Application of cryptographic and verification techniques to the security and privacy of information systems

Marc Beunardeau

THÈSE DE DOCTORAT

de l'Université de recherche Paris Sciences et Lettres

Préparée à l'Ecole Normale Supérieure

Cryptographie Appliquée à la Sécurité des Systèmes d'Information

Ecole doctorale n°386

l'École Doctorale Sciences Mathématiques de Paris Centre

Spécialité Informatique

Soutenue par
Marc BEUNARDEAU
le 15 janvier 2019

Dirigée par David NACCACHE

COMPOSITION DU JURY :

M. Joye Marc
OneSpan, Rapporteur

M. Coron Jean-Sebastien
Université du Luxembourg, Rapporteur

M. Pointcheval David
Ecole Normale Supérierue, Président du jury

M. Pierre-Alain Fouque
Université de Rennes I, examinateur

Mme. Lysyanskaya Anna
Brown University, examinatrice

M. Preneel Bart
KU Leuven, examinateur

ENS

PSL RESEARCH UNIVERSITY PARIS

Remerciements

Je tiens à remercier David Naccache, Professeur à l'Ecole Normale Supérieure, qui m'a encadré tout au long de cette thèse et qui m'a fait partager ses brillantes intuitions qui m'ont aidées à saisir de nombreux problèmes. Qu'il soit aussi remercié pour les nombreuses conversations sur des sujets aussi variés qu'interessants.

J'adresse tous mes remerciements à Marc Joye et Jean-Sébastien Coron de l'honneur qu'ils m'ont fait en acceptant d'être rapporteurs de cette thèse.

J'exprime ma gratitude à Pierre-Alain Fouque, Anna Lysanskaya, David Pointcheval et Bart Preneel qui ont bien voulu participer à mon jury.

Je tiens aussi à remercier Michel Léger, Directeur du laboratoire d'innovation d'Ingenico, qui m'a accueilli pendant trois ans au sein de son laboratoire. C'est grâce à lui que j'ai pu concilier avec bonheur recherche théorique et appliquée pendant cette thèse. Merci également à toute l'équipe du Lab et particulierement à Aisling, Hiba et Rémi avec qui j'ai particulièrement aprrecié travailler et discuter.

Je tiens à remercier tous les membres de l'équipe de sécurité de l'Ecole Normale Supérieure, et tous mes co-autheurs avec qui les nombreux échanges m'ont permis de progresser (et de publier :)).

Enfin, un grand merci a ma famille et mes amis qui m'ont soutenu pendant ces trois ans et ont patiemment écouté mes tentatives de vulgarisation.

Contents

Part I

Introduction

Chapter 1

Preliminaries

Contents

The first thing people think of when they hear of cryptography is that it is used for secret communications. While it is not the purpose of this thesis to re-explain in detail cryptography and its applications, we begin by over-viewing several examples of the usage of cryptography. The purpose of this part is to give the non-specialist reader an intuition about what cryptography is, and how large its applications are. We will also seize the occasion to introduce mathematical definitions of the various concepts. Some of these definitions are usually not given, or not discussed in mainstream cryptographic papers, and are therefore not in our chapters. The reasons there are not is usually the limited number of pages in a paper, and the fact that the community implicitly assumes that everybody knows and understand them, or that they do not to use such formalism. However, we enjoy the non-limitation of space in this thesis to go a bit deeper. By doing so it allows the cryptographer to recall precise models such as Turing machines (e.g. Section 1.4.1), and the non-cryptographer to understand more easily subtle notions such as security games and security notions (e.g. Figure 1.3) that are usually left undiscussed in nowadays cryptographic papers. Arguably modern cryptography started with the Kerckhoff's principle [MP08] which states that the security of a cryptosystem should only rely on the key's secrecy, not on how this key is used. Before that, security by obscurity (i.e. the way we encrypt is secret) was preferred. It was vulnerable to reverse engineering and leaks from the designers and users of the system. On the other hand, respecting this principle allows two things. First if an encryption device is stolen by the adversary then we simply have to change the key instead of changing the cryptosystem. Second, respecting this principle allows the analysis of cryptosystems by everyone, and therefore better designs. Nowadays the vast majority of cryptosystems respect this principle, and it allowed cryptography to become a science. It still happens that well analysed and efficient solutions are not yet found, which sometimes makes the industry use security by obscurity. This is the case for example in obfuscation and white-box cryptography (because no cryptographic solution exist) and format preserving encryption where proven solutions are inefficient Section 1.5.1.1.

One could also argue that the real start of modern cryptography was the invention of the computer and its ancestors (electro-mechanical machines), that allowed designing and attacking systems with costly computations. The most famous example was probably the enigma machine during the Second World War by the Germans. It could be used easily by German soldiers thanks to an electro-mechanical device and was cryptanalysed by Polish and British cryptographers (notably Alan Turing) thanks to another electro-mechanical device called the cryptologic bomb.

1.1 Foreword

This thesis can be seen as a collection of papers that the author co-wrote during his work at ENS and Ingenico. While we did choose not to include all our results here, some of our work being too far from others, it can still be seen as non-standard in the sense that this thesis is not focused on one precise topic. The purpose of this introduction throughout examples is twofold. One is to place the many different works in the different parts of cryptography. The other is to explain how the different papers of this thesis are consistent. The last part (Section 1.5) introduces some of the areas of cryptography, and aims to show how the topics treated in this thesis do not come from the theoretical cryptography. Of course, we do use its results and methodologies, but the topics can be considered as 'practice-oriented', in the sense that our problems are inspired by the real world problems. On the other hand, a part of cryptography aims at more fundamental problems, which could serve a more philosophical or aesthetic purpose, and which should find applications that have not yet been foreseen. An interesting reading that can help the reader to better understand our point of view is Rogaway's positional paper [Rog16]. A different view that argues that cryptography takes its roots in a mathematical truth rather than the real world can be found in [Gol06]. Since the author belongs to a security department and working in industry,

it is natural that our thesis adopts the first point of view (without denying the interest of the second). Unfortunately, this will not be done by exposing projects done at Ingenico for two reasons: industry projects are outside the scope of a scientific thesis, and disclosing information that should not be is a risk we are not willing to take.

1.2 Symmetric Cryptography

Symmetric (or secret key) cryptography is the oldest and simplest form of cryptography. It exists since Antiquity, and until the 70's was the only known form of encryption. It assumes that both parties have agreed beforehand on a shared secret (the secret key). Then they mix the secret with their message, in a reversible way thus allowing decryption. Formally a secret key cryptosystem is a family of permutations on the message space, indexed by the set of secret keys.

1.2.1 Shannon's Principle

The first formalisation of secret key cryptosystem was made by Claude Shanon in 1949 [Sha49]. In this article, Shannon introduces two concepts. The first one is *information theory*. The second one is *block ciphers*. Historically there have been two types of ciphers. Some use substitution i.e. they replace a message chunk by another one. The correspondence between different chunks constitutes the secret key. For example, if the set of messages is the English language, we can pick a permutation on the alphabet, and replace every message letter by the image of the letters under the chosen permutation. Choosing an appropriate permutation (a randomly chosen one would have a good chance to do the trick) allows complex relation between the key and the message. On the other hand, choosing one that keeps some structure (e.g. changing vowels to vowels and consonants to consonants) will make deciphering easier. For bit strings (which is the most widespread real-world application) the structure can be logical, or arithmetical. However, even with a good permutation, it is not sufficient since the English language (or structured data such as XML documents) has lots of structure (e.g. letters do not have the same frequency). A simple cryptanalysis would be, for example, to guess that the image of 'e' by the secret permutation is the letter appearing the most in the ciphertext. If the guess is correct then the keys space is reduced from 26! to 25! with very little information. The leakage rate is far from optimal.

To avoid this, we need a second kind of ciphers: permutations ciphers. Letting the message space be strings of fixed length n on a fixed alphabet, we choose the secret key to be a permutation σ of $\{1, \cdots, n\}$. The encryption of a message $a_1 \cdots a_n$ is simply $a_{\sigma(1)} \cdots a_{\sigma(n)}$. This alone is insecure, but combined with substitution ciphers as it is done in modern ciphers, this allows to thwart frequential analysis. Ideally, if one message bit is changed, about half of the bits of the ciphertext are flipped.

1.2.2 Block Ciphers and Stream Ciphers

Moderns ciphers can be divided into two categories: stream ciphers and block ciphers. A stream cipher takes as input a string of any length, whereas a block cipher takes a fixed length input (usually around 128 bits) called the block size. Block ciphers are used as a primitive for modes of operations, which call several time a block cipher, and add padding to encrypt messages longer than the block size, and of length that is not a multiple of the block size. Achieving a goal using primitives, as it is done for a symmetric scheme with a block cipher, is a standard technique[1] in cryptography and in computer science in general. It allows evaluating the efficiency and security

[1] Often referred as composition

of the different components modularly. Apart from specific cases (e.g. [CCF+16]), block ciphers are generally preferred over stream ciphers.

1.2.3 Examples : DES and AES

Two of the most used block ciphers are the Data Encryption Standard (DES), and the Advanced Encryption Standard (AES), known initially as Rijndael [DR99].

DES DES was introduced by the National Institute of Standards and Technology (NIST) in 1976 as the new standard block cipher. Due to various attacks, and the increase of computational power the original version which using 56 bit keys is now considered insecure. A variant called triple DES that applies two times the DES permutation, with in between the inverse DES permutation, all three with different keys (using therefore a 168 bit key) is still in use in old infrastructure (e.g. the EMV protocol for payment).

AES Due to the weakening of the DES, the NIST launched a competition to create a new symmetric encryption standard in 1997. Four years later Rijndael is chosen and renamed AES. Its structure is based on a Substitution Permutation Network. We give a visual example of such an SPN in Figure 1.1

The precise AES specifications can be found in [MVM09].

1.2.4 Beyond Encryption

Cryptography is not only about encryption. In this section, we will briefly describe a few other widely used primitives.

Message Authentication Code Message Authentication Codes (MACs) ensure integrity. Since the Internet is open, attackers can do more than simply eavesdrop communications. For example, they can modify messages, which can be catastrophic for security applications. Integrity is the desired property that every modified message is detected as such and discarded by its receiver. When a common secret key sk is shared, this can be achieved with MACs. We attach to the message m a tag t, which is a function of m and sk, such that from the knowledge of multiple m, t one cannot derive m', t' such that t' is a correct tag for m'. Thus modifying a message will result in a rejection by the legitimate receiver.

Authenticated Encryption The good practice is to use encryption with MAC, to make authenticated encryption. For long confidentiality and integrity were not combined, and authenticated encryption was done 'by hand' combining ciphers and MAC. Due to a lot of misuses (e.g. counterintuitive attacks leveraging reuse of the same secret key for both can be done, or simply not thinking that you need integrity while you do) cryptographers now advocate the use of Authenticated Encryption (AE). Following this paradigm, the CESAER competition was organised, similarly to the AES one, to create a standard authenticated encryption scheme. Details on this competition can be found in https://competitions.cr.yp.to/caesar.html. Nowadays the most utilized scheme is the AES-Galois Counter Mode (AES-GCM), which uses AES as a primitive to build an AE scheme.

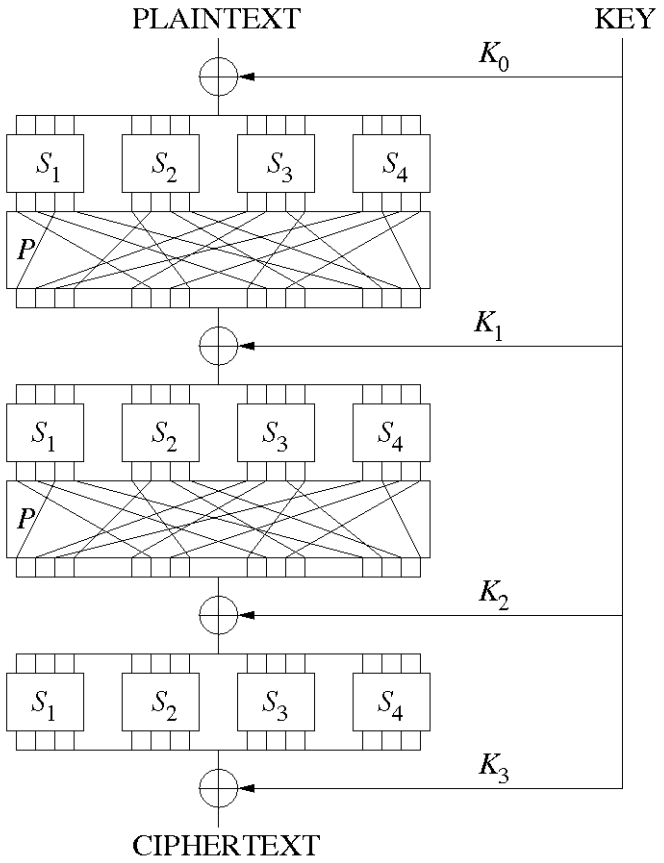

Figure 1.1: Illustration of an SPN. First, the key is added to the message with a simple eXclusive OR (XOR). Then the message is divided into small chunks, each being passed through an S-box (a permutation on a $\{0,1\}^x$ with small x). This step implements confusion: complex non-linear operations can be done efficiently since they are done on small chunks. Then a linear operation implements diffusion: every cipher part now depends on various message and key parts. Our figure describes four rounds with different sub keys all derived from the key. Note that the last permutation is removed, since it can be reversed.

Hash Function A (cryptographic) hash function $h : \{0,1\}^* \to \{0,1\}^n$ is a deterministic function easy to compute with n usually equal to 256 or a bit less. Their purpose is to remove any pattern linking inputs and outputs. Usually, this is achieved by mixing arithmetic and logical operations, which are both fast to do, and which underlying structures 'destroy' each other. Some of the wanted properties are:

- Pre-image resistance: given $h(m)$ it is hard to compute m
- Second pre-image resistance: given m_1 it is hard to find $m_2 \neq m_1$ such that $h(m_1) = h(m_2)$
- Collision resistance: it is hard to find m_1, m_2 such that $h(m_1) = h(m_2)$
- Random Oracle Model (ROM): This is a not a property that hash function can have in real-life. For many proofs, the preceding properties are insufficient, so cryptographers use idealised hash functions. We use and define this model in Section 3.3.

Trivially Collision resistance implies Second pre-image resistance. We reason by contrapositive, assuming second pre-image resistance does not hold, we are given m_1, and find m_2 such that $h(m_1) = h(m_2)$. We can then use the same messages m_1, m_2 to show that collision resistance does not hold. Hash functions can be used to build more complex cryptographic functions. A famous example is the construction a hash function based MAC as introduced in [BCK96] and called HMAC. The basic idea is to mix in a particular way the secret key and the message using hash functions. One can notice that the constant length of the output of hash functions are desirable to verify the integrity of long messages rapidly. If h is a hash function, K a secret key, $K' = h(K)$, m a message, ipad and opad two constants of size n, $a||b$ is the concatenation of the strings a and b, $a \oplus b$ is the bit wise XOR of the strings a and b, then $\text{HMAC}(K, m) = H((K' \oplus \text{opad})||(H(K' \oplus \text{ipad})||m)$. The security of this construction depends on the security of h and the length of k.

The current recommended hash functions are the Secure Hash Algorithm 2 (SHA-2) family [oCoST12] and the SHA-3 family [BDPA09]. SHA-1 [EJ01] was introduced by the NSA, and is has recently been fully broken [SBK+17]. SHA-3 (initially Keccak) was the winner of the SHA competition organised by the NIST.

1.3 Asymmetric Cryptography

Asymmetric cryptography or public key cryptography was invented by James H. Ellis in 1970 at the UK Government Communications Headquarters. However, it remained secret until Whitfield Diffie and Martin Hellman rediscovered and published it in [DH06]. The purpose of public key cryptography is to solve the problem of key distribution. Using secret key cryptography is possible only if the communicating parties already met and agreed on a common secret. Although it is not handy, such a key distribution scheme was still doable for military or diplomatic applications. However, for Internet communication it is impossible since communicating parties do not know each others in real life. To solve that, a public key algorithm has two keys: one secret, one public. Knowing the public key only allows encrypting, but knowing the secret one allows decrypting. Users publish their public keys so that everybody can send encrypted message to their, and no one but the secret-key owner would be able to decrypt. To enable this, we rely on algebraic relations between the public key and secret key, and trapdoor one-way functions (i.e. functions that do can only be efficiently computed in one way unless one has a specific knowledge called the trapdoor).

1.3.1 Diffie-Hellman

Diffie and Hellman published the first public key algorithm which realised a key exchange: two participants, Alice and Bob, send messages over a *public* channel (e.g. the Internet), and at the end of the exchange they both have knowledge of a common secret that one cannot derive by eavesdropping.

Discrete Logarithm Diffie-Hellman's function is trapdoor the exponentiation in a finite field. Fixing a prime p and a generator g of the finite field of order p, computing $g^x \bmod p$ is easy (i.e. polynomial time, see Section 1.4.1) by fast exponentiation. The converse, which is finding x given $g^x \bmod p$ is hard. Now to exchange a key, two parties agree on public parameter p and g. Alice generates a random x and sends to Bob $g^x \bmod p$. Note that nor Bob neither an eavesdropper can infer x. Bob does the same with another random y. After these two messages Alice knows x and $Z = