E. Shannon. "Communication Theory of Secrecy Systems". In: *The Bell System Technical Journal* 28.4 (Oct. 1949). A footnote on the initial page says: "The material in this paper appeared in a confidential report, 'A Mathematical Theory of Cryptography', dated Sept. 1, 1945 ([421]), which has now been declassified."., pp. 656–715. URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6769090 (cit. on pp. 12, 15, 37, 197).

[423] Zhenqing Shi, Bin Zhang, Dengguo Feng, and Wenling Wu. "Improved Key Recovery Attacks on Reduced-Round Salsa20 and ChaCha". In: *ICISC 12: 15th International Conference on Information Security and Cryptology*. Ed. by Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon. Vol. 7839. Lecture Notes in Computer Science. Springer, Heidelberg, Nov. 2013, pp. 337–351. DOI: 10.1007/978-3-642-37682-5_24 (cit. on pp. 79, 98).

[424] Akihiro Shimizu and Shoji Miyaguchi. "Fast Data Encipherment Algorithm FEAL". In: *Advances in Cryptology – EUROCRYPT'87*. Ed. by David Chaum and Wyn L. Price. Vol. 304. Lecture Notes in Computer Science. Springer, Heidelberg, Apr. 1988, pp. 267–278. DOI: 10.1007/3-540-39118-5_24 (cit. on pp. 6, 34).

[425] Victor Shoup. *A Computational Introduction to Number Theory and Algebra*. 2nd ed. http://shoup.net/ntb. Cambridge University Press, 2009, pp. I–XVI, 1–517. ISBN: 9780521516440 (cit. on p. 123).

[426] Marcos A. Simplício Jr., Leonardo C. Almeida, Ewerton R. Andrade, Paulo C. F. dos Santos, and Paulo S. L. M. Barreto. *Lyra2: Password Hashing Scheme with improved security against time-memory trade-offs*. Cryptology ePrint Archive, Report 2015/136. https://eprint.iacr.org/2015/136. 2015 (cit. on p. 147).

[427] R. Slipetskyy. “Security Issues in OpenStack”. MA thesis. Norwegian University of Science and Technology, 2011 (cit. on p. 51).

[428] J. E. Smith and G. S. Sohi. “The microarchitecture of superscalar processors”. In: *Proceedings of the IEEE* 83.12 (Dec. 1995), pp. 1609–1624. ISSN: 0018-9219. DOI: 10.1109/5.476078 (cit. on p. 3).

[429] J. L. Smith. *The Design of Lucifer, A Cryptographic Device for Data Communications*. RC 3326. IBM Research, 1971 (cit. on p. 16).

[430] Standardization Administration of China. *Control Network LONWORKS Technology Specification — Part 1: Protocol Specification*. GB/Z 20177.1-2006. 2006 (cit. on pp. 133, 144).

[431] Marc Stevens. *Fast Collision Attack on MD5*. Cryptology ePrint Archive, Report 2006/104. https://eprint.iacr.org/2006/104. 2006 (cit. on p. 2).

[432] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. “The First Collision for Full SHA-1”. In: *Advances in Cryptology – CRYPTO 2017, Part I*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10401. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2017, pp. 570–596. DOI: 10.1007/978-3-319-63688-7_19 (cit. on pp. 2, 34).

[433] Marc Stevens, Arjen K. Lenstra, and Benne de Weger. “Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities”. In: *Advances in Cryptology – EUROCRYPT 2007*. Ed. by Moni Naor. Vol. 4515. Lecture Notes in Computer Science. Springer, Heidelberg, May 2007, pp. 1–22. DOI: 10.1007/978-3-540-72540-4_1 (cit. on p. 2).

[434] Marc Stevens, Alexander Sotirov, Jacob Appelbaum, Arjen K. Lenstra, David Molnar, Dag Arne Osvik, and Benne de Weger. “Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate”. In: *Advances in Cryptology – CRYPTO 2009*. Ed. by Shai Halevi. Vol. 5677. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2009, pp. 55–69. DOI: 10.1007/978-3-642-03356-8_4 (cit. on pp. 2, 51).

[435] Bozhan Su, Wenling Wu, Shuang Wu, and Le Dong. “Near-Collisions on the Reduced-Round Compression Functions of Skein and BLAKE”. In: *CANS 10: 9th International Conference on Cryptology and Network Security*. Ed. by Swee-Huay Heng, Rebecca N. Wright, and Bok-Min Goi. Vol. 6467. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2010, pp. 124–139 (cit. on pp. 72, 73).

[436] Makoto Sugita, Mitsuru Kawazoe, Ludovic Perret, and Hideki Imai. "Algebraic Cryptanalysis of 58-Round SHA-1". In: *Fast Software Encryption – FSE 2007*. Ed. by Alex Biryukov. Vol. 4593. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2007, pp. 349–365. DOI: 10.1007/978-3-540-74619-5_22 (cit. on p. 38).

[437] Anne Tardy-Corfdir and Henri Gilbert. "A Known Plaintext Attack of FEAL-4 and FEAL-6". In: *Advances in Cryptology – CRYPTO'91*. Ed. by Joan Feigenbaum. Vol. 576. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 1992, pp. 172–181. DOI: 10.1007/3-540-46766-1_12 (cit. on p. 34).

[438] Erik Tews, Ralf-Philipp Weinmann, and Andrei Pyshkin. "Breaking 104 Bit WEP in Less Than 60 Seconds". In: *Information Security Applications, 8th International Workshop, WISA 2007, Jeju Island, Korea, August 27-29, 2007, Revised Selected Papers*. Ed. by Sehun Kim, Moti Yung, and Hyung-Woo Lee. Vol. 4867. Lecture Notes in Computer Science. Springer, 2007, pp. 188–202. ISBN: 978-3-540-77534-8. URL: https://doi.org/10.1007/978-3-540-77535-5%5C_14 (cit. on pp. 77, 136).

