Designing and Improving Code-based Cryptosystems

Vom Fachbereich Informatik der
Technischen Universität Darmstadt genehmigte

Dissertation

zur Erlangung des Grades
Doktor rerum naturalium (Dr. rer. nat.)

von

Dipl.-Math. Mohammed Meziani

geboren in Beni Sadden, Marokko.

Referenten:	Prof. Dr. Johannes Buchmann Dr. Pierre-Louis Cayrel Prof. Dr. Ayoub Otmani
Tag der Einreichung:	12.05.2014
Tag der mündlichen Prüfung:	03.06.2013
Hochschulkennziffer:	D 17

Darmstadt 2014

Wissenschaftlicher Werdegang

November 2008 - 2012

Promotionsstudent am Lehrstuhl von Prof. Dr. Johannes Buchmann, Fachbereich Informatik, Fachgebiet Theoretische Informatik - Kryptographie und Computeralgebra -, und Mitarbeiter im Projekt ”Kryptographische Primitives” des Arbeitsbereichs ”Sichere Daten” im Center for Advanced Security Research Darmstadt (CASED).

April 2002 - Oktober 2007

Studium der Mathematik mit Schwerpunkt Informatik an der Technischen Universität Darmstadt.

Oktober 1992 - Oktober 2000

Studium der Angewandten Mathematik mit Schwerpunkt Statistik an der Sidi Mohammed Ben Abdellah-Universität-Fes, Marokko.

Acknowledgement

I sincerely thank Allah, my God, the Most Gracious, Most Merciful for enabling me to accomplish my Ph.D. successfully and for often putting so many good people in my way.

In completing my Ph.D. thesis I owe a great debt to many people. I wish to extend my deep thanks gratitude and appreciation to everyone contributed to the successful completion of my thesis.

First and foremost, I would like to express my deep sense of gratitude and thanks to my research supervisor Prof. Dr. Johannes Buchmann for his valuable advice, constructive criticism, patient guidance, encouragement and his extensive discussions around my work. I also must express my heartiest thanks to Dr. Pierre-Louis Cayrel for his sincere efforts, interest and time he have kindly spent to complete my thesis. I also gratefully thank Prof. Dr. Ayoub Otmani for his help and for agreeing to be my coreferee.

I also thank my closest friends, Sidi Mohamed El Yousfi Aloui, Mohamed Saied Emam Mohamed, Rachid El Bensarkhani, Sami Alsouri, and Özgür Dagdelen for their help and support. Special thanks go to Stanislav Bulygin, Gerhard Hoffmann, and Robert Niebuhr for their interesting and fruitful collaboration. Additionally, I would like to thank my CDC and CASED colleagues for creating a peaceful and beautiful office atmosphere. Also, I thank CASED for their financial support.

Finally, very special thanks go to my parents, brothers and sisters, who always supported me in all my pursuits; my wife for her inspirational patience and support; my little son Ibrahim, who kept me smiling during the last year of my PhD pursuit. Thank you !.

List of Publications

[PUB1] Mohammed Meziani and Rachid El Bansarkhani. An Efficient and Secure Coding-based Authenticated Encryption. In Roberto Di Pietro and Javier Herranz and Ernesto Damiani and Radu State editors, Data Privacy Management and Autonomous Spontaneous Security, 7th International Workshop, DPM 2012, and 5th International Workshop, SETOP 2012, volume 7731 of Lecture Notes in Computer Science, pages 43-60. Springer, 2013.

[PUB2] Rachid El Bansarkhani and Mohammed Meziani: "An Efficient Lattice-based Secret Sharing Construction. In Ioannis G. Askoxylakis and Henrich Christopher Pöhls and Joachim Posegga editors, Information Security Theory and Practice. Security, Privacy and Trust in Computing Systems and Ambient Intelligent Ecosystems- 6th IFIP WG 11.2 International Workshop, WISTP 2012, volume 7322 of Lecture Notes in Computer Science, pages 160-168, Springer, 2012.

[PUB3] Mohammed Meziani, Gerhard Hoffmann and Pierre-Louis Cayrel. Improving the Performance of the SYND Stream Cipher. In Aikaterini Mitrokotsa and Serge Vaudenay editors, AFRICACRYPT, volume 7374 of Lecture Notes in Computer Science, pages 99-116. Springer, 2012.

[PUB4] Robert Niebuhr, Mohammed Meziani, Stanislav Bulygin and Johannes Buchmann. Selecting paramaters for secure McEliece-based cryptosystems. In the International Journal of Information Security, volume 11, number 3, pages 137-147, Springer 2012.

[PUB5] Mohammed Meziani, Pierre-Louis Cayrel, Sidi Mohamed El Yousfi Alaoui. 2SC: An Efficient Code-Based Stream Cipher. In Tai-Hoon Kim and Hojjat Adeli and Rosslin John Robles and Maricel O. Balitanas editors, Information Security and Assurance - International Conference, ISA 2011, volume 200 of Communications in Computer and Information Science, pages 33-42, Springer, 2011.

[PUB6] Mohammed Meziani, Özgür Dagdelen, Pierre-Louis Cayrel, Sidi Mohamed El Yousfi Alaoui. S-FSB: An Improved Variant of the FSB Hash Family. In Tai-Hoon Kim and Hojjat Adeli and Rosslin John Robles and Maricel O. Balitanas editors, Information Security and Assurance - International Conference, ISA 2011, volume 200 of Communications in Computer and Information Science, pages 132–145, Springer, 2011.

[PUB7] Sidi Mohamed El Yousfi Alaoui, Pierre-Louis Cayrel, Mohammed Meziani. Improved Identity-Based Identification and Signature Schemes Using Quasi-Dyadic Goppa Codes. In Tai-Hoon Kim and Hojjat Adeli and Rosslin John Robles and Maricel O. Balitanas editors, Information

Security and Assurance - International Conference, ISA 2011, volume 200 of Communications in Computer and Information Science, pages 146–155, Springer, 2011.

[PUB8] Pierre-Louis Cayrel, Sidi Mohamed El Yousfi Alaoui, Gerhard Hoffmann, Mohammed Meziani, Robert Niebuhr. Recent Progress in Code-Based Cryptography. In Tai-Hoon Kim and Hojjat Adeli and Rosslin John Robles and Maricel O. Balitanas editors, Information Security and Assurance - International Conference, ISA 2011, volume 200 of Communications in Computer and Information Science, pages 21–32, Springer, 2011.

[PUB9] Mohammed Meziani, Sidi Mohamed El Yousfi Alaoui, Pierre-Louis Cayrel. Hash Functions Based on Coding Theory. In the proceedings of the 2nd Workshop on Codes, Cryptography and Communication Systems (WCCCS 2011), pages 32–37, June 2011.

[PUB10] Pierre-Louis Cayrel and Mohammed Meziani. Post-quantum Cryptography: Code-Based Signatures. In Tai-Hoon Kim and Hojjat Adeli editors, Advances in Computer Science and Information Technology, AST/UCMA/ISA/ACN 2010 Conferences, volume 6059 of Lecture Notes in Computer Science, pages 82–99. Springer, 2010.

[PUB11] Mohammed Meziani and Pierre-Louis Cayrel. A Muti-Signature based on Coding Theory". In the proceeding of the International Conference on Cryptography, Coding and Information Security (ICCCS 2010), volume 63 March 2010 ISSN 2070-3724, 2010.

Zusammenfassung

In der modernen Kryptographie basiert die Sicherheit der meisten beweisbar sicheren kryptographischen Primitiven auf schwierigen Problemen aus der Zahlentheorie wie beispielsweise das Faktorisierungs- und das diskrete Logarithmusproblem. Allerdings allein auf die Hartnäckigkeit dieser Probleme vertrauen scheint riskant zu sein. Im Jahr 1994 zeigte Peter Shor wie beide genannten Probleme in polynomieller Zeit (und somit effizient) mit Hilfe von Quantencomputern gelöst werden können.

Im Gegensatz dazu, sollen kryptographischen Primitive, welche auf Probleme aus der Kodierungstheorie basieren, gegen Quantuncomputerangriffe resistent sein und die uns heute bekannten Angriffe benötigen exponentielle Laufzeit. Neben der Post-Quantum Sicherheit bieten Code basierte Systeme weitere Vorteile fr die heutigen Anwendungen aufgrund ihrer hervorragenden algorithmischen Effizienz. In der Tat sind sie schneller als herkömmliche Kryptosysteme wie RSA, da sie nur sehr einfache Operationen wie Verschiebungen und XORs benötigen, anstatt den blichen teuren Berechnungen ber groe Zahlen. Trotz herausstechender Effizienz leiden die meisten Code basierten Systeme von zu groenSchlsselgren. Die Einführung von Codes mit algebraischer Struktur wie quasi-zyklische und quasi-dyadische Codes, half das Schlüsselgröenproblem zu berwinden, allerdings hat sich ergeben, dass sie anfällig auf algebraische Kryptoanalyse waren.

Diese Dissertation leistet einen Beitrag zur Forschung und Entwicklung von Code-basierten Kryptosysteme. Insbesondere interessieren wir uns fr die Entwicklung sowie die Verbesserung der drei wichtigen Primitive: Stromchiffren und Hash-Funktionen. Wir untersuchen die FSB Hashfunktion und die SYND Stromchiffre und zeigen wie deutlich ihre Effizienz verbessert werden kann, whrend die Sicherheitsreduktionen auf die gleichen NP-vollständigen Probleme erhalten und gültig bleiben.

Unabhängig von diesen Ergebnissen, adressieren und lösen wir das Problem der Auswahl geeigneter Parametern für den Goppa Code basierten McEliece Kryptosystem. Basierend auf dem Lenstra-Verheul Modell bieten wir auch, zum ersten Mal, ein Framework, das ermöglicht eine Auswahl von optimalen Parametern zu bestimmen, welche die gewünschte Sicherheitsstufe in einem bestimmten und konkreten Jahr erfüllt.

Abstract

In modern cryptography, the security of the most secure cryptographic primitives is based on hard problems coming from number theory such as the factorization and the discrete logarithm problem. However, being mainly based on the intractability of those problems seems to be risky. In 1994, Peter Shor showed how these two problems can be solved in polynomial time using a quantum computer.

In contrast, cryptographic primitives based on problems from coding theory are believed to resist quantum computer based attacks and the best known attacks have exponential running time. Along with post-quantum security, code-based systems offer other advantages for present-day applications due to their excellent algorithmic efficiency. Actually, they run faster than traditional cryptosystems like RSA, since they only require very simple operations like shifts and XORs instead of expensive computations over big integers. However, although efficient, most code-based schemes suffer from considerably large key sizes. Codes with algebraic structure such as quasi-cyclic and quasi-dyadic codes, were proposed to overcome the key size issue, but it has been shown to be insecure against algebraic cryptanalysis.

This thesis contributes to the research and development of code-based cryptosystems. In particular, we are interested in developing as well as improving three important primitives: stream ciphers and hash functions. We study the FSB hash function and the SYND stream cipher and find a way to considerably improve their efficiency, while maintaining the security reduction to the same NP-complete problems.

Independently of these results, we address and solve the problem of selecting appropriate parameter sets for the binary Goppa code-based McEliece cryptosystem. Based on the Lenstra-Verheul model, we also provide, for the first time, a framework allowing to choose optimal parameters that offer a desired security level in a given year.

Contents

Chapter 1

Introduction

Today cryptography is the cornerstone of information security. It is used for many applications such as electronic commerce, e-banking, computer password, etc. It can be divided into two families: *symmetric* (or private-key cryptography), and *asymmetric* (or public-key cryptography). In the first case, a single key is used for both encryption and decryption. Symmetric cryptography includes many cryptographic primitives such as stream ciphers and hash functions. In the asymmetric setting, two different keys are required: one is the public key, used to encrypt, and the other is the private key needed to decrypt and therefore must be kept secret.

Nowadays, many public-key systems are available; and the three most widely used are: RSA, Rabin and ElGamal. The security of RSA is based on the intractability of the integer factorization problem. In the Rabin system, the underlying problem is computationally equivalent to factoring. The security of ElGamal public-key cryptosystem is related to the hardness of the discrete logarithm problem. However, Shor [Sho94] proposed a quantum computer algorithm, which can solve these two problems in polynomial time. Therefore, all public-key cryptosystems resting on the difficulty of these two problems could be broken by a large quantum computer. For this reason, a significant amount of effort in the cryptographic community bas been devoted to design new cryptographic schemes which are resistant to future quantum computer algorithms. The schemes are known as post-quantum cryptographic primitives [BBD08]. Cryptosystems based on error correcting codes are one of the four promising alternatives [BBD08] that are believed to possess the potential to achieve this goal. In these systems, the problem that is used is drawn from coding theory, namely the problem of decoding general linear codes, which is known to be NP-hard [BMvT78]. The oldest-known schemes belonging to this group are the McEliece public-key system [McE78] and its dual Niederreiter's variant [Nie86]. These two systems are still secure, in the sense that no attack able to realize a total break in an acceptable time has been proposed up to date. All published algorithms have an exponential running time. In addition to post-quantum security, code-based cryptosystems have some advantages over conventional public-key cryptosystems, for example RSA. They possess fast encryption and decryption algorithm (for comparison see the benchmark data made available by eBATS benchmarking project [BL]). However, they also suffer from two major problems that seriously limit their practical usability: the public key size is quite large, and the transmission rate is low.

Results and outline of the thesis

This thesis contributes to designing three classes of code-based cryptosystems, namely stream-ciphers, hash functions, and authenticated encryptions scheme as well as to solving the problem of selecting secure parameters for the McEliece-like cryptosystems. After introducing the required notions of code-based cryptography for understanding this thesis in Chapter 2, we present the following results and discuss the main open problems as well as future research directions in the respective chapter.

Code-based Stream Ciphers (Chapter 3). In chapter 3, we first start with the description of all existing code-based stream ciphers that have been proposed in the literature, namely the Pseudo-random generator due to Fischer and Stern [FS96], and the SYND stream cipher proposed by Gaborit et al [GLS07]. Then, we present our three contributions in this area. The first contribution consists in designing a new code-based stream-cipher following the sponge construction, called 2SC, which stands for "Sponge Code-based Stream Cipher". This cipher runs faster than previous proposals, but suffers from the drawback of possessing large matrices. The second contribution consists in improving the SYND stream cipher in terms of speed by replacing the transformation used in SYND by a new one without loss of security reduction to the regular syndrome decoding problem. The new resulting cipher, called XSYND (eXtended SYND), performs all previous constructions in terms of performance in practice and is shown to be provably secure in the sense that if any adversary is able to distinguish the key stream produced by XSYND, he can solve a hard instance of the regular syndrome decoding problem. Futhermore, XSYND requires small storage capacity compared to 2SC. The last contribution is to show how to construct a parallel variant of XSYND, called Parallel SYND (in short PSYND) and hence obtaining a faster stream cipher, whose security is still based on the same problem as SYND and XSYND. At the time of writing of this thesis, we are not aware of any similar code-based stream cipher that has less storage requirements and a faster key stream generation.

Code-based Hash Functions (Chapter 4). This chapter deals with the design of hash functions based on coding theory. We first start by describing the Fast Syndrome Based hash family [AFS03, AFS05, FGS07] (in short FSB) and recalling its main features. Then, we present our main contribution, which consists in showing how to incorporate the ideas of FSB and the sponge construction due to Bertoni et al. [BDPA07] to design a variant of FSB hash function, called Sponge-FSB (in short SFSB). The security of this variant is based on the same problems as FSB, and outperforms FSB in terms of speed. Our experimental results show that our proposal is up to 30 % faster in practice than FSB using appropriate parameters.

Parameters Choice for the McEliece-like cryptosystems (Chapter 5). In this chapter, we address the problem of choosing optimal parameters for the McEliece cryptosystem that provide security until a given year and give detailed recommendations. Following the Lenstra-Verheul model, which uses a set of explicitly formulated parameter settings, combined with existing data points about future hardware and software developments, we propose parameters that provide the desired security level until a given year and optimize the key sizes.

Chapter 2

Preliminaries and Definitions

In this chapter, we will introduce the shared notations and some mathematical definitions for all following chapters. Furthermore, we recall a number of essential security notions and basic tools that are needed for understanding the rest of this thesis.

2.1. Mathematical notation

Scalar, Vectors, Sets, and Matrices. We write scalars using italic Roman lowercase letters (e.g. x) or, sometimes, italic Greek lowercase (e.g. α). Vectors are denoted by bold fonts (e.g. $\mathbf{x}$). The coefficients of a vector are noted using the same letter but with the bold removed. For instance, the ith coefficient of the vector $\mathbf{y}$ is written y_i. We use sans-serif fonts to denote matrices (e.g. A). The transpose of a matrix A (resp.of a vector $\mathbf{x}$) is denoted $\mathsf{A}^\top$ (resp. $\mathbf{x}^\top$). The coefficient located at the intersection of the ith row and the jth column of the matrix A is denoted $a_{i,j}$. We write sets using upper-case letters (e.g. S). We write usual sets of numbers using the blackboard font (e.g. $\mathbb{N}$ for the natural numbers). The explicit definition of a set is denoted using the curly brackets (e.g. $S = \{1,2,\ldots,10\} = \{i \,|\, i = 1,\ldots,10\}$). The length of the binary string $\mathbf{x}$ is denoted $|\mathbf{x}|$. A finite field consisting of q elements is written $\mathbb{F}_q$.

Operators. The symbol "$\|$" is used to indicate the concatenation operator. If $\mathbf{x}$ and $\mathbf{y}$ are two equal-sized vectors, then $\mathbf{x} \oplus \mathbf{y}$ denotes their bitwise XOR. We denote by "$\cdot$" the matrix-vector multiplication. The symbol "$\leftarrow$" is used for assignment, while "$=$" as well as "$\equiv$" for comparison. When using "$\equiv$", we mean equality modulo an equivalence relation (e.g. $a \equiv b \mod q$). The inner product of two bit vectors a and b (of the same size) is defined by $\langle a,b \rangle \equiv \sum_i a_i b_i \mod 2$.

Functions. Regarding functions, we use standard mathematical notations. That is, if f is a function that takes as input elements from a set X and outputs elements from a set Y, then we denote this by $f : X \to Y$. The set X is called the domain of f and Y the range of f.

Distributions. Let S be a finite set. The statement $x \xleftarrow{\$} S$ means that x is distributed uniformly over the set S. If D is a distribution, then we denote by $x \xleftarrow{\$} D$ the event that x is a random variable selected according to D. We denote by U_n the uniform distribution over the set $\mathbb{F}_2^n = \{0,1\}^n$. A function $\mu : \mathbb{N} \to [0,1]$ is called negligible if for all polynomials p, there exists some $n_0 > 1$ such that for all $n > n_0, \mu(n) < |\frac{1}{p(n)}|$.

2.2. Standard notions

Algorithms. Algorithms are written using lowercase calligraphic letters (e.g. $\mathcal{A}$). The execution of an algorithm $\mathcal{A}$ with inputs x to produce an output a is written as $a \leftarrow \mathcal{A}(x)$ or $\mathcal{A}(x) = a$. An algorithm is said to be efficient, if it runs in Probabilistic Polynomial Time (PPT).

Computational indistinguishability and pseudorandomness. Indistinguishability is a fundamental notion in complexity theory. It originates from [GM82] and was presented in a more general way in [Yao82a]. Informally, two probability distributions are computationally indistinguishable if no efficient algorithm (called the "distinguisher") can tell them apart better than with a negligible probability. Formally, this can be stated as follows. Let $(X_n)_n$ and $(Y_n)_n$ be sets of probability distributions, where X_n and Y_n are probability distributions over $\{0,1\}^{p(n)}$ for some polynomial $p(n)$. We say that $(X_n)_n$ and $(Y_n)_n$ are computationally indistinguishable if for all non-uniform PPT distinguisher $\mathcal{D}$, there exists a negligible function $\varepsilon(n)$ such that

$$\forall n \in \mathbb{N},\ \Pr[t \leftarrow X_n, \mathcal{D}(t) = 1] - \Pr[t \leftarrow Y_n, \mathcal{D}(t) = 1] < \varepsilon(n).$$

Based on the definition of computational indistinguishability, we next want to define the notion of pseudo-random distributions. Let U_n denote the uniform distribution over $\{0,1\}^n$. We say that a distribution is pseudo-random if it is indistinguishable from the uniform distribution.

2.3. Coding Theory & Cryptography

In this section, we start with a short introduction to coding theory. Then we briefly recall some concepts and definitions from coding theory and code-based cryptography that we will rely upon later. For more details we refer the reader to the books [MS77, Lin98].

2.3.1. Introduction

The theory of error-correcting codes was originally introduced in 1948 by Claude Shannon in his paper "A Mathematical Theory of Communication" [Sha48]. The study of error-correcting codes is called coding theory. This field is concerned with sending digital information over a noisy channel that adds errors to the transmitted data. Its main goal is to construct coding systems that can detect and correct such errors. The applications of coding theory include, for example, satellite communication, data transmission, data storage, mobile communication, file transfer, and digital audio/video transmission. The core idea of coding theory is to systematically introduce some redundancy to messages

for allowing transmission errors not only to be detected but also to be corrected. In other words, the sender first selects a message, which is represented as a string of symbols over some alphabet. This message is encoded (encoding process) into a longer string over the same alphabet, called a codeword, and then transmitted over a noisy channel. The channel adds errors (or noise) by modifying some of the characters of the transmitted string, then delivers the corrupted string to the receiver. The receiver finally tries to decodes the delivered message by using some knowledge (decoding process), hopefully to the intended message. This idea is similar to that used by McEliece to construct the first code-based cryptosystem [McE78].

2.3.2. Basic Definitions from Codes

Definition 2.3.1 (Linear Error-Correcting code)**.** *A q-ary (linear) error-correcting code (or code) $\mathcal{C}$ of length n over $\mathbb{F}_q$ is a subspace of $\mathbb{F}_q^n$. The dimension k of $\mathcal{C}$ is the dimension of $\mathcal{C}$ as an $\mathbb{F}_q$-vector space. Elements of $\mathbb{F}_q^n$ are called words and elements of $\mathcal{C}$ are called codewords. The difference $n-k$ is called the co-dimension of $\mathcal{C}$. A code with these features is called an $[n,k]$ code. For $q=2$, $\mathcal{C}$ is called a binary linear code. The code rate R of $\mathcal{C}$ is defined as the ratio between its dimension and its length, i.e. $R=\frac{k}{n}$.*

Definition 2.3.2 (Hamming weight, Hamming distance)**.** *The Hamming weight (or weight) $\mathtt{wt}(\mathbf{x})$ of a word $\mathbf{x}$ is the number of non-zero entries in $\mathbf{x}$. The Hamming distance $d(\mathbf{x},\mathbf{y})$ between two words $\mathbf{x}$, $\mathbf{y}$ is the number of entries i such that $x_i \neq y_i$.*

Definition 2.3.3 (Regular word)**.** *A word $\mathbf{x}$ of length n and weight $\mathtt{wt}(\mathbf{x})=\omega$ is called regular, if it is composed of ω blocks of length $\frac{n}{\omega}$, where each block has only a single non-zero entry.*

Definition 2.3.4 (2-Regular word)**.** *A 2-Regular word is defined as a sum of two regular words. It is of length n and weight less or equal to 2ω.*

Definition 2.3.5 (Minimum distance, relative distance)**.** *The minimum distance d (or just distance) of code $\mathcal{C}$ is the smallest distance between distinct codewords, i.e. $d=\min\{d(\mathbf{x},\mathbf{y});\mathbf{x},\mathbf{y}\in\mathcal{C},\mathbf{x}\neq\mathbf{y}\}$. The ratio $\frac{d}{n}$ is called the relative distance of $\mathcal{C}$, where n is the code length.*

If $\mathcal{C}$ is a linear code, the minimum distance d of $\mathcal{C}$ is the same as the minimum weight of the non-zero codewords of $\mathcal{C}$, i.e. $d=\min\{\mathtt{wt}(\mathbf{x});\mathbf{x}\in\mathcal{C},\mathbf{x}\neq\mathbf{0}\}$. If the minimum weight d of an $[n,k]$ code is known, we refer to the code as an $[n,k,d]$ code.
For decoding a word belonging to a subset S of the space $\mathbb{F}_q^n$ relative to a code, the usual method is to link a codeword having minimum distance to that word. To this purpose, we use the so-called error-correcting (or decoding) algorithm.

Definition 2.3.6 (Error-correcting algorithm)**.** *Let $\mathcal{C}$ be an $[n,k]$ code defined over $\mathbb{F}_q$, S a subset of the space $\mathbb{F}_q^n$, and δ a positive integer. An δ-error correcting algorithm $\mathcal{A}_C$ for $\mathcal{C}$ is defined by the following equation:*

$$\forall x\in S,\ \mathcal{A}_C(x)=\{\mathbf{y}\in\mathcal{C};d(\mathbf{x},\mathbf{y})\leq\delta\}.$$

Let $\mathcal{C}$ be an $[n,k,d]$ code, and a word $\mathbf{x}$ in $\mathbb{F}_q^n$. For any positive integer δ with $\delta\leq\lfloor\frac{(d-1)}{2}\rfloor$, there exists at most one codeword $\mathbf{y}\in\mathcal{C}$ satisfying $d(\mathbf{x},\mathbf{y})\leq\delta$. As a result, the word $\mathbf{x}$ is uniquely decodable

to codeword $\mathbf{x}$. The positive integer $\lfloor \frac{(d-1)}{2} \rfloor$ is called the error-correction capability of the code $\mathcal{C}$. As we can see, the minimum distance d is important in determining the error-correction capability of an $[n,k,d]$ code. The higher the minimum distance, the more errors the code can correct.

The two most common ways to define a linear $[n,k]$ code are whether with a generator matrix or with a parity check matrix. Such matrices are defined as follows.

Definition 2.3.7 (Generator matrix)**.** *A generator matrix for an $[n,k]$ code $\mathcal{C}$ is any $k \times n$ matrix G whose rows form a basis for $\mathcal{C}$. In this case, we have $\mathcal{C} = \{\mathbf{z} \cdot \mathsf{G}, \mathbf{z} \in \mathbb{F}_q^k\}$. If $\mathsf{G} = [\mathsf{I}_k \parallel \mathsf{A}]$, where I_k is the $k \times k$ identity matrix and A is an $k \times (n-k)$ matrix , then we say that G is in systematic (or standard) form.*

Definition 2.3.8 (Parity check matrix)**.** *A parity check matrix H for an $[n,k]$ code $\mathcal{C}$ is any $(n-k) \times n$ matrix defined by $\mathcal{C} = \{\mathbf{x} \in \mathbb{F}_q^n, \mathsf{H} \cdot \mathbf{x}^\top = \mathbf{0}\}$.*

It is easy to check that if $\mathsf{G} = [\mathsf{I}_k \parallel \mathsf{A}]$ is a generator matrix for an an $[n,k]$ code $\mathcal{C}$ in systematic form, then $\mathsf{H} = [-\mathsf{A}^\top \parallel \mathsf{I}_{n-k}]$ is a parity check matrix for $\mathcal{C}$.

Definition 2.3.9 (Syndrome)**.** *Let $\mathcal{C}$ be an $[n,k]$ code over $\mathbb{F}_q$, and let H be a parity check matrix for $\mathcal{C}$. For any $\mathbf{x} \in \mathbb{F}_q^n$, the syndrome of $\mathbf{x}$ is the vector $\mathbf{s}_\mathsf{H} = \mathsf{H} \cdot \mathbf{x}^\top \in \mathbb{F}_q^{n-k}$.*

As we can see, the syndrome depends on the choice of the parity check matrix H. Therefore, it is more suitable to denote the syndrome by $\mathbf{s}_\mathsf{H}$ to emphasize this dependance. However, for simplicity of notation, the index H is eliminated whenever there is no risk of ambiguity.

The Gilbert-Varshamov (GV) bound is a lower bound on rate of a code. It provides a sufficient condition for the existence of a linear code. It was actually proved in two independent works, first for general random codes by Gilbert [Gil] and then for linear random codes by Varshamov [Var57]. Before defining this bound, we need the following definition.

Definition 2.3.10 (Entropy function)**.** *For a positive integer $q \geq 2$, the q-ary entropy function $h_q : [0,1] \to \mathbb{R}$ is defined as follows: $h_q(x) = -x\log_q(x) - (1-x)\log_q(1-x)$.*
Of special interest is the binary $(q=2)$ entropy function: $h_2(x) = -x\log_2(x) - (1-x)\log_2(1-x)$.

The function h_q is continuous and strictly increasing on $[0, 1-\frac{1}{q}]$ with $h_q(0) = 0$ and $h_q(1-\frac{1}{q})$. The binary entropy function h_2 is symmetric with respect to line $x = \frac{1}{2}$ and satisfies $h_2(1-x) = h_2(x)$. Furthermore, for $z \in [0,1]$, the inverse $h_q^{-1}(z)$ is defined as the unique $x \in [0, 1-\frac{1}{q}]$ such that $h_q(x) = z$.

Definition 2.3.11 (Gilbert-Varshamov bound)**.** *Let n, k, and d be positive integers such that $2 \leq d \leq n$ and $1 \leq k \leq n$. If $\sum_{i=0}^{d-2} \binom{n-1}{i} (q-1)^i < q^{n-k}$, then there exists a linear $[n,k]$-code over $\mathbb{F}_q$ with minimum distance at least d.*

There exists a asymptotic version of the GV bound, which is stated using the relative distance and the entropy function h_q. This version reads as follows.

Definition 2.3.12 (Asymptotic Gilbert-Varshamov bound)**.** *Let $q \geq 2$. For every $0 \leq \delta < 1 - \frac{1}{q}$, and $0 < \varepsilon \leq 1 - h_q(\delta)$, there exists a code with rate $R \geq 1 - h_q(\delta) - \varepsilon$, and relative distance δ.*

2.3.3. Some Examples of Codes

In this section we briefly describe some important classes of linear codes that we will use in the subsequent chapters of this thesis. These codes have introduced to construct a number of cryptographic primitives from coding theory. We will start by presenting Goppa codes, followed by (quasi-) cyclic codes.

Goppa codes

These codes were first defined by V .D. Goppa in [Gop70]. They constitute a family of $q-$ary linear codes and can be defined as follows. Let q be an arbitrary prime power and m, n be two positive integers such that $m \geq 2$ and $n \leq q^m$. Let also $L = \{\alpha_1, \alpha_2, \cdots, \alpha_n\}$ be a subset of distinct elements of $\mathbb{F}_{q^m}$.

Definition 2.3.13 ($q-$ary Goppa codes)**.** *Let t be an integer such that $0 < t < n$. Let $g(x) \in \mathbb{F}_{q^m}[x]$ be a polynomial of degree t, called Goppa polynomial, such that $g(\alpha_i) \neq 0$, for all $1 \leq i \leq n$. The $q-$ary Goppa code $\mathcal{G}_q(L,g)$ is defined by*

$$\mathcal{G}_q(L,g) = \{\mathbf{x} \in \mathbb{F}_q^n, \sum_{i=0}^{n} \frac{x_i}{x-\alpha_i} \equiv 0 \mod g(\mathbf{x})\}.$$

Another way to describe Goppa codes is to consider the definition using the parity check matrix (see[MS77] for more details). In this definition, the Goppa code is defined as the the set of vectors $\mathbf{a} \in \mathbb{F}_{q^m}^n$, whose coordinates are labeled with the elements of L in the following way $\mathbf{a} = (a_{\alpha_1}, a_{\alpha_2}, \cdots, a_{\alpha_n})$ and satisfy the equation $\mathsf{H} \cdot \mathbf{a}^\top = \mathbf{0}$, where H is a parity check matrix given by

$$\mathsf{H} = \begin{bmatrix} \frac{1}{g(\alpha_1)} & \frac{1}{g(\alpha_2)} & \cdots & \frac{1}{g(\alpha_n)} \\ \frac{\alpha_1}{g(\alpha_1)} & \frac{\alpha_2}{g(\alpha_2)} & \cdots & \frac{\alpha_n}{g(\alpha_n)} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\alpha_1^{t-1}}{g(\alpha_1)} & \frac{\alpha_2^{t-1}}{g(\alpha_2)} & \cdots & \frac{\alpha_n^{t-1}}{g(\alpha_n)} \end{bmatrix}$$

Since $\mathbb{F}_{q^m}$ can be seen as a m-dimensional vector space over $\mathbb{F}_q$, then the matrix H can be written as a matrix over $\mathbb{F}_q$ of size $mt \times n$.

If $g(\mathbf{x})$ is irreducible, then $\mathcal{G}_q(L,g)$ is an irreducible Goppa code. In this thesis, we only consider binary Goppa codes, i.e. $q = 2$, which are denoted $\mathcal{G}(L,g)$. For such codes we have the following results.

Theorem 2.3.1 ([MS77])**.** *Let $\mathcal{G}(L,g)$ be an $[n,k,d]$ Goppa code. Then we have*

- *$k \geq n - mt$, where t is the degree of g*
- *$d \geq 2\ deg(g^*) + 1$, where g^* is the square-free polynomial, which has the highest degree and divides g*
- *there exists $deg(g^*)$-error correcting algorithm for $\mathcal{G}(L,g)$.*

Theorem 2.3.2 ([vdV90])**.** *Let $g(x) \in \mathbb{F}_{2^m}[x]$ be a square-free polynomial of degree t with no roots in $\mathbb{F}_{2^m}$. Let also $\mathcal{G}(L,g)$ be the corresponding $[n,k]$ Goppa code. If we choose $t < 2^{\frac{m}{2}-1}$, then we have $k = n - mt$.*

Cyclic codes

Cyclic codes were first introduced by Prange [Pra57] in 1957. They form a fundamental subclass of linear codes and have wide applications in data storage systems and in communication systems due to their interesting algebraic structure and efficient encoding/decoding algorithms. They can be described as follows.

Definition 2.3.14 (Cyclic code)**.** *An $[n,k]$ linear code $\mathcal{C}$ is cyclic if the cyclic shift of a codeword $\mathbf{x} \in \mathcal{C}$ is also a codeword in $\mathcal{C}$. That is, $\mathbf{x} = (x_1, x_2, \cdots, x_n) \in \mathcal{C} \Rightarrow \mathbf{x}' = (x_n, x_1, \cdots, x_{n-1}) \in \mathcal{C}$.*

Another way to define cyclic codes are cyclic (or circulant) matrices.

Definition 2.3.15 (Cyclic (or circulant) matrix)**.** *Let A be a square matrix of size $n \times n$. A is cyclic if every row of the matrix is a cyclic shift of the row above, i.e.*

$$\mathsf{A} = \begin{bmatrix} a_1 & a_2 & \cdots & a_n \\ a_n & a_1 & \cdots & a_{n-1} \\ \vdots & \ddots & \ddots & \vdots \\ a_2 & \cdots & a_n & a_1 \end{bmatrix}$$

As we can see, this matrix is fully specified by its first row (or column), which allows to reduce the size of the storage memory. Instead of storing the whole matrix, one need to store only its first row (or column).

Quasi-Cyclic (QC) codes

Quasi-cyclic codes are a generalization of cyclic codes and can be defined as follows.

Definition 2.3.16 (Quasi-cyclic code)**.** *An $[n,k]$ linear code $\mathcal{C}$ with $n = mn_0$ and $k = mk_0$ is called quasi-cyclic if the cyclic shift of a codeword $\mathbf{x} \in \mathcal{C}$ by n_0 symbols is also a codeword in $\mathcal{C}$. That is, $\mathbf{x} = (x_1, x_2, \cdots, x_n) \in \mathcal{C} \Rightarrow \mathbf{x}' = (x_{n-n_0}, \cdots, x_1, \cdots, x_{n-n_0-1}) \in \mathcal{C}$.*

A cyclic code is quasi-cyclic with $n_0 = 1$. As before, a QC code can be defined using a quasi-cyclic matrix as follows.

Definition 2.3.17 (Quasi-cyclic matrix)**.** *An $k \times n$ block matrix with $n = mn_0$ and $k = mk_0$ is called quasi-cyclic if each block is a cyclic matrix of size $m \times m$.*

In code-based cryptography, (QC) codes are used to construct many cryptographic primitives in order to improve their efficiency in practice. Some of these primitives will be presented in the following chapters.

2.3.4. Computational Problems

The security of the cryptosystems presented in this thesis rely on the hardness of the subsequent problems: The Syndrome Secoding (SD) problem and its two variants the Regular Syndrome Decoding (RSD), and the 2-Regular Null Syndrome Decoding (2-NRSD) problem. In this thesis we will use only the binary versions of these problems. That is, we consider $q = 2$.

Definition 2.3.18 (Syndrome decoding (SD))**.** *Given an* $(n-k) \times n$ *binary matrix* H*, an* $(n-k)$*-bit vector* $\mathbf{s}$*, and an integer* $\omega > 0$*. Find an n-bit word* $\mathbf{x}$ *of weight* ω *such that* $\mathsf{H} \cdot \mathbf{x}^\top = \mathbf{s}$*.*

This problem is proven NP-complete in [McE78] and decades of research in coding theory indicate that it is hard in the average case [Bar98]. Note that its extended variant $\mathbb{F}_q$ of this problem is also showed NP-complete in [Bar94].

A special case of this problem is called the regular syndrome decoding (RSD) problem, which only has solutions in the set of regular words. This is proven NP-complete in [AFS05], and reads as follows.

Definition 2.3.19 (Regular Syndrome decoding (RSD))**.** *Given an* $(n-k) \times n$ *binary matrix* H*, an* $(n-k)$*-bit vector* $\mathbf{s}$*, and an integer* $\omega > 0$*. Find n-bit regular word* $\mathbf{x}$ *of weight* ω *such that* $\mathsf{H} \cdot \mathbf{x}^\top = \mathbf{s}$*.*

A further NP-complete variant of the SD problem, introduced also in [AFS05], is the 2-Regular Null Syndrome Decoding (2-NRSD), which can be stated as follows.

Definition 2.3.20 (2-Regular Null Syndrome Decoding (2-NRSD))**.** *Given an* $(n-k) \times n$ *binary matrix* H*, and an integer* $\omega > 0$*. Find an n-bit word* $\mathbf{x}$ *of weight less than or equal to* 2ω *such that* $\mathsf{H} \cdot \mathbf{x}^\top = \mathbf{0}$*.*

In practice, all generic known algorithms to solve the above problems run in exponential time. We will discuss some of these algorithms in more details later.

2.4. Some Cryptographic Primitives

Cryptographic primitives are the most basic building blocks for creating cryptographic systems, that are designed to achieve security properties such as confidentiality, authentication or anonymity. Such systems include (authenticated) encryption schemes, hash functions and stream ciphers. In this thesis, we will look at how to design these three cryptographic systems from coding theory, whose security is based on the problems introduced in the previous section. For this purpose, we briefly recap the definition and properties of such primitives.

2.4.1. Encryption Schemes

An encryption scheme is a (mathematical) algorithm where plaintext is converted into so called ciper-text. There are two basic classes of encryption: symmetric encryption (or secret key encryption) and asymmetric encryption (or public key encryption). In symmetric encryption, a single key is used both for encryption and decryption, while in asymmetric encryption, two different keys are used, one for encryption and one for decryption.

As mentioned in Chapter 1 , the most famous class of asymmetric encryption schemes based on the hardness of the syndrome decoding problem contains the McEliece and Niederreiter encryption scheme [McE78, Nie86], which are actually equivalent from the security point of view as shown in [LDmW94]. In the following lines, we briefly explain how these systems work. The parameters of these systems are n, k, t with $\omega << n$.

The McEliece Public Key Cryptosystem. The McEliece cryptosystem (in its original version) uses a binary irreducible Goppa code as a trapdoor. This trapdoor is the knowledge of the Goppa polynomial. The McEliece PKC can be described as follows:

- **Key Generation:** Generate the following matrices:
  - G': $k \times n$ generator matrix of a $[n,k]$ binary irreducible Goppa code $\mathcal{G}$ with error-correcting capability ω
  - S: $k \times k$ binary non-singular matrix
  - P: $n \times n$ random permutation matrix

  Public Key: (G, ω) with $\mathsf{G} = \mathsf{SG}'\mathsf{P}$.

  Secret Key: $(\mathsf{S}, \mathcal{A}_{\mathcal{G}}, \mathsf{P})$, where $\mathcal{A}_{\mathcal{G}}$ is an efficient ω-error correcting algorithm for $\mathcal{G}$.
- **Encryption:** To encrypt the plaintext $\mathbf{x} \in \mathbb{F}_2^k$
  - choose randomly a word $\mathbf{e} \in \mathbb{F}_2^n$ of weight ω
  - compute the ciphertext $\mathbf{y}$ as $\mathbf{y} = \mathbf{x} \cdot \mathsf{G} \oplus \mathbf{e}$
- **Decryption:** To decrypt a ciphertext $\mathbf{y}$
  - compute the inverses P^{-1} and S^{-1}
  - calculate $\mathbf{y} \cdot \mathsf{P}^{-1} = \mathbf{x} \cdot \mathsf{SG}' \oplus \mathbf{e} \cdot \mathsf{P}^{-1}$
  - apply the algorithm $\mathcal{A}_{\mathcal{G}}$ for $\mathcal{G}$ to recover $\mathbf{x} \cdot \mathsf{S}$.
  - compute the plaintext $\mathbf{x} = \mathbf{x} \cdot \mathsf{S} \cdot \mathsf{S}^{-1}$

The Niederreiter Public Key Cryptosystem. This system is a (dual) variant of the McEliece cryptosystem. It uses a parity check matrix instead a generator matrix. The plaintext $\mathbf{x} \in \mathbb{F}_2^n$ of weight ω, while the corresponding ciphertext is a syndrome $\mathbf{y} \in \mathbb{F}_2^{n-k}$. The Niederreiter cryptosystem consists of three algorithms:

- **Key Generation:**
  - H': $(n-k) \times n$ parity check matrix of a $[n,k]$ binary irreducible Goppa code $\mathcal{G}$ with error-correcting capability ω
  - S: $(n-k) \times (n-k)$ binary non-singular matrix
  - P: $n \times n$ random permutation matrix

  Public Key: (H, ω) with $\mathsf{H} = \mathsf{SH}'\mathsf{P}$.

  Secret Key: $(\mathsf{S}, \mathcal{A}_{\mathcal{G}}, \mathsf{P})$, where $\mathcal{A}_{\mathcal{G}}$ is an efficient ω-error correcting algorithm for $\mathcal{G}$.
- **Encryption:** To encrypt plaintext $\mathbf{x} \in \mathbb{F}_2^n$ of weight ω, compute $\mathbf{y} = \mathsf{H} \cdot \mathbf{x}^\top$
- **Decryption:** To recover $\mathbf{x}$
  - compute the inverses P^{-1} and S^{-1}
  - calculate $\mathsf{S}^{-1} \cdot \mathbf{y} = \mathsf{H}'\mathsf{P} \cdot \mathbf{x}^\top$
  - apply the algorithm $\mathcal{A}_{\mathcal{G}}$ for $\mathcal{G}$ to recover $\mathsf{P} \cdot \mathbf{x}^\top$.
  - compute the plaintext $\mathbf{x}$ via $\mathbf{x}^\top = \mathsf{P}^{-1}\mathsf{P} \cdot \mathbf{x}^\top$

2.4.2. Stream Ciphers

Stream ciphers are a fundamental class of symmetric encryption algorithms, which transform a sequence of plaintext symbols (usually binary digits) one at a time, into a sequence of ciphertext symbols by combining plaintext symbols with a pseudo-random sequence, called keystream sequence. This latter is generated using a (known) initial vector and a secret key. Stream ciphers are commonly divided into classes: synchronous or self-synchronizing.
The former is one of the topics of this thesis and Chapter 3 is especially devoted to stream ciphers based on coding theory.
Informally, a synchronous stream cipher can be defined as follows.

Definition 2.4.1 (Synchronous stream cipher)**.** *A synchronous stream cipher is one in which the keystream is produced independently of the plaintext and the ciphertext.*

Let K be a secret key and IV be an initial vector. We denote by s_i an internal state of the cipher. In a synchronous stream cipher, the encryption (and decryption) process consists of the following steps:

- *Initialization:* The aim of this step is to produce a (pseudo-random) initial state, s_0, by $s_0 = f(K, IV)$, where f is an initialization function, whose arguments are K and IV.
- *Update:* In this step, an internal state s_i is updated as $s_{i+1} = g(s_i)$, where g is a function, called update or next state function.
- *Output:* The key stream, x_i, is produced by $x_i = h(s_i)$, where h is a function, called output function.
- *Encryption/Decryption:* The ciphertext y_i is obtained by combining x_i with the plaintext p_i using a combining function k as follows: $y_i = k(x_i, p_i)$. In the most proposed stream ciphers, the function k is simply the bitwise XOR-operator, i.e. $y_i = x_i \oplus p_i$. The decryption is then given by $p_i = x_i \oplus y_i$.

2.4.3. Hash Functions

A hash function is an important cryptographic primitive used in many applications and protocols for secure communication such as digital signatures, data integrity, and identification protocols. A detailed overview on hash functions can be found for example in [Pre93]. An informal definition for hash functions would be the following.

Definition 2.4.2 (Hash function)**.** *A hash function h is a computationally efficient function, which maps binary strings of arbitrary length to binary strings of some fixed length, called digest or hash. That is, $h : \{0,1\}^* \to \{0,1\}^t$, where t is the hash length.*

In principle, to be of cryptographic use, a hash function must fulfill three fundamental security requirements: preimage resistance, second preimage resistance and collision resistance. The importance of these requirements is application dependent. To explain these requirements, let $h : \{0,1\}^* \to \{0,1\}^t$ be a hash function, where t is the hash length in bits. A hash computation of a plaintext $x \in \{0,1\}^*$ is expressed as $h(x) = y \in \{0,1\}^t$.

1. **Preimage resistance:** Given any hash value y, it is "computationally infeasible" or "hard" to find a plaintext (or message) x such that $h(x) = y$. In other words, it must be hard to invert h from y to get x. This property is also known as one-wayness.

2. **Second preimage resistance:** Given a plaintext x and its corresponding hash value $h(x)$, it is "computationally infeasible" or "hard" to find another plaintext x' such that

   $$x' \neq x \text{ and } h(x') = h(x).$$

3. **Collision resistance:** It is "computationally infeasible" or "hard" hard to find any two plaintexts (or message) x' and x such that

   $$x' \neq x \text{ and } h(x') = h(x).$$

A hash function f that transforms a fixed-length input and to a shorter, fixed-length output, is called a compression function. That is, $f : \{0,1\}^{\ell} \rightarrow \{0,1\}^{t}$, with $\ell > t$.

In practice, for designing hash function, the following modes of operation can be utilized.

The Merkle-Damgård construction (MD). The simplest and most commonly approach for designing hash functions is to iterate a compression function on the plaintext to be hashed. A compression function is a mapping, which takes a fixed length input and returns a shorter, fixed-length output. This design principle is called the Merkle-Damgård construction [Mer89, Dam89] (MD), and it works briefly as follows. Let f be a compression function. First the plaintext to be hashed (with padding) P, is broken up into equal-sized blocks, i.e. $P = (p_1, p_2, \cdots, p_l)$. Then the temporary hash values h_i, called the chaining variable or the internal state, are computed as

$$h_l = f(p_l \parallel h_{l-1}) \,,$$

where $h_0 = IV$ is a given initial value and h_l is the final hash for P. Figure 2.1 illustrates the MD-construction.

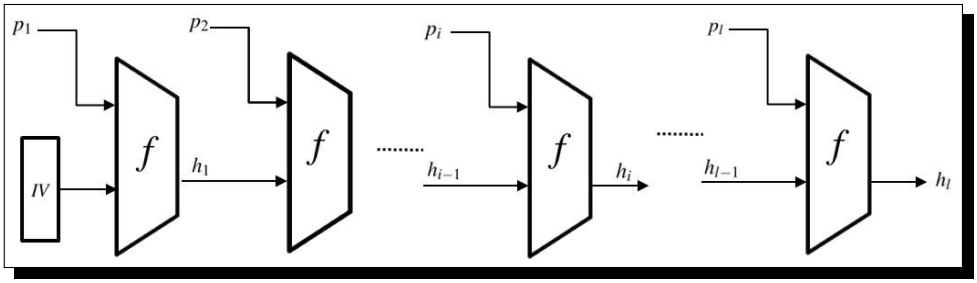

Figure 2.1.: The Merkle-Damgård construction using a compression function f.

The security of this construction is mainly based on the security of the underlying compression function. More precisely, Merkle [Mer89] and Damgård [Dam89] showed that as long as the compression function is collision resistant, the resulting hash function is guaranteed to be collision resistant. Unfortunately, the MD-construction has been shown to be vulnerable to certain generic attacks such as multi-collision attacks [Jou04] and long-message second-preimage attacks [KS05]. Motivated by these attacks, new methods of designing hash functions have proposed. One of these methods, called the Sponge construction is described in the following.

The Sponge construction (SG). In contrast to the MD-construction, the sponge construction [BDPA07] is a recently proposed and very interesting hash design, which uses a random permutation or a random transformation f, instead of a compression function and supports variable length outputs. If f is a random permutation, this construction is called P-sponge, otherwise, it is called a T-sponge. The sponge construction operates on an internal state having a fixed size $b = r + c$, where r is the bit rate and c the capacity of the sponge. Initially, the state is equal to the all-zeros vector, i.e. 0^b. Basically, the sponge construction as depicted in Figure 2.2, proceeds in the following steps:

- **Absorbing step:** In this step, the plaintext to be hashed is first padded using a padding rule and cut into r-bit blocks such that the last block absorbed shall not be zero, i.e. $P = (p_1, p_2, \cdots, p_l)$ with $|p_i| = r$ for all i and $p_l \neq 0^r$. Then each block is XOR-ed with the r-bit part of the current state s_i, intervealed by the application of f, resulting in the next state s_{i+1}, i.e.

  $$s_{i+1} = f(s_i \oplus (p_i \parallel 0^c)), \text{ with } s_0 = 0^b.$$

  This process will be iteratively repeated until all blocks are processed.

- **Squeezing step:** In this step, the state continues to be updated (or permuted) by f followed by outputting only the r-bit part of the resulting state at each iteration as output blocks. We denote these blocks by h_i. The hash value h of length ℓ consists of the concatenation of all these blocks, i.e. $h = h_1 \parallel h_2 \parallel \cdots \parallel h_N$ such that $\ell = Nr$. The number of iterations (or output blocks) is chosen by user in order to get the desired hash length.

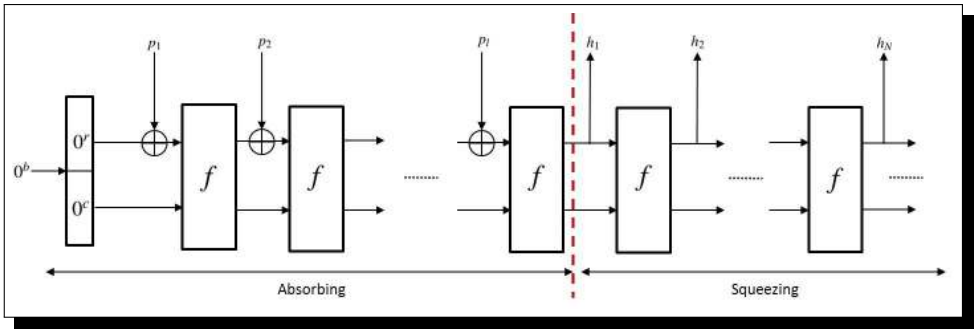

Figure 2.2.: The sponge construction based on the permutation/transformation f.

The security of SG construction depends on its capacity c, hash length ℓ, and transforation or permutation f. It has been shown in [BDPA08] that when the internal permutation (resp. internal transformation) is modeled as a randomly chosen permutation (resp. random chosen transformation), the expected security bounds offered by a ℓ-bit sponge-based hash function are given in Table 2.1.

Type	Collision	Preimage	2^{nd} Preimage
P-Sponge	$\min(2^{\frac{c}{2}}, 2^{\frac{\ell}{2}})$	$\min(2^{\frac{c}{2}}, 2^{\ell})$	$\min(2^{\frac{c}{2}}, 2^{\ell})$
T-Sponge	$\min(2^{\frac{c}{2}}, 2^{\frac{\ell}{2}})$	$\min(2^{c}, 2^{\ell})$	$\min(\frac{2^c}{\|P\|}, 2^{\ell})$

Table 2.1.: The security bounds of the sponge construction against collision, preimage, and 2^{nd} Preimage attacks, where the quantity $|P|$ is the size of the plaintext to be hashed.

On the other hand, if the SG construction is keyed [BDPA11b], i.e. the message is prefixed with a secret key of length t, then the best known attack against this construction is exhaustive key search as

long as the size of the message queries is upper bounded by 2^a with $a << c$ and $c \geq t + a + 1$.

Chapter 3

Code-based Stream Ciphers

This chapter starts with a brief overview of existing constructions of code-based stream ciphers in Section 3.1. Namely, the pseudo-random generator due to Fisher and Stern [FS96], followed by the SYND stream cipher, proposed by Gaborit et al. [GLS07]. Then, it presents our three contributions in this field: the 2SC [MCY11], XSYND [MHC12], and PSYND [MHC] stream cipher. Following the sponge construction, we present the 2SC stream cipher in Section 3.2, while we describe the XSYND cipher in Section 3.3, which is an improved version of the SYND cipher in terms of performance. Finally, the PSYND cipher we present in Section 3.4 can be regarded as a parallel version of XSYND and outperforms all previous constructions in terms of efficiency.

3.1. Previous Work

This section briefly gives an overview of proposed pseudo-random number generators, which use error-correcting codes.

3.1.1. The Fischer-Stern's Pseudo-Random Generator

At Eurocrypt 1996, Fischer and Stern [FS96] presented the first code-based pseudo-random number generator (FS-PRNG) based on the syndrome decoding problem. This generator uses a collection of functions, denoted here as $(f_n)_{n\geq 0}$, whose domains $(D_n)_{n\geq 0}$ are given by

$$D_n = \left\{ (\mathsf{H}, x) \in \mathbb{F}_2^{\ell \times n} \times \mathbb{F}_2^n, \mathtt{wt}(x) = \omega \right\}.$$

and the collection $(f_n)_{n\geq 0}$ are defined by

$$f_n : D_n \to \mathbb{F}_2^{\ell(n+1)} \tag{3.1}$$

$$(\mathsf{H}, x) \mapsto f_n(x) = \left(\mathsf{H}, \mathsf{H} \cdot x^\top \right). \tag{3.2}$$

Thus inverting a function f_n implies solving instances of the syndrome decoding (SD) problem. Hence, if one chooses (n, ℓ, ω) such that the collection corresponds to hard instances of the SD problem, then it can be regarded as a collection of one-way functions. So, for parameter sets (n, ℓ, ω)

satisfying the GV bound, we have the following fact: "*For sets of parameters* (n, ℓ, ω) *satisfying the GV bound the collection* $(f_n)_{n\geq 0}$ *is one-way.*"

Note that the functions f_n are expansion functions. They accept strings of length $\ell n + \log_2(\binom{n}{\omega})$ bits and output strings having length $\ell(n+1)$ bits, since $\log_2(\binom{n}{\omega}) < \ell$.
Starting from a function f_n defined over D_n, Fischer and Stern proposed an iterative construction of a pseudo-random number generator, which produces $\ell - \log_2(\binom{n}{\omega})$ key stream bits in each round. One iteration of this generator is illustrated in Algorithm 1.

Algorithm 1 One round of Fischer-Stern PRNG

Input : a seed e_0 of length $\lceil \log_2 \binom{n}{\omega} \rceil$ bits
Output : a bit string z of length $\ell - \lceil \log_2 \binom{n}{\omega}$

$x \leftarrow \phi(e_0)$ // convert e_0 into a word x of length n and weight ω using Algorithm 2.
$y \leftarrow \mathsf{H} \cdot x^\top$ // multiply x by H
$(y_1, y_2) \leftarrow y$ // split y into y_1 and y_2 with $|y_1| = \lceil \log_2 \binom{n}{\omega} \rceil$ and $|y_2| = \ell - \lceil \log_2 \binom{n}{\omega} \rceil$
$z \leftarrow y_2$ //output z

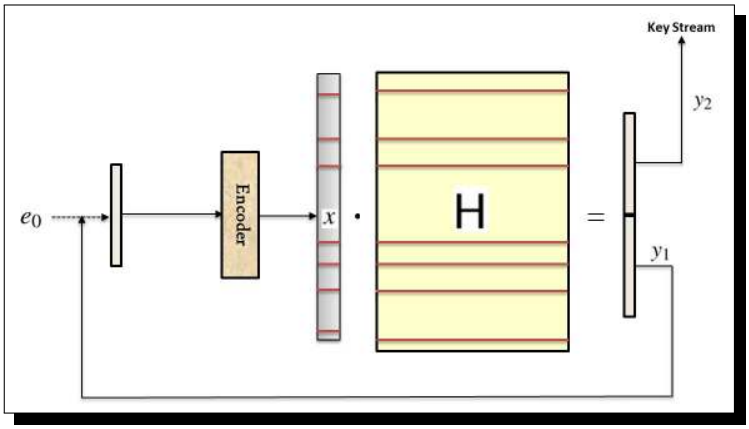

Figure 3.1.: A diagram of FS-PRNG.

For converting bit strings of size $\lceil \log_2 \binom{n}{\omega} \rceil$ into words of length n and weight ω, the Fischer-Stern PRNG uses the encoding algorithm shown in Algorithm 2.

Security analysis. The randomness of the key stream produced by G_n is proven in [FS96] under two assumptions:

(1) the collection of functions $(f_n)_{n\geq 0}$ and

(2) the matrix H is indistinguishable from a random one.

The proof is done by contradiction and briefly works as follows: Assume that the key stream produced by G_n is not pseudo-random. One first constructs a distinguisher whose input is a string generated by G_n, say $G_n(z)$ for some random $z \in \mathbb{F}_2^n$, and a random string $r \in \mathbb{F}_2^n$ and whose output is 1 with different probability. Then one uses this distinguisher to build a predicator allowing to correctly

Algorithm 2 Guillot's algorithm ϕ

Input : an integer e between 0 and $\binom{n}{\omega} - 1$
Output : a binary word $x = (x_1, x_2, \cdots, x_n)$ of length n and weight ω

$t \leftarrow \binom{n}{\omega}$, $t' \leftarrow 0$, $j \leftarrow n$
while $j > 0$ **do**
  $t' \leftarrow t \cdot \frac{j-\omega}{j}$
  if $e \leq t'$ **then**
    $x_j \leftarrow 0$
    $t \leftarrow t'$
  else
    $x_j \leftarrow 1$
    $e \leftarrow e - t'$
    $t \leftarrow t \cdot \frac{\omega}{n}$
  end if
  $j \leftarrow j - 1$
end while

guess the inner product of x and r with success probability at least $\frac{1}{2} + \frac{1}{2p(n)}$, for every polynomial $p(n)$. In doing so, they obtain a contradiction to the one-wayness of $(f_n)_{n\geq 0}$ using the Goldreich-Levin Theorem [GL89].

Performance. As explained above, a string $y_2 \in \mathbb{F}_2^{\ell - \lceil \log_2 \binom{n}{w} \rceil}$ is produced in each iteration. To do so, one needs to first multiply an $\ell \times n$ matrix H by a string from $\mathbb{W}_{n,w}$. This can solely be performed by XORing ω columns of H leading to $\ell\omega$ binary operations. The columns positions are determined by the indexes j of x with $x_j = 1$ in Algorithm 2, which needs the computations of binomial coefficients and necessitates arithmetic operations on large integers. This approximately requires $O(n^2 \log_2(n))$ binary operations. As a consequence, the whole cost of generating $\ell - \lceil \log_2 \binom{n}{\omega} \rceil$ bits amounts to around $O(n^2 \log_2(n)) + \ell\omega$ binary operations. In practice, the authors claimed that their system outputs 3500 bits per second on a SUN Sparc10 station using $(n, \ell, \omega) = (512, 256, 55)$.
In order to increase the performance of the generator, Fischer and Stern proposed to precompute the binomial coefficients and store them in a table. For a code with parameters (n, ℓ, ω), the memory needed to store these coefficients is $(\omega \ell n)$ bits, since we need ωn entries, each of them of size ℓ bits. Furthermore, a space of $\ell(n-\ell)$ bits due to the matrix H is required.

Proposed parameters. Their values are listed in Table 3.1 with $n = 2\ell$ and $\log_2 \binom{n}{\omega} < \ell$, for which the GV-bound condition is fulfilled.

3.1.2. The SYND stream cipher

Motivated by the inefficiency of Fischer-Stern PRNG [FS96], Gaborit et al. [GLS07] proposed the SYND stream cipher as an improved variant with two main features: introducing quasi-cyclic matrices reduces the storage capacity and replacing the above encoder by a new one. This so called regular encoder is used in [AFS05], and considerably speeds up the key stream generation. As for most stream

n	ℓ	ω	key/IV (bits)	speed (cycles/byte)	sec-level $\log_2$(# bin.Ops)
512	256	55	247	360410	60
728	364	71	331	38140	78
728	364	78	353	40330	85
1024	512	100	468	25810	100
1024	512	110	613	26970	120

Table 3.1.: Proposed parameters for FS-PRNG in [FS96].

ciphers, keystream generation of SYND consists of three phases: the initialization, the update and the output steps.

(1) Initialization: It is depicted in Figure 3.2. The aim of this phase is to produce an initial state e_0 using a secret key K and an initial vector IV of the same length $\ell/2$ bits. This is achieved as follows. Let g_1 and g_2 be two syndrome maps defined by

$$g_1 : \mathbb{F}_2^\ell \to \mathbb{F}_2^\ell \tag{3.3}$$
$$x \mapsto g_1(x) = \mathsf{A} \cdot (\phi(x))^\top \tag{3.4}$$

and

$$g_2 : \mathbb{F}_2^\ell \to \mathbb{F}_2^\ell \tag{3.5}$$
$$x \mapsto g_2(x) = \mathsf{B} \cdot (\phi(x))^\top \tag{3.6}$$

A graphical illustration of g_i $(i = 1,2)$ is shown in Figure 3.3. The matrices A and B are two random cyclic of the same size $\ell \times n$ with $\ell = \omega \log_2(n/\omega)$and $x \mapsto \phi(x)$ is an encoding algorithm, called a (n,ω) *regular encoder*, transforming an ℓ bits string into regular words of length n and weight ω. This encoder is presented in Algorithm 3. The purpose of this algorithm is to speed up the vector-matrix multiplication, which only consists in XORing of ω columns of the underlying matrix. Here we assume that n/ω is an integer such that $n/\omega = 2^\alpha$ for some $\alpha > 0$. Note that in [AFG+08] a further regular encoding algorithm has been introduced, which mixes the input bits to produce a regular word.

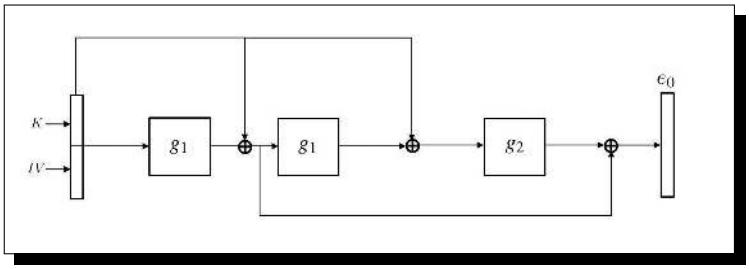

Figure 3.2.: A diagram of the initialization function f used in SYND.

Algorithm 3 Regular Encoder ϕ

Input : x a binary string of ℓ bits with $\ell = \omega \log_2(n/\omega)$
Output : a regular word $z = \phi(x)$ of length n and weight ω.

$z = (z_1, \cdots, z_n) \leftarrow 0^n$ *(initializing with n zeros)*
for $i = 1$ **to** $\omega - 1$ **do**
  Extract the $\log_2(n/\omega)$ right bits of x
  Convert those bits into an integer k between 0 and $n/\omega - 1$
  $z_{(i \cdot n/\omega)+k} \leftarrow 1$
  Shift x to the right by $\log_2(n/\omega)$ bits
end for

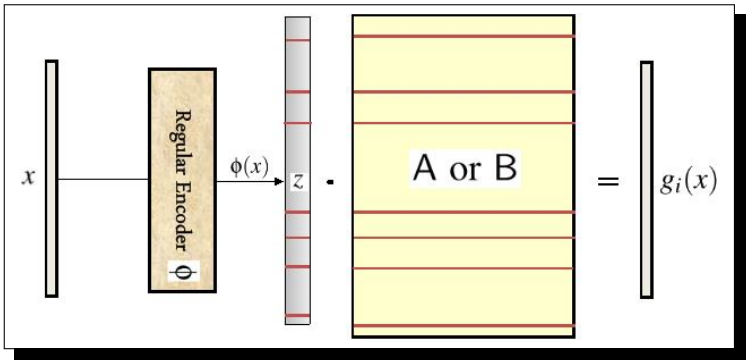

Figure 3.3.: A diagram of mappings $x \to g_i(x)$.

Starting from the mapping $(g_i)_{i=1,2}$, a further mapping f is build as an initialization function to generate an initial state e_0 of length ℓ. This function is defined by

$$f : \mathbb{F}_2^{\ell/2} \times \mathbb{F}_2^{\ell/2} \to \mathbb{F}_2^{\ell}$$
$$(x\|y) \mapsto f(x\|y) = (x\|y) \oplus g_1(x\|y) \oplus g_2((x\|y) \oplus g_1((x\|y) \oplus g_1(x\|y)))$$

Where $(x\|y)$ denotes the concatenation of x and y. Obviously computing e_0 exactly requires three XOR operations and three function evaluations. We can estimate the number of XORs needed to evaluate $f(x||y)$: the evaluation of g_1 (or g_2) can be performed in $\omega\ell$ binary XORs and each XOR-operation of two binary vectors of length ℓ needs ℓ binary XORs. This means, that the generation of e_0 requires $3\ell(1+\omega)$ binary operations.

(2) Update: During this phase, the initial state is updated several times (say λ times) by calling the mapping g_1 to produce an internal state e_{i+1} as $e_{i+1} \leftarrow g_1(e_i)$ with $e_0 = f(K, IV)$.

(3) Output: In this step, the resulting internal state e_i is fed through the mapping g_2 to generate the keystream, which is XOR-ed with the cleartext to get a ciphertext. The whole process of key stream generation is depicted in Figure 3.4

Security analysis. In the above description, the main building blocks of the SYND stream cipher are the functions g_1 and g_2. Thus the security of SYND can be reduced to the RSD problem presented

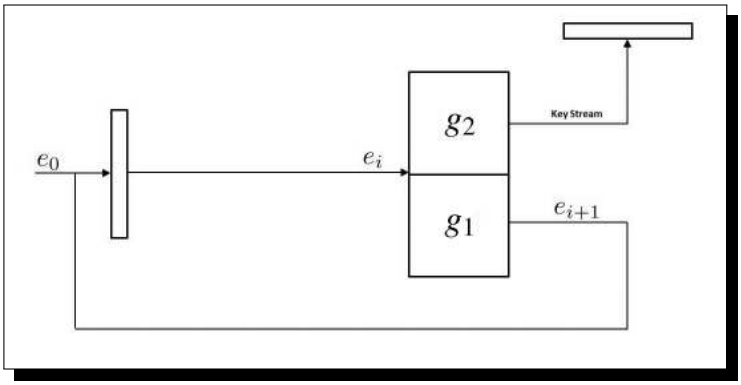

Figure 3.4.: A diagram of the key stream generation of SYND.

earlier. Indeed, state recovery or key recovery consists in inverting either g_1 or g_2, which is equivalent to solving instances of regular syndrome decoding problem. Furthermore, the authors of SYND did not deliver an explicit security reduction proof. They only pointed out that the proof is a generalization of [BGP09] and [FS96] and the proof will be detailed in the full version. The main result of their proof is the following: If there exists an algorithm $\mathcal{A}$ distinguishing a random bit sequence from the sequence of the bit key stream produced through a known random (or a quasi-cyclic random) $\ell \times n$ matrix multiplied by an unknown randomly chosen regular word e in time T and advantage ε, then there exists an algorithm $\mathcal{A}'$ that can recover e in time $T' \approx \frac{2^7 \ell^2 \lambda^2 T}{\varepsilon^2}$, where λ is the number of iterations.

Performance. Theoretically the performance of SYND can be expressed as the number of binary XORs to produce ℓ bits of the keystream. To do this, we have to count the average number of XORs in each step. As mentioned above, the initialization step requires about $3\ell(1+\omega)$ binary XORs. The complexity of the update and the output phase is about $N\omega$ and ω binary XORs respectively, where N is the number of rounds made during the update process. Thus, the whole complexity to produce a keystream of ℓ bits is about

$$3\ell + \ell\omega(2N+3) \text{ binary operations}$$

Regarding the storage requirements, one needs to store ℓ bits coming from key size and initial vector and the n bits of first row of the random quasi-cyclic matrix of size $\ell \times n$.

Proposed parameters. They are shown in Table 3.2. For efficiency reason, $n = 8192$, $\ell = 256$ and $\omega = 32$ are the proposed parameter. The implemenation results given in [GLS07] provide 27 cycles/byte on Pentium IV running at 3.4 GHz versus 26 cycles/byte for AES-CTR according the best AES-implementation in 2007. At the time of writing this chapter, there exists no-free implementation of SYND made by the authors. The only available implementation of SYND has been first presented in [MCY11]. An optimized version of this implementation is recently proposed in [CSM] (see the subsequent section) and shows that SYND only runs at 30.27 instead of 26 cycles/byte as claimed in [GLS07]. We will make our comparison based on this optimized implementation [CSM]. The security levels listed below are estimated according to the best known attack [MMT11].

$n = 8192$				
ω	ℓ	key/IV size (bits)	speed (cycles/byte)	sec-level $\log_2$(# bin.Ops)
32	256	128	26	90
48	384	192	47	155
128	1024	512	83	370

Table 3.2.: Performance of SYND given in [GLS07]

3.2. The 2SC Stream Cipher

In this section, the first contribution in the context of stream ciphers will be presented. We propose here a novel stream cipher, called the sponge code-based stream cipher (in short 2SC), following the sponge construction. The main goal of this section is to show how to design a new stream cipher, which runs much faster than the SYND cipher described in the previous section.

In the description of 2SC, we will preserve the same notations as before. The main parameters are (n, ℓ, ω) with $\ell = \omega \log_2(n/\omega) = r + c$, where r and c are the parameters characterizing the sponge construction.

3.2.1. Description of the 2SC cipher

The 2SC is a family of synchronous stream cipher supporting the key/initial vector (IV) lengths of 144, 208, and 352 bits, respectively. As for most stream ciphers, the key stream generation process consists of two phases:

(1) initialization: an initial state of the cipher is created using the key K and the initial vector IV having the same length $\ell/2$, and

(2) the key stream generation: the state is repeatedly updated (Update step) and used to generate key stream bits (Squeezing step).

These two phases use two different functions f and g and are defined as follows.

Initialization. The initialization function f takes a key K and an initial vector IV and returns an initial state as follows:

$$\begin{aligned} f : \mathbb{F}_2^{|K|} \times \mathbb{F}_2^{|IV|} &\to \mathbb{F}_2^{\ell} \\ (x_1, x_2) &\mapsto f(x_1, x_2) = f_1\left(f_1^{[r]}(x_1 \| 0^c) \oplus x_2, f_1^{[c]}(x_1 \| 0^c)\right), \end{aligned}$$

where "||" denotes the concatenation and "0^t" is the all-zero vector of size t. We write $f_1^{[r]}(z)$ (resp. $f_1^{[c]}(z)$) the r-bit (resp. c-bit) part of the output $f_1(z)$ for an input z from $\mathbb{F}_2^{\ell}$, where f_1 is defined by:

$$f_1 : \mathbb{F}_2^\ell \to \mathbb{F}_2^\ell$$
$$x \mapsto f_1(x) = \mathsf{A} \cdot (\phi(x))^\top.$$

Here, the function $x \mapsto \phi(x)$ is a regular encoder described above in Algorithm 3, which converts a ℓ-bit string into a regular word of length n and weight ω. The matrix A is a random binary matrix of size $\ell \times n$. The whole process of generating an initial state is shown in Figure 3.5.

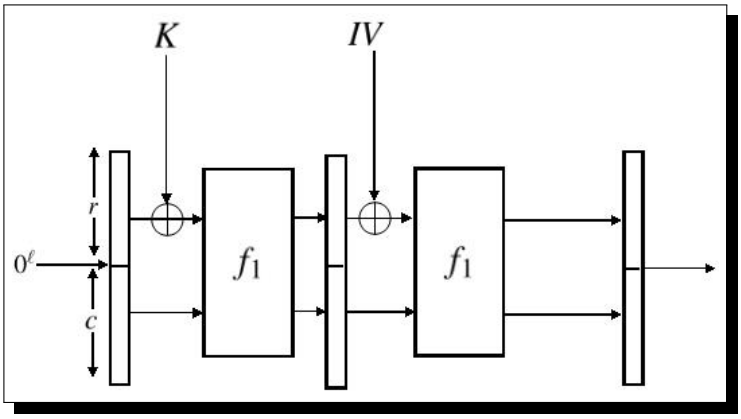

Figure 3.5.: The Initialization function f of the 2SC stream cipher, where $f_1(x) = \mathsf{A} \cdot (\phi(x))^\top$.

Update. During this step, an additional function g is used to update the internal state several times. The number of times (say λ) that g is run is chosen by the user, affecting both the security and the efficiency of the construction. The function g is defined by:

$$g : \mathbb{F}_2^\ell \to \mathbb{F}_2^\ell \tag{3.7}$$
$$x \mapsto g(x) = \mathsf{B} \cdot (\phi(x))^\top, \tag{3.8}$$

where A is a binary random matrix having the same size as B.

Squeezing. Let e_0 be the initial state returned by f and $e_{\lambda-1} = g^{(\lambda-1)}(e_0)$, where $g^{(\lambda-1)}$ is the composition of g with itself $(\lambda - 1)$ times. The keystream of SC consists of r-bit blocks $(z_i)_{i\geq 1}$ computed as follows:

- z_1 consists of the first r bits of the internal state $e_{\lambda-1}$ after calling g, i.e. $z_1 = g^{[r]}(e_{\lambda-1})$
- For $i \geq 2$, $z_i = g^{[r]}(e_{\lambda+i-2})$.

The entire process explaining Update and Squeezing steps is shown in Figure 3.6.

Having the key stream bits z_i, the ciphertext c_i is obtained by combining the plainttest block m_i with z_i using the XOR-operation as in the one-time pad encryption scheme.

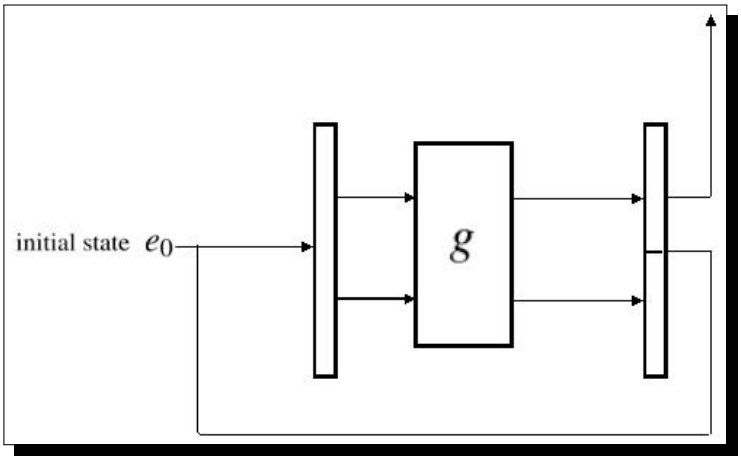

Figure 3.6.: The Update and Squeezing phases of the 2SC cipher, where $g(x) = \mathsf{B} \cdot (\phi(x))^{\top}$.

3.2.2. Security Analysis

The security of 2SC is discussed in this subsection. We first show that, the output of 2SC is pseudo-random, i.e., the probability to distinguish the key stream output by 2SC from a random sequence is negligible. Then we analyze the security of 2SC from practical point of view by demonstrating that, in practice, it is hard to recover states or reconstruct the secret data (K and IV) from the key stream.

Pseudorandomess of the key stream. In order to prove that the output of 2SC is pseudo-random, we define some useful concepts.

Definition 3.2.1. *A family $\mathcal{U}$ of hash functions $u \in \mathcal{U}$ with $u : X \to Y$ is called universal if for all $x \neq x'$ we have*

$$Prob[u(x) = u(x') \mid u \text{ sampled randomly from } \mathcal{U}] \leq \frac{1}{b},$$

where $x, x' \in X$ and $b = |Y|$.

Next, we introduce the Subset Sum Problem (SSP), which is closely related to the syndrome decoding problem. The SSP has been proved NP-complete by Karp in [Kar72] and can be stated as follows.

Definition 3.2.2 (Subset Sum Problem (SSP))**.** *Given n integers $(h_1, \cdots, h_n)$, each of ℓ bits, and and an integer y called the target, find a subset $S \subset \{1, \cdots, n\}$ such that $\sum_{j \in S} h_j = y \mod 2^{\ell}$.*

As stated in [IN96], this problem is equivalent to inverting the function

$$g_h(S) = \sum_{j \in S} h_j = y \mod 2^{\ell}. \tag{3.9}$$

This function maps an n-bit string to ℓ-bit string. When the cardinality $|S|$ of S is upper bounded by a fixed integer ω (i.e. $|S| \leq \omega$), we get an instance of the (regular) syndrome decoding stated earlier. More precisely, take $|S| = \omega$; then the elements in S can be interpreted as the positions of the non-zero coordinates of an incidence vector x. Thus x has weight $|S| = \omega$. The elements $(h_1, \cdots, h_n)$ are the

rows of a matrix H of size $\ell \times n$. The target y is the syndrome such that $\mathsf{H} \cdot x^\top = y$.

Without loss of generality, the transformation f in the initialization step and the equivalent transformation g can be regarded as a mapping $x \mapsto u(x) = \mathsf{H} \cdot x^\top$, x is a regular word, because the encoding function ϕ is bijective.

In what follows, we will use $\mathcal{R}$ to denote the set of regular words having length n and weight ω and $\mathcal{H}$ to indicate the set of binary random matrices of size $\ell \times n$. In order to prove that the family $\mathcal{U} = \{u : u(x) = \mathsf{H} \cdot x^\top, x \in \mathcal{R}, \mathsf{H} \in \mathcal{H}\}$ is universal, we need the following lemma.

Lemma 3.2.1. *There exists, on average, only one solution of each instance RSD(n, ℓ, ω), where $\ell = \omega \log 2(n/\omega)$.*

Proof. Let $N_{rsd}(n, \ell, \omega)$ denotes the expected number of solutions of an instance RSD(n, ℓ, ω). This number is defined as the number of regular words divided by the number of the syndromes, i.e. $N_{rsd}(n, \ell, \omega) = \frac{\binom{n}{\omega}^\omega}{2^\ell}$. By replacing ℓ by the value $\omega \log 2(n/\omega)$, we obtain $N_{rsd}(n, \ell, \omega) = 1$.

□

Proposition 3.2.1. *The family $\mathcal{U} = \{u : u(x) = \mathsf{H} \cdot x^\top, x \in \mathcal{R}, \mathsf{H} \in \mathcal{H}\}$ is universal.*

Proof. From Lemma 3.2.1, we know that there exists on average only one regular word that solves the syndrome decoding problem. Thus, it follows that for all for all $x \neq x'$

$$\Pr[\mathsf{H} \cdot x^\top = \mathsf{H} \cdot {x'}^\top \mid \mathsf{H} \text{ sampled randomly from } \mathcal{H}] = 0 \leq \frac{1}{2^\ell}$$

□

Now we prove the following theorem.

Theorem 3.2.1. *The probability of distinguishing the output of each $u \in \mathcal{U}$ from a random sequence of length ℓ is negligible.*

Proof. The proof is deduced from [IN96] and works as follows. As explained above, the family $\mathcal{U}$ can be seen as a collection g_h defined as in equation (3.9). Due to Proposition 3.2.1 this collection of transformations is universal and therefore, as proved in [IN96], we can apply the Leftover hash lemma [IZ89] to show that if $\ell < \gamma n$ for some real number $\gamma < 1$, then the expected distinguishibility of $g_h(S) = \mathsf{H} \cdot x^\top$ and a random $y \in \mathbb{F}_2^\ell$ is at most $2^{\frac{-(1-\gamma)n}{2}}$.
In our setting, γ can be obtained as follows:

$$\psi(n, \omega) = \frac{\ell}{n} = \frac{\log_2(n/\omega)}{(n/\omega)}.$$

For simplicity, we can assume that $n > 4\omega$. In this case, the function ψ tends to zero when n is chosen to be large enough. Consequently, there exists an n_0 such that for all $n \geq n_0$, $\psi(n, \omega)$ is upper bounded by a constant $\gamma < 1/2$. Thus, for values of the code length n such that $\frac{(1-\gamma)n}{2}$ is large enough, the probability of distinguishing the output of any function $u \in \mathcal{U}$ from a random sequence of length ℓ is negligible.

□

Pseudorandomness of the initial state. The initialization process of the 2SC consists of two stages. During the first stage, the secret key K is introduced to generate a pre-initial state. For suitably chosen parameters (n, ℓ, ω), as indicated in [GLS07], the underlying syndrome mapping behaves like a random function, since its outputs are indistinguishable from a random sequence. Therefore, the pre-initial state is pseudo-random. During the second stage, the outer r-bit part of this state is first XORed with a secret initial value. Then, the resulting ℓ-bit vector is fed to the function f to produce the initial state. This process can be viewed as XORing ω random columns of a random matrix, resulting in a random ℓ-bit initial state.

Best known attacks. In practice, an adversary against the security of 2SC is faced with two problems. On the one hand, knowing the blocks z_i of r bits does not allow him to get the remaining c bits; the larger the capacity, the more secure the system is. On the other hand, even having successfully guessed those bits, the adversary must solve an instance of the RSD problem. However, solving the RSD problem efficiently is as difficult as SD in average case, for an appropriately chosen parameter set. Indeed, all known attacks for SD are fully exponential; in fact, only three kinds of algorithms can attack the SD-based systems: Information Set Decoding (ISD), the Generalized Birthday Algorithm (GBA), and structural decoding. Which of the two approaches is more efficient depends on the parameters and the cryptosystem. In our setting, each instance of RSD has on average one solution due to the form of the regular words; here the best known attack is the GBA, as shown in [FS09]. The most recent GBA against code-based crytosystems is proposed in [FS09] and will be used to select secure parameters for 2SC.

Remark 3.2.1. *One could also use Time Memory trade-off attacks against stream ciphers. This attack was first introduced in [Hel80] as a generic method of attacking block ciphers. To avoid it, one must adjust the cipher parameters as shown in [HS05, Gol97], i.e., the IV should be at least as large as the key, and the state should be at least twice the key.*

3.2.3. Parameters Choice and Implementation Results

Suitable parameters (n, ℓ, ω) for 2SC should provide both efficiency and high security against all known attacks. Firstly, we account for Time Memory Trade-Off attacks (see section 5.1) and choose (n, ℓ, ω) such that $\ell = \omega \log_2(n/\omega) \geq 2|IV| \quad \text{and} \quad |IV| \geq |K|$.
Since $|IV| = |K| = r$, we obtain $\ell \leq 2c$. We use the following strategy for selecting secure parameters for 2SC: according the sponge construction, we first fix c such that $c/2$ is at least the desired security level, then choose the remaining parameters (n, ℓ, ω) accordingly.

We have implemented 2SC to test a large set of potential parameters for a number of security levels. In practice, optimal parameters for this scheme should also take into account these three main implementation-specific requirements: the ratio $\frac{\ell}{c}$, selecting an appropriate block size for the regular encoding, and the use of int-wise (rather than byte-wise) XORing. A large value of $\frac{\ell}{c}$ yields a large value of r, hence allowing for better performance. We implement the regular encoding such that it uses shift operations, thus efficiently using processor architecture. The choice $\log_2(n/\omega) = 16$ was the most promising block size in terms of the computation time in our implementation. Finally, int-wise XORing reduces computation time by four times compared to byte-wise XORing. Our parameters should thus ensure that we can perform int-wise XORing.

Putting everything together, the choice of ω, ℓ and c is a tradeoff decision. On the one hand, a small ω leads to fewer XOR operations during matrix multiplication. On the other hand, a small ω implies a small ℓ ($\ell = \omega \log_2(n/\omega)$). Making n large will help in increasing ℓ. But at the same time the matrix will become very big. Last but not least, the smaller c is chosen, the more efficient the computation is, because r becomes larger.

In order to compare the speed of 2SC with the speed of SYND [GLS07], we have optimized the implementations of SYND and 2SC proposed in [MCY11] with the same techniques using the parameter sets proposed in [GLS07] and [MCY11] respectively. As mentioned earlier, the results given in [GLS07] can not be checked, since no freely-available implementation of SYND exists. On our own implementations, we obtained the results presented in Table 3.3 and Table 3.8. For comparable security levels, 2SC runs faster than SYND. At the same time, SYND needs significantly larger key sizes compared to 2SC. However, 2SC suffers from the drawback of having to store large matrices. A graph showing comparison between SYND and 2SC performance for the same security levels is given in Figure 3.7. In this Figure, the red graph (resp. green graph) represents the interpolation curve that provides estimates for the performance of SYND (resp. of 2SC) with respect to the expected security level based on the optimal parameters we found. As one can see, the SYND's curve is linear, while the 2SC's curve is a quadratic function.

Security Level	n	ℓ	ω	Key/IV size (bits)	Speed (cycles/byte)
90	8192	256	32	128	30.27
170	8192	512	64	256	41.50
250	8192	1024	128	512	149.94

Table 3.3.: Performance of SYND using quasi-cyclic codes

Security Level	n	ℓ	ω	c	Key/IV size (bits)	Speed (cycles/byte)
90	$13 \cdot 2^{19}$	384	24	240	144	25.12
170	$7 \cdot 2^{17}$	544	34	336	208	33.22
250	$29 \cdot 2^{17}$	928	58	576	352	80.05

Table 3.4.: Performance of 2SC using quasi-cyclic codes

3.3. The XSYND Cipher

As seen in the previous section, the 2SC stream cipher outperforms the SYND cipher in terms of speed, but it requires huge storage capacity. This section presents our second contribution [MHC12], which shows how to improve considerably the performance the SYND cipher without using a regular

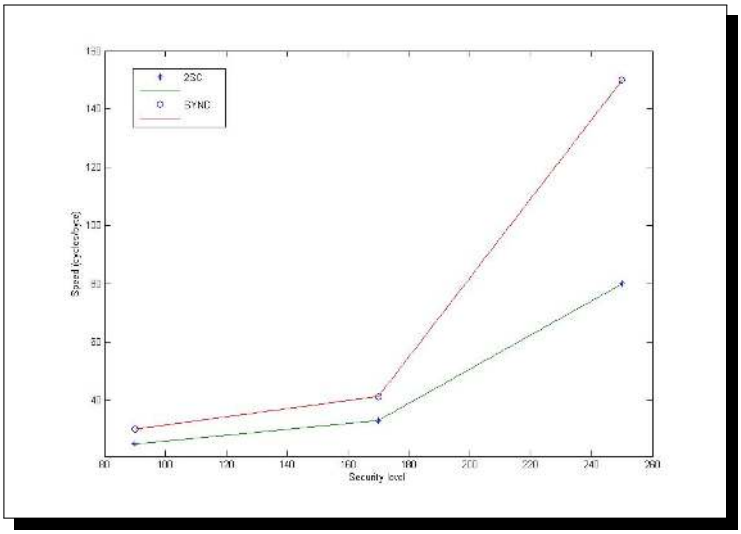

Figure 3.7.: A graphical performance comparison between 2SC and SYND.

encoder and without compromising the security of the modified SYND stream cipher. Our new proposal, called the eXtended SYND (in short XSYND), uses a generic state transformation which is also reducible to the regular syndrome decoding problem, but has better computational characteristics than the regular encoding. Moreover, unlike SYND, we show how the security reduction of our XSYND works.

3.3.1. Description of the XSYND cipher

For describing we will use the same notations as before. The XSYND is obtained from SYND by making the following modifications. Firstly, we modify the f function (Figure 3.2) such that it requires only two, rather than three function evaluations, without loss of security. We denote the new function by f' and depict it in Fig. 3.8. Note that this modification does not affect the recovery of the secret K or the initial vector IV. In fact, it is straightforward to prove that, given an initial state e_0 output by f', if an adversary can extract K and IV from e_0, it can also easily solve an instance $\mathrm{RSD}(n,\ell,\omega)$. The new function f' function is defined by:

$$f'(x) = y \oplus g_2(y); \quad y = x \oplus g_1(x); \quad \forall x = (K, IV) \in \mathbb{F}_2^{\ell/2} \times \mathbb{F}_2^{\ell/2}.$$

The second modification in XSYND is to avoid the regular encoding $x \mapsto \phi(x)$ in functions g_i $(i = 1,2)$ described in Figure 3.3 by using the Randomize-Then-Combine paradigm due to Bellare et al. [BGG94, BGG95, BM97]. This paradigm is shown in Figure 3.9. More precisely, given an input x consisting of ω blocks $x_1, x_2, \ldots, , x_\omega$, each block being α bits (where α is chosen at will), we first feed each block through a random function $\mathcal{F}_i$, obtaining an output y_i, i.e., $\mathcal{F}_i(x_i) = y_i$. The values $y_1, y_2, \cdots, y_\omega$ are then combined by bitwise XOR to produce the final output $y = y_1 \oplus y_2 \oplus \cdots \oplus y_\omega$.

In XSYND, we use the following function $\mathcal{F}_i$: let H be a random binary matrix of size $\omega\alpha \times \omega \cdot 2^\alpha$, consisting of ω submatrices $\mathsf{H}_1 \ldots \mathsf{H}_\omega$ of size $\omega\alpha \times 2^\alpha$ (we write $\mathsf{H} = \mathsf{H}_1 || \ldots || \mathsf{H}_\omega$). If we write the submatrices as $\mathsf{H}_i = (h_i^{(0)}, h_i^{(1)}, \ldots, h_i^{(2^\alpha - 1)})$, where $h_i^{(j)} \in \mathbb{F}^{\omega\alpha}$ for $j \in \{0, 1, \ldots, 2^\alpha - 1\}$, then we can

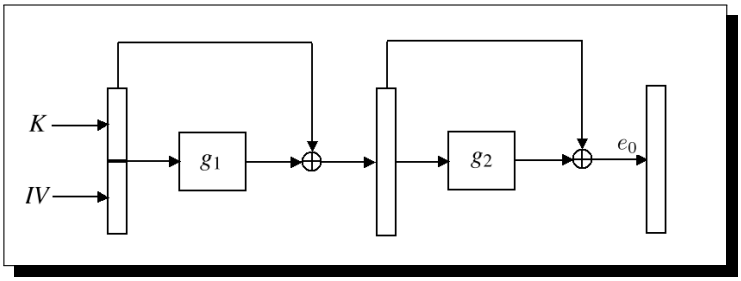

Figure 3.8.: The initialization function f' of XSYND

define $\mathcal{F}_i$ by $\mathcal{F}_i(x_i) = y_i = h_i^{(j)}$ if and only if the decimal value $< x_i >_\alpha$ of x_i is equal to j. We have 2^α possible value for each y_i, depending on the integer value of the block x_i. In this way, we redefine the functions g_i as follows:

$$g_1(x) = a_1^{(<x_1>_\alpha)} \oplus a_2^{(<x_2>_\alpha)} \oplus \cdots \oplus a_\omega^{(<x_\omega>_\alpha)} \text{ with } \mathsf{A} = \mathsf{A}_1||\ldots||\mathsf{A}_\omega \text{ and } \mathsf{A}_i = (a_i^{(0)}, a_i^{(1)}, \ldots, a_i^{(2^\alpha-1)})$$

$$g_2(x) = b_1^{(<x_1>_\alpha)} \oplus b_2^{(<x_2>_\alpha)} \oplus \cdots \oplus b_\omega^{(<x_\omega>_\alpha)} \text{ with } \mathsf{B} = \mathsf{B}_1||\ldots||\mathsf{B}_\omega \text{ and } \mathsf{B}_i = (b_i^{(0)}, b_i^{(1)}, \ldots, b_i^{(2^\alpha-1)})$$

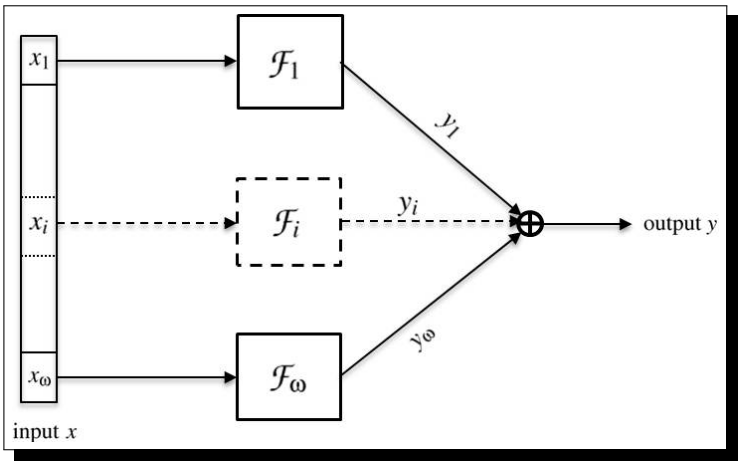

Figure 3.9.: Randomize-then-combine paradigm proposed in [BGG94]

Figure 3.10 illustrates the new function g_1 introduced in XSYND.

3.3.2. Security of XSYND

In this subsection we analyze the security of XSYND from both the theoretical and practical point of view. In the theoretical part, we show first that the security of the core mapping introduced in XSYND is directly reducible to the syndrome decoding problem. More precisely, this mapping can be expressed as a product of a parity check matrix by a regular word such that the security of our proposal is equivalent of that of SYND. After that, we prove that distinguishing the key stream generated by XSYND from truly random sequence is reducible to solving an instance of regular word. Our proof is

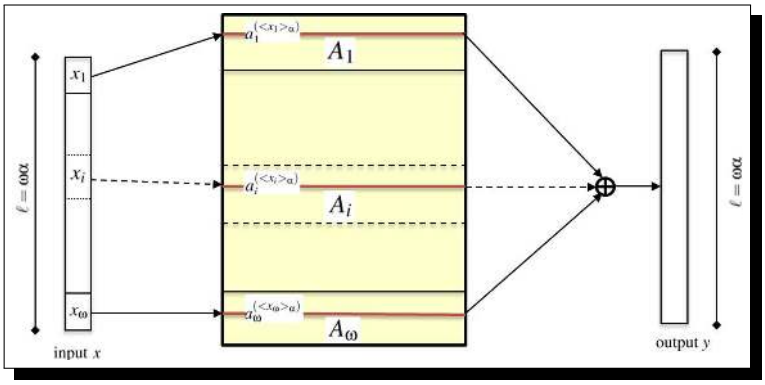

Figure 3.10.: The new functions g_1

mainly based on Goldreich-Levin Theorem. In the practical part, we analyze the security of XSYND faces the best known algorithms.

Theoretical Security. Our analysis is done in two steps. In the first step, we show that it is hard to recover the secret state x given $g_1(x)$ and $g_2(x)$. More precisely, we show that inverting $g_i(x)$ is reducible to the RSD problem. In the second step, we prove that XSYND is a pseudo-random generator, meaning that the key stream produced by XSYND is indistinguishable from truly random sequences.

Step 1: We consider general transformations $\mathcal{T}$ defined as:

$$\mathcal{T}(x) = a_1^{(<x_1>_\alpha)} \oplus a_2^{(<x_2>_\alpha)} \oplus \cdots \oplus a_\omega^{(<x_\omega>_\alpha)},\ \forall x = (x_1, \ldots, x_\omega) \in \mathbb{F}_2^{\omega\alpha}.$$

In this transformation, $a_i^{(j)}$ for $j = 0, \ldots, 2^\alpha$ is the $(j+1)^{th}$ column of the i^{th} submatrix A_i of a random binary matrix A of size $\omega\alpha \times \omega 2^\alpha$. Note that both g_1 and g_2 are particular instantiations of $\mathcal{T}$, for random matrices A and B. Our argument in this step is as follows: we first show that (1) for each x there exists a regular word z such that $\mathcal{T}(x) = \mathsf{A} \cdot z^\top$, then prove that (2) learning x from $y = \mathcal{T}(x)$ is equivalent to finding a regular word z such that $\mathsf{A} \cdot z^\top = y$ (this is an instantiation of RSD(n, ℓ, ω) for $\ell = \omega\alpha$ and $n = \omega 2^\alpha$). Thus, under the RSD assumption, the modified XSYND protocol security can be reduced to the hardness of RSD.

First consider (1). We write $\mathsf{A} = \mathsf{A}_1 | \ldots | \mathsf{A}_\omega$ as in the previous subsection, for $\omega\alpha \times 2^\alpha$ submatrices A_i. Each submatrix has columns $a_i^{(0)}, \ldots, a_i^{(2^\alpha - 1)}$. We note that any regular word z is in fact a word of length $n = \omega 2^\alpha$ and weight ω, whose intger entries $z_1, \ldots, z_\omega$ indicate the positions of its non-zero entries (and each z_i is a unique value between $(i-1)2^b + 1$ and $i2^b$ since the word is regular). Let $x' = (x'_1, \ldots, x'_\omega)$ be a state in decimal notation of the ℓ-bit vector x, i.e., $x'_i =< x_i >_\alpha$ for $i = 1, \cdots, \omega$. We associate each x' with a value z whose decimal notation is $(z_1, \ldots, z_\omega)$ for $z_i = (x'_i + 1) + (i-1)2^\alpha$. The reverse transformation of z to x' is obtained as follows:

$$\begin{cases} x'_1 \equiv z_1 - 1 \pmod{2^\alpha} \\ x'_2 \equiv z_2 - 1 \pmod{2^\alpha} \\ \cdots \quad \cdots \quad \cdots \quad \cdots \\ x'_\omega \equiv z_\omega - 1 \pmod{2^\alpha} \end{cases}$$

It is easy to check that:

$$\mathsf{A}\cdot z^{\top} = a_1^{(<x_1>_\alpha)} \oplus a_2^{(<x_2>_\alpha)} \oplus \cdots \oplus a_\omega^{(<x_\omega>_\alpha)} = \mathcal{T}(x).$$

Toy Example. Let us consider $\omega = 3$ and $\alpha = 2$. Then the matrix A should be $(3\cdot 2)\times(3\cdot 2^2) = 6\times 12$ and binary. Consider in this example the following matrix A:

$$\mathsf{A} = \left[\begin{array}{cccc|cccc|cccc} a_1^{(0)} & a_1^{(1)} & a_1^{(2)} & a_1^{(3)} & a_2^{(0)} & a_2^{(1)} & a_2^{(2)} & a_2^{(3)} & a_3^{(0)} & a_3^{(1)} & a_3^{(2)} & a_3^{(3)} \\ \hline 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \end{array}\right]$$

Let us consider a state x' in decimal form, with $x' = (2,1,0)$, corresponding to $x = [100100]$. Compute z in decimal form according to the formula $z_i = (x'_i + 1) + (i-1)2^\alpha$. Thus $z_1 = 3, z_2 = 6$, and $z_3 = 9$. In binary notation, z_i denotes the positions of z's non-zero entries, i.e. $z = [0010|0100|1000]$. We can now verify that for this z we have

$$\mathcal{T}(x) = a_1^{(2)} \oplus a_2^{(1)} \oplus a_3^{(0)} = [001111] = \mathsf{A}\cdot z^{\top}.$$

Now let us consider the security reduction of general transformations $\mathcal{T}$ to the RSD problem, i.e. step (2) outlined above. We have shown that for each input value x we can find a regular word z of weight ω such that $\mathsf{A}\cdot z^{\top} = \mathcal{T}(x)$. Assume that there exists an adversary that can invert $\mathcal{T}(x)$, i.e. given $y = g(x)$, the adversary outputs x. Then the same adversary computes z as above and can thus, given a matrix A, and a value $y = \mathcal{T}(x) = \mathsf{A}\cdot z^{\top}$, this adversary can output the regular word z. This is exactly an instantiation of $\mathrm{RSD}(n,\ell,\omega)$ for $\ell = \omega\alpha$ and $n = \omega 2^\alpha$. In conclusion, we can reduce the security of XSYND to the hardness of the RSD problem.

Step 2: In this step, we prove that XSYND is a pseudo-random generator. Our proof is an adaption of that given for the Fischer-Stern's PRNG [FS96]. We will show that if there exists an algorithm that is able of distinguishing a random bit string from the output of the mapping $x \to (g_1(x), g_2(x))$, then this algorithm can be converted into a predicator that can predicts the inner product of an input x and a random bit string chosen at random. Before doing so, we state the following assumptions.

1. *Indistinguishability (A1):* The binary matrices A and B (both of size $\ell\times n$) are computationally indistinguishable from uniform matrices of the same dimensions.
2. *Regular syndrome decoding (RSD) (A2):* The family of mappings defined as $g_{\mathsf{M}}(z) = \mathsf{M}\cdot z^{\top}$ for an uniform $2\ell\times n$ binary matrix M is one-way on the set of all regular words of length n and weight ω.

As shown before, the mapping $x \to g_1(x)$ (resp. $x \to g_2(x)$) can be regarded as $\mathcal{F}_u(z) = \mathsf{A}\cdot z^{\top}$ (resp. $\mathcal{F}_o(z) = \mathsf{B}\cdot z^{\top}$), where A and B are binary matrices, both of size $\ell\times n$, and z is taken from the set of

regular words and related to the input x. Therefore, from now on, we will use $\mathcal{F}_u$ (resp. $\mathcal{F}_o$) instead of g_1 (resp. g_2).

From A and B we create a $2\ell \times n$ block matrix M by stacking them vertically, i.e.

$$\mathsf{M} = \left(\frac{\mathsf{A}}{\mathsf{B}}\right)$$

In this case, we can write the mapping $x \to (g_1(x), g_2(x))$ as $g_{\mathsf{M}}(z) = \mathsf{M} \cdot z^{\top} = (\mathcal{F}_u(z), \mathcal{F}_o(z))$. Consequently, in order to prove that XSYND is a pseudo-random generator, it is sufficient to prove that the output of $z \to g_{\mathsf{M}}(z)$ is pseudo-random as proved in [FS96]. Our proof is based on the Goldreich-Levin Theorem [GL89], which says that, for any one-way function, the inner product of its argument and a randomly chosen bit string is a hardcore bit (or hardcore predicate). Recall that the inner product of two bit strings a and b (of the same size) is defined by

$$\langle a, b \rangle = \sum_i a_i b_i \mod 2.$$

Formally, this theorem can be stated as follows:

Theorem 3.3.1 (Goldreich-Levin theorem [GL89])**.** *Let $f : \mathbb{F}_2^{\lambda(n)} \to \mathbb{F}_2^{\mu(n)}$ be a one-way function. For every PPT algorithm $\mathcal{A}$, for all polynomials p and all but finitely many n's,*

$$\Pr[\mathcal{A}(f(x), \nu) = \langle x, \nu \rangle] \leq \frac{1}{2} + \frac{1}{p(n)}$$

where the probability is taken over x uniformly chosen x and $\nu \in \mathbb{F}_2^{\lambda(n)}$.

Using this theorem we now prove that that XSYND is a pseudo-random generator.

Theorem 3.3.2. *Suppose n, ℓ, and ω are chosen such that the indistinguishability and the regular syndrome decoding assumptions hold. Then the output distribution of XSYND is computationally indistinguishable from a truly random distribution. That is, XSYND is a pseudo-random generator.*

Proof. Our proof is by contradiction. Let us assume that an 2ℓ-bit output of the mapping $g_{\mathsf{M}}(z) = \mathsf{M} \cdot z^{\top}$ is not pseudo-random, and there exists a distinguisher $\mathcal{D}$, which is capable to differentiate this output of from a 2ℓ-bit random string υ. More precisely, $\mathcal{D}$ takes as input $2\ell \times n$ binary random matrix M and a random $\upsilon \in \{0,1\}^{2\ell}$ as a candidate being equal to $\mathsf{M} \cdot z^{\top}$ for some unknown regular word z. In the event that $\mathsf{M} \cdot z^{\top} = \upsilon$, $\mathcal{D}$ outputs 1 with probability above $\frac{1}{2} + \frac{1}{p(n)}$, for every polynomial $p(n)$. Otherwise, when υ is chosen uniformly from $\{0,1\}^{2\ell}$, $\mathcal{D}$ outputs 1 with probability at most $\frac{1}{2}$. Formally, the distinguisher $\mathcal{D}$ behaves as follows:

$$\begin{cases} \Pr[\mathcal{D}(\mathsf{M}, \upsilon) = 1] \geq \frac{1}{2} + \frac{1}{p(n)}, & \text{if } \upsilon = \mathsf{M} \cdot z^{\top}, \text{ for some regular word } z \\ \Pr[\mathcal{D}(\mathsf{M}, \upsilon) = 1] < \frac{1}{2}, & \text{if } \upsilon \text{ is taken uniformly from } \{0,1\}^{2\ell} \end{cases}$$

As next step, we will build an algorithm $\mathcal{P}$, which uses the distinguisher $\mathcal{D}$ as subroutine. This algorithm will predicts the inner product $\langle z, \nu \rangle$ with probability at least $\frac{1}{2} + \frac{1}{2p(n)}$, where z is an unknown regular word (an input of g_{M}) and ν a randomly chosen n-bit string. To this end, let write $\nu = (\nu_1, \cdots, \nu_n)$. In addition, let σ be the number of the positions j such that where $z_i = \nu_j = 1$, i.e. the size of the intersection $z \cap \nu$ and ρ its parity, i.e. the inner product $\langle z, \nu \rangle$. Then the algorithm $\mathcal{P}$ takes as input $g_{\mathsf{M}}(z)$ and ν and executes the following steps:

- Select a random $\rho' \in \{0,1\}$ as candidate to ρ
- Choose randomly $\xi \in \{0,1\}^{2r}$
- Build a new $2\ell \times n$ binary matrix $\widehat{\mathsf{M}} = (\widehat{m}_1, \cdots, \widehat{m}_n)$ such that for every $j \in \{1, \cdots, n\}$ it holds

$$\widehat{m}_j = \begin{cases} m_j + \xi & \text{if } \nu_j = 1, \\ m_j & \text{if } \nu_j = 0 \end{cases}$$

- Feed the distinguisher with $\widehat{\mathsf{M}}$ and $g_{\mathsf{M}}(z) + \rho' \cdot \xi$
- If the distinguisher outputs 1, then output $\rho' = \rho$. Otherwise, output the opposite of ρ'.

Now, we show next that $\mathcal{P}$ predicts the inner product $\langle z, \nu \rangle$ with probability above $\frac{1}{2} + \frac{1}{2p(n)}$. We have to consider two events:

(1) $\mathtt{E}_1$:"ρ is guessed correctly". Then the prognosticated value for the inner product $\langle z, \nu \rangle$ is correct if the distinguisher outputs 1. The distribution seen by the distinguisher on $(\widehat{\mathsf{M}}, g_{\mathsf{M}}(z) + \rho' \cdot \xi)$ is identical to the distribution on input $(\mathsf{M}, g_{\mathsf{M}}(z))$. By construction, this is the case with probability at least $\frac{1}{2} + \frac{1}{p(n)}$.

(2) $\mathtt{E}_2$:"ρ is not guessed correctly". The distinguisher receives uniformly distributed inputs because of the randomness of ξ. It then returns 1 with probability $\frac{1}{2}$.

Since $\Pr[\mathtt{E}_1] = \Pr[\mathtt{E}_2] = \frac{1}{2}$, we conclude that the overall probability of correctly predicting the inner product $\langle z, \nu \rangle$ is at least $\frac{1}{2} + \frac{1}{2p(n)}$. This contradicts the Theorem 3.3.1 because of the RSD assumption.

□

Practical Security This section presents what are provably the most generic attacks against XSYND. We will only address the hardness of inverting the mapping $\mathcal{T}$ defined in the previous section, since this is the main building block of XSYND design. If an attacker can invert $\mathcal{T}$, then she can recover the secret key and recover inner states.
In what follows, we denote by $\mathtt{WF}_Y(n, \ell, \omega)$ the work factor (i.e. number of binary operations) required to solve the instance $\mathrm{RSD}(n, \ell, \omega)$ by using an algorithm Y. Furthermore, in estimating the complexity of each attack against XSYND we use $\ell = \omega\alpha$ with $\alpha = \log_2\left(\frac{n}{\omega}\right)$.
There are essentially three types of known attacks that are applicable to XSYND:

1. **Linearization Attacks.** There are two types of linearization attacks that are relevant for XSYND, namely the Bellare-Micciancio (BM) attack [BM97] against the XHASH function [BM97], and the attack due to Saarinen [Saa07]. We discuss these attacks below.

   (a) The Bellare-Micciancio's attack (BM). This is a preimage attack proposed by Bellare and Micciancio [BM97] against the so-called XHASH mapping. This attack relies on finding a

linear dependency among ω ℓ-bit vectors, where ω is the number of vectors XORred together and ℓ, the length (in bits) of the target value. This is likely to succeed if the value ω is close to ℓ. More precisely, let l and k be two positive integers. Let f be a random function with $f : \mathbb{F}_2^l \mapsto \mathbb{F}_2^\ell$. Let $[i]$ denote the binary representation of an integer i. Based on f, the XHASH mapping is defined as

$$\text{XHASH}(x) = f([1]|x_1) \oplus \cdots \oplus f([\omega]|x_\omega), \text{ with } x = (x_1, x_2, \ldots, x_\omega).$$

The BM attack finds a preimage x of a given $z = \text{XHASH}(x) \in F_2^\ell$ as follows. First, one finds ω-bit string $y = (y_1, \ldots, y_\omega)$, with $y_i \in \mathbb{F}_2$, such that $\text{XHASH}(x^y) = z$, where $x^y = x_1^{y_1} \ldots x_\omega^{y_\omega}$. To achieve this, one first computes 2ω values $\beta_i^k = f([i]|x_i^j)$ for $k \in \{0,1\}$ and $i \in \{1, \ldots, \omega\}$; the next step is to try to solve the following system of equations over $\mathbb{F}_2$ using linear algebra:

$$\begin{cases} y_i \oplus \bar{y}_i = 1, \quad i \in \{1, \ldots, \omega\}, \\ \oplus_{i=1}^{\omega} \beta_i^0(j) y_i \oplus \beta_i^1(j) \bar{y}_i = z(i), \quad j \in \{1, \ldots, \ell\}. \end{cases}$$

Here, $\beta_i^0(j)$ (resp. $\beta_i^1(j)$) denotes the j^{-th} bit of β_i^0 (resp. β_i^1) and $\bar{y}_i = 1 - y_i$ are the unknowns. This system has $\ell + \omega$ equations in $2w$ unknowns and is easy to solve when $w = r + 1$. More generally, it was shown in [BM97] that for all $y \in \mathbb{F}_2^\omega$ the probability to have $\text{XHASH}(x^y) \neq z$ is at most $2^{\ell-\omega}$. That is, the complexity of inverting XHASH is at least $2^{\ell-\omega}$; in our notation,

$$\mathtt{WF}_{\text{BM}}(n, \ell, \omega) \geq 2^{\ell-\omega} = 2^{(\alpha-1)\omega}.$$

(b) The Saarinen's attack (SA). This attack is due to Saarinen [Saa07] and it was proposed against the FSB [AFS05] hash function. The main idea behind this attack is reducing the problem of finding collisions or preimages to that of solving systems of equations. This attack is very efficient when $\ell < 2\omega$. We briefly show how this attack works in our setting, where we must invert the map $\mathcal{T}$.

As shown in section 5.1, $\mathcal{T}(x) = \mathsf{A} \cdot z^\top$, where A is the random binary matrix of size $\ell \times n$, whose entries define $\mathcal{T}$, and z is a regular word of length n and weight ω. We can in turn write $\mathsf{A} \cdot z^\top$ out as follows:

$$y = \oplus_{i=1}^{\omega} a_{(i-1)\frac{n}{\omega} + x_i + 1}, \; 0 \leq x_i \leq \frac{n}{\omega}, \tag{3.10}$$

where $x = (x_1, \ldots, x_\omega)$ and a_j denotes the j^{-th} column of A. For simplicity, assume that $x_i \in \{0,1\}$. In this case, we define a constant ℓ-bit vector c and an additional $\ell \times \omega$ binary matrix H as follows.

$$c = \oplus_{i=1}^{\omega} a_{(i-1)\frac{n}{\omega}+1}, \; \mathsf{A} = [b_1 \cdots b_\omega] \text{ with } b_i = a_{(i-1)\frac{n}{\omega}+1} \oplus a_{(i-1)\frac{n}{\omega}+2}. \tag{3.11}$$

It is easy to check that $y = B \cdot x + c$. As a consequence if $\ell = \omega$, then H is square and we can find the preimage x from y as:

$$x = B^{-1} \cdot (y \oplus c), \tag{3.12}$$

where B^{-1} denotes the inverse of H. Note that this inverse exists with probability without proof of $\prod_{i=1}^{\ell}(1 - 1/2^i) \approx 0.29$ for ℓ moderately large. The expected complexity of this attack is the the workload of inverting H, which is al most $0.29 \cdot \ell^3$. It has been proved in [Saa07] that the same complexity is obtained even if $\ell \leq 2\omega$.

In the opposite direction, Saarinen also extended his attack for the case when $\omega \leq \ell/\theta$ for $\theta > 1$ and $x_i \notin \{0,1\}$. In this case, the complexity is about $2^\ell/(\theta+1)^w$. Moreover, the recent result [BLPS11b] shows that if $\theta = 2\beta$, for $\beta > 1$, this complexity becomes $2^\ell/(\beta+1)^{2\omega}$. As consequence we obtain:

$$\mathtt{WF}_{\mathrm{SA}}(n,\ell,\omega) \geq \begin{cases} 2^\ell/(\theta+1)^\omega & \text{if } \omega \leq \ell/\theta \\ 2^r/(\theta+1)^{2\omega} & \text{if } \omega \leq \ell/2\theta \end{cases}$$

which can be rewritten in our setting as:

$$\mathtt{WF}_{\mathrm{SA}}(n,\ell,\omega) \geq \begin{cases} (\frac{2^\alpha}{\theta+1})^\omega & \text{if } \theta \leq \alpha \\ (\frac{2^\alpha}{(\theta+1)^2})^\omega & \text{if } \theta \leq \alpha/2 \end{cases}$$

2. **Generalized Birthday Attacks (GBA).** This class of attacks attempt to solve the following, so-called k-sum problem: given k random lists $L_1, L_2, \ldots, L_k$ of ℓ-bit strings selected uniformly and independently at random, find $x_1 \in L_1, x_2 \in L_2, \ldots, x_k \in L_k$ such that $\oplus_{i=1}^k x_i = 0$. For $k = 2$, a solution can be found in time $2^{\ell/2}$ using the standard birthday paradox. For $k > 2$ Wagner's algorithm [Wag02] and its extended variants [AFS05, Ber07, MS09, FS09] can be applied. When $k = 2^{j-1}$ and $|L_i| > 2^{\ell/j}$, Wagner's algorithm can find at least one solution in time $2^{\ell/j}$.

Let us explain the main idea behind a GBA algorithm for $k = 4$. Let $L_1, \ldots, L_4$ be four lists, each of length $2^{\ell/3}$. The algorithm proceeds in two iterations. In the first iteration, we build two new lists $L_{1,2}$ and $L_{3,4}$. The list $L_{1,2}$ contains all sums $x_1 \oplus x_2$ with $x_1 \in L_1$ and $x_2 \in L_2$ such that the first $\ell/3$ bits of the sum are zero. Similarly, $L_{3,4}$ contains all sums $x_3 \oplus x_4$ with $x_3 \in L_3$ and $x_4 \in L_4$ such that the first $\ell/3$ bits of the sum are zero. So the expected length of $L_{1,2}$ is equal to $2^{-\ell/3} \cdot |L_1| \cdot |L_2| = 2^{\ell/3}$. Similarly, the expected length of $L_{3,4}$ is also $2^{\ell/3}$. In the second iteration of the algorithm, we construct a new list L'_1 containing all pairs $(x'_1, x'_2) \in L_{1,2} \times L_{3,4}$ such that the first $\ell/3$ bits of the sum $x'_1 \oplus x'_2$ are zero. Then the probability that $x'_1 \oplus x'_2$ equals zero is $2^{-2\ell/3}$. Therefore, the expected number of matching sums is $2^{-2\ell/3} \cdot |L_{1,2}| \cdot |L_{3,4}| = 1$. So we expected to find a solution. This idea can be generalized for $k = 2^{j-1}$ by repeating the same procedure $j-2$ times. In each iteration a, we construct lists, each containing $2^{\ell/j}$ elements that are zero on their first $a\ell/j$ bits, until obtaining, on average, one ℓ-bit element with all entries equal to 0.

We estimate the security of XSYND against GBA attacks by using the GBA algorithm from [FS09]. This algorithm attempts to find a set of indices $I = \{1, 2, \cdots, 2^\gamma\}$ satisfying $\oplus_{i \in I} \mathsf{H}_i = 0$, where H_i are columns of the matrix H. As shown in [FS09], the algorithm is applicable when $\binom{2^\alpha \omega}{2^{(1-\gamma)}\omega} \geq 2^{\alpha\omega+\gamma(\gamma-1)}$. Under this condition, the cost of solving an instance RSD problem with parameters (n,ℓ,ω) is given by:

$$\mathtt{WF}_{\mathrm{GBA}}(n,\ell,\omega) \geq \left(\frac{\omega\alpha}{\gamma} - 1\right) 2^{\frac{\omega\alpha}{\gamma}-1}.$$

Note that the recent result in [NCB11] shows that the time and memory efficiency of GBA attacks can be improved, but only by a small factor. This improvement is taken into account when proposing parameters for XSYND in the following subsection.

3. **Information Set Decoding (ISD).** ISD is one of the most important generic algorithm for decoding errors in an arbitrary linear code. An ISD algorithm consists (in its simplest form) in finding a valid, so-called information set, which is a subset of k error-free positions amongst the n positions of each codeword. Here, k is the dimension and n the length of the code. The validity of this set is checked by using Gaussian elimination on the $\ell \times n$ parity check matrix H. If we denote by $p(n,\ell,\omega)$ the probability of finding a valid information set and by $c(\ell)$ the cost of Gaussian elimination, then the overall cost of ISD algorithms equals the ratio $c(r)/p(n,\ell,\omega)$.

   In the following, we estimate the cost of finding a solution to the regular syndrome decoding (RSD) problem, i.e. we wish to invert the map $\mathcal{T}$. Let $n_s(n,\ell,\omega)$ be the expected number of solutions of RSD instance. This quantity is:

$$n_s(n,\ell,\omega) = \frac{\binom{n}{\omega}^w}{2^\ell} = 1,$$

   because $\ell = \omega \log_2(\frac{n}{\omega})$. In addition, let $p_v(n,\ell,\omega)$ be the probability that a given information set is valid for one given solution of RSD. As shown in [AFS05], $p(n,\ell,\omega)$ can be approximated by: $p(n,\ell,\omega) \approx p_v(n,\ell,\omega) \cdot n_s(n,\ell,\omega)$.

   Furthermore, as shown in [AFS05], $p_v(n,\ell,\omega)$ is given by:

$$p_v(n,\ell,\omega) = \binom{r}{n}^\omega = \left(\frac{\log_2(n/w)}{n/\omega}\right)^\omega$$

   We thus conclude that the probability of selecting a valid set to invert RSD is equal to: $p(n,\ell,\omega) = \binom{\alpha}{2^\alpha}^\omega$.
   Hence, the cost $\mathtt{WF}_{\mathrm{ISD}}(n,\ell,\omega)$ of solving an instance of RSD with parameters (n,ℓ,ω) is approximately:

$$\mathtt{WF}_{\mathrm{ISD}}(n,\ell,\omega) \approx c(\ell) \cdot \left(\frac{2^\alpha}{\alpha}\right)^\omega . \tag{3.13}$$

   If we assume that the complexity of Gaussian elimination is ℓ^3, then $\mathtt{WF}_{\mathrm{ISD}}(n,\ell,\omega)$ becomes:

$$\mathtt{WF}_{\mathrm{ISD}}(n,\ell,\omega) \approx (\omega\alpha)^3 \cdot \left(\frac{2^\alpha}{\alpha}\right)^\omega . \tag{3.14}$$

   In practice, we use the lower bound for ISD algorithms presented in [MMT11] to estimate the security of XSYND faces ISD attacks and show our results in Table 3 .

Remark 3.3.1. *One could also use Time Memory trade-off attacks against stream ciphers. This attack was first introduced in [Hel80] as a generic method of attacking block ciphers. To make this attack unfeasible, one must adjust the cipher parameters as shown in [Gol97, HS05], i.e., the initial vector should be at least as large as the key, and the state should be at least twice the key.*

Table 3 briefly summarizes the expected complexity of the previous attacks against XSYND.

Attack	The binary logarithm of the complexity: $\log_2(\mathtt{WF}_{(.)}(n,\ell,\omega))$ with $\ell = \omega\alpha$ and $n = \omega 2^{\alpha}$
BM	$\omega(\alpha-1)$
SA	$\begin{cases}\omega(\alpha-\log_2(\theta+1)), & \text{if } \theta \leq \alpha \\ \omega(\alpha-2\log_2(\theta+1)), & \text{if } \theta \leq \alpha/2\end{cases}$
GBA	$\omega\alpha/\gamma+\log_2(\omega\alpha/\gamma-1)-1$ for $\gamma \in \mathbb{N}$
ISD	$\omega(\alpha-\log_2(\alpha))+3\log_2(\omega\alpha)$

Table 3.5.: The estimated complexities of possible attacks against XSYND.

3.3.3. Parameters and Experimental Results

Taking into account all the previous attacks against XSYND, we select 'optimal' parameters that offer both a desired level of security and a satisfying efficiency. First, we choose $\ell = 2|IV|$ and $|IV| = |K|$ to avoid the Time Memory Trade-Off attacks according [Gol97, HS05]. For efficiency reasons we then fix $\alpha = \log_2(n/\omega) = 8$ and for each security level λ we vary ω to obtain both high performance and a complexity of solving the RSD problem of at least 2^{λ}.

We have tested a large set of potential parameters for a number of security levels. Table 3.6 presents the optimal parameter sets (n,ℓ,ω) resulted from running our implementation for several security levels. Note that in our implementation, we only use random binary codes without any particular structure. But it is possible to find parameters providing the same security levels when the parity check matrix is quasi-cyclic as in [GLS07]. In this case, ℓ has to be a prime and 2 is primitive root of the finite field $\mathbb{F}_r^*$ in order to guarantee the randomness property of quasi-cyclic codes as demonstrated in [GZ07].

Table 3.6.: Proposed parameters for XSYND.

Security Level	n	ℓ	ω	Key/IV size (bits)	Speed of XSYND (cycles/byte)
90	8192	256	32	128	14.92
120	12288	384	48	192	16.98
160	16384	512	64	256	35.40
200	20480	640	80	320	43.68
240	24576	768	96	384	55.42
280	28672	896	112	448	77.09

The results shown in Table 3.6 are for a pure C/C++ implementation with additional use of C/C++-Intrinsics). The operating system was Debian 6.0.3, the source has been compiled with gcc (Debian 4.4.5-8) 4.4.5. All results have been gained on an AMD Phenom(tm) 9950 Quad-Core Processor, running at a clock rate of 1300 MHz. Due to the row-major convention of C/C++, the two matrices A resp. B have been used and stored in transposed form. In order to compare the speed of XSYND with the claimed speed of SYND [GLS07] and 2SC [MCY11] (Table 3.8), we have tested our implementation using the parameter sets suggested in [GLS07]. Our results presented in Table 3.7 show that, for comparable security levels (90 and 250), XSYND runs faster than SYND [GLS07] and 2SC cipher [CSM], but its speed is compara-

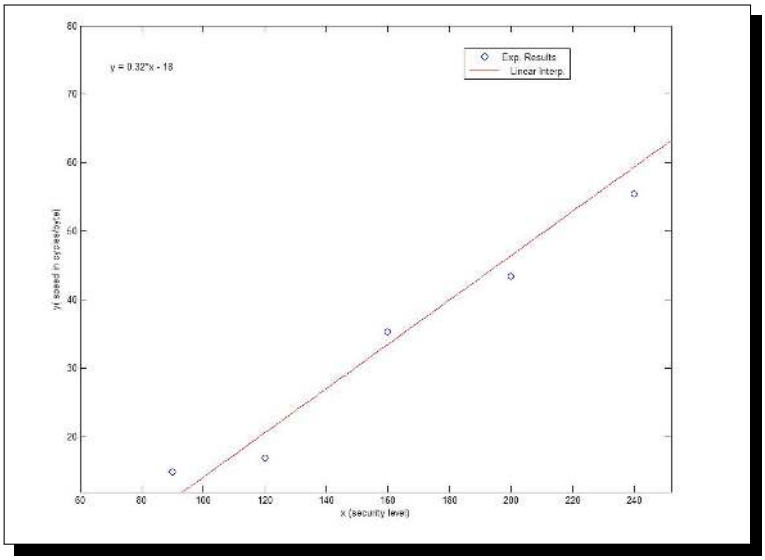

Figure 3.11.: The behavior of speed of XSYND in function of security level.

ble to that of 2SC for 170-bit security.

Security Level	n	ℓ	ω	key/IV size (bits)	speed of SYND (cycles/byte) [GLS07] — [CSM]	speed of XSYND (cycles/byte)
90	8192	256	32	128	27 — 30.27	14.92
170	8192	512	64	256	53 — 41.05	35.18
250	8192	1024	128	512	83 — 149.94	55.69

Table 3.7.: Performance of XSYND compared to that of SYND using the same parameters in [GLS07].

Table 3.8.: Parameters and performance of 2SC cipher given in [CSM].

Security Level	n	ℓ	ω	key/IV size (bits)	Speed (cycles/byte)
90	1572864	384	24	144	25.12
170	2228224	544	34	208	33.22
250	3801088	928	58	352	80.05

3.4. The PSYND Cipher

3.4.1. Motivation

From the description of the XSYND stream cipher and its original variant XSYND, it is straightforward to see that for producing a key stream block of SYND, the mapping g_2 first has to wait for

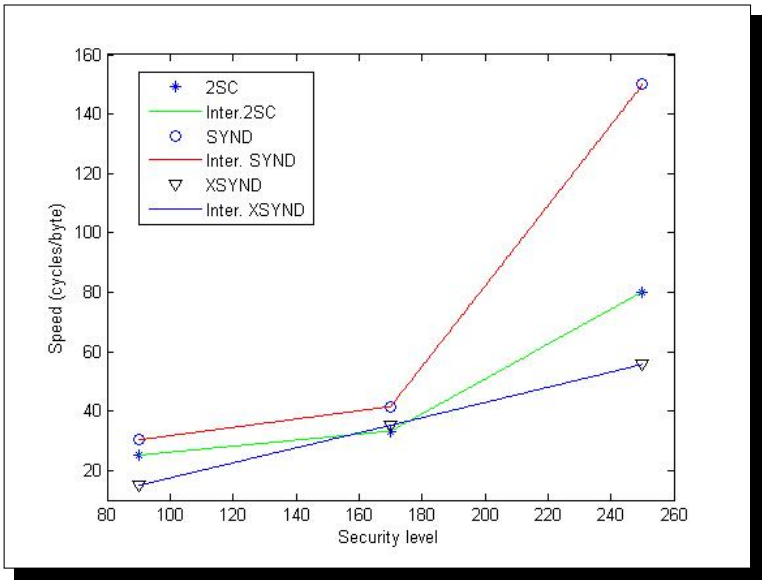

Figure 3.12.: Speed comparison between SYND , 2SC and XSYND.

function evaluation of g_1 every time. This a bottleneck of this two schemes, because it makes the whole process not fully parallelizable using the state-of-the art programming techniques. The main purpose of this section is to show how to design a SYND-like stream cipher that can be completely implemented in a parallel manner, and hence has better computationally features than the XSYND stream cipher, and can be furthermore shown to be provably secure at the same time.

3.4.2. Description of the PSYND cipher

In this subsection we provide a detailed description of the PSYND cipher and its basic ingredients. The letters in its name stand for "Parallel SYND".

As in XSYSND and SYND, the PSYND is a synchronous stream cipher and parameterized by a set of positive integers (n, ℓ, ω) satisfying $\ell = \omega\alpha$, where $\omega < \ell < n$, and $\alpha = \log(n/\omega)$. This set determines the size of an internal state, the key length, and the size of an initial vector (IV). For security reasons, the IV has the same length as the secret key, both have $\ell/2$ bits. The PSYND is composed of two major blocks, `Initializer`, and `Generator`, both implicitly use two transformations G_1 and G_2 that are similar to the mappings g_1 and g_2 of the XSYND stream cipher, respectively. (See Figure 3.10).

`Initializer`: $F : \mathbb{F}_2^{|K|} \times \mathbb{F}_2^{|IV|} \to \mathbb{F}_2^{\ell} \times \mathbb{F}_2^{\ell}$.
It takes as input the initialize vector IV and the key K in order to calculate the initial state $s_0 = (x_0, y_0)$, according the following steps:

$$\begin{cases} \lambda = K \| IV \\ \beta = \lambda \oplus G_1(\lambda) \\ \gamma = \lambda \oplus G_2(\lambda) \\ F(\lambda) = (\beta \oplus G_2(\beta), \gamma \oplus G_1(\gamma)) = (x_0, y_0) = s_0 \end{cases}$$

During the initialization stage no output bits are returned, only after s_0 is produced, keystream generation is allowed to take place. A graphical illustration of F is presented in Figure 3.13.

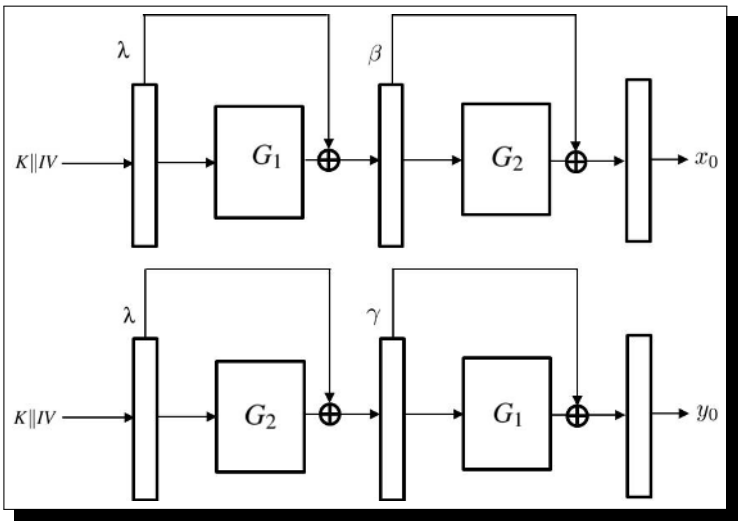

Figure 3.13.: Block diagram of Initializer F

Generator: $\mathtt{G} : \mathbb{F}_2^\ell \times \mathbb{F}_2^\ell \to \mathbb{F}_2^\ell \times \mathbb{F}_2^\ell$.
It takes as input the initial state s_0 generated by F, and produces an 2ℓ-bit string $(u_i, v_i), i \geq 1$ in each round. A simple block diagram of this generator is shown in Figure 3.14.
The creation of $\{s_{i+1}\}_{i \geq 0}$ works as follows. Starting with $s_i = (x_i, y_i)$, the generator $\mathtt{G}$ outputs (u_i, v_i) and updates s_i (or computes the subsequent state $s_{i+1} = (x_{i+1}, y_{i+1})$) by executing the following computations (in parallel):

$$x_{i+1} \leftarrow \phi_1(x_i) \text{ and } y_{i+1} \leftarrow \phi_2(y_i) \quad \texttt{(STEP 1)} \tag{3.15}$$
$$v_i \leftarrow \phi_2(x_i) \text{ and } u_i \leftarrow \phi_1(y_i) \quad \texttt{(STEP 2)} \tag{3.16}$$

During this process, only u_i and v_i are returned as output and made visible to an adversary.

Note that G_1 and G_2 are called at the same time with different inputs, so that the implementation of G can be fitted to the parallelism of modern CPUs. The whole process of the keystream generation of PSYND can be described by a single function $\mathtt{h}$ that takes as input a 2ℓ-bit string and expands it into a 4ℓ-bit string in each iteration. This function is defined as follows:

$$\mathtt{h}(x,y) := \mathtt{f}(x,y) \| \mathtt{g}(x,y) \in \mathbb{F}_2^{4\ell}, \tag{3.17}$$

where $(x,y) \to \mathtt{f}(x,y) := (G_1(x), G_2(y))$ is a update function, that refreshes the current state (x,y), while $(x,y) \to \mathtt{g}(x,y) := (G_1(y), G_2(x))$ is an output function producing an 2ℓ-bit string which form

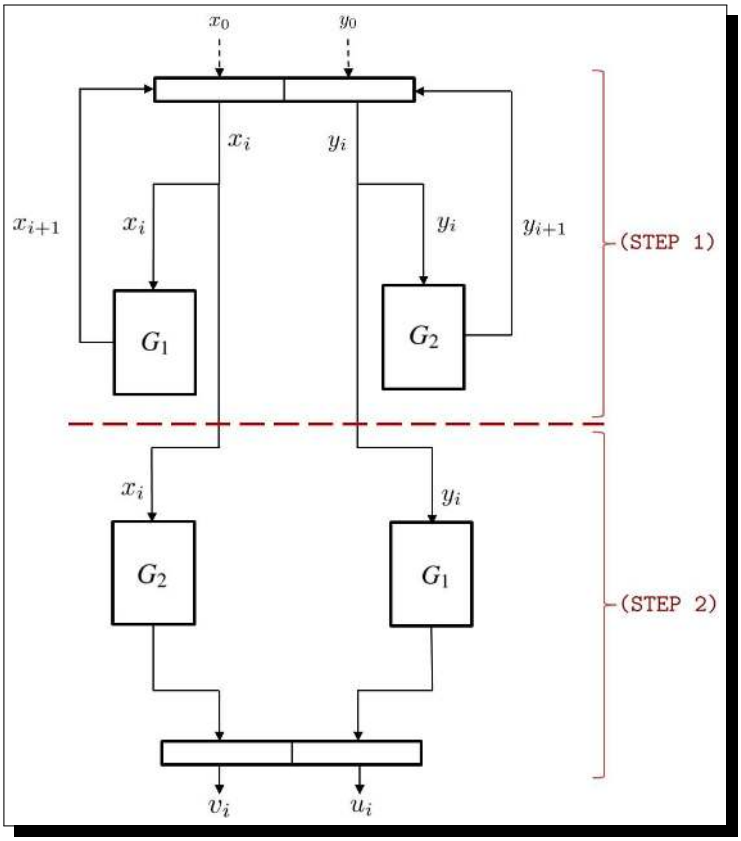

Figure 3.14.: Diagrammatic representation of the PSYND's Generator G

the keystream bits of PSYND.

3.4.3. Security of the cipher

The security of PSYND is discussed. More precisely, we will prove that the PSYND is a pseudo-random generator. Then we analyze its security from practical point of view by identifying all the best known attacks that can applicable against it.

Theoretical security. Here we will prove that the PSYND is a pseudo-random generator under the assumptions made in the previous section, namely the indistinguishability and RSD assumption. In order to prove that, we need the following lemmas.

Lemma 3.4.1. *If the assumption* A2 *holds, then the transformations G_1 and G_2 defined in PSYND are one-way.*

Proof. To prove this Lemma, it is sufficient to show that G_1 (and G_2) can be transformed into a function selected from the collection defined in A2. This claim has been proved in subsection 3.3.2.

□

Lemma 3.4.2. *If the assumption* `A2` *holds, then the transformations* `f` *and* `g` *are both one-way.*

Proof. This straightforward results from [GIL+90, Yao82b], that state that for every collection of one-way functions $\mathcal{F} = \{f_k\}_{k\in S}$, where S is a finite set, the collection $\mathcal{F}_n = \{f_{i_1,\cdots,i_n}\}_{i_1,\cdots,i_n\in S^n}$, whose elements are defined as $f_{i_1,\cdots,i_n}(x_1,\cdots,x_n) = (f_{i_1}(x_1),\cdots,f_{i_n}(x_n))$ is also one-way. In addition, this is even true when having a single mapping (i.e., when $i_1 = \cdots = i_n$). In our setting, we have $S = \{1,2\}$. □

For proving the pseudo-randomness of the key stream produced by PSYND, we will prove that the sequence pair (u_i, v_i) is pseudo-random, meaning that it is indistinguishable from an 2ℓ-bit random string. We will do this by induction. Before doing this, we prove the following theorem.

Theorem 3.4.1. *Let* `R` *and* `L` *be two functions defined over* $\mathbb{F}_2^\ell$ *by*

$$\mathtt{R}(x) = G_1(x)\|G_2(x) \;\; \textit{and} \;\; \mathtt{L}(y) = G_2(y)\|G_1(y)$$

Related to `R` *and* `L`*, we define two* 2ℓ*-bit strings generator* $\mathsf{G_R}$ *and* $\mathsf{G_L}$ *depicted in Figure 3.15 and Figure 3.16, respectively.*

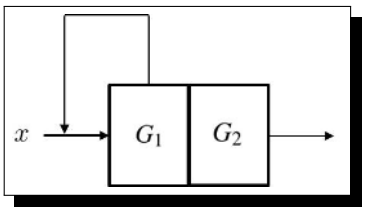

Figure 3.15.: Illustration of $\mathsf{G_R}$

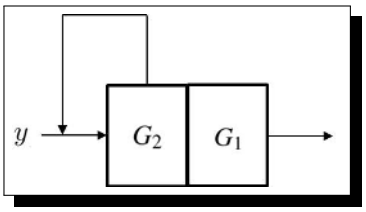

Figure 3.16.: Illustration of $\mathsf{G_L}$

The function G_1 (resp. G_2) is the update (resp. output) function of $\mathsf{G_R}$ contrary to $\mathsf{G_L}$.
If the assumptions `A1` *and* `A2` *hold, then $\mathsf{G_R}$ and $\mathsf{G_L}$ are both pseudo-random generators.*

Proof. It is sufficient to prove this statement for $\mathsf{G_R}$, since they are similar. The proof is similar to that of XSYND (Theorem 3.3.2. Note that `R` and is `L` are both expansion functions with expansion factor 2ℓ.
As shown earlier the functions G_1 (resp. G_2) can be rewritten as $G_1(x) = \mathsf{A}\cdot z^\top$ (resp. $G_2(x) = \mathsf{B}\cdot z^\top$), where z is a regular word that corresponds to the input x. Hence, if we vertically stack A and B, we obtain a new $2\ell \times n$ matrix M given by

$$\mathsf{M} = \begin{pmatrix}\mathsf{A}\\ \mathsf{B}\end{pmatrix}.$$

This matrix satisfies assumption `A1`. By doing so, we can write $\mathtt{R}(x) = \mathsf{M}\cdot z^\top$. So, it would be sufficient to prove that the output of $\mathsf{M}\cdot z^\top$ is pseudo-random. Now, we show by contradiction that the output of `R` is pseudo-random. Suppose the opposite. Then there is a distinguisher Ψ that can make a distinction between this output and an 2ℓ-bit random sequence u. This distinguisher accepts as input an $2\ell \times n$ binary random matrix A and a random $u \in \{0,1\}^{2\ell}$ as a candidate being equal to $\mathsf{M}\cdot z^\top$ for some unknown regular word z. If $\mathsf{M}\cdot z^\top = u$, Ψ outputs 1 with probability above $\frac{1}{2} + \frac{1}{p(n)}$, for every polynomial $p(n)$. Otherwise, when u is chosen uniformly from $\{0,1\}^{2\ell}$, Ψ outputs 1 with probability at most $\frac{1}{2}$.
More precisely, the behavior of Ψ is the following:

$$\begin{cases} \Pr[\Psi(\mathsf{M},u)=1] \geq \frac{1}{2}+\frac{1}{p(n)}, & \text{if } u = \mathsf{M}\cdot z^{\top}, \text{ for some regular word } z \\ \Pr[\Psi(\mathsf{M},u)=1] < \frac{1}{2}, & \text{if } u \text{ is taken uniformly from } \{0,1\}^{2\ell} \end{cases}$$

Our next step is to construct an algorithm Θ which calls Ψ as a subroutine, in order to predict the dot product of unknown regular word z and a random chosen η-bit sequence v (i.e., $\langle z,v\rangle$) with probability at least $\frac{1}{2}+\frac{1}{2p(n)}$. To achieve this, we write $v=(v_1,\cdots,v_\eta)$ and define π to be the number of the positions j such that $z_i=v_j=1$, i.e. the size of the intersection $z\cap v$. Let σ be its parity, i.e. the inner product $\langle z,v\rangle$. By doing so, on inputs $\mathsf{M}\cdot z^{\top}$ and v, the algorithm Θ will perform the following:

- Choose a random $\sigma' \in \{0,1\}$ as candidate to σ
- Choose randomly $\delta \in \{0,1\}^{2\ell}$
- Construct a new $2\ell \times n$ binary matrix $\widehat{\mathsf{M}} = (\widehat{a}_1,\cdots,\widehat{a}_n)$ such that for every $j \in \{1,\cdots,n\}$ it holds

$$\widehat{a}_j = \begin{cases} a_j+\delta & \text{if } v_j=1, \\ a_j & \text{if } v_j=0 \end{cases}$$

  where $(a_1,\cdots,a_\eta)$ is the decomposition of M.
- Supply the distinguisher Ψ with $\widehat{\mathsf{M}}$ and $\mathsf{M}\cdot z^{\top}+\sigma'\cdot\delta$
- If Ψ outputs 1, then outputs $\sigma'=\sigma$. Otherwise, returns the opposite of σ'.

We now show that Θ predicts the dot product $\langle z,v\rangle$ with probability above $\frac{1}{2}+\frac{1}{2p(n)}$.
There are to distinct cases to treat:

(1) $\mathtt{C}_1$:= $\{\sigma$ guessed correctly$\}$. Then the predicted value for $\langle z,v\rangle$ is correct if the distinguisher outputs 1. But the distribution seen by the distinguisher on $(\widehat{\mathsf{M}},\mathsf{M}\cdot z^{\top}+\sigma'\cdot\delta)$ is similar to the distribution on input $(\mathsf{M},\mathsf{M}\cdot z^{\top})$. By construction, this occurs with probability at least $\frac{1}{2}+\frac{1}{p(n)}$.

(2) $\mathtt{C}_2$:= $\{\sigma$ not guessed correctly$\}$. The distinguisher receives uniformly distributed inputs because of the randomness of δ and therefore outputs 1 with probability $\frac{1}{2}$.

Seeing that $\Pr[\mathtt{C}_1]=\Pr[\mathtt{C}_2]=\frac{1}{2}$, we can conclude that the entire success probability of predicting the dot product $\langle z,v\rangle$ is at least $\frac{1}{2}+\frac{1}{2p(n)}$. This leads to a contradiction with the Goldreich-Levin Theorem [GL89] due to assumption `A2`. As consequence, the function `R` (and also `L`) is pseudo-random.

□

Now we are ready to present the main result concerning the pseudo-randomness of the PSYND stream cipher. The proof is inductive over the iteration number and involves Theorem 3.4.1.

Theorem 3.4.2. *If we choose parameters (n,ℓ,ω) such that assumptions `A1` and `A3` are met, then the PSYND cipher is a pseudo-random generator.*

Proof. To this end, we will show that the sequences u_i and v_i are pseudo-random. We will prove this by induction over i.

- Base case : $i = 1$. From Theorem 3.4.1 we conclude that $\mathtt{R}(x_0) = G_1(x_0)\|G_2(x_0)$ and $\mathtt{L}(y_0) = G_2(y_0)\|G_1(y_0)$ are both 2ℓ-bit pseudo-random sequence. This implies also $u_0 = G_2(x_0)$ and $v_0 = G_1(y_0)$ are pseudo-random sequence. The same argument holds for $x_1 = G_1(x_0)$ and $y_1 = G_2(y_0)$.
- Induction step: Assume that u_j and v_j (and also x_j and y_j) are pseudo-random sequences for some positive integer j (the induction hypothesis). We will show that u_{j+1} and v_{j+1} are also pseudo-random sequences. We have the following relations:

$$\begin{cases} u_{i+1} = G_1(y_{i+1}), \;\; y_{j+1} = G_2(y_j) \\ v_{i+1} = G_2(x_{i+1}), \;\; x_{j+1} = G_1(x_j) \end{cases}$$

By the induction hypothesis, the sequences $\mathtt{R}(x_j) = G_1(x_j)\|G_2(x_j)$ and $\mathtt{L}(y_j) = G_2(y_j)\|G_1(y_j)$ are pseudo-random. That means, $G_1(x_j) = x_{j+1}$ and $G_2(y_j) = y_{j+1}$ are pseudo-random and applying Theorem 3.4.1 on these sequences completes the inductive step and therefore the PSYND cipher is a pseudo-random number generator.

□

After proving the pseudorandomness of the key stream of the PSYND stream cipher, we now want to discuss the security of the initialization process.

Security of the initialization process. We will prove that the probability of recovering the secret key is negligible. As described before, the generation of the initial state $s_0 = (x_0, y_0)$ of PSYND is done as follows:

$$\begin{cases} \lambda = K\|IV \\ \beta = \lambda \oplus G_1(\lambda) \\ \gamma = \lambda \oplus G_2(\lambda) \\ F(\lambda) = (\beta \oplus G_2(\beta), \gamma \oplus G_1(\gamma)) = (x_0, y_0) = s_0 \end{cases}$$

Assume that one can somehow know the initial state s_0. So, to recover the secret key and IV, one has to solve the following equations:

$$x_0 = \beta \oplus G_2(\beta) \text{ with } \beta = \lambda \oplus G_1(\lambda) \tag{3.18}$$
$$y_0 = \gamma \oplus G_1(\gamma) \text{ with } \gamma = \lambda \oplus G_2(\lambda) \tag{3.19}$$

To do this, it is sufficient to solve the equations

$$x' = x \oplus G_2(x), \tag{3.20}$$
$$y' = y \oplus G_1(y), \tag{3.21}$$

where x' and y' are both known, while x and y are unknown variables. As shown before, G_2 and G_1 produce pseudo-random outputs for random inputs. That means, that the variables x and y are also pseudo-random. For the sake of simplicity, one can suppose that $x' = 0$ and $y' = 0$. In this case, the equations (6) and (7) become

$$x \oplus G_2(x) = 0, \tag{3.22}$$
$$y \oplus G_1(y) = 0, \tag{3.23}$$

As a result, to solve this system one has to find fixed points for the transformations G_1 and G_2. But, what is the success probability for finding such points. In order to estimate this probability, we use the following result, which is known as the Piling-Up Lemma [Mat94].

Lemma 3.4.3. *For each value ($1 \leq i \leq t$), let Z_i be a random variable over $\{0,1\}$, independent of Z_j for all $j \neq i$, such that*

$$\Pr(Z_i = 1) = p_i, \ \forall i \in \{1, \cdots, t\}$$
$$\Pr(Z_i = 0) = 1 - p_i$$

Then $\Pr(Z_1 \oplus Z_2 \oplus \cdots \oplus Z_t = 0) = \frac{1}{2} + 2^{t-1} \prod_{i=1}^{t} (p_i - \frac{1}{2})$

The equation (3.23) can be rewritten as XORing of $\omega + 1$ unknown ℓ-bit strings, whose sum equals 0. Thus if x is randomly chosen from $\mathbb{F}_2^\ell$, then the probability that it is a fixed point for G_2 is equal to $\frac{1}{2}^\ell$. Actually, the row entries of the underlying matrix of G_2 and x can be associated to independent random variables $(Z_i^{(j)})$ defined over $\{0,1\}$, where $i \in \{1, \cdots, \omega+1\}$ and $j \in \{1, \cdots, \ell\}$ are the column and row positions, respectively. With this setting, we have $\Pr[Z_i^{(j)} = 1] = p_i = \frac{1}{2}$. Using Lemma 3.4.3 we obtain

$$\Pr(\phi_2(x) = x) = \prod_{i=1}^{\ell} \Pr(Z_1^{(j)} \oplus Z_2^{(j)} \cdots \oplus Z_{\omega+1}^{(j)}) = \frac{1}{2^\ell}. \tag{3.24}$$

Thus for a large value of ℓ , this probability is negligible. In PSYND the value of ℓ is at least 128.

Practical security. To asses the security of the PSYND family from practical point of view, this paragraph briefly describes the best known generic algorithms for attacking PSYND, but focuses only on attacks that aim at recovering the secret key or internal states. Key or state recovery in PSYND can be done by inverting the underlying functions G_1 and G_2 involved in the initialization and key stream generation procedure, as they constitute the major components of PSYND's desing.

Best known attacks: to our best knowledge, there basically exist three different potential algorithms: Linearisation Attacks (LA), Generalized Birthday Attacks (GBA), and Information Set Decoding (ISD). All these attacks have been described earlier in subsection 3.3.2.

Other type of attacks: Here we discuss some other attacks that could be applicable against the PSYND stream cipher.

Exhaustive key search. It is the simplest attack against any cipher and consists in trying every possible key in turn until the correct key is found. However, this attack do not seem to be applicable against PSYND due to the large cardinality of the key space. The key length used in PSYND is at least to 128

bits.

Guess-and-Determine attacks. The basic idea of this kind of attacks is try to find the value of unknowns variables in a cipher by guessing some of them and deducing another from the guessed unknowns variables. Let us demonstrate how this attack can be applied against PSYND. Assume that IV is known, that is $\frac{\omega}{2}$ selected columns of the whole matrix of size $\omega\alpha \times \omega 2^{\alpha}$ are known. Suppose that an adversary aims at attacking the initialization phase of PSYND for recovering the secret key. Assume that she can only guess $u\alpha$ bits of the secret key and the remaining $\frac{\omega}{2}\alpha$ bits are still unknown. Those bits correspond to $\frac{\omega}{2} - u$ unknown columns, each of them has length ℓ bits. Thus, the complexity to recover the secret key is $\ell^{\frac{\omega}{2}-u}$ binary operations. For example, for the instance $\text{PSYND}_{(8192,256,32)}$, this attack requires 2^{120} bops if only 8 bits of the secret key are successfully guessed.

Time-memory trade-off attacks. A time-memory tradeoffs attack, originally presented by Hellman in [Hel80], is applicable when the state size of the cipher is too small. As pointed out in [Gol97, HS05], the TMTA can be avoided by taking the following conditions into account: the initial value should be at least as large as the key, and the state should be at least twice the key. The PSYND's parameters proposed in the next section fulfil this condition. If one allows the precomputation complexity to be greater than the complexity of exhaustive key search, it is possible to get a time-memory trade-off attack which is faster than exhaustive key search. This is valid for any type of stream cipher.

Dieharder tests. Dieharder tests*, developed by G. Marsaglia, is a set of statistical tests designed to examine the randomness of random numbers. These tests were performed on the internale state (including the initial state) as well as on the output of the PSYND cipher. As a result, PSYND seems to behave as a true random number generator, as it passes successfully all tests in the Dieharder suite.

3.4.4. Parameters and experimental results

Some implementation details The PSYND stream cipher has been implemented using C/C++ programming language. The implementation employs the intrinsic functions and the underlying matrices have been generated using the system time as a seed. For our tests we used an Intel Core 2 Duo E8400, running at a clock frequency of 3.0 GHz. It has L1 cache size of 2×32 KB for the data caches, and a L2 cache size of 6 MB. The sources have been compiled with gcc, version 4.6.1, and the tests have be carried out under Linux Ubuntu 6.10. For each test run, we load the matrices A and B from a file and generate the initial state. At the moment, we used a fixed key K and a fixed initial vector IV. For each iteration of the output phase the indices of the XORed columns are unknown a priori. Therefore, our scheme has difficulty to take advantage particularly of the L1 cache. The probability that cache lines can be reused is quite low, which is also confirmed by our results. The bigger the matrices A and B become, the more the advantages of the L1 cache get lost. In order to overcome this problem, we executed the two output steps in parallel. For the implementation of threads, we used the `pthread` library. Each output step has been represented by its own thread. Thereby we create two output streams which can then be used by the application.

*http://www.phy.duke.edu/ rgb/General/dieharder.php

Parameters choice The PSYND's parameters should be selected with great caution as bad choice could considerably affect the speed and the security of the system. By construction, the main parameters are α and ω due to the relations $n = \omega 2^{\alpha}$ and $n = \omega\alpha$. Smaller ω offer good performance, and larger α increase security and therefore controlling these two parameters allows for obtaining an optimal efficiency–security tradeoff curve. According to this rule, the choice of α and ω in our implementation meets the following constraints: α is equal to 8, and ω is a multiple of 16, allowing ℓ to be a multiple of 32 (or 64 depending on the CPU architecture) for a full use of the word-size XORs.

Experimental results Putting all of the above conditions together, we found a large set of parameters (n, ℓ, ω) providing a high performance using different key/IV size and offering required security levels. Table 3.9 presents some "optimal" parameters sets (n, ℓ, ω) in which we give the provided security level, and the size of the secret key K and the initial vector IV. The security is estimated according to best known attacks described earlier.

security	n	ℓ	ω	K/IV size	speed (cpb)
90	8192	256	32	128	9.43
128	12288	384	48	192	12.09
180	16384	512	64	256	16.84
300	20480	640	80	320	29.92
400	24576	768	96	768	41.8

Table 3.9.: Some parameters for the PSYND cipher.

Comparison with XSYND. Here we want to compare the PSYND cipher with the XSYND cipher [MHC12]. The comparison is done in terms of the speed and the storage requirements. The speed is measured in cycles per byte (cpb), while the storage space in bits. The data comparison results are summarized in Table 3.10 and plotted in Figure 3.17. As one can deduce from this table, the PSYND outperforms XSYND in terms of performance. For instance, $\text{PSYND}_{(8192,256,32)}$ runs much faster than $\text{XSYND}_{(8192,256,32)}$, for the same security level of 90 bits. As mentioned in [GLS07], adding of the quasi-cyclic structure in the matrices of PSYND, could significantly improve the PSYND system in terms storage space, since the required space amount (in bits) to be stored equals n instead of $n\ell$ bits. This modification however could worsen the performance and threaten the security of the system, despite the pseudo-randomness property of quasi-cyclic codes satisfying some constraints on parameters [GZ07]. For that reasons, we did not implement this kind of codes in the PSYND family.

Comparison with the eSTREAM candidates. The eSTREAM portfolio contains 7 ciphers, 4 of them are software-oriented stream ciphers (profile 1): HC-128, Rabbit, Salsa 20/12, and SOSEMANUK. A detailed description of these ciphers can be found in the eSTREAM book [RB08]. Table 3.11 summarizes some data about these stream-ciphers including the key/IV size and the performance. These data were reported in the Bernstein's eSTREAM benchmarking paper [Ber]. The speed measurements were performed on a computer possessing a four-core 2394MHz Intel Core 2 Quad Q6600 6fb processor. As one can deduce from this table, the PSYND familiy is much slower than these proposals. For instance, HC-128 supporting 128-bit key and 128-bit IV and offering 128-bit

scheme	parameters (n, ℓ, ω)	key/IV size (bits)	speed (cpb)	estimated security
XSYND	$(8192, 256, 32)$	128	14.92	90
	$(12288, 384, 48)$	192	16.98	128
	$(16384, 512, 64)$	256	35.40	160
	$(20480, 640, 80)$	320	43.68	200
	$(24576, 768, 96)$	384	55.42	240
PSYND	$(8192, 256, 32)$	128	9.43	90
	$(12288, 384, 48)$	192	12.09	128
	$(16384, 512, 64)$	256	16.84	160
	$(20480, 640, 80)$	320	29.92	200
	$(24576, 768, 96)$	384	41.8	240

Table 3.10.: Performance comparison of PSYND with XSYND for the same security levels.

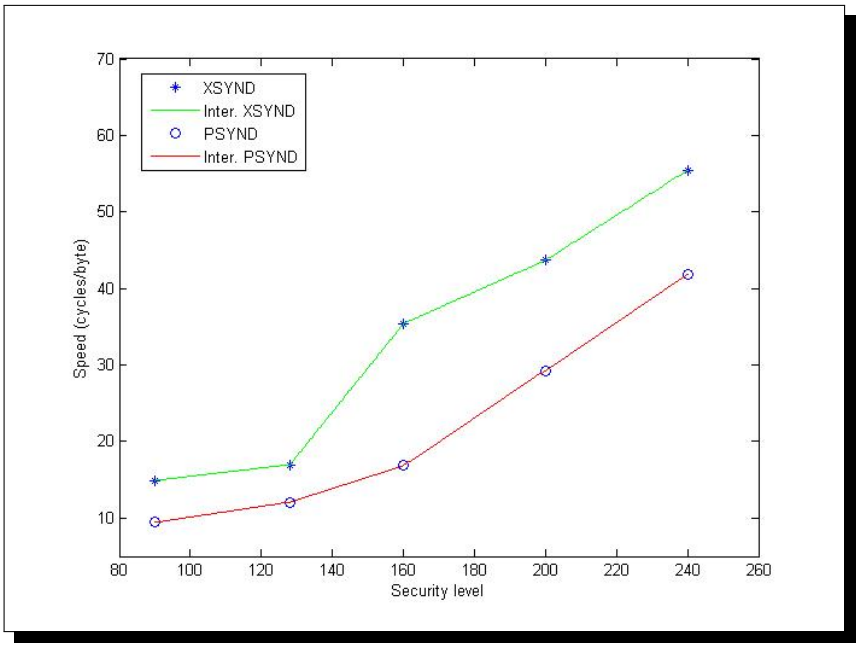

Figure 3.17.: Graphical comparison between XSYND and PSYND in terms of speed for the same security levels.

security, runs with a speed of 2.34 cpb, while $\text{PSYND}_{(8192,256,32)}$ runs at 9.43 cpb and but only provides 90-bit security. However, compared to the 128-bit AES-CTR running at 12.59 cpb, the instance $\text{PSYND}_{(12288,384,48)}$ with a key and IV of 192 bit, delivers the same performance.

Primitive	key size	IV size	speed (cpb)
HC-128	128	128	2.34
Rabbit	128	64	2.34
Salsa 20/12	128	64	2.54
SOSEMANUK	128	64	3.54
AES-CTR	128	128	12.59
PSYND	128	128	9.43

Table 3.11.: Performance of some software-oriented eSTREAM's candidates, as reported in [Ber].

3.5. Conclusion and Open Problems

Starting from the Fischer-Stern pseudo-random number generator, we have describe three code-based constructions of stream ciphers and shown how to improve the efficiency the existing ones. First, we showed how to use the sponge construction to design a stream cipher, called 2SC. This runs much faster than the SYND stream cipher, but it uses larger matrices than SYND. Then, we propose the XSYND cipher as an improved variant of SYND in terms of performance. XSYND uses a generic state transformation which is directly reducible to the regular syndrome decoding problem, but has better computational characteristics than the regular encoding introduced in the SYND system. Furthermore, we deliver a security proof for XSYND, which shows that if there exist a distinguisher for XSYND, there exist a solver that can solve a hard instance of the regular syndrome decoding. In the last section of this chapter, we showed how to parallelize the XSYND in order to double the output size and hence increase further its performance leading to a new construction, called PSYND. This cipher outperforms all previous constructions. For instance, for 80 bit security, the PSYND with 128 bits key size runs at 9.43 cycles per byte, while the original proposal SYND only runs at 36 cycles per bytes, both use the same parameter set. However, despite its security reduction to the RSD problem, its efficiency is unfortunately incomparable to that of the software-oriented candidates (profile 1) from the eSTREAM project. As future work, we suggest a hardware implementation of PSYND to see whether the performance can be further improved and to make a comparison with profile 2 of eSTREAM project.

Chapter 4

Code-based Hash Functions

This chapter investigates the design of hash functions based on the SD problem. We first start by describing the Fast Syndrome Based hash [AFS05, FGS07, AFG+08] family (FSB) and recalling its main features in Section 4.2. Then, we present in Section 4.3 our main contribution, which consists in showing how to incorporate the ideas of FSB and the sponge construction due to Bertoni et al. [BDPA07] to design a variant of FSB hash function, called Sponge-FSB (in short S-FSB). The security of this variant is based on the same problems as FSB, and discussed in Subsection 4.3.2, while the proposed parameters and our implementation results are given in Subsection 4.3.3. Finally, we describe the RFSB hash function, which is another variant of FSB and provide the reported results according to the available RFSB implementations.

4.1. Introduction

As defined in Chapter 2, a cryptographic hash function satisfying certain security properties, plays an important role in many cryptographic applications such as digital signatures, pass-word protection and pseudo-random number generation. Over last years, a long list of hash functions have been proposed in the literature. Following cryptanalytical advances, most of them widely used in practice such as SHA-1 [EJ01] have been found to be insecure [CR06, WYY05]. This has called into question the long-term security of later algorithms that share a similar design like SHA-2 family [Nat08]. As a reaction, National Institute of Standards and Technology (NIST) announced a publicly available contest, called SHA-3 (or the Advanced Hash Standard (AHS)), to develop new family of hash functions. Initially, 64 candidates have been submitted following different design principles, and only 14 of the competing designs were selected in the second round of the contest. One of the submissions that did not pass to this round, was the Fast Syndrome-Based hash function , which we describe in the subsequent section.

4.2. Fast Syndrome Based Hash Family

4.2.1. Description of FSB hash familiy.

In 2003, Augot et al. [AFS03] introduced the Syndrome-Based hash family (in short SB), which iterates a so called compression function according to Merkle-Damgård's design principle [Mer89, Dam89] (MD). The SB is the first hash function which uses binary random codes and has security reduction to NP-complete problems from coding theory. At Mycrypt 2005, Augot et al. [AFS05] proposed an improved variant of SB, called Fast Syndrome-Based hash family (in short FSB), in terms of speed by introducing the so-called regular encoder that convert a bit string of certain length into a regular word. In 2007, Finiasz et al. [FGS07] showed how to increase the FSB's efficiency in two ways: (1) adding a final compression transformation in order to obtain a desired hash length as well as to achieve a security level equal to half the output length; (2) using quasi-cyclic codes instead of purely random ones in order to get a short description for the hash function, allowing the underlying matrix to be fit in the cache of a standard CPU, and thus considerably increasing the speed of FSB. The FSB construction with these two modifications has been submitted to the SHA-3 competition, but it did not pass the second round because of its performance. In what follows, we describe the SHA-3 FSB's proposal [AFG+08].

The FSB hash function follows Merkle-Damgård's construction [Mer89, Dam89] based on a compression function $\mathcal{F}$ from coding theory, as shown in Figure 4.1. This function (geen-framed) compresses s input bits to ℓ bits ($s > \ell$) and is defined by

$$\begin{aligned} \mathcal{F} : \mathbb{F}_2^s &\to \mathbb{F}_2^\ell \\ x &\mapsto \mathcal{F}(x) = \mathsf{M} \cdot (\phi(x))^\top, \end{aligned}$$

Where M is a quasi-cyclic matrix of size $\ell \times n$, composed of ℓ block matrices M_i of size $\ell \times \ell$, i.e. $\mathsf{M} = \mathsf{M}_1 \parallel \mathsf{M}_2 \parallel \cdots \parallel \mathsf{M}_\ell$, and $s = \omega \log_2(n/\omega)$. Furthermore, this compression is parameterized by a number prime p such that 2 is a generator of $\mathbb{F}_p$. This prime determines $\frac{n}{\ell}$ pre-defined vectors, each having p bits. Each vector generates a block matrix $\hat{\mathsf{M}}_i$ of size $p \times p$, which should be truncated to M_i. The function $x \to \phi(x)$ is an encoding algorithm, which takes inputs of size s bits and returns regular words of length n and weight ω according to Algorithm 4.

Algorithm 4 FSB's regular encoder ϕ

Input : x a binary string of s bits with $s = \omega \log_2(n/\omega)$
Output : a regular word $e = \phi(x)$ of length n and weight ω.

$e = (e_1, \cdots, e_n) \leftarrow 0^n$ (*initializing with n zeros*)
$j = (j_1, \cdots, j_\omega)$
Write $x = z \parallel t$ with $|z| = \ell$, $|t| = s - \ell$
Split $z = (z_1 || \cdots || z_\omega)$ with $|z_i| = \frac{\ell}{\omega}$ $(0 \le z_i \le 2^{\frac{\ell}{\omega}} - 1)$
Split $t = (t_1 || \cdots || t_{\frac{s-\ell}{\omega}})$ with $|t_i| = \frac{s-\ell}{\omega}$ $(0 \le t_i \le 2^{\frac{s-\ell}{\omega}} - 1)$
for $i = 0$ **to** $\omega - 1$ **do**
  $j \leftarrow i\frac{n}{\omega} + z_i + t_i 2^{\frac{\ell}{\omega}}$
  $e_j \leftarrow 1$
end for

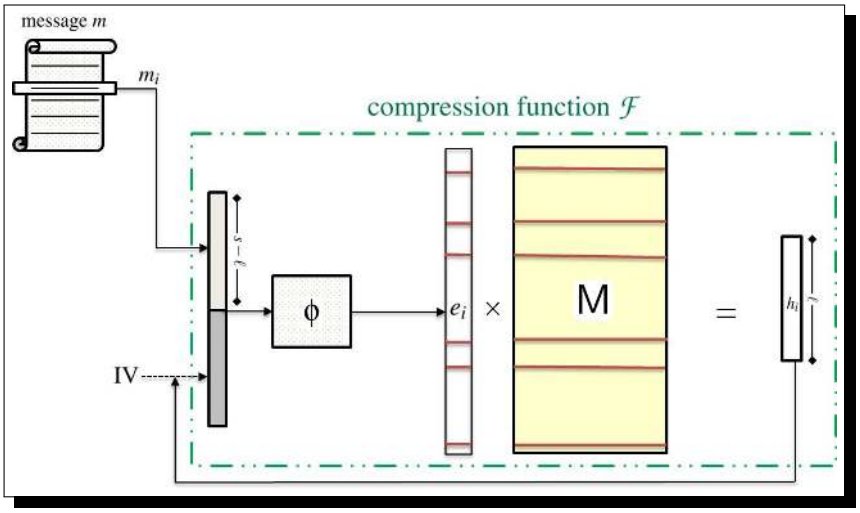

Figure 4.1.: Fast Syndrome-Based Hash function without a final transformation.

The FSB's hashing process is shown in Figure 4.1. The message $m = (m_1, m_2, \ldots, m_k)$ (with padding) to hash is first splitten into blocks m_i, each of size $s - \ell$ bits. Then the intermediate hash value h_i, for $i = 1, \cdots, k$, is computed as

$$h_i = \mathcal{F}(m_i \parallel h_{i-1}),$$

where $h_{i-1} = IV$ is an ℓ-bit initial value. All IV's used in FSB are equal to 0^ℓ. This process will continue until the compression function finishes processing the entire message to produce the last result h_k, called pre-final hash. To obtain the hash value of the message by FSB hash function, h_k is then fed through a second compression function $\mathcal{G}$ to produce the desired hash length. The main reason for using such function is that the FSB's compression $\mathcal{F}$ cannot achieve a $\frac{\ell}{2}$-bit (resp. ℓ-bit security) against collision attack (resp. inversion attack).
For all versions of FSB, the Whirpool hashing algorithm [BR00] is used as a final compression function because of its nice properties. It possesses a high degree of non-linearity and behaves well even its output is shortened to achieve a smaller size.

4.2.2. Theoretical security of FSB

As the FSB hash function follows the MD-design, its security is directly related to the security of the underlying compression function $\mathcal{F}$. More precisely, Merkle and Damgård showed that if the compression function is collision resistant, then the iterated hash function is collision resistant as well. As demonstrated in [AFG+08] that the FSB compression function $\mathcal{F}$ has the following security reduction:

- *Collision resistance.* Finding a collision for $\mathcal{F}$ is at least as hard as finding a codeword of length n and weight $\leq 2\omega$, which means solving an instance of the 2-regular Null-Syndrome Decoding problem.
- *Preimage resistance.* Inverting $\mathcal{F}$ is at least as hard as finding a codeword of length n and weight ω, which means solving an instance of the the Regular Syndrome Decoding problem.

- *Second preimage resistance.* Finding a second preimage for $\mathcal{F}$ is at least as hard as finding a codeword of length n and weight ω in a code, having a parity check matrix of size $(\ell - \omega) \times (n - \omega)$.

4.2.3. Practical security of FSB

In practice, the FSB's compression function $\mathcal{F}$ has to resist the following attacks: Information Set Decoding algorithm [AFS05] (ISD), Generalized Birthday Attack (GBA) [Wag02, CJ04], and some other cryptanalytic techniques [Saa07, FL08, KK06]. All parameter sets proposed for FSB were selected so as to withstand all these attacks. As reported in [AFG+08], the security level provided by a FSB version with output size k bits (denoted FSB_k) against collision, inversion and second perimage are respectively $\frac{k}{2}$, k, and $k-l$, where l is the logarithm to the base 2 of the message length. Note that in [AFS05], a detailed analysis of the complexities of a ISD and GBA algorithm is given.

4.2.4. Efficiency of FSB

Theoretically the speed of FSB hash function depends mainly on the speed of the compression function $\mathcal{F}$, which can be estimated as the number of bitwise XOR-operations required in each iteration to process one bit of the input. As shown in [AFS05], this number, we denote here N_{fsb}, is a function in (n, ℓ, ω), which is defined by

$$\text{N}_{\text{FSB}}(n, \ell, \omega) = \frac{\ell\omega}{\omega \log_2(\frac{n}{\omega}) - \ell}$$

Using differential calculus shows that the minimum of N_{FSB} is always attained for $\omega_0 = 2\ln(\ell)$, independently of n . Its minimum is equal to $\frac{\omega}{\ln(\frac{n}{\omega})-1}$. So, for fixed ω, large values of n will further improve the performance of FSB, however this will augment the size of the matrix to be used, and hence increase the number of cache misses, which immediately affects the speed of the FSB.

4.2.5. Parameters choice for FSB.

Parameters for FSB are selected according the following rules:

- Choose ℓ in order to get the desired hash length and the security level required.
- ℓ must be a multiple of 32 or 64 depending on the CPU architecture, in order to use the word-size XORs.
- Choose n and ω such that
  - $\frac{n}{\omega}$ a power of 2 (by construction) for reading an integer number of input bits at a time.
  - the size $\ell \times n$ is smaller for avoiding the cache misses.
  - ω is close to ω_0

Five FSB's instances [AFG+08] have been proposed, namely FSB_k where output size $k \in \{160, 224, 256, 384, 512\}$. Their parameter sets are shown in Table 4.1 below.

Instance	n	ℓ	ω	p	s
FSB_{160}	$5 \cdot 2^{18}$	640	80	653	1120
FSB_{224}	$7 \cdot 2^{18}$	896	112	907	1568
FSB_{256}	2^{21}	1024	128	1061	1792
FSB_{384}	$23 \cdot 2^{16}$	1472	184	1483	2392
FSB_{512}	$31 \cdot 2^{16}$	1984	248	1987	3224

Table 4.1.: Parameters for the five instances of FSB hash function, where $s = \omega \log_2(\frac{n}{\omega})$ and p is the smallest prime number such that $p \geq \ell$ and 2 is a generator of $\mathbb{F}_p$.

4.3. The S-FSB Hash function

In this section we present our construction, which is an improved variant of FSB family in terms of performance. We call it S-FSB hash family, which stands for Sponge Fast Syndrome Based hash family. To do so, we will use the same notations as in the previous section and define four positive integers n,ℓ, r and c such that the ratio $\frac{n}{\omega}$ is a power of 2, and $\ell = r + c = \omega \log_2(\frac{n}{\omega})$.

4.3.1. Description of S-FSB

The main idea behind S-FSB is to use the sponge design principle [BDPA07] based on a one-to-one transformation $\mathcal{T}$ rather than a compression function $\mathcal{F}$ (Figure) according to the MD-paradigm [Mer89, Dam89] used in FSB. This transformation is similar to that of the SYND stream cipher, which we have presented in Chapter 3. It is defined by

$$\mathcal{T} : \mathbb{F}_2^{\ell} \rightarrow \mathbb{F}_2^{\ell} \tag{4.1}$$

$$x \mapsto \mathsf{M} \cdot (\phi(x))^{\top}. \tag{4.2}$$

Where M is a random binary matrix of size $\ell \times n$ and the mapping $x \rightarrow \phi(x)$ is a regular encoding algorithm as in FSB. For plugging this transformation into the sponge construction, we take ℓ width, r the rate, and c the capacity such that $\ell = r + c$.

As explained in 2.4.3 the message m to be hashed must be padded according to a sponge-compliant padding rule such that the last block be non-zero and broken into blocks m_i of r bits. That is, $m = (m_1, \cdots, m_k)$. Then the ℓ bits of the state are all zeros, i.e. $IV = 0^{\ell}$ and the the sponge construction proceeds in two steps:

- **Absorbing step:** this step is illustrated in Figure 4.2. Each message block m_i is proceeded as follows: Let s_{i-1} be the ℓ-bit input of round i such that $s_{i-1} = s_{i-1}^{(1)} \parallel s_{i-1}^{(2)}$ with $|s_{i-1}^{(1)}| = r$ and $|s_{i-1}^{(2)}| = c$. The output s_i of round i is then computed as

  $$s_i = \mathcal{T}(s_{i-1}^{(1)} \oplus m_i, s_{i-1}^{(2)}) \text{ with } s_0 = IV = 0^{\ell}$$

  When all input blocks are processed, the S-FSB construction switches to the squeezing step.

- **Squeezing step:** in this step the internal state $(s_i)_{i\geq k}$ should be first updated by

$$s_{i+1} = \mathcal{T}(s_i) \text{ with } i \geq k$$

and then truncated to b ($b \leq r$) bits to produce the pre-final hash values h_i. The final l-bit hash value h of the message m is obtained as the concatenation of those hash values, i.e. $h = h_1 \parallel h_2 \parallel \cdots \parallel h_d$, where d is the number of output blocks, which is chosen at will by the user such that $l = db$. In S-FSB, we take $d = 2$, and $b = r = l$.

Figure 4.3 explains how the squeezing step works.

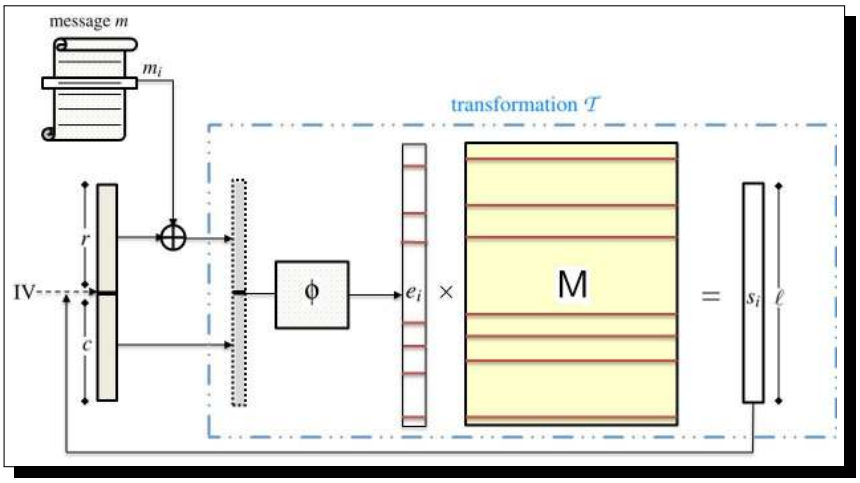

Figure 4.2.: Absorbing step of S-FSB hash function.

As in FSB hash function, the performance of S-FSB depends directly on the number of the bitwise XOR operations computed at each round to treat the r bits of one message block. That is, one needs first r XORs for the bitwise addition and then ℓ XORs of ω columns of the matrix M. This result to $r + \ell\omega$ binary XOR-operations. Since the number of bits of each message block is r, the number of expected binary XORs (denoted by $\mathrm{N}_{\mathrm{sfsb}}$) in average for each message input bit is:

$$\mathrm{N}_{\text{S-FSB}}(\omega, r, c) = \frac{r+\ell\omega}{r} = \frac{r+(r+c)\omega}{r} = 1+\omega(1+\frac{c}{r}), \tag{4.3}$$

where $\ell = r + c = \omega\log_2(\frac{n}{\omega})$.

This results to

$$\mathrm{N}_{\text{S-FSB}}(n, \omega, r, c) = 1 + \frac{\omega^2\log_2(\frac{n}{\omega})}{r} = 1 + \frac{\omega^2\log_2(\frac{n}{\omega})}{r} \tag{4.4}$$

This function depends on three main parameters ω, r, and c and is the main measure to estimate the theoretical performance of S-FSB.

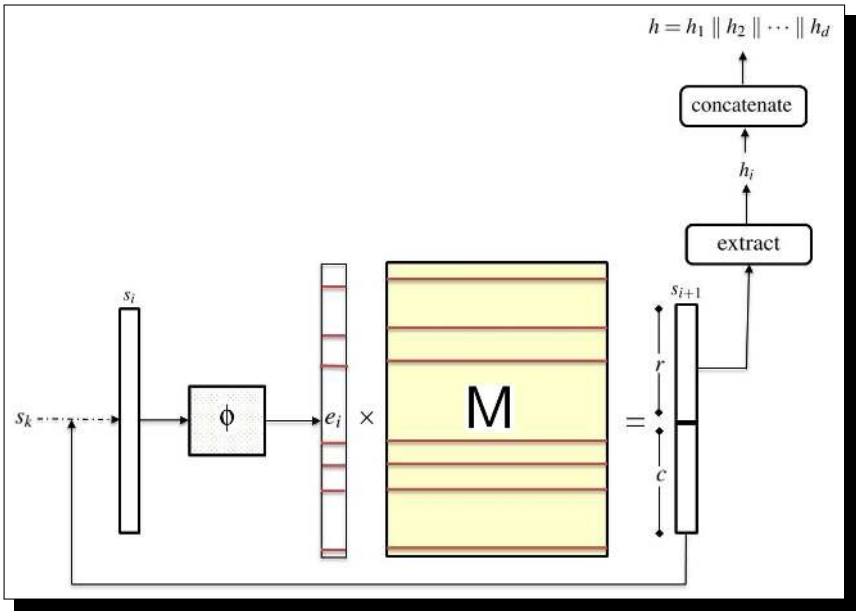

Figure 4.3.: Squeezing step of S-FSB hash function.

4.3.2. Security Analysis

In this subsection, we analyze the security of the S-FSB hash function. We first show how the security of S-FSB is reducible to the two variants of the syndrome decoding problems as in FSB. More precisely, finding pre-images (resp. collisions) for S-FSB is related to the hardness of the regular syndrome decoding pro (resp. 2-null-regular syndrome decoding) problem. Then we identify all best-known attacks against the S-FSB hash function and estimate their minimal complexities.

Theoretical Security

For analyzing the theoretical security of S-FSB, we need to introduce the following definitions to understand how generic attacks work against sponge-based hash functions. With generic attacks, we mean attacks that do not exploit special properties of the underlying transformation.
In our analysis we will denote by $[e]_c$ and $[e]_r$ the inner and the outer state of an ℓ-bit state e, respectively. That is, $e = [e]_r \| [e]_c$ with $\ell = r + c$.

Definition 4.3.1 (Absorbing function)**.** *The absorbing function* $\texttt{abs}(\cdot)$ *of a sponge construction* $\mathcal{S}$*, takes as input a padded message x of length multiple of r and returns the value of the state e obtained after absorbing x, i.e.* $\texttt{abs}(x) = e$.

Definition 4.3.2 (Path)**.** *Let* $\texttt{abs}(\cdot)$ *be the absorbing function. An input x is called path to the state e if* $\texttt{abs}(x) = e$.

Definition 4.3.3 (Squeezing function)**.** *The squeezing function* $\texttt{sqz}(\cdot)$ *of a sponge construction* $\mathcal{S}$*, takes as input an* ℓ*-bit state e given at the beginning of the squeezing step and returns an l-bit string the output truncated to l bits of* $\mathcal{S}$.

In general, it is difficult to find an ℓ-bit state e that satisfies $\mathtt{sqz}(e) = z$, when the length of z is large enough.

Definition 4.3.4 (Output binding problem)**.** *Given a random string z. Output binding problem is to find a state e such that $\mathtt{sqz}(e) = z$.*

In output binding the string z is not mandatory equal to the result of the squeezing of a state e and hence the equation $\mathtt{sqz}(e) = z$ may admit no solution. The expected number of solutions of this problem is $2^{\ell-|z|}$. If $\ell < |z|$, the probability to find a solution is roughly $2^{\ell-|z|}$. On the other hand, if the string z has been actually obtained by squeezing an state e, then we talk about the state recovery problem, which reads as follows.

Definition 4.3.5 (State recovery problem)**.** *State recovery problem consists in finding a state e, given a string z verifying $\mathtt{sqz}(e) = z$.*

Definition 4.3.6 (State collision)**.** *A state collision is a pair of two distinct paths x, x' such that $\mathtt{abs}(x) = \mathtt{abs}(x')$.*

Definition 4.3.7 (Inner collision)**.** *Let x, x' be to distinct paths with $\mathtt{abs}(x) = e$, and $\mathtt{abs}(x') = e'$. The pair (x,x') is said to be an inner collision if $e_c = e'_c$.*

Definition 4.3.8 (Output collision)**.** *Let $\mathcal{S}$ be a sponge construction. An output collision (or just collision) for $\mathcal{S}$ is a pair of distinct paths x, x' that have the same hash value under $\mathcal{S}$, i.e. $\mathcal{S}(x) = \mathcal{S}(x')$.*

The following proposition shows the relation between inner collision and state collisions.

Proposition 4.3.1. *Let $\mathcal{S}$ be a sponge construction with absorbing function $\mathtt{abs}(\cdot)$. The problem of finding state collisions for $\mathcal{S}$ is equivalent to the problem of finding inner collisions for $\mathcal{S}$.*

Proof. First we show that a state collision implies an inner collision. Let x, x' be a state collision. Then, by definition we have $\mathtt{abs}(x) = \mathtt{abs}(x')$, which implies that $[\mathtt{abs}(x)]_c = [\mathtt{abs}(x')]_c$, and hence x, x' is also an inner collision.
Conversely, assume the existence of inner collisions. That means, there exist two distinct paths z and z' with $[\mathtt{abs}(z)]_c = [\mathtt{abs}(z')]_c$. In order to construct a state collision, we perform the following steps:

- Compute the r-bit parts of z and z', i.e. $[\mathtt{abs}(z)]_r$ and $[\mathtt{abs}(z')]_r$,
- Find two r-bit strings y and y' such that $[\mathtt{abs}(z)]_r \oplus y = [\mathtt{abs}(z')]_r \oplus y'$.
- Set $t = z \parallel y$ and $t' = z' \parallel y'$.

By doing so, t and t' are distinct and form a state collision for $\mathcal{S}$. Indeed, the states $([\mathtt{abs}(z)]_r \oplus y) \parallel [\mathtt{abs}(z)]_c$ and $([\mathtt{abs}(z')]_r \oplus y') \parallel [\mathtt{abs}(z')]_c$ are equal and hence lead to the same value under $\mathtt{abs}(\cdot)$. Furthermore, any pair of the form $t \parallel t^*$ and $t' \parallel t^*$, where t^* is an arbitrary input block leads to an output collision, independent of the hash length.

□

In order to analyze the security of S-FSB hash function, we first explain how the sponge construction based on a random transformation can be generically cryptanalyzed. We then apply these cryptanalysis techniques when replacing the random transformation by the code-based one, which is

defined by equation 4.1. The most techniques described here are given in [BDPA11a].

Let $\mathcal{S}$ be a sponge construction with absorbing function $\texttt{abs}(\cdot)$, and squeezing function $\texttt{sqz}(\cdot)$. Let $\mathcal{F}$ denote a random transformation and $\mathcal{T}$ be the function defined in S-FSB. For attacking the sponge construction in general, the following strategies can be used to find (output) collisions, preimages, and second preimages.

Output collisions. As proven in Proposition 4.3.1, the existence of inner collisions plays an important role to build collisions in the hash value. Therefore, in order to end up with collision resistance in the hash values, it is sufficient to prevent inner collisions. As shown in [BDPA11a], an inner collision is clearly producible with workload $\min(2^{\frac{c+3}{2}}, 2^{\frac{l+3}{2}})$, when a random transformation f is used, where l is the hash length. However, when looking for inner collisions in S-FSB based on $\mathcal{T}$, the security is equivalent to the security of an FSB variant mapping to c bits. More precisely, let Let (n, ω, r, c) be the parameter sets for an S-FSB hash function scheme. Moreover, let $n = n_r + n_c$ and $\omega = \omega_r + \omega_c$ be the corresponding columns of H and weights of the input regular word belonging to the first r bits and the last c bits, respectively, i.e. $n_r = (n \cdot r)/s$ and $\omega_r = (\omega \cdot r)/s$. Thus, the workload to produce an inner collision for S-FSB is exactly as an FSB with parameters (n_r, c, ω_r). In [AFG+08], it is shown that finding a collision for the function $\mathcal{T}$ applied on FSB is at least as difficult as finding a word of weight $\leq 2\omega$ and vice versa. That meas, finding inner (and also output) collisions in S-FSB requires solving an instance of the 2-Null Regular Syndrome Decoding problem with parameters (n_r, c, ω_r). Moreover, we need to consider the entire mapping $\mathcal{T} : \mathbb{F}_2^\ell \to \mathbb{F}_2^\ell$ according to state collisions. Here, we need to ensure that the instantiation of an 2-NRSD with parameters (n, ℓ, ω) is hard. To sum up, S-FSB comes up with the following bound on the workload to produce a collision.

Proposition 4.3.2 (collision resistance)**.** *Let h be an S-FSB(n, w, r, c) hash function scheme instantiated with parameters (n, ω, r, c) where $n = n_r + n_c$ and $\omega = \omega_r + \omega_c$. For any adversary $\mathcal{A}$ the lower bound of the workload to output two distinct input messages x, x' mapping to the same hash value is* $\min(RSD(n_r, c, 2\omega_r), RSD(n, \ell, 2\omega))$.

Proof. Let $\mathcal{A}$ be an adversary successfully returning two preimages x, x' mapping to the same hash value y of an S-FSB instance with parameter sets (n, ω, r, c). We show that given $\mathcal{A}$ we build an adversary $\mathcal{B}$ either providing a collision in FSB with parameter sets (n_r, c, ω_r) or solving efficiently a syndrome decoding problem $\mathrm{SD}(n, s, w)$. Due to the sponge construction, an inner collision suffices to produce (multi) collisions. An inner collision implies solving a FSB instance with parameter sets (n_r, c, ω_r). If $\mathcal{A}$ outputs efficiently a (outer) collision which consequently means an inner collision as well, $\mathcal{B}$ simply returns both preimages as a collision for the FSB instance. The workload solving $FSB(n_r, c, \omega_r)$ equals $RSD(n_r, c, 2\omega_r)$, proven in [AFG+08].

If $\mathcal{A}$ strategy neglects inner collisions and approaches directly to find two distinct preimages to the mapping $\mathcal{T}$ of S-FSB, similar to FSB, finding a collision in $\mathcal{T}$ for random binary matrix H equals the 2-RNSD problem, i.e. $\mathcal{B}$ returns $\bar{x} = x + x'$ given by $\mathcal{A}$ as the solution of the 2-RNSD with parameter (n, s, ω). □

(Second) Preimages. Let l be the hash length. To find a preimage for $\mathcal{S}$ based on $\mathcal{F}$, an attacker $\mathcal{A}$ has the following two approaches to succeed. Firstly, $\mathcal{A}$ could bind the output $h \in \{0,1\}^l$ to a state $s \in \{0,1\}^\ell$. Then, $\mathcal{A}$ inverts $\mathcal{T}$ (if possible) to reach a state $t = \mathcal{F}^{-1}(s) = t_r || t_c$. At that point,

a possible path $t_{pre} = t_{pre,r}||t_{pre,c} \in \{0,1\}^{\ell}$ is required, which leads to the same capacity as t, i.e. $t_{pre,c} = t_c$. Hence, $\mathcal{A}$ takes the path to t_{pre} and let $t_{pre,r} \oplus t_r)$ be absorbed by $\mathcal{T}$. This results to a desired preimage of output h. The overall workload amounts to the cost of inverting $\mathcal{T}$ plus $2^{c/2} + 2^{l-r}$. If inverting $\mathcal{T}$ is impossible, the output is bound directly to state t. We require that the first r bits of $\mathcal{T}(t)$ matches to the l bits of h. The workload of finding such a state U is of order $2^l + 2^{c-1}$.

Now, we analyze the security against preimage attacks when we replace the random function $\mathcal{F}$ by $\mathcal{T}$ as this is the case in S-FSB. Since $\mathcal{T}$ is invertible (even if it is hard) both approaches described above are realizable.

The first method needs to invert $\mathcal{T}$ or equivalently, solving an instance of the regular syndrome decoding. Thus, the cost to find a preimage to mapping H costs $\mathrm{RSD}(n,\ell,\omega)$. Finding a path to state t_{pre} costs exactly as an inner collision. A more efficient way to produce a preimage is given by the second approach. First, we find a state $t \in \{0,1\}^{\ell}$ such that $\mathcal{T}(t)$ matches the l bits of output h. Therefore, we require to solve an RSD instance with parameters (n,ℓ,ω) and end up with an expected workload of the order $RSD(n,\ell,\omega) + RSD(n_r,c,\omega_r)$ whereas the latter term corresponds the workload for finding a path to the desired state.

A second preimage $x' \in \{0,1\}^{\ell}$ to a given image $y = \mathcal{T}(x)$ is obtained either by pursuing the strategy of finding a preimage as described above or by finding a second path to one of the inner states of message x. In [BDPA07], it is shown that for a random function $\mathcal{S}$ the workload to output a second preimage is of order $2^c/|P|$ if $|P| < 2^{c/2}$ where $|P|$ denotes the bit length of a given preimage P. Note that the expected workload is at least as large as outputting an inner collision since a second preimage implies an inner collision.

When replacing the random function $\mathcal{F}$ by transformation $\mathcal{T}$, finding a second path to one of the inner states results in an expected workload of inverting an syndrome decoding instance with inputs (n_r,c,ω_r) divided by the the path length $|P|$. To summarize, we obtain the following proposition concerning the (second) preimage resistance of S-FSB.

Proposition 4.3.3 (Preimage Resistance)**.** *Let S-FSB$_{(n,\omega,r,c)}$ be an S-FSB hash function scheme instantiated with parameters (n,ω,r,c) with $n = n_r + n_c$ and $\omega = \omega_r + \omega_c$. For any adversary $\mathcal{A}$ the expected workload to find a message x such that $h =$ S-FSB$_{(n,\omega,r,c)}(x)$ to a given $h \in \{0,1\}^l$ is of order* $\min(RSD(n,\ell,\omega) + RSD(n_r,c,\omega_r), 2^l + 2^{c-1})$,

Proposition 4.3.4 (Second Preimage Resistance)**.** *Let S-FSB$_{(n,\omega,r,c)}$ be a hash function scheme instantiated with parameters (n,ω,r,c) with $n = n_r + n_c$ and $\omega = \omega_r + \omega_c$. For any adversary $\mathcal{A}$ the expected workload find a message x' with $x \neq x'$ to a given message $x \in \{0,1\}^m$ such that S-FSB$_{(n,\omega,r,c)}(x) =$ S-FSB$_{(n,\omega,r,c)}(x')$ is of order* $\min(RSD(n_r,c,\omega_r), 2^{c/2})/|P|$, *where $|P|$ denotes the bit length of a given preimage P.*

Practical Security

As shown previously, the security of S-FSB is related to the hardness of solving instances of two problems: RSD and 2-NRSD problem. In practice, to assess this security regarding the collision and (second) preimage resistance, we have to identify all known applicable attacks and to estimate the minimal complexities required to execute these attacks. As far as we know, there exist two kind of attacks: Information Set Decoding (ISD), Generalized Birthday Attack (GBA). The essential idea behind these algorithms is explained in the previous Chapter.

Information Set Decoding (ISD). Let $P_r(n,\ell,\omega)$ be the probability that a given information set is valid for one given solution of RSD. Let denote by $N_r(n,\ell,\omega)$ the expected number of solution of RSD. As stated in [AFS05], the probability $P(n,\ell,\omega)$ can approximated by $P(n,\ell,\omega) = P_r(n,\ell,\omega) \times N_r(n,\ell,\omega)$. Since there exist $\left(\frac{n}{\omega}\right)^{\omega}$ regular words, then the average number of solutions of RSD is

$$N_r(n,\ell,\omega) = \frac{\left(\frac{n}{\omega}\right)^{\omega}}{2^{\ell}}.$$

In our setting, we have $\ell = \omega \log 2(n/\omega)$. This results in $N_r(n,\ell,\omega) = \frac{\left(\frac{n}{\omega}\right)^{\omega}}{\left(\frac{n}{\omega}\right)^{\omega}} = 1$. That means, we have only one solution to RSD, on average. Furthermore, as shown in [AFS05], the $P_r(n,\ell,\omega)$ is given by

$$P_r(n,\ell,\omega) = \left(\frac{s}{n}\right)^{\omega} = \left(\frac{\log_2(n/\omega)}{n/\omega}\right)^{\omega}$$

If we set $\log_2(n/\omega) = \beta$, for some integers β, then the final probability of selecting a valid set to invert RSD equals to:

$$P(n,\ell,\omega) = P_r(n,\ell,\omega) \times N_r(n,\ell,\omega) = \left(\frac{\beta}{2^{\beta}}\right)^{\omega} \text{ with } \beta = \log_2(n/\omega). \tag{4.5}$$

To estimate the cost of finding collisions, we have to evaluate the complexity of solving the 2-RNSD problem stated above. This can be done in the same way as in [AFS05]. We compute the number of two-regulars words, then we multiply it by the probability of the validity, to get the total probability of choosing a valid set. This probability, denoted by P_I, is given by:

$$P_I(n,\ell,\omega) = \left(\frac{\omega}{n}\right)^{\omega} \left[\binom{\log_2(n/\omega)}{2} + 1\right]^{\omega}$$

For simplicity, we can assume that $\beta \geq 2$. So, we get an upper bound for this probability, denoted by P_C, which is equal to:

$$P_C(n,\ell,\omega) = \left(\frac{\beta^2}{2^{\beta+1}}\right)^{\omega} \text{ with } \beta = \log_2(n/\omega). \tag{4.6}$$

From the equation (4.6), we conclude that the probability for a random information set to be valid in case of collisions search is larger by a factor $(\frac{\beta}{2})^{\omega}$ compared to the probability for a random information set to be valid in case of finding preimages, where $\beta = \log_2(n/\omega)$.

In practice, there exists a lower bound for information set decoding attacks, presented in [BTP11]. Moreover, a new variant of ISD algorithm [BLPS11a] was developed for estimating the hardness of solving the 2-Regular Null Syndrome Decoding problem (2-RNSD). These algorithms run faster than the lower bounds given in [FS09]. The parameters we propose in the next section are chosen to resist all these attacks.

Generalized Birthday Attack (GBA). As in Chapter 3, we use the the attack from Matthieu and Sendrier [FS09], which relies on the Generalized Birthday Problem introduced by Wagner [Wag02], whose idea is as follows.

For a given integer α, to find a set of indexes $I = \{1, 2, \cdots, 2^\alpha\}$ verifying

$$\bigoplus_{i \in I} H_i = 0.$$

To find this set I, one has to compile 2^α lists of $2^{\frac{\ell}{\alpha+1}}$ elements containing distinct columns of the matrix H of size $\ell \times n$. These lists are then pairwise combined to get $2^{\alpha-1}$ lists of XORs of 2 columns of H. In the resulting lists, only 2 columns starting with $\frac{\ell}{\alpha+1}$ zeros are kept, instead of all the possible columns. Then, the new lists are pairwise merged to obtain $2^{\alpha-2}$ lists of XORs of 4 columns of H. Only 4 columns of H starting with $2\frac{\ell}{\alpha+1}$ zeros, are kept. This process will be continued, until only two lists are left. These two lists will contain $2^{\frac{\ell}{\alpha+1}}$ XORs of $2^{\alpha-1}$ columns of H having $(\alpha-1)\frac{\ell}{\alpha+1}$ zeros at the beginning. After that, the standard birthday algorithm can be applied to get one solution. Since all lists treated above, have the same size $\frac{\ell}{\alpha+1}$, the complexity of GBA is at least in $O\left(\frac{\ell}{\alpha+1} 2^{\frac{\ell}{\alpha+1}}\right)$.

As we can see in this algorithm, the number of XORed columns was a power of 2. However, this does not hold in general because the weight w can be any number. So if w is not a power of 2, one can modify the above algorithm such that one can back in the general case of GBA by imposing the following condition on α: $\frac{1}{2^\alpha}\binom{n}{\frac{2\omega}{2^\alpha}} \geq 2^{\frac{\ell-\alpha}{\alpha}}$ (see [FS09] for more details). This condition can be rewritten as:

$$\binom{2^\beta \omega}{2^{(1-\alpha)}\omega} \geq 2^{\beta\omega + \alpha(\alpha-1)} \tag{4.7}$$

where $\log_2(n/\omega) = \beta$. In this case, one gets a lower bound of the cost of solving an instance SD problem with parameters (n, ℓ, ω) as follows:

$$\left(\frac{\omega\beta}{\alpha} - 1\right) 2^{\frac{\omega\beta}{\alpha}-1}. \tag{4.8}$$

As we can see, for fixed weight ω, this complexity is an increasing function in n. So, to avoid the GBA attack, we have to choose large n.

In [BLN+09] an implementation of GBA is presented against the compression function of FSB. This implementation includes two techniques introduced in [Ber07] in order to mount GBA on computers, which do not have enough storage capacity to hold all list entries. However, the complexity of this attack is still exponential. Since our scheme is based on the FSB compression function, we claim that our proposal is secure against this implementation.

Other possible attacks. In addition to the previous attacks, it was shown in [GLP08] that the sponge-based hash functions can be attacked by slide attacks. This kind of attacks was introduced in [BW99] by Biryukov et.al for cryptanalyzing iterative block ciphers. For attacking a sponge-like construction, the self-similarity issue can be exploited, meaning that all the blank rounds behave identically. As noted in [GLP08], a simple defense against slide attacks consists in adding nonzero

constant just before running the blank rounds. This can be achieved by a convenient padding such that the last block of the message is different from null vector. That is exactly, what we are used in our construction. Therefore, our proposal is secure against slide attacks. In [Saa07], the so-called linearization attack (LA) was proposed against FSB to find collisions. The key idea is to reduce the problem of finding collisions to a linear algebra problem that can be solved in polynomial time, when the ratio ℓ/ω is up to 2. Furthermore, as shown in [Saa07], this attack can still be applied if $\ell > \omega$. It can be extended even to $\ell > 2\omega$ with complexity $O(\ell^3 \left(\frac{3}{4}\right)^{\ell-2\omega})$. So, to avoid the LA attack, we have to choose $\ell > 2\omega$.

4.3.3. Parameters Choice

When selecting parameters for S-FSB, we have to look for parameters providing the desired security with least processing cost required to hash one bit of the message. As mentioned in Section 4.3, this cost can be theoretically measured using the function N_{sfsb} defined by the following equation

$$\mathrm{N}_{S-FSB}(n,\omega,r,c) = 1 + \frac{\omega^2 \log_2(n/\omega)}{r} = 1 + \frac{\omega\ell}{r} = 1 + \frac{\omega(r+c)}{r} = 1 + \omega\left(1 + \frac{c}{r}\right) \tag{4.9}$$

We observe that for increasing values of c, this function is an implicitly increasing quantity in ω and n. So, if we want to have a good performance, then we have to choose small values of c (as small as possible) and select w and n such that the value of r are large. But from security point of view, we should choose ℓ greater than $2\omega+1$ to withstand the linearization attack mentioned earlier. Furthermore, to avoid inner and outer collisions, the running time of solving instances of RSD and 2-RNSD with parameters (n,ℓ,ω) and (n_r,c,ω_r) according the best known collision attack, must be larger than the desired security.

Starting from those conditions, we propose three parameter sets (n,ℓ,ω,c) that provide different security levels. Those sets of parameters are presented in Table 4.2 together with the corresponding numbers of XORs and the complexities of the ISD and GBA attacks.

						Preimage		Collision	
Hash size l	n	ℓ	ω	c	N_{S-FSB}	GBA	ISD	GBA	ISD
160	$3\cdot 2^{19}$	384	24	240	64.0	2^{130}	2^{99}	2^{86}	2^{91}
224	$17\cdot 2^{17}$	544	34	336	88.9	2^{150}	2^{144}	2^{114}	2^{122}
256	$39\cdot 2^{17}$	624	39	296	90.5	2^{246}	2^{172}	2^{129}	2^{148}

Table 4.2.: Proposed parameters for S-FSB

4.3.4. Performance and Comparison

S-FSB has been implemented on a 2.53 GHz Pentium Core2 Duo, running Linux (Ubuntu 10.04) 32 Bit with 6MB of cache and 4GB of RAM. The C compiler is GCC, version 4.4.3 with -O3 optimization. The performance of the three versions of S-FSB is reported in Table 4.3. This performance was measured on a message of size 1 GB. The file hash time in the third row was measured by repeated

calls to the `clock()` function to get the current millisecond clock value and subtracted the stop time from the start time. The number of samples we performed is about one million. To get the speed expressed in cycles per bytes, we multiplied the measured hash time by the CPU frequency and divided the result by the file size in bytes. The C-code of S-FSB can be found in [Cay11].

In order to compare our results with those of FSB SHA-3 proposal [INR07], we ran the C-code of FSB on the same desktop and we obtained the results presented in Table 4.5. As we can see, the S-FSB is more efficient than FSB by a factor of 1.44 (30%). Furthermore, we have small storage capacity comparable to FSB. We leverage quasi-cyclic codes in our implementation. Despite these improvements, the S-FSB hash family remains slower than the existing hash functions like the SHA-2 family.

Hash size (bits)	File size (MB)	File hash time (s)	Speed (cpb)
160	1000	66.90	160
224	1000	84.48	201
256	1000	75.63	183

Table 4.3.: Performance of S-FSB in [Cay11] .

Hash size (bits)	File size (MB)	File hash time (s)	Speed (cpb)
160	1000	87.76	212
224	1000	102.99	248
256	1000	109.38	264

Table 4.4.: Performance of FSB SHA-3 proposal in [INR07].

It is worth mentioning that in [CSM] optimized implementations of FSB and S-FSB are proposed and available on [San12]. Using the same parameters and for the same hash size as above, their implementation results are listed below.

Hash size (bits)	Speed of FSB (cpb)	Speed of S-FSB(cpb)
160	110	79
224	131	99
256	204	172

Table 4.5.: Performance of FSB and S-FSB in [CSM].

4.4. The RFSB Hash Function

In this section, we describe the RFSB hash function [BLPS11b] (Really Fast Syndrome Based hash function), which is a further enhanced variant of FSB in terms of efficiency.

4.4.1. Description of the RFSB hash function

RFSB [BLPS11b] was also proposed in 2011 and slightly changes the way the quasi-cyclic matrices are handled. The shift is made during the sum up, depending only on the rank in this sum. It also takes advantage of hardware considerations to improve the efficiency a lot. RFSB selects the xored columns in a simpler way, which alleviate the matrix a lot. Theses columns are then shifted and properly xored. This approach was chosen to simplify the implementations as mush as possible. Like FSB, the RFSB follows also the MD-design and its compression function G takes s-bit inputs and returns ℓ bit strings, where $s = \omega b > \ell$ for some positive integer b verifying $n = \omega 2^b$. Formally, the function G is described as follows. Let $\mathbf{x} = (x_1, x_2, \cdots, x_\omega)$ be an s-bit input of G, where each x_i of length b bits, which is represented by an integer y_i between 0 and $2^b - 1$. Let H be a matrix of size $\ell \times n$. As $n = \omega 2^b$, H contains ω block matrices, each of size $\ell \times 2^b$. That is, $\mathsf{H} = \mathsf{H}_1 \parallel \mathsf{H}_2 \parallel \cdots \parallel \mathsf{H}_\omega$. According to the MD-construction, $\mathbf{x}$ is composed of a $(s-\ell)$-bit message block and the current state of size ℓ. The next state $G(\mathbf{x})$ is computed by:

$$G(\mathbf{x}) = h_1^{(y_1)} \oplus h_2^{(y_2)} \oplus \cdots \oplus h_\omega^{(y_\omega)}.$$

Where $h_j^{(y_j)}$ is a certain column of H_j, for $j = 1, \cdots, \omega$. Figure 4.4 shows how $G(\mathbf{x})$ is calculated. A detailed RFSB hash algorithm can be found in [MAC11, Algorithm 5].

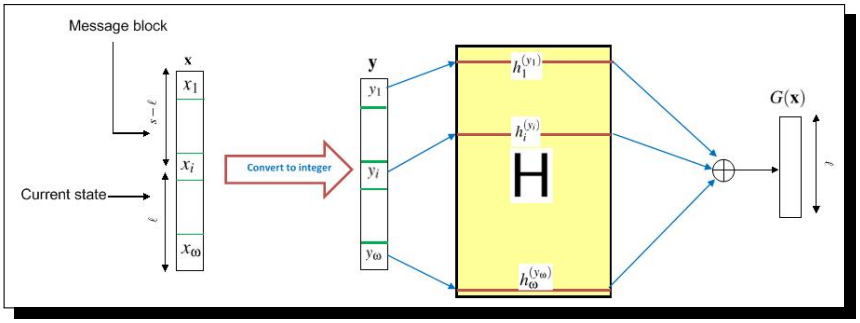

Figure 4.4.: Compression function of RFSB hash function.

The specifications of RFSB in [BLPS11b] only take into account one set of parameters, which is intended to use the hardware registers at their best. This set is $(n, \ell, \omega) = (28672, 509, 112)$.

4.4.2. Security of RFSB

In [BLPS11b], a detailed security analysis of RFSB is done showing that it is secure against all the best attacks like ISD and GBA algorithms. RFSB is designed to provide 128-bit security. From a theoretical point of view, it is straightforward to prove that the security of RFSB is related to the hardness of solving instances of RSD and 2-NRSD problems.

4.4.3. Performance of RFSB

In [BLPS11b] the authors report an implementation of RFSB with $\ell = 509$ (noted by RFSB-509) that outperforms SHA-256 on Intel Core 2 Quad Q9550 CPUs at 13.62 versus 15.26 cycles/byte. The latest measurements available on eBASH project* show that RFSB-509 runs even at 10.64 cycles/byte while SHA-256 remains at 15.31 cycles/byte on the same platform. Furthermore, [RWMC11] presents a software implementation for RFSB with $\ell \in \{227, 379, 509, 1019\}$ and provides performance measurements for all four instances. However, the speeds in [RWMC11] are not close to the speeds reported in the original RFSB paper (e.g., the speed of RFSB-509 is 120.5 cycles/byte on an Intel i7 CPU). In addition, the authors in [CSM] suggest a implementation for RFSB-509 and reports a speed only of 17.26 cycles/byte. Table 4.6 gives speed information about RFSB variants according to the available implementations.

Variant	Speed (cycles/byte) in [RWMC11]	Speed (cycles/byte) in [CSM]	Speed (cycles/byte) in [BLPS11b]
RFSB-227	42.4	-	-
RFSB-379	62.8	-	-
RFSB-509	120.5	17.26	13.62
RFSB-1019	152.8	-	-

Table 4.6.: Performance of RFSB.

4.5. Conclusion and Open Problems

We have shown how to construct a provably secure variant of the FSB hash family following the sponge construction. Although its speed is better than the original FSB hash function and its security is reducible to the hardness of solving to NP-complete problems, it suffers from some drawbacks. It remains far slower than SHA-1 family, because it possesses a long initialization step for generating the underlying matrix. Furthermore, its description is large (this is also the case for FSB), since it require big matrices generated from about two millions bits of the digits of π. A simple way to reduce the size of these matrices is to introduce a constant weight encoder, which requires only small parameters. However, weight encoders run slower than regular ones. They will slow down the computation too much, and hence worsen the performance as well as the security of the hash function. Hence, an efficient construction of such a encoder would be an asset but remains an open problem. Another useful trick to increase the speed of the hash function is to renounce the underlying encoder and use another technique to perform the matrix-vector multiplication without any encoding algorithm as introduced in RFSB.

*http://bench.cr.yp.to/ebash.html

Chapter 5

Parameters Selection for the McEliece-like Cryptosystems

5.1. Motivation

Public key encryption plays a vital role in securing sensitive data in practical applications. When public key cryptosystems are used in such applications appropriate key sizes must be selected. For number-theory-based systems such as RSA, and EC-ElGamal, Lenstra and Verheul proposed a framework on how to select appropriate keys that provide security until a given year. In code-based cryptography, as far we know, this is still an open problem. In this chapter we address this issue and show how to select optimal parameters for the McEliece cryptosystem that provide security until a given year and give detailed recommendations. This is our main contribution, which has been appeared in [NMBB12].

5.2. Our Security Model

In this section we introduce the security model we use. This model is obtained by adapting the Lenstra-Verheul framework based on Moores Law and is essentially made up of three parts. First we begin by analyzing the McEliece cryptosystem from security point of view. Then we explain how the Lenstra-Verheul model works and finally we give a sensitive analysis, which quantifies the robustness of our model, and it allows users to apply our results even if they have different assumptions about the correct values.

Security of the McEliece cryptosystem

For attacking the McEliece cryptosystem, there exist to families of algorithms. The structural and decoding attacks. The former aim at recovering the private key from the public key. [EOS06] gives a detailed overview of these attacks. The latter attempts to derive the plainttext from a given cipertext, and is based on Information Set Decoding (ISD) technique. It seems to outperform all other techniques in terms of complexity. Therefore, our security analysis will based on the complexity of this

kind of attack.

Many ISD algorithms have been developed and proposed in the literature. The most important of these are presented in Table 5.2, together with their respective complexity to decode a $(1024, 524, 50)$ Goppa code (these are the original McEliece parameters).

Year	Algorithm	Binary logarithm of complexity
1986	Adams-Mejier [AM89]	80.7
1988	Lee-Brickell [LB88]	70.89
1989	Stern [Ste89]	66.21
1994	Canteaut-Chabanne [CC94]	65.5
1998	Canteaut-Chabaud [CC98]	64.1
2008	Bernstein-Lange-Peters [BTP08]	60.4
2009	Finiasz-Sendrier [FS09]	59.9

Table 5.1.: Complexity of ISD algorithms against $(1024, 524, 50)$ McEliece cryptosystem

For estimating the security level, denoted by $S(n,k,t)$, against a given (n,k,t) Goppa code, we use the lower bounds of ISD algorithms proposed in the paper [NCBBb] (extended version of [NCBBa]), which is based on the lower bounds in [FS09] and the idea of [BLP11].

Remark 5.2.1. *In [FGO$^+$10] a distinguisher against binary Goppa codes of high rate has been proposed. This distinguisher uses the algebraic techniques introduced in [FOPT10] and works only under certain requirements on the parameters. The parameters we propose, however, do not satisfy these constraints, and are therefore secure against the techniques in [FGO$^+$10].*

Remark 5.2.2. *In [JJ02] a further ISD algorithm has been published, which attempts to solve only one SD instance out of many. However no asymptotic analysis of the advantage of this algorithm is provided when attacking multiple targets instead of 1. Against the (1024,524,50) McEliece cryptosystem with a single ciphertext, this algorithm requires at least $2^{68.1}$ binary operations.*

Lenstra-Verheul model

In [LV01] Lenstra and Verheul (LV) proposed a mathematical model providing key length recommendations for public-key cryptosystems based on integer factorization (IF), discrete logarithm (DL), and elliptic curve DL. This is the first important work that uses a mathematical approach for the determination of secure key sizes based on concrete parameters. After the introduction of this model, several papers made use of it to find appropriate key lengths for cryptographic primitives (see, for example, [MQSW01], and [Sze08]). Furthermore, many companies have used this model to estimate the accepted key length for their cryptographic applications. For instance, in 2004, McAfee, the computer security company applied the LV-model to find the minimal key size for SSL connections [Ara04]. Another interesting organization is the BlueKrypt company which hosts the website `www.keylength.com`. This site has an implementation of the LV-model and summarizes reports from well-known organizations allowing the evaluation of the minimum security requirements for some symmetric and asymmetric systems in the future. The LV-model is explained in more detail below.

The LV-model is based on a number of assumptions that combine the impact of cryptanalytic progress and the effect of changes in computing environment. The key points of this model on which the choice of parameters depends are the following:

1. **`Security margin:`** It is the year s which is used to "anchor" the extrapolation. In [LV01] the default value of s is 1982 which represents the last year for which it is assumed that a 56-bit key DES cryptosystem provides adequate security for commercial use. The computational effort for breaking the 56-bit DES system was estimated to be $5 \cdot 10^5$ MIPS-years.
   In order to estimate the security level provided at a given year, Lenstra and Verheul define a function $\mathrm{IMY}(y)$. This abbreviation stands for "Infeasible number of MIPS-years* for year y", and it refers to the minimum computational effort that is expected to be infeasible to do in year y.
   In general, we define $\mathrm{IMY}(y)$ in such a way that a successful attack using tens of thousands of year-y CPUs requires more than 100 years to finish. The number of CPUs is a rough estimate for the effort a security agency might put into an attack. The number of years is derived from the fact that US law used to require some national secrets to be protected for 75 years†.

2. **`Computing environment:`** This estimates the changes in computational power available to attackers. This estimation is based on a slight variation of Moore's law by introducing three variables a, b, and c that specify the changes in hardware speed, IT budget, and price over time. The definitions of these variables and their default values are as follows:

   - a is the expected average number of months in which processor speed and memory size increase by a factor of two. The default value is $a = 18$, which is the value specified by Moore's law and is so far in line with current hardware developments. In this paper we are going to use the same value due to the fact that over the last years, hardware development has resulted in a doubling of transistors (for a fixed price) every 12–24 months‡. Thus, a default of 18 is a compromise of this historic data. Also, opinions differ in whether hardware development will slow down or new technologies will further accelerate it;
   - $c \in \{0,1\}$ indicates how to interpret the variable a. For $c = 0$, the amount of computing power and memory *which is available to an attacker* doubles every a months, while for $c = 1$, the computing power and RAM *for a given price* double every a months. We will use $c = 1$ since the historic trend mentioned above refers to a fixed price.
   - b is defined as the average number of months it takes for IT budgets to double. According to historic data§, the US Gross National Product has doubled approx. every 10.5 years over the last 30 years. Since the exact growth varies every year, we will use an average value to extrapolate over a larger period of time. Our default setting for b is 120.

3. **`Cryptanalysis:`** This refers to the expected cryptanalytic progress. It is measured by the number of months r it is expected for cryptanalytic attacks to become twice as effective. We estimate this number by attacks against code-based cryptosystems only, since the cryptanalytic

*MIPS = million instructions per second

†For example, the report on the Kennedy assassination; see `http://en.wikipedia.org/wiki/John_F._Kennedy_assassination`

‡See http://wi-fizzle.com/compsci/

§See http://www.bea.gov

development can be very different for other cryptosystems. Lenstra and Verheul's default value is $r = 18$. In code-based cryptography, we find it reasonable to assume that the pace of future cryptanalytic developments and their impact will be relatively close to what we have seen from 1988 until 2009. By applying a linear regression on data points listed in Table 5.2, we get a line whose slope roughly equals -0.41 meaning that a twofold attack efficiency improvement will happen in each $1/0.41 \approx 2.44$ years. Also the value of r is $r = 2.44 \cdot 12 \approx 29.27$. In this paper, we take $r = 30$, which corresponds to 2.5 years.

Based on these points, Lenstra and Verheul give a formula allowing to determine lower bounds for the algorithmic complexity that offer a specified security margin at least until year y in the future (independent of the concrete asymmetric cryptosystem). To do this, they show how IMY(y) is estimated from the points above. Given that breaking the DES system takes $5 \cdot 10^5$ MIPS-years, which was infeasible in the year $s = 1982$, the function $\mathrm{IMY}(y)$ is defined by:

$$\mathrm{IMY}(y) = 5 \cdot 10^5 \cdot 2^{12(y-s)/a} \cdot 2^{12c(y-s)/b} \quad \text{MIPS-years.} \tag{5.1}$$

With our default settings, it follows that in year y a computational complexity of

$$\mathrm{IMY}(y) = 5 \cdot 10^5 \cdot 2^{\frac{23}{30}(y-1982)} \quad \text{MIPS-years} \tag{5.2}$$

provides an acceptable level of security. The next step is to convert this lower bound expressed in MIPS-years to a lower bound for the number of binary operations. In order to do that, we use as a data point the result [BTP08] that approximately $2^{60.4}$ binary operations are needed to break the original McEliece with parameters $(1024, 524, 50)$; expecting cryptanalytic developments by a factor $2^{12(y-2008)/r}$ (with $r = 30$), we claim that a sufficient condition for security level, denoted by $S(n,k,t)$, of a McEliece instance with parameter set (n,k,t) providing an adequate security until a given year y is the following:

$$S(n,k,t) \geq \frac{\mathrm{IMY}(y) \cdot 2^{12(y-2008)/30} \cdot 2^{60.4}}{1.7 \cdot 10^5}. \tag{5.3}$$

As in [LV01, Page 9], $S(n,k,t)$ is defined as the expected runtime of the fastest algorithm published today for attacking the McEliece cryptosystem with the parameter set (n,k,t). In our case, this corresponds to the lower bounds presented in [NCBBa, NCBBb]. The value $1.7 \cdot 10^5$ is expressed in MIPS-years and obtained from the fact that the attack by Bernstein et al. [BTP08] required 1400 CPU days on Q6600 quad processors. Assuming that a Q6600 processor [BTP08] does approximately $44,000$ MIPS (SiSoft Sandra benchmark and [AJ07]), this corresponds to $1.7 \cdot 10^5$ MIPS-years.

Therefore, the inequality (5.3) becomes:

$$S(n,k,t) \geq 2.9412 \cdot 2^{\frac{23}{30}(y-1982)+\frac{12}{30}(y-2008)+60.4} \tag{5.4}$$

Analysis and discussion

In this subsection, we provide a sensitivity analysis for the values we explained in the previous subsection. More specifically, we estimate and discuss the impact that a different value of each variable has on the resulting security level.

Security margin 1982 and DES-56 bit

The function IMY(y) was "anchored" by defining 1982 as the last year in which breaking the DES scheme with 56-bit key was considered infeasible. The choice to use DES with 56-bit for this definition is arbitrary; the function, therefore, is defined using the number of operations required to break the DES scheme, and it is thus independent of which cryptosystem was used for the definition.
Any other year and/or cryptosystem can be used for the definition, e.g. AES or RSA. Using the data from the 2008 attack by Bernstein et al. [BTP08] that a 44,000 MIPS CPU breaks the original McEliece parameters in 1400 CPU-days, the attack complexity estimated as 2^{60} operations corresponds to $2^{17.3}$ MIPS years. An attack complexity of 2^{80} operations, which was considered the "smallest general-purpose level" of security[¶] in 2008, corresponds to $2^{37.3}$ MIPS-years, very close to our estimate of $IMY(2008) = 2^{38.8}$.

Moore's Law (parameters a and c)

The original Moore's Law refers to the number of transistors on an integrated circuit [Moo98]. Moore estimated this number to double every two years[‖]. The number of MIPS of a CPU depends on the number of transistors, but also on the clock speed. These two factors taken together increase the chip performance by a factor of two every 18 months [Hou] (estimated by David House, an Intel executive). For our sensitivity analysis, we will consider a 10% error in this estimate, i.e. a range between 16 and 20 months for a twofold performance increase. The value $c = 1$ is in line with past developments, but we will show the impact of $c = 0$ below.

Budget (parameter b)

Our choice for the value of b is based on the budget development of the US, since it constitutes the largest economic power worldwide. However, countries like China have a much higher economic growth; some analysts expect China to overtake the US in the near future, doubling the US economic power in 2030[**]. This growth corresponds to a twicefold increase in economic power in 6 years. Even though GDP of China is smaller than that of the US (about 40% in 2010) and the faster growth is therefore on a smaller baseline, we will assume a range of 72–120 for the value of b.

Cryptanalytic progress (parameter r)

For more than two decades, cryptanalytic progress has improved the efficiency of the fastest attack algorithm by a factor of two every 30 months. While every individual attack algorithm has a lower bound for its complexity (see, for example, [BTP08, FS09, NCBBa]), many new attacks have been developed which improved the previous bounds. As in the case of Moore's Law, it is unclear whether generic attack algorithms have a lower bound for their complexity that cannot be improved, thereby slowing down cryptanalytic progress, or whether new cryptanalytic tools will increase the progress. We will therefore consider a larger range for r, from 20 to 40 months.

[¶] www.keylength.com, ECRYPT II recommendations

[‖] See `http://www.intel.com/technology/mooreslaw/` or `http://en.wikipedia.org/wiki/Moore's_law`

[**] J. Lin, World Bank's chief economist, on March 23rd, 2011

Parameter	Our value	Expected range	Impact Absolute	Impact Percent
a	18	16–20	5.7	4.3%
b	120	72–120	4.5	3.5%
c	1	0/1	6.8	5.2%
r	30	20–40	8.4	6.4%

Table 5.2.: Impact of different input values to our model. Impact is the absolute and percent change in the required security level for the year 2050. For example, ranging parameter a from 16 to 20 changes the security level between 126.4 and 136.7 bit, an absolute change of 5.7 bit and a relative change of 4.3.

It can be seen from Table 5.2 that even very pessimistic assumptions (from an user's point of view) do not lead to dramatic changes in the required security level. For example, assuming the most pessimistic value for all four parameters above raises the required security until at least 2050 from 131 to 149 bit, an increase of 18.6 bits or 14.2%. Table 5.4 (page 75) applies the optimistic and pessimistic assumptions described above and shows optimal parameters for selected years.

5.3. Parameters selection

Our methodology

The problem of estimating secure parameters for the McEliece cryptosystem for security until at least a given year consists in obtaining, for the security level S calculated in (5.3), a set of parameters that achieves this security level and provides the smallest key size among all other such sets. To solve this problem, we use the following methodology:

1. Based on simplified theoretical arguments we show that there exists an optimal information rate $R^* = k/n$ with $R^* \approx 0.8$ such that for a given key size the maximum of security is achieved at this rate.
2. We show how an instance attaining maximum security for a given key size can be used to solve the problem of finding the optimal key size for a given security level.
3. We present an algorithm that we use to find optimal instances that have a rate of ≈ 0.74, corresponding to the arguments from 1.

As pointed out in [Sen02], the complexity of the ISD-algorithms is roughly estimated by

$$C(n,R) = p(n)2^{-t(n,R)\log_2(1-R)}, \tag{5.5}$$

where $p(n)$ is some polynomial in n and $t(n,R)$ the error-correcting capability. For the classical ISD the degree of p is 3 and it gets lower for improvements. In the case of a t-error correcting Goppa code of length n and dimension $k = n - t\lceil \log_2 n \rceil$, the above formula becomes

$$C_G(n,R) = p(n)2^{(c(R)n/\log_2 n)(1+o(1))}, \tag{5.6}$$

where $c(R) = -(1-R)\log_2(1-R)$ is the complexity coefficient. In [Sen02] it is also mentioned that, neglecting $p(n)$ and concentrating only on the exponential part, the following can be shown: for

a given code length n, the highest complexity is achieved at an information rate of $1-e^{-1} \approx 0.63$. Although we will compute our table using the lower bounds from [FS09], we would first like to provide some theoretical evidence that the optimal rate exists also for the problem of the smallest key size. Considering that the numerous improvements of the ISD enhance only the polynomial part significantly, the reasoning appears to be sound. In the following lemma we simplify $C_G(n,R)$ to

$$C_G(n,R) = 2^{(c(R)n/\log_2 n)(1+o(1))}, \tag{5.7}$$

similarly to [Sen02].

Lemma 5.3.1. *Given the key size K, the maximum complexity of an ISD-like algorithm as per (5.7) is achieved at an information rate $R^* \approx 0.8$.*

Proof. First, from the formula $K = R(1-R)n^2$ we have that $n = \sqrt{K/(R(1-R))}$. Now if we substitute this expression in (5.7) we obtain

$$C_G(K,R) = 2^{\left(c_K(R)\sqrt{K}/\log_2 \sqrt{\frac{K}{R(1-R)}}\right)(1+o(1))},$$

where $c_K(R) = -\log_2(1-R)\sqrt{\frac{1-R}{R}}$. So in order to maximize $C_G(K,R)$ for a given K we need to maximize the term $c_K(R)\sqrt{K}/\log_2 \sqrt{\frac{K}{R(1-R)}}$ for K. Fixing K and taking a derivative we have the following equation for obtaining the point of maximum:

$$\begin{aligned} &c'_K(R)\frac{K_s}{\log_2 \frac{K_s}{\sqrt{(1-R)R}}} \\ &+c_K(R)\frac{K_s(1-2R)\ln 2}{2R(1-R)\log_2^2 \frac{K_s}{\sqrt{(1-R)R}}} = 0, \end{aligned}$$

where $K_s = \sqrt{K}$. This simplifies to

$$c'_K(R) + c_K(R)\frac{(1-2R)\ln 2}{2R(1-R)\log_2 \frac{K_s}{\sqrt{(1-R)R}}} = 0. \tag{5.8}$$

Consider now the equation

$$c'_K(R) = 0. \tag{5.9}$$

The solution of this equation is a root of the equation

$$\frac{\ln(1-R)}{R} = -2$$

and numerically this root is $R^* \approx 0.8$. This shows that the function $c_K(R)$ is bounded: $0 < c_K(R) < c_K(R^*)$. Now considering that for K, and thus for K_s, large enough the second summand of (5.8) is negligible, we are left with the equation (5.9). The root of this equation is R^*. □

Now let us show that having the fact that the maximum complexity for the given key size is attained at some R^*, the minimum key size for the given security level is achieved for a code with the same rate R^*.

Proposition 5.3.1. *Let the security level S^* be given. Let $C(K,R)$ be the complexity of a decoding algorithm A for a code with the key size K and rate R. We impose the following formal assumptions on $C(K,R)$:*

(a) *$C(K,R)$ is continuous on $]0,\infty[\times[0,1]$.*

(b) *C is unbounded in K for all R:*
$\forall R \in]0,1[: C(K,R) \to \infty, K \to \infty$.

(c) *C is increasing in K:*
$\forall K_2 > K_1 > 0 \; \forall R \in]0,1[: C(K_1,R) < C(K_2,R)$.

Further, assume that for given K the maximum complexity of A is achieved at R^*:

(d) $\forall K > 0 \; \forall R \neq R^* : C^*(K) := C(K,R^*) > C(K,R)$.

Then the McEliece cryptosystem that satisfies the security level S^* w.r.t A with the smallest possible key size has an underlying Goppa code of rate R^*.

Proof. Due to (a) the function $C^*(K)$ is continuous and due to (c) is strictly increasing. Now because of (b) there exists a solution to $C^*(K) = S^*$. And because of the above mentioned properties of C^* this solution is unique: $C^*(K^*) = S^*$. Finally, the claim of the proposition follows from $S^* = C(K^*,R^*) > C(K^*,R) \; \forall R \neq R^*$. □

Remark 5.3.1. *Conditions (a)–(c) are natural for any complexity function of a decoding algorithm. The property (d) is true at least for ISD-like algorithms as we have seen in Lemma 5.3.1.*

So now we may expect the following to happen in our table. Although we use more advanced lower bounds from [FS09] we still expect that for given K the maximum security will be achieved at some R^*, the same for all K. As we have mentioned, this is due to the fact that the improvements of the ISD algorithm do not seem to improve much on the exponential part. Moreover, because of the same reason we expect this R^* not to differ significantly from the value 0.8 predicted by Lemma 5.3.1. Having this, we then use Proposition 5.3.1 to construct an algorithm that with arbitrary precision finds an instance with the smallest key possible that achieves the given security level S. This algorithm is depicted below (see Algorithm 5). In this algorithm, the value of S is calculated via the inequality (5.4), the interval $[R_{start}, R_{end}]$ is chosen large enough and contains 0.8: we take an information rate which ranges from $R_{start} = 0.6$ to $R_{end} = 0.85$. All other parameters are chosen so that it is feasible to complete the algorithm in a reasonable time. For the key size, we set $K_{up} = 200$ kB as an upper bound and use the step size $K_{step} = 1$ kB. Moreover, we use the lower bound formula from [FS09] as a function C.

Proposed Parameters

Our results are presented in Table 5.3 which shows the following information:

- Year: the year until which data security is required. Historic data is given mainly to allow comparison with other sources.
- Symmetric key size: the symmetric key size required to ensure data security, calculated in accordance with Lenstra and Verheul's approach.

Algorithm 5 Search$(S, C, K_{step}, K_{up}, R_{step}, R_{start}, R_{end})$

Input:
- Security level S
- Complexity function $C(n,R)$ satisfying $(a)-(d)$ of Proposition 5.3.1
- Step for the key size search K_{step}
- Search upper bound for the key size K_{up}
- Step for the rate search R_{step}
- Rate search interval bounds R_{start}, R_{end}

Output: n_{out} and R_{out} such that
- The key size is the smallest possible up to steps K_{step} and R_{step}

for $K = K_{step}$ **to** K_{up} **do**
  for $R = R_{start}$ **to** R_{end} **do**
    $n \leftarrow \sqrt{\frac{K}{R(1-R)}}$
    if $C(n,R) \geq S$ **then**
      return n and R
    end if
    $R \leftarrow R + R_{step}$
  end for
end for

- Lower bound for $\log_2(S(n,k,t))$: the $\log_2$ of the minimum number of binary operations (required to break a McEliece cryptosystem) that are expected still to be infeasible in the respective year.
- The last two columns are a translation of the required symmetric key size into parameters relevant in practice, i.e. the number of MIPS years that render a cryptosystem infeasible to break, and the corresponding number of years on a modern Quad core CPU.

5.4. Conclusion and Open Problems

In this work we have addressed the problem of selecting optimal parameters for the McEliece cryptosystem based on binary Goppa codes. This problem was to find instances of the McEliece cryptosystem that are expected to remain secure at least until a given year and providing the smallest key sizes. The computations were modelled using the Lenstra-Verheul framework which is based on Moore's Law and other assumptions about future developments. For this problem, we have presented detailed parameter recommendations. This allows (potential) users of the McEliece cryptosystem to optimize the parameter choice, thereby improving the applicability of code-based cryptography. We have also shown the fact that all such optimal instances have information rate close to 0.74.

As a next step, we suggest a comprehensive analysis of concrete application scenarios. As we have illustrated above, in these scenarios constraints, as well as the trade-offs between the code properties, strongly depend on the details of the application, e.g. available bandwidth, acceptable response times, or (typical) message size. This analysis would provide further insights into the current strengths and limitations of code-based cryptography, thereby also suggesting new research focuses for the future.

Table 5.3.: Proposed parameters for the McEliece cryptosystem – optimized for public key size

Year	Symmetric Key Size	Lower bound for $\log_2 S(n,k,t)$	McEliece parameters (n,k,t) and public key size (kB)		IMY(y) (MIPS-years)	Corresponding number of years[a]
2011	79	85	(1652, 1203, 42)	66	$2.47 \cdot 10^{12}$	$5.61 \cdot 10^{7}$
2012	80	87	(1687, 1226, 43)	69	$4.19 \cdot 10^{12}$	$9.52 \cdot 10^{7}$
2013	80	88	(1702, 1219, 45)	72	$7.14 \cdot 10^{12}$	$1.62 \cdot 10^{8}$
2014	81	89	(1770, 1306, 43)	74	$1.21 \cdot 10^{13}$	$2.75 \cdot 10^{8}$
2015	82	90	(1823, 1368, 42)	76	$2.07 \cdot 10^{13}$	$4.70 \cdot 10^{8}$
2016	83	91	(1833, 1356, 44)	79	$3.51 \cdot 10^{13}$	$7.98 \cdot 10^{8}$
2017	83	92	(1845, 1356, 45)	81	$5.98 \cdot 10^{13}$	$1.36 \cdot 10^{9}$
2018	84	93	(1877, 1387, 45)	83	$1.02 \cdot 10^{14}$	$2.32 \cdot 10^{9}$
2019	85	95	(1951, 1481, 43)	85	$1.73 \cdot 10^{14}$	$3.93 \cdot 10^{9}$
2020	86	96	(1955, 1463, 45)	88	$2.94 \cdot 10^{14}$	$6.68 \cdot 10^{9}$
2021	86	97	(1983, 1479, 46)	91	$5.01 \cdot 10^{14}$	$1.14 \cdot 10^{10}$
2022	87	98	(2013, 1508, 46)	93	$8.52 \cdot 10^{14}$	$1.94 \cdot 10^{10}$
2023	88	99	(2018, 1491, 48)	96	$1.45 \cdot 10^{15}$	$3.30 \cdot 10^{10}$
2024	89	101	(2104, 1596, 46)	99	$2.47 \cdot 10^{15}$	$5.61 \cdot 10^{10}$
2025	89	102	(2106, 1576, 48)	102	$4.20 \cdot 10^{15}$	$9.55 \cdot 10^{10}$
2026	90	103	(2135, 1604, 48)	104	$7.14 \cdot 10^{15}$	$1.62 \cdot 10^{11}$
2027	91	104	(2157, 1614, 49)	107	$1.21 \cdot 10^{16}$	$2.75 \cdot 10^{11}$
2028	92	105	(2198, 1654, 49)	110	$2.07 \cdot 10^{16}$	$4.70 \cdot 10^{11}$
2029	93	106	(2220, 1664, 50)	113	$3.52 \cdot 10^{16}$	$8.00 \cdot 10^{11}$
2030	93	108	(2241, 1673, 51)	116	$5.98 \cdot 10^{16}$	$1.36 \cdot 10^{12}$
2032	95	110	(2344, 1784, 50)	122	$1.73 \cdot 10^{17}$	$3.93 \cdot 10^{12}$
2034	96	112	(2440, 1877, 50)	129	$5.01 \cdot 10^{17}$	$1.14 \cdot 10^{13}$
2036	98	115	(2496, 1920, 51)	135	$1.45 \cdot 10^{18}$	$3.30 \cdot 10^{13}$
2038	99	117	(2440, 1776, 59)	144	$4.20 \cdot 10^{18}$	$9.55 \cdot 10^{13}$
2040	101	119	(2521, 1854, 59)	151	$1.22 \cdot 10^{19}$	$2.77 \cdot 10^{14}$
2042	103	122	(2623, 1964, 58)	158	$3.52 \cdot 10^{19}$	$8.00 \cdot 10^{14}$
2044	104	124	(2662, 1979, 60)	165	$1.02 \cdot 10^{20}$	$2.32 \cdot 10^{15}$
2046	106	126	(2691, 1973, 63)	173	$2.95 \cdot 10^{20}$	$6.70 \cdot 10^{15}$
2048	107	129	(2798, 2088, 62)	181	$8.53 \cdot 10^{20}$	$1.94 \cdot 10^{16}$
2050	109	131	(2804, 2048, 66)	189	$2.47 \cdot 10^{21}$	$5.61 \cdot 10^{16}$

[a] on a 2.4 GHz Intel Core 2 Quad Q6600

Table 5.4.: Comparison of parameters using optimistic versus pessimistic assumptions (from a users point of view) for selected years.

	Optimistic scenario			Pessimistic scenario		
Year	Lower bound for $\log_2 S(n,k,t)$	McEliece parameters (n,k,t) and public key size (kB)		Lower bound for $\log_2 S(n,k,t)$	McEliece parameters (n,k,t) and public key size (kB)	
2020	95	(1902, 1390, 47)	87	98	(2047, 1541, 46)	95
2030	105	(2220, 1664, 50)	113	112	(2396, 1801, 53)	131
2040	142	(2453, 1811, 57)	116	126	(2730, 2045, 60)	171
2050	127	(2732, 2024, 62)	175	139	(3108, 2342, 66)	219

List of Figures

List of Tables

Bibliography

[AFG+08] D Augot, M. Finiasz, P. Gaborit, S. Manuel, and N. Sendrier. SHA-3 proposal: FSB. Submission to NIST, 2008. (Cited on pages 18, 49, 50, 51, 52, and 57.)

[AFS03] D. Augot, M. Finiasz, and N. Sendrier. A fast provably secure cryptographic hash function. Cryptology ePrint Archive, Report 2003/230, 2003. `http://eprint.iacr.org/`. (Cited on pages 2 and 50.)

[AFS05] D. Augot, M. Finiasz, and N. Sendrier. A Family of Fast Syndrome Based Cryptographic Hash Functions. In E. Dawson and S. Vaudenay, editors, *Mycrypt 2005*, volume 3715, pages 64–83. Springer, 2005. (Cited on pages 2, 9, 17, 33, 34, 35, 49, 50, 52, and 59.)

[AJ07] Au-Ja.de. Intel core 2 Quad Q6600, May 2007. 2007. (Cited on page 68.)

[AM89] C.M. Adams and H. Meijer. Security-related Comments Regarding McEliece Public-key Cryptosystem. *IEEE Trans. Inform. Theory*, 35(2):454–455, 1989. (Cited on page 66.)

[Ara04] R. Araujo. The Need for Strong SSL Ciphers. 2004. (Cited on page 66.)

[Bar94] S. Barg. Some new np-complete coding problems. *Problems Inform. Transmission*, 30(3):23–28, 1994. (Cited on page 9.)

[Bar98] A. Barg. *Complexity issues in coding theory*, volume 1, pages 649–754. Elsevier Science, Amsterdam, 1998. (Cited on page 9.)

[BBD08] D. J. Bernstein, J. Buchmann, and E. Dahmen. *Post Quantum Cryptography*. Springer-Verlag, 2008. (Cited on page 1.)

[BDPA07] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Sponge Functions. In *ECRYPT Hash Workshop 2007*, 2007. (Cited on pages 2, 13, 49, 53, and 58.)

[BDPA08] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. On the indifferentiability of the sponge construction. In *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 181–197. Springer, 2008. (Cited on page 13.)

[BDPA11a] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Cryptographic sponge functions. 2011. http://sponge.noekeon.org/CSF-0.1.pdf. (Cited on page 57.)

[BDPA11b] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. On the security of the keyed sponge construction. In *SKEW*, 2011. (Cited on page 13.)

[Ber] D. J. Bernstein. Which phase-3 estream ciphers provide the best software speeds ? `http://cr.yp.to/streamciphers/phase3speed-20080225.pdf`. (Cited on pages 46, 48, and 77.)

[Ber07] D. J. Bernstein. Better price-performance ratios for generalized birthday attacks. In *Workshop Record of SHARCS07: Special-purpose Hardware for Attacking Cryptographic Systems (2007)*, 2007. (Cited on pages 34 and 60.)

[BGG94] M. Bellare, O. Goldreich, and S. Goldwasser. Incremental cryptography: The case of hashing and signing. In *Proceedings of the 14th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '94, pages 216–233. Springer, 1994. (Cited on pages 27, 28, and 76.)

[BGG95] M. Bellare, O. Goldreich, and S. Goldwasser. Incremental cryptography and application to virus protection. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, STOC '95, pages 45–56. ACM, 1995. (Cited on page 27.)

[BGP09] C. Berbain, H. Gilbert, and J. Patarin. Quad: A multivariate stream cipher with provable security. *J. Symb. Comput.*, 44(12):1703–1723, 2009. (Cited on page 20.)

[BL] D. J. Bernstein and T. Lange. eBACS: ECRYPT Benchmarking of Cryptographic Systems. `http://bench.cr.yp.to`. (Cited on page 1.)

[BLN+09] D. J. Bernstein, T. Lange, R. Niederhagen, C. Peters, and P. Schwabe. FSBDay: Implementing wagner's generalized birthday attack against the SHA-3 candidate FSB, 2009. (Cited on page 60.)

[BLP11] D. J. Bernstein, T. Lange, and C. Peters. Smaller decoding exponents: Ball-collision decoding. In *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 743–760. Springer, 2011. (Cited on page 66.)

[BLPS11a] D. J. Bernstein, T. Lange, C. Peters, and P. Schwabe. Faster 2-regular information-set decoding. *IACR Cryptology ePrint Archive*, 2011:120, 2011. (Cited on page 59.)

[BLPS11b] D. J. Bernstein, T. Lange, C. Peters, and P. Schwabe. Really fast syndrome-based hashing. In A. Nitaj and D. Pointcheval, editors, *Progress in Cryptology–AFRICACRYPT 2011*, volume 6737 of *LNCS*, pages 134–152. Springer, 2011. `http://cryptojedi.org/papers/#rfsb`. (Cited on pages 34, 63, and 64.)

[BM97] M. Bellare and D. Micciancio. A new paradigm for collision-free hashing: incrementality at reduced cost. In *Proceedings of the 16th annual international conference on Theory and application of cryptographic techniques*, EUROCRYPT'97, pages 163–192. Springer, 1997. (Cited on pages 27, 32, and 33.)

[BMvT78] E. Berlekamp, R. McEliece, and H. van Tilborg. On the Inherent Intractability of Certain Coding Problems. *IEEE Trans. Inform. Theory*, 24(3):384–386, 1978. (Cited on page 1.)

[BR00] P. S. L. M. Barreto and V. Rijmen. The WHIRLPOOL Hashing Function. 2000. Revised May 2003. Available: `http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html` (2009/06/24). (Cited on page 51.)

[BTP08] D. J. Bernstein, T.Lange, and C. Peters. Attacking and defending the mceliece cryptosystem. In Johannes Buchmann and Jintai Ding, editors, *PQCrypto*, volume 5299 of *Lecture Notes in Computer Science*, pages 31–46, 2008. (Cited on pages 66, 68, and 69.)

[BTP11] D. J. Bernstein, T.Lange, and C. Peters. Smaller decoding exponents: Ball-collision decoding. In Phillip Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 743–760, 2011. (Cited on page 59.)

[BW99] A. Biryukov and D. Wagner. Slide attacks. In *FSE*, volume 1636 of *LNCS*, pages 245–259. Springer, 1999. (Cited on page 60.)

[Cay11] P.L-. Cayrel. 2011. http://www.cayrel.net/research/code-based-cryptography/code-based-cryptosystems/article/implementation-of-code-based-hash. (Cited on pages 62 and 77.)

[CC94] A. Canteaut and H. Chabanne. A further improvement of the work factor in an attempt at breaking McEliece's cryptosystem. Research Report RR-2227, INRIA, 1994. (Cited on page 66.)

[CC98] A. Canteaut and F. Chabaud. A New Algorithm for Finding Minimum-Weight Words in a Linear Code: Application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511. *IEEE Transactions on Information Theory*, 44(1):367–378, 1998. (Cited on page 66.)

[CJ04] J.-S. Coron and A. Joux. Cryptanalysis of a provably secure cryptographic hash function. Cryptology ePrint Archive, Report 2004/013, 2004. `http://eprint.iacr.org/`. (Cited on page 52.)

[CR06] C. De Cannière and C. Rechberger. Finding sha-1 characteristics: General results and applications. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT*, volume 4284 of *LNCS*, pages 1–20. Springer, 2006. (Cited on page 49.)

[CSM] P-L Cayrel, Q. Santos, and M. Meziani. Efficient Software Implementations of Code-based Hash Functions and Stream-Ciphers. (Cited on pages 20, 36, 37, 62, 64, and 77.)

[Dam89] I. Damgård. A Design Principle for Hash Functions. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO '89, Proc.*, volume 435 of *LNCS*, pages 416–427. Springer, 1989. (Cited on pages 12, 50, and 53.)

[EJ01] D. Eastlake and P. Jones. Us secure hash algorithm 1 (sha1), 2001. (Cited on page 49.)

[EOS06] D. Engelbert, R. Overbeck, and A. Schmidt. A summary of mceliece-type cryptosystems and their security. Cryptology ePrint Archive, Report 2006/162, 2006. `http://eprint.iacr.org/`. (Cited on page 65.)

[FGO+10] J.-C. Faugère, V. Gauthier, A. Otmani, L. Perret, and J.-P. Tillich. A distinguisher for high rate mceliece cryptosystems. *IACR Cryptology ePrint Archive*, 2010:331, 2010. (Cited on page 66.)

[FGS07] M. Finiasz, P. Gaborit, and N. Sendrier. Improved fast syndrome based cryptographic hash functions. In V. Rijmen, editor, *ECRYPT Hash Workshop 2007*, 2007. (Cited on pages 2, 49, and 50.)

[FL08] P.-A. Fouque and G. Leurent. Cryptanalysis of a hash function based on quasi-cyclic codes. In Tal Malkin, editor, *CT-RSA*, volume 4964 of *LNCS*, pages 19–35. Springer, 2008. (Cited on page 52.)

[FOPT10] J.-C. Faugère, A. Otmani, L. Perret, and J.-P. Tillich. Algebraic cryptanalysis of mceliece variants with compact keys. In *Proceedings of the 29th Annual international conference*

on Theory and Applications of Cryptographic Techniques, EUROCRYPT'10, pages 279–298. Springer, 2010. (Cited on page 66.)

[FS96] J.-B. Fischer and J. Stern. An efficient pseudo-random generator provably as secure as syndrome decoding. In *EUROCRYPT'96: Proc. of the 15th annual international conference on Theory and application of cryptographic techniques*, pages 245–255. Springer, 1996. (Cited on pages 2, 15, 16, 17, 18, 20, 30, 31, and 77.)

[FS09] M. Finiasz and N. Sendrier. Security Bounds for the Design of Code-based Cryptosystems. In Mitsuru Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009*, number 5912 in LNCS, pages 88–105. Springer, 2009. (Cited on pages 25, 34, 59, 60, 66, 69, 71, and 72.)

[Gil] E. N. Gilbert. A comparison of signalling alphabets. *Bell System Technical Journal*. (Cited on page 6.)

[GIL+90] O. Goldreich, R. Impagliazzo, L. Levin, R. Venkatesan, and D. Zuckerman. Security preserving amplification of hardness. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, SFCS '90, pages 318–326 vol.1. IEEE Computer Society, 1990. (Cited on page 41.)

[GL89] O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *STOC '89: Proc. of the twenty-first annual ACM symposium on Theory of computing*, pages 25–32. ACM, 1989. (Cited on pages 17, 31, and 42.)

[GLP08] M. Gorski, S. Lucks, and T. Peyrin. Slide attacks on a class of hash functions. In *Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '08, pages 143–160. Springer-Verlag, 2008. (Cited on page 60.)

[GLS07] P. Gaborit, C. Laudaroux, and N. Sendrier. Synd: a very fast code-based cipher stream with a security reduction. In *IEEE Conference, ISIT'07*, pages 186–190, Nice, France, 2007. (Cited on pages 2, 15, 17, 20, 21, 25, 26, 36, 37, 46, and 77.)

[GM82] S. Goldwasser and S. Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *STOC*, pages 365–377, 1982. (Cited on page 4.)

[Gol97] J.Dj. Golic. Cryptanalysis of alleged a5 stream cipher. In *Proceedings of the 16th annual international conference on Theory and application of cryptographic techniques*, EUROCRYPT'97, pages 239–255. Springer, 1997. (Cited on pages 25, 35, 36, and 45.)

[Gop70] V .D. Goppa. A New Class of Linear Correcting Codes. In *Probl. Pered. Info.*, volume 6, pages 24–30, 1970. (Cited on page 7.)

[GZ07] P. Gaborit and G. Zémor. Asymptotic improvement of the gilbert-varshamov bound for linear codes. volume abs/0708.4164, 2007. (Cited on pages 36 and 46.)

[Hel80] M. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26:401–406, 1980. (Cited on pages 25, 35, and 45.)

[Hou] D. House. Myths of moore's law. `http://news.cnet.com/Myths-of-Moores-Law/2010-10713-1014887.html`. (Cited on page 69.)

[HS05] J. Hong and P. Sarkar. Rediscovery of time memory tradeoffs. Cryptology ePrint Archive, Report 2005/090, 2005. `http://eprint.iacr.org/`. (Cited on pages 25, 35, 36, and 45.)

[IN96] R. Impagliazzo and M. Naor. Efficient cryptographic schemes provably as secure as subset sum. *J. Cryptology*, 9(4):199–216, 1996. (Cited on pages 23 and 24.)

[INR07] INRIA. 2007. http://www-rocq.inria.fr/secret/CBCrypto/index.php?pg=fsb. (Cited on pages 62 and 77.)

[IZ89] R. Impagliazzo and D. Zuckerman. How to recycle random bits. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, SFCS '89, pages 248–253. IEEE Computer Society, 1989. (Cited on page 24.)

[JJ02] T. Johansson and F. Jönsson. On the complexity of some cryptographic problems based on the general decoding problem. *IEEE Transactions on Information Theory*, 48(10):2669–2678, 2002. (Cited on page 66.)

[Jou04] A. Joux. Multicollisions in iterated hash functions. application to cascaded constructions. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 306–316, 2004. (Cited on page 12.)

[Kar72] R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*. Plenum Press, 1972. (Cited on page 23.)

[KK06] J. Kelsey and T. Kohno. Herding hash functions and the nostradamus attack. In *Proceedings of the 24th annual international conference on The Theory and Applications of Cryptographic Techniques*, EUROCRYPT'06, pages 183–200. Springer, 2006. (Cited on page 52.)

[KS05] J. Kelsey and B. Schneier. Second preimages on n-bit hash functions for much less than 2^n work. In *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 474–490. Springer, 2005. (Cited on page 12.)

[LB88] P.J. Lee and E.F. Brickell. An Observation on the Security of McEliece's Public-key Cryptosystem. In *EUROCRYPT '88, Lect. Notes in CS*, pages 275–280, 1988. (Cited on page 66.)

[LDmW94] Y. X. Li, R. H. Deng, and X. m. Wang. On the equivalence of mceliece's and niederreiter's public-key cryptosystems. *IEEE Transactions on Information Theory*, pages 271–271, 1994. (Cited on page 9.)

[Lin98] J. H. Van Lint. *Introduction to Coding Theory*. Springer-Verlag, 3rd edition, 1998. (Cited on page 4.)

[LV01] A. K. Lenstra and E. R. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology*, 14:255–293, 2001. (Cited on pages 66, 67, and 68.)

[MAC11] M. Meziani, S. M. El Yousfi Alaoui, and P-L. Cayrel. Hash functions based on coding theory. In *the 2nd Workshop on Codes, Cryptography and Communication Systems (WCCCS 2011)*, pages 32–37, 2011. (Cited on page 63.)

[Mat94] M. Matsui. Linear cryptanalysis method for des cipher. In *Workshop on the theory and application of cryptographic techniques on Advances in cryptology*, EUROCRYPT '93, pages 386–397. Springer, 1994. (Cited on page 44.)

[McE78] R.J. McEliece. A public-key cryptosystem based on algebraic coding theory. *DNS Progress Report*, pages 114–116, 1978. (Cited on pages 1, 5, and 9.)

[MCY11] M. Meziani, P.-L. Cayrel, and S. M. Alaoui El Yousfi. 2SC: An Efficient Code-Based Stream Cipher. In T.-H. Kim, H. Adeli, R. J. Robles, and M. O. Balitanas, editors, *ISA*, volume 200 of *Communications in Computer and Information Science*, pages 111–122. Springer, 2011. (Cited on pages 15, 20, 26, and 36.)

[Mer89] R. C. Merkle. One Way Hash Functions and DES. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO '89, Proc.*, volume 435 of *LNCS*, pages 428–446. Springer, 1989. (Cited on pages 12, 50, and 53.)

[MHC] M. Meziani, G. Hoffmann, and P.-L. Cayrel. PSYND: A Parallel Variant of the XSYND Stream Cipher. In *MoCrySEn 2013 (submitted)*. (Cited on page 15.)

[MHC12] M. Meziani, G. Hoffmann, and P.-L. Cayrel. Improving the Performance of the SYND Stream Cipher. In Aikaterini Mitrokotsa and Serge Vaudenay, editors, *AFRICACRYPT*, volume 7374 of *Lecture Notes in Computer Science*, pages 99–116. Springer, 2012. (Cited on pages 15, 26, and 46.)

[MMT11] A. May, A. Meurer, and E. Thomae. Decoding random linear codes in $\tilde{O}(2^{0.054n})$. In *Proceedings of the 17th international conference on The Theory and Application of Cryptology and Information Security*, ASIACRYPT'11, pages 107–124. Springer-Verlag, 2011. (Cited on pages 20 and 35.)

[Moo98] G.E. Moore. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1):82–85, 1998. (Cited on page 69.)

[MQSW01] A. Menezes, M. Qu, D. Stinson, and Y. Wang. Evaluation of Security Level of Cryptography: ESIGN Signature Scheme. CRYPTREC Project, Japan, Jan. 2001. (Cited on page 66.)

[MS77] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error Correcting Codes*. North-Holland, 1977. (Cited on pages 4 and 7.)

[MS09] L. Minder and A. Sinclair. The extended k-tree algorithm. In *Proc. of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'09, pages 586–595, 2009. (Cited on page 34.)

[Nat08] National Institute of Standards and Technology (NIST). *Secure Hash Standard*, October 2008. (Cited on page 49.)

[NCB11] R. Niebuhr, P.-L. Cayrel, and J. Buchmann. Improving the efficiency of Generalized Birthday Attacks against certain structured cryptosystems. In *WCC 2011*, LNCS, pages 163–172. Springer, Apr 2011. (Cited on page 34.)

[NCBBa] R. Niebuhr, P.-L. Cayrel, S. Bulygin, and J. Buchmann. On lower bounds for information set decoding over $\mathbb{F}_q$. In *SCC 2010, Royal Holloway, University of London, London, UK 2010*. (Cited on pages 66, 68, and 69.)

[NCBBb] R. Niebuhr, P.-L. Cayrel, S. Bulygin, and J. Buchmann. On lower bounds for information set decoding over $\mathbb{F}_q$ and on the effect of partial knowledge. In *"Symbolic Computation and Cryptography II" (unpublished) (2011)*. (Cited on pages 66 and 68.)

[Nie86] H. Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Problems of Control and Information Theory. Problemy Upravlenija i Teorii Informacii*, 15:159–166, 1986. (Cited on pages 1 and 9.)

[NMBB12] R. Niebuhr, M. Meziani, S. Bulygin, and J. Buchmann. Selecting parameters for secure mceliece-based cryptosystems. *International Journal of Information Security*, 11(3):137–147, 2012. (Cited on page 65.)

[Pra57] E. Prange. Cyclic error-correcting codes in two symbols. September 1957. No. AFCRC-TN-57-103. ASTIA Document No. AD133749. (Cited on page 8.)

[Pre93] B. Preneel. *Analysis and design of cryptographic hash functions*. Doctoral dissertation, Katholieke Universiteit Leuven, 1993. (Cited on page 11.)

[RB08] M. Robshaw and O. Billet, editors. *New Stream Cipher Designs: The eSTREAM Finalists*. Springer-Verlag, 2008. (Cited on page 46.)

[RWMC11] L. Rothamel, M. Weiel, M. Meziani, and P.-L. Cayrel. Report cryptography lab ss2011 implementation of the rfsb hash function. 2011. www.cayrel.net/IMG/pdf/Report.pdf. (Cited on page 64.)

[Saa07] M.-J. O. Saarinen. Linearization attacks against syndrome based hashes. In K. Srinathan, C. Pandu Rangan, and Moti Yung, editors, *INDOCRYPT*, volume 4859 of *LNCS*, pages 1–9. Springer, 2007. (Cited on pages 32, 33, 52, and 61.)

[San12] Q. Santos. 2012. http://perso.ens-lyon.fr/quentin.santos/. (Cited on page 62.)

[Sen02] N. Sendrier. On the Security of the McEliece Public-key Cryptosystem. In M. Blaum, P.G. Farrell, and H. van Tilborg, editors, *Information, Coding and Mathematics*, pages 141–163. Kluwer, 2002. Proceedings of Workshop honoring Prof. Bob McEliece on his 60th birthday. (Cited on pages 70 and 71.)

[Sha48] C. E. Shannon. A mathematical theory of communication. *Bell system technical journal*, 27, 1948. (Cited on page 4.)

[Sho94] P. W. Shor. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In *SFCS '94: Proc. of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134. IEEE Computer Society, 1994. (Cited on page 1.)

[Ste89] J. Stern. A method for finding codewords of small weight. In *Proc. of Coding Theory and Applications*, pages 106–113, 1989. (Cited on page 66.)

[Sze08] G. Szewczyk. The dynamic ciphers - New concept of long-term protecting. *Annales Universitatis Apulensis Series Oeconomica*, 2(10), 2008. (Cited on page 66.)

[Var57] R. R. Varshamov. Estimate of the number of signals in error correcting codes. *Dokl. Acad. Nauk SSSR*, 117:739741, 1957. (Cited on page 6.)

[vdV90] M. van der Vlugt. The true dimension of certain binary goppa codes. *IEEE Transactions on Information Theory*, 36(2):397–398, 1990. (Cited on page 7.)

[Wag02] D. Wagner. A generalized birthday problem. In Moti Yung, editor, *CRYPTO*, volume 2442 of *LNCS*. Springer, 2002. (Cited on pages 34, 52, and 60.)

[WYY05] X. Wang, Y. L. Yin, and H. Yu. Finding collisions in the full sha-1. In *In Proceedings of Crypto*, pages 17–36. Springer, 2005. (Cited on page 49.)

[Yao82a] A. C. Yao. Theory and application of trapdoor functions. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 80–91. IEEE Computer Society, 1982. (Cited on page 4.)

[Yao82b] A. C. Yao. Theory and applications of trapdoor functions (extended abstract). In *FOCS*, pages 80–91. IEEE Computer Society, 1982. (Cited on page 41.)