[439] Tyge Tiessen. "Polytopic Cryptanalysis". In: *Advances in Cryptology – EUROCRYPT 2016, Part I*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9665. Lecture Notes in Computer Science. Springer, Heidelberg, May 2016, pp. 214–239. DOI: 10.1007/978-3-662-49890-3_9 (cit. on p. 36).

[440] G. S. Tjaden and M. J. Flynn. "Detection and Parallel Execution of Independent Instructions". In: *IEEE Transactions on Computers* 19.10 (Oct. 1970), pp. 889–895. ISSN: 0018-9340. URL: http://dx.doi.org/10.1109/T-C.1970.222795 (cit. on p. 3).

[441] R. M. Tomasulo. "An Efficient Algorithm for Exploiting Multiple Arithmetic Units". In: *IBM Journal of Research and Development* 11.1 (Jan. 1967), pp. 25–33. ISSN: 0018-8646. URL: http://dx.doi.org/10.1147/rd.111.0025 (cit. on p. 3).

[442] Johannes Trithemius. *Polygraphiae Libri Sex*. Oppenheim, 1518. URL: http://reader.digitale-sammlungen.de/resolve/display/bsb11057927.html (cit. on p. 1).

[443] Eran Tromer, Dag Arne Osvik, and Adi Shamir. "Efficient Cache Attacks on AES, and Countermeasures". In: *Journal of Cryptology* 23.1 (Jan. 2010), pp. 62–74. DOI: 10.1007/s00145-009-9049-y (cit. on p. 5).

[444] Yukiyasu Tsunoo, Teruo Saito, Hiroyasu Kubo, Tomoyasu Suzaki, and Hiroki Nakashima. "Differential Cryptanalysis of Salsa20/8". In: *The State of the Art of Stream Ciphers (SASC)*. 2007 (cit. on p. 98).

[445] D. M. Tullsen, S. J. Eggers, and H. M. Levy. “Simultaneous multithreading: Maximizing on-chip parallelism”. In: *Computer Architecture, 1995. Proceedings., 22nd Annual International Symposium on*. June 1995, pp. 392–403 (cit. on p. 3).

[446] Stanley Tzeng and Li-Yi Wei. “Parallel white noise generation on a GPU via cryptographic hash”. In: *Proceedings of the 2008 symposium on Interactive 3D graphics and games*. I3D ’08. Redwood City, California: ACM, 2008, pp. 79–87. ISBN: 978-1-59593-983-8. URL: http://doi.acm.org/10.1145/1342250.1342263 (cit. on p. 41).

[447] International Telecommunication Union. *Measuring digital development*. Tech. rep. 2021. URL: https://www.itu.int/en/ITU-D/Statistics/Pages/facts/default.aspx (cit. on p. 1).

[448] Paul C. van Oorschot and Michael J. Wiener. “Parallel Collision Search with Cryptanalytic Applications”. In: *Journal of Cryptology* 12.1 (Jan. 1999), pp. 1–28. DOI: 10.1007/PL00003816 (cit. on pp. 29, 30).

[449] S. R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar. “An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS”. In: *Solid-State Circuits, IEEE Journal of* 43.1 (2008), pp. 29–41. URL: http://dx.doi.org/10.1109/JSSC.2007.910957 (cit. on p. 39).

[450] Mathy Vanhoef and Frank Piessens. “All Your Biases Belong to Us: Breaking RC4 in WPA-TKIP and TLS”. In: *USENIX Security 2015: 24th USENIX Security Symposium*. Ed. by Jaeyeon Jung and Thorsten Holz. USENIX Association, Aug. 2015, pp. 97–112 (cit. on p. 2).

[451] Serge Vaudenay. “An Experiment on DES Statistical Cryptanalysis”. In: *ACM CCS 96: 3rd Conference on Computer and Communications Security*. Ed. by Li Gong and Jacques Stern. ACM Press, Mar. 1996, pp. 139–147. DOI: 10.1145/238168.238206 (cit. on p. 37).

[452] Serge Vaudenay. “Decorrelation: A Theory for Block Cipher Security”. In: *Journal of Cryptology* 16.4 (Sept. 2003), pp. 249–286. DOI: 10.1007/s00145-003-0220-6 (cit. on pp. 31, 37).

[453] Serge Vaudenay. “On the Lai-Massey Scheme”. In: *Advances in Cryptology – ASIACRYPT’99*. Ed. by Kwok-Yan Lam, Eiji Okamoto, and Chaoping Xing. Vol. 1716. Lecture Notes in Computer Science. Springer, Heidelberg, Nov. 1999, pp. 8–19. DOI: 10.1007/978-3-540-48000-6_2 (cit. on p. 17).

[454] Serge Vaudenay. "On the Security of CS-Cipher". In: *Fast Software Encryption – FSE'99*. Ed. by Lars R. Knudsen. Vol. 1636. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 1999, pp. 260–274. DOI: 10.1007/3-540-48519-8_19 (cit. on pp. 33, 36).

[455] Serge Vaudenay. "Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS..." In: *Advances in Cryptology – EUROCRYPT 2002*. Ed. by Lars R. Knudsen. Vol. 2332. Lecture Notes in Computer Science. Springer, Heidelberg, Apr. 2002, pp. 534–546. DOI: 10.1007/3-540-46035-7_35 (cit. on pp. 20, 22, 77).

[456] Serge Vaudenay and Martin Vuagnoux. "Passive-Only Key Recovery Attacks on RC4". In: *SAC 2007: 14th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Carlisle M. Adams, Ali Miri, and Michael J. Wiener. Vol. 4876. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2007, pp. 344–359. DOI: 10.1007/978-3-540-77360-3_22 (cit. on p. 136).

[457] G. S. Vernam. "Cipher Printing Telegraph Systems For Secret Wire and Radio Telegraphic Communications". In: *American Institute of Electrical Engineers, Transactions of the* XLV (Jan. 1926), pp. 295–301. ISSN: 0096-3860 (cit. on p. 13).

[458] Janos Vidali, Peter Nose, and Enes Pasalic. "Collisions for variants of the BLAKE hash function". In: *Inf. Process. Lett.* 110.14-15 (2010), pp. 585–590. URL: https://doi.org/10.1016/j.ipl.2010.05.007 (cit. on p. 67).

[459] Michael Vielhaber. *Breaking ONE.FIVIUM by AIDA an Algebraic IV Differential Attack*. Cryptology ePrint Archive, Report 2007/413. https://eprint.iacr.org/2007/413. 2007 (cit. on p. 36).

[460] Sebastiano Vigna. "On the probability of overlap of random subsequences of pseudorandom number generators". In: *Inf. Process. Lett.* 158 (2020), p. 105939. URL: https://doi.org/10.1016/j.ipl.2020.105939 (cit. on p. 46).

[461] John von Neumann. "Various techniques used in connection with random digits". In: *Monte Carlo Method*. Ed. by A. S. Householder, G. E. Forsythe, and H. H. Germond. Vol. 12. National Bureau of Standards Applied Mathematics Series. Washington, D.C.: U.S. Government Printing Office, 1951, pp. 36–38 (cit. on p. 4).

[462] David Wagner. "The Boomerang Attack". In: *Fast Software Encryption – FSE'99*. Ed. by Lars R. Knudsen. Vol. 1636. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 1999, pp. 156–170. DOI: 10.1007/3-540-48519-8_12 (cit. on p. 36).

[463] David Wagner. “Towards a Unifying View of Block Cipher Cryptanalysis”. In: *Fast Software Encryption – FSE 2004*. Ed. by Bimal K. Roy and Willi Meier. Vol. 3017. Lecture Notes in Computer Science. Springer, Heidelberg, Feb. 2004, pp. 16–33. DOI: 10.1007/978-3-540-25937-4_2 (cit. on p. 37).

[464] John Walker. *A Pseudorandom Number Sequence Test Program*. http://www.fourmilab.ch/random/. Jan. 2008 (cit. on p. 50).

[465] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. “Finding Collisions in the Full SHA-1”. In: *Advances in Cryptology – CRYPTO 2005*. Ed. by Victor Shoup. Vol. 3621. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2005, pp. 17–36. DOI: 10.1007/11535218_2 (cit. on pp. 2, 34, 51).

[466] Xiaoyun Wang and Hongbo Yu. “How to Break MD5 and Other Hash Functions”. In: *Advances in Cryptology – EUROCRYPT 2005*. Ed. by Ronald Cramer. Vol. 3494. Lecture Notes in Computer Science. Springer, Heidelberg, May 2005, pp. 19–35. DOI: 10.1007/11426639_2 (cit. on pp. 2, 34, 51).

[467] Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. “Efficient Collision Search Attacks on SHA-0”. In: *Advances in Cryptology – CRYPTO 2005*. Ed. by Victor Shoup. Vol. 3621. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2005, pp. 1–16. DOI: 10.1007/11535218_1 (cit. on pp. 34, 51).

[468] Mark N. Wegman and Larry Carter. “New Hash Functions and Their Use in Authentication and Set Equality”. In: *Journal of Computer and System Sciences* 22 (1981), pp. 265–279 (cit. on p. 21).

[469] Gordon Welchman. *The Hut Six Story: Breaking the Enigma Codes*. 2nd ed. Classic Crypto Books, Oct. 1997. ISBN: 978-0947712341 (cit. on p. 1).

[470] David J. Wheeler and Roger M. Needham. “TEA, a Tiny Encryption Algorithm”. In: *Fast Software Encryption – FSE’94*. Ed. by Bart Preneel. Vol. 1008. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 1995, pp. 363–366. DOI: 10.1007/3-540-60590-8_29 (cit. on p. 6).

[471] Michael J. Wiener. “The Full Cost of Cryptanalytic Attacks”. In: *Journal of Cryptology* 17.2 (Mar. 2004), pp. 105–124. DOI: 10.1007/s00145-003-0213-5 (cit. on p. 28).

[472] John Wilkins. *Mercury, or the Secret and Swift Messenger. Shewing, how a man may with privacy and speed communicate his thoughts to a friend at any distance*. London: I. Norton, 1641, p. 172. URL: https://archive.org/details/gu_mercuryorthes00wilk (cit. on p. 1).

[473] Robert S. Winternitz and Martin E. Hellman. “Chosen-Key Attacks on a Block Cipher”. In: *Cryptologia* 11.1 (1987), pp. 16–20. URL: http://dx.doi.org/10.1080/0161-118791861749 (cit. on p. 27).

[474] Aaram Yun, Je Hong Park, and Jooyoung Lee. "On Lai-Massey and quasi-Feistel ciphers". In: *Des. Codes Cryptography* 58.1 (2011), pp. 45–72. URL: http://dx.doi.org/10.1007/s10623-010-9386-8 (cit. on p. 17).

[475] I. A. Zabotin, G. P. Glazkov, and V. B. Isaeva. *Cryptographic Protection for Data Processing Systems: Cryptographic Transformation Algorithm (GOST 28147-89)*. Translated from Russian by Aleksandr Malchik, Sun Microsystems Laboratories, Mountain View, California, with editorial and typographic assistance from Whitfield Diffie, Sun Microsystems, Mountain View, California. 1989. URL: https://github.com/mjosaarinen/gost-r34.11-94 (cit. on p. 16).

[476] Fahad Zafar and Marc Olano. "Tiny encryption algorithm for parallel random numbers on the GPU". In: *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*. I3D '10. Washington, D.C.: ACM, 2010, 1:1–1:1. ISBN: 978-1-60558-939-8. URL: http://doi.acm.org/10.1145/1730971.1730973 (cit. on p. 49).

[477] Fahad Zafar, Marc Olano, and Aaron Curtis. "GPU random numbers via the tiny encryption algorithm". In: *Proceedings of the Conference on High Performance Graphics*. HPG '10. Saarbrucken, Germany: Eurographics Association, 2010, pp. 133–141. URL: http://portal.acm.org/citation.cfm?id=1921479.1921500 (cit. on pp. 41, 44, 49).

[478] Xin-jie Zhao, Tao Wang, and Yuan-yuan Zheng. *Cache Timing Attacks on Camellia Block Cipher*. Cryptology ePrint Archive, Report 2009/354. https://eprint.iacr.org/2009/354. 2009 (cit. on p. 5).

[479] Yuliang Zheng, Tsutomu Matsumoto, and Hideki Imai. "Impossibility and Optimality Results on Constructing Pseudorandom Permutations (Extended Abstract)". In: *Advances in Cryptology – EUROCRYPT'89*. Ed. by Jean-Jacques Quisquater and Joos Vandewalle. Vol. 434. Lecture Notes in Computer Science. Springer, Heidelberg, Apr. 1990, pp. 412–422. DOI: 10.1007/3-540-46885-4_41 (cit. on p. 17).

[480] Yuliang Zheng, Tsutomu Matsumoto, and Hideki Imai. "On the Construction of Block Ciphers Provably Secure and Not Relying on Any Unproved Hypotheses". In: *Advances in Cryptology – CRYPTO'89*. Ed. by Gilles Brassard. Vol. 435. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 1990, pp. 461–480. DOI: 10.1007/0-387-34805-0_42 (cit. on p. 17).

[481] Thilo Zieschang. "Combinatorial Properties of Basic Encryption Operations (Extended Abstract)". In: *Advances in Cryptology – EUROCRYPT'97*. Ed. by Walter Fumy. Vol. 1233. Lecture Notes in Computer Science. Springer, Heidelberg, May 1997, pp. 14–26. DOI: 10.1007/3-540-69053-0_2 (cit. on p. 6).

# Appendix A

# Appendix to Chapter 6

## A.1 Automated Search Sample Code

```
from z3 import *

W = 64 # word size
TARGET_D = 0 # target weight

def get_new_variable():
  if not hasattr(get_new_variable, "counter"):
    get_new_variable.counter = 0  # it doesn't exist yet, so initialize it
  t = BitVec("y%u" % get_new_variable.counter, W)
  get_new_variable.counter += 1
  return t

def hw(x):
  return Sum([ ZeroExt(W-1, Extract(i,i,x)) for i in range(W) ])

def valid_and(a, b, c):
  return (~(a|b) & c) == 0

def prob_and(a, b, c):
  return hw(a|b)

def make_and(a, b):
  global S
  global D
  t = get_new_variable()
  S.add( valid_and(a, b, t) )
  D += prob_and(a, b, t)
  return t

def H(a, b):
  return make_and(a << 1, b << 1) ^ a ^ b;

def ROTR(x, c):
  return Concat(Extract(c-1,0,x), Extract(W-1, c, x))

def G(a, b, c, d):
  a = H(a, b); d ^= a; d = ROTR(d,  8)
```

```
  c = H(c, d); b ^= c; b = ROTR(b, 19)
  a = H(a, b); d ^= a; d = ROTR(d, 40)
  c = H(c, d); b ^= c; b = ROTR(b, 63)
  return a, b, c, d

S = Solver()
D = BitVecVal(0, W)

X = [ BitVec(f'x{i}', W) for i in range(4) ]
S.add(Or([ x != 0 for x in X ]))
Z = G(*X)
S.add([ BitVec(f'z{i}', W) == Z[i] for i in range(4) ])

S.add(ULE(D, TARGET_D))
print(S.to_smt2())
```

## A.2 Selected Differentials

### A.2.1 Experimental Verification of Automated Search

The first table shows the results from our verification of trails found using automated search, see Section 6.2.4. Notation is used as follows. $w_e$: expected weight, $\#S$: number of samples, $v_e$: expected value of input/output pairs adhering the differential, $v_m$: measured value of input/output pairs adhering the differential, $w_m$: measured weight. After that we list the differentials in 32- and 64-bit $F^{1.5}$ that we used to perform the verification.

| | | | NORX32 | | | NORX64 | | |
|---|---|---|---|---|---|---|---|---|
| $w_e$ | $\#S$ | $v_e$ | $v_m$ | $v_m - v_e$ | $w_m$ | $v_m$ | $v_m - v_e$ | $w_m$ |
| 12 | $2^{28}$ | 65536 | 65652 | +116 | 11.997 | 65627 | +91 | 11.997 |
| 13 | $2^{29}$ | 65536 | 65788 | +252 | 12.994 | 65584 | +48 | 12.998 |
| 14 | $2^{30}$ | 65536 | 65170 | −366 | 14.008 | 65476 | −60 | 14.001 |
| 15 | $2^{31}$ | 65536 | 65441 | −95 | 15.002 | 65515 | −21 | 15.000 |
| 16 | $2^{32}$ | 65536 | 65683 | +147 | 15.996 | 65563 | +27 | 15.999 |
| 17 | $2^{33}$ | 65536 | 65296 | −240 | 17.005 | 65608 | +72 | 16.998 |
| 18 | $2^{34}$ | 65536 | 65389 | −147 | 18.003 | 65565 | +29 | 17.999 |

| $\delta_0$ | | | | $\delta_1$ | | | | $w$ |
|---|---|---|---|---|---|---|---|---|
| 00000000 | 00000400 | 80000080 | 80000000 | 00000000 | 00000000 | 00000000 | 80001000 | 12 |
| 00000000 | 80000400 | 80000080 | 00000000 | 00000000 | 00000000 | 00000000 | 21012100 | |
| 00000000 | 80000000 | 80808080 | 80000000 | 00000000 | 00000000 | 00000000 | 10808080 | |
| 00000000 | 80000000 | 80800000 | 80000080 | 00000000 | 00000000 | 00000000 | 10008080 | |
| 80000000 | 00000000 | 00000400 | 80000180 | 80001000 | 00000000 | 00000000 | 00000000 | 13 |
| 00000000 | 00000000 | 80000400 | 80000080 | 21012100 | 00000000 | 00000000 | 00000000 | |
| 80000000 | 00000000 | 80000000 | 80808080 | 10808080 | 00000000 | 00000000 | 00000000 | |
| 80000080 | 00000000 | 80000000 | 80800000 | 10008080 | 00000000 | 00000000 | 00000000 | |
| 80000080 | 80000000 | 00000000 | 00000400 | 00000000 | 80001000 | 00000000 | 00000000 | 14 |
| 80000180 | 00000000 | 00000000 | 80000400 | 00000000 | 21012100 | 00000000 | 00000000 | |
| 80808080 | 80000000 | 00000000 | 80000000 | 00000000 | 10808080 | 00000000 | 00000000 | |
| 80800000 | 80000080 | 00000000 | 80000000 | 00000000 | 10008080 | 00000000 | 00000000 | |
| 00000400 | 80000000 | 00000400 | 40100000 | 00100000 | 00000000 | 00000000 | 00000000 | 15 |
| 80000400 | 80000000 | 00000000 | 00100200 | 00200021 | 00000000 | 00000000 | 00000000 | |
| 80000000 | 80018000 | 00000400 | 00000000 | 80000010 | 00000000 | 00000000 | 00000000 | |
| 80000000 | 00800000 | 00040400 | 40000600 | 00000010 | 00000000 | 00000000 | 00000000 | |
| 00000400 | 80000080 | 80000000 | 00000000 | 00000000 | 00000000 | 80003000 | 00000000 | 16 |
| 80000400 | 80000080 | 00000000 | 00000000 | 00000000 | 00000000 | 63016100 | 00000000 | |
| 80000000 | 81808080 | 80000000 | 00000000 | 00000000 | 00000000 | 31808080 | 00000000 | |
| 80000000 | 80800000 | 80000080 | 00000000 | 00000000 | 00000000 | 30008080 | 00000000 | |
| 00000000 | 00000400 | 80000080 | 80000000 | 00000000 | 00000000 | 00000000 | 80001000 | 17 |
| 00000000 | 80000400 | 80000080 | 00000000 | 00000000 | 00000000 | 00000000 | 21012100 | |
| 00000000 | 80000000 | 80838780 | 80000000 | 00000000 | 00000000 | 00000000 | 10808080 | |
| 00000000 | 80000000 | 80800000 | 80000080 | 00000000 | 00000000 | 00000000 | 10008080 | |
| 00000400 | 00000000 | 80000000 | C0000200 | 00100000 | 00000000 | 00000000 | 00606001 | 18 |
| 80000400 | 00000000 | 00000000 | 00000200 | 00200021 | 00000000 | 00000000 | C24242C0 | |
| 80000000 | 00000000 | 80000000 | 00000000 | 80000010 | 00000000 | 00000000 | 61010160 | |
| 80000000 | 00000000 | 80000080 | C0000000 | 00000010 | 00000000 | 00000000 | 60010160 | |

| $\delta_0$ | | | | $\delta_1$ | | | | $w$ |
|---|---|---|---|---|---|---|---|---|
| 8000000000000000 | 0000000000000000 | 0000000000040000 | 8000000000000080 | 8000001000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 12 |
| 0000000000000000 | 0000000000000000 | 8000000000040000 | 8000000000000080 | 2100002001010000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | |
| 8000000000000000 | 0000000000000000 | 8000000000000000 | 8000808000000080 | 1080000000808000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | |
| 8000000000000080 | 0000000000000000 | 8000000000000000 | 0080800000000000 | 1000000000808000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | |
| 4000001000000000 | 0000000000040000 | 8000000000000000 | 0000000000040000 | 0000000000000000 | 0000100000000000 | 0000000000000000 | 0000000000000000 | 13 |
| 0000001000020000 | 8000000000040000 | 8000000000000000 | 0000000000000000 | 0000000000000000 | 0000200000000021 | 0000000000000000 | 0000000000000000 | |
| 0000000000000000 | 8000000000000000 | 8000008000000000 | 0000000000040000 | 0000000000000000 | 8000000000000010 | 0000000000000000 | 0000000000000000 | |
| 4000000000020000 | 8000000000000000 | 0000800000000000 | 0000000004040000 | 0000000000000000 | 0000000000000010 | 0000000000000000 | 0000000000000000 | |
| 0000000000040000 | 8000000000000080 | 8000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 8000001000000000 | 0000000000000000 | 14 |
| 8000000000040000 | 8000000000000080 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 2100002001010000 | 0000000000000000 | |
| 8000000000000000 | 8003808000000080 | 8000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 1080000000808000 | 0000000000000000 | |
| 8000000000000000 | 0080800000000000 | 8000000000000080 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 1000000000808000 | 0000000000000000 | |
| 0000000000000000 | 00000000000C0000 | 8000000000000080 | 8000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 8000001000000000 | 15 |
| 0000000000000000 | 8000000000040000 | 8000000000000080 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 2300006001010000 | |
| 0000000000000000 | 8000000000000000 | 8000808000000080 | 8000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 1180000000808000 | |
| 0000000000000000 | 8000000000000000 | 0080800000000000 | 8000000000000080 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 1000000000808000 | |
| 0000000000040000 | 4000001000080000 | 0000000000040000 | 8000000000000000 | 0000000000000000 | 0000000000000000 | 0000100000000000 | 0000000000000000 | 16 |
| 0000000000000000 | 0000001000020000 | 8000000000040000 | 8000000000000000 | 0000000000000000 | 0000000000000000 | 0000200000000021 | 0000000000000000 | |
| 0000000000040000 | 0000000000000000 | 8000000000000000 | 8000008000000000 | 0000000000000000 | 0000000000000000 | 8000000000000010 | 0000000000000000 | |
| 0000000004040000 | C0000000000E0000 | 8000000000000000 | 0000800000000000 | 0000000000000000 | 0000000000000000 | 0000000000000010 | 0000000000000000 | |
| 8000000000000080 | 8000000000000000 | 0000000000000000 | 0000000000040000 | 0000000000000000 | 8000007000000000 | 0000000000000000 | 0000000000000000 | 17 |
| 8000000000000080 | 0000000000000000 | 0000000000000000 | 8000000000040000 | 0000000000000000 | E300006001010000 | 0000000000000000 | 0000000000000000 | |
| 8000808000000180 | 8000000000000000 | 0000000000000000 | 8000000000000000 | 0000000000000000 | 7180000000808000 | 0000000000000000 | 0000000000000000 | |
| 0080800000000000 | 8000000000000080 | 0000000000000000 | 8000000000000000 | 0000000000000000 | 7000000000808000 | 0000000000000000 | 0000000000000000 | |
| 0000000000040000 | 8000000000000000 | 0000000000040000 | 400000F000000000 | 0000100000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 18 |
| 8000000000040000 | 8000000000000000 | 0000000000000000 | 0000001000020000 | 0000200000000021 | 0000000000000000 | 0000000000000000 | 0000000000000000 | |
| 8000000000000000 | 8000008000000000 | 0000000000040000 | 0000000000000000 | 8000000000000010 | 0000000000000000 | 0000000000000000 | 0000000000000000 | |
| 8000000000000000 | 0000800000000000 | 000000000C040000 | 4000000000020000 | 0000000000000010 | 0000000000000000 | 0000000000000000 | 0000000000000000 | |

### A.2.2 Probability-1 Differentials in G

Using automated search we could show that there are exactly 3 probability-1 differentials in both versions (32- and 64-bit) of G.

| | Differences | | | |
|---|---|---|---|---|
| $\delta_0$ | 80000000 | 80000000 | 80000000 | 00000000 |
| $\delta_1$ | 00000000 | 00000001 | 80000000 | 00000000 |
| $\delta_0$ | 80000000 | 00000000 | 80000000 | 80000080 |
| $\delta_1$ | 80000000 | 00000000 | 00000000 | 00000000 |
| $\delta_0$ | 00000000 | 80000000 | 00000000 | 80000080 |
| $\delta_1$ | 80000000 | 00000001 | 80000000 | 00000000 |

| | Differences | | | |
|---|---|---|---|---|
| $\delta_0$ | 8000000000000000 | 8000000000000000 | 8000000000000000 | 0000000000000000 |
| $\delta_1$ | 0000000000000000 | 0000000000000001 | 8000000000000000 | 0000000000000000 |
| $\delta_0$ | 8000000000000000 | 0000000000000000 | 8000000000000000 | 8000000000000080 |
| $\delta_1$ | 8000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 |
| $\delta_0$ | 0000000000000000 | 8000000000000000 | 0000000000000000 | 8000000000000080 |
| $\delta_1$ | 8000000000000000 | 0000000000000001 | 8000000000000000 | 0000000000000000 |

### A.2.3 Best Differential Characteristics for $F^4$

The following two tables show the best differential trail in $F^4$ that we were able to find. The values $\delta_0$ and $\delta_4$ are in- and output difference, respectively, and $\delta_1$, $\delta_2$, and $\delta_3$ are internal differences. The differences are listed after a single application of F, respectively, and the values $w_i$, with $i \in \{0, \ldots, 3\}$, are the corresponding differential weights.

| $\delta_0$ | | | | $w_0$ | $\delta_1$ | | | | $w_1$ |
|---|---|---|---|---|---|---|---|---|---|
| 80140100 | 90024294 | 84246020 | 92800154 | 172 | 40100000 | 00000400 | 80000000 | 00000400 | 11 |
| e4548300 | 52240214 | e0202424 | d0004054 | | 00100200 | 80000400 | 80000000 | 00000000 | |
| c4464046 | 00a08480 | c1008108 | 90d43134 | | 00000000 | 80000000 | 80008000 | 00000400 | |
| e200c684 | e2eac480 | a4848881 | 06915342 | | 40000200 | 80000000 | 00800000 | 00040400 | |
| $\delta_2$ | | | | $w_2$ | $\delta_3$ | | | | $w_3$ |
| 00000000 | 00000000 | 00000000 | 00000000 | 44 | 04042425 | 00100002 | 00020000 | 02100000 | 357 |
| 00000000 | 00000000 | 00000000 | 00000000 | | 04200401 | 42024200 | 20042024 | 20042004 | |
| 00000000 | 80000000 | 00000000 | 00000000 | | 10001002 | 80000200 | 25250504 | 10021010 | |
| 00000000 | 00000000 | 00000000 | 00000000 | | 10020010 | 00001002 | 00000210 | 04252504 | |
| $\delta_4$ | | | | | | | | | |
| c4001963 | 804da817 | 0c05b60e | 12220503 | total weight: 584 | | | | | |
| 9072b909 | 185b792a | cc0d56cd | 7e0ac646 | | | | | | |
| 80116300 | 100c2800 | 8f003320 | 3b270222 | | | | | | |
| 01056104 | 88000041 | 92002824 | 04210001 | | | | | | |

| $\delta_0$ | | | | $w_0$ | $\delta_1$ | | | | $w_1$ |
|---|---|---|---|---|---|---|---|---|---|
| 00900824010288c5 | 4000443880011086 | 224012044220ac43 | e004044484049520 | 349 | 8000000800050000 | 8000000000000000 | 4000000000000000 | 0000001000020080 | 27 |
| 4080882001010885 | 4600841880821086 | a3c0721444632c43 | c224440007849504 | | 8000000800040000 | 8000000000000000 | c000000000040000 | 8000001000020080 | |
| 81600850830b0484 | 840080c080868000 | 8004449040c14400 | 8102101840908a80 | | 0000000000000000 | 8000008000000000 | c000004000040000 | 4000808000020080 | |
| 6191548c08000581 | 0200004006038044 | 8104f01c8702c0e0 | 60605084938886e3 | | 0000000000010080 | 0000800000000000 | 8000400004040000 | 80808000020000c0 | |
| $\delta_2$ | | | | $w_2$ | $\delta_3$ | | | | $w_3$ |
| 8000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 12 | 0000000000000000 | 0000000000000000 | 0000100000000000 | 0000202000000001 | 448 |
| 8000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | | 4200404002020040 | 0000000000000000 | 0000000000000000 | 0000200000000021 | |
| 8000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | | 8000000000000010 | 2100000001010020 | 0000000000000000 | 0000000000000000 | |
| 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | | 0000000000000000 | 0000000000000010 | 2000000001010020 | 0000000000000000 | |
| $\delta_4$ | | | | | | | | | |
| 321a4500060e4e2e | 27404405026e500e | 3806422387200a08 | 8c40f4a0884c0820 | total weight: 836 | | | | | |
| 71540fb858cb9902 | ee018cc282747980 | c714164174ce3eb9 | 1a49a091101191e1 | | | | | | |
| 786680d0e46406cb | 14440844013274e6 | 03a843203f071b7c | 09a840c00c0ccc78 | | | | | | |
| 4000404a22120005 | 07220c4202016240 | 2aa4200a0a041a62 | 84a468682000601c | | | | | | |

## A.2.4 Best Iterative Differentials for F

| | Differences | | | | $w$ |
|---|---|---|---|---|---|
| $\delta_0 = \delta_1$ | 818c959b | 00186049 | eb5b7984 | 791c6da1 | 512 |
| | 677b513d | 80000400 | 00000227 | 5293655f | |
| | 00809a2b | bfa98bff | c08b8e89 | 0000711c | |
| | 800027c3 | f984eb5b | 6d81f915 | b5aaa99d | |

| | Differences | | | | $w$ |
|---|---|---|---|---|---|
| $\delta_0 = \delta_1$ | 0000000100000000 | 0000000000000000 | f77c78b200000d04 | 0000000000000000 | 843 |
| | be7fffeffe0f349f | 0000000000000000 | 6c07fbd200000001 | ff1ab5be4e7500be | |
| | 0060c54927018000 | 0000000000000000 | 0000000000000000 | b603fde900000000 | |
| | b6035caf00000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | |

### A.2.5 Best Differentials having Equal Columns of weight 44 in F

| | Differences | | | |
|---|---|---|---|---|
| $\delta_0$ | 80000000 | 80000000 | 80000000 | 80000000 |
| | 80000000 | 80000000 | 80000000 | 80000000 |
| | 80000000 | 80000000 | 80000000 | 80000000 |
| | 00000000 | 00000000 | 00000000 | 00000000 |
| $\delta_1$ | 00102001 | 00102001 | 00102001 | 00102001 |
| | 42624221 | 42624221 | 42624221 | 42624221 |
| | a1010110 | a1010110 | a1010110 | a1010110 |
| | 20010110 | 20010110 | 20010110 | 20010110 |

| | Differences | | | |
|---|---|---|---|---|
| $\delta_0$ | 8000000000000000 | 8000000000000000 | 8000000000000000 | 8000000000000000 |
| | 8000000000000000 | 8000000000000000 | 8000000000000000 | 8000000000000000 |
| | 8000000000000000 | 8000000000000000 | 8000000000000000 | 8000000000000000 |
| | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 |
| $\delta_1$ | 0000102000000001 | 0000102000000001 | 0000102000000001 | 0000102000000001 |
| | 4200604002020021 | 4200604002020021 | 4200604002020021 | 4200604002020021 |
| | a100000001010010 | a100000001010010 | a100000001010010 | a100000001010010 |
| | 2000000001010010 | 2000000001010010 | 2000000001010010 | 2000000001010010 |

# Appendix B

# Appendix to Chapter 7

```
import os

def ROT8(x, c):
  return ((x%256 << c%8) | (x%256 >> -c%8)) % 256

def OMADigest(m,k):
  a = [0] * 8
  m = m[:] + [0] * (-len(m) % 144)
  for l in range(0, len(m), 144):
    b = m[l:l+144]
    for i in range(18):
      for j in range(7, -1, -1):
        if (k[i%12] >> (7 - j)) & 1:
          a[j] = (a[(j+1)%8] + b[8*i+7-j] + ROT8(~(a[j] + j), 1)) % 256
        else:
          a[j] = (a[(j+1)%8] + b[8*i+7-j] - ROT8(~(a[j] + j), -1)) % 256
  return a

def EN14908(r, m, k):
  mlen, a = len(m) - 1, r[:]
  while True:
    for i in range(6):
      for j in range(7, -1, -1):
        b = 0 if mlen < 0 else m[mlen]
        mlen -= 1
        if k[i] & (1 << (7 - j)):
          a[j] = a[(j+1)%8] + b + ROT8(~(a[j] + j), 1)
        else:
          a[j] = a[(j+1)%8] + b - ROT8(~(a[j] + j), -1)
    if mlen < 0:
      break
  return a

def RC4Encrypt(X,key):
    def RC4(key, b):
      B,S,i,j,l=[],range(256),0,0,len(key)
      while i < 256:
          j = (j + S[i] + key[i%l]) & 0xff
          S[i], S[j] = S[j], S[i]
          i += 1
      i, j = 1, 0
      while b:
          t = S[i]
          j = (j + S[i]) & 0xff
          S[i], S[j] = S[j], S[i]
```

```
            B += [S[(S[i]+S[j]) & 0xff]]
            b -= 1
            i = (i + 1) & 0xff
        return B
    S = RC4(key,len(X))
    for i in xrange(len(X)):
        X[i] ^= S[i]
    return X

def OSGPKeyDerive(k):
  k1 = EN14908([0x81, 0x3f, 0x52, 0x9a, 0x7b, 0xe3, 0x89, 0xba], [], k)
  k2 = EN14908([0x72, 0xb0, 0x91, 0x8d, 0x44, 0x05, 0xaa, 0x57], [], k)
  return k1 + k2

def OSGPEncrypt(m, k):
  k_ = OSGPKeyDerive(k)
  a  = OMADigest(m, k)
  for i in range(8):
    k_[i] ^= a[i]
  return RC4Encrypt(m, k_) + a

def OSGPDecrypt(c, k):
  assert(len(c) >= 8)
  k_ = k_ = OSGPKeyDerive(k)
  a  = c[-8:]
  for i in range(8):
    k_[i] ^= a[i]
  m = RC4Encrypt(c[:-8], k_)
  return OMADigest(m, k) == a, m

# Test vector
m = [0x02,0x02,0x00,0x30,0x00,
     0x03,0x7f,0x30,0xea,0x6d,
     0x00,0x00,0x00,0x0d,0x00,
     0x20,0x98,0x00,0x31,0xc3,
     0x00,0x08,0x00,0x00,0x00,
     0x00,0x00,0x11]
k = [0xDF] * 12
a = [0xdb, 0xe5, 0xcd, 0xe5, 0x07, 0xb1, 0xcb, 0x3d]
assert(OMADigest(m, k) == a)

def OMABackward(a,m,k,n):
  a, m = a[:], m[:] + [0] * (-len(m) % 144)
  for l in range(n):
    i, j = l // 8, l % 8
    if (k[(17-i)%12] >> (7 - j)) & 1:
      x = ROT8(a[j] - a[(j+1)%8] - m[143-8*i-j], -1)
    else:
      x = ROT8(a[(j+1)%8] + m[143-8*i-j] - a[j], 1)
    a[j] = (~x - j) % 256
  return a

def OMAForward(a,m,k,n):
  a, m = a[:], m[:] + [0] * (-len(m) % 144)
  for l in range(n, 144):
    i, j = l // 8, 7 - l % 8
    if (k[i%12] >> (7 - j)) & 1:
      a[j] = (a[(j+1)%8] + m[8*i+7-j] + ROT8(~(a[j] + j), 1)) % 256
    else:
      a[j] = (a[(j+1)%8] + m[8*i+7-j] - ROT8(~(a[j] + j), -1)) % 256
  return a

m = map(ord, os.urandom(144))
k = map(ord, os.urandom(12))
a = OMADigest(m, k)
```

```
assert( OMAForward([0]*8, m, k, 0) == OMADigest(m,k) )
assert( OMAForward(OMABackward(a,m,k,8),m,k,144-8) == a )

def TagGenOracle(m,init=[True]):
    if init[0]:
      print '[ORACLE] k = ' + str(k)
      init[0] = False
    return OMADigest(m,k)

def TagCheckOracle(m,a):
  return TagGenOracle(m) == a

def OSGPEncryptOracle(m, init=[True]):
  return OSGPEncrypt(m, k)

def OSGPCheckOracle(c):
  ok, _ = OSGPDecrypt(c, k);
  return ok

def Algorithm_74():
  m = map(ord, os.urandom(144))
  a = TagGenOracle(m)
  k = [0] * 12
  for i in range(12):
    for j in range(8):
      m_ = m[:]
      m_[136-8*i-j-1] ^= 0x80
      a_ = TagGenOracle(m_)
      b  = OMABackward(a,m,k,8*i)
      b_ = OMABackward(a_,m_,k,8*i)
      k[(17-i)%12] |= ((b[j] ^ b_[j])&1) << (7 - j)
  return k

print 'Algorithm 7.4: ' + str(Algorithm_74())

def Algorithm_75():
  def RecoverByte(a, b):
    x = (a[7] ^ b[7]) & 1
    for i in xrange(0,7):
        x |= ((a[6-i] ^ b[6-i] ^ a[7-i] ^ b[7-i]) & 1) << (i+1)
    return x
  k = [0] * 12
  m = map(ord, os.urandom(144))
  a = TagGenOracle(m)
  for i in range(12):
    m_ = m[:]
    m_[136-8*i-8] ^= 0x80
    a_ = TagGenOracle(m_)
    b  = OMABackward(a,m,k,8*i)
    b_ = OMABackward(a_,m_,k,8*i)
    k[(17-i)%12] = RecoverByte(b, b_)
  return k

print 'Algorithm 7.5: ' + str(Algorithm_75())

def Algorithm_76():
  def recurse(m,a,k,i=0):
    if i >= 12:
      a_ = OMADigest(m,k)
      return a_ == a
    m_ = m[:]
    m_[128-8*i:] = map(ord, os.urandom(144-(128-8*i)))
    a_ = TagGenOracle(m_)
    for x in range(256):
      k[(17-i)%12] = x
```

```
      b = OMABackward(a, m, k, 8*i + 16)
      b_ = OMAForward(b, m_, k, 128 - 8*i)
      if a_ == b_ and recurse(m,a,k,i+1):
          return True
    return False
  k = [0] * 12
  m = map(ord, os.urandom(144))
  a = TagGenOracle(m)
  recurse(m,a,k)
  return k

print 'Algorithm 7.6: ' + str(Algorithm_76())

def Algorithm_77():
  k = [0] * 12
  c = OSGPEncryptOracle(map(ord, os.urandom(96+8)))
  for i in range(96):
    c_ = c[:]
    c_[i+0] ^= 0x80
    c_[i+1] ^= 0x80
    c_[i+8] ^= 0xC0
    if OSGPCheckOracle(c_):
      continue
    c_[i+8] = c[i+8] ^ 0x40
    k[((i+8)//8)%12] |= (0 if OSGPCheckOracle(c_) else 1) << ((i+8)%8)
  return k

print 'Algorithm 7.7: ' + str(Algorithm_77())

# Key-recovery attack from Section 7.2.4, using additive differences
def Algorithm_78():
  k = [0] * 12
  m = map(ord, os.urandom(96+8))
  a = TagGenOracle(m)
  for i in range(96):
    m_ = m[:]
    m_[i+0] = (m[i+0] + 0x02) % 256
    m_[i+1] = (m[i+1] - 0x02) % 256
    m_[i+8] = (m[i+8] - 0x01) % 256
    if TagCheckOracle(m_, a): continue
    m_[i+8] = (m[i+8] - 0xfc) % 256
    if TagCheckOracle(m_, a):
      k[((i+8)//8)%12] |= 1 << ((i+8)%8)
      continue
    m_[i+8] = (m[i+8] - 0x81) % 256
    k[((i+8)//8)%12] |= (0 if TagCheckOracle(m_, a) else 1) << ((i+8)%8)
  return k

print 'Algorithm 7.8: ' + str(Algorithm_78())
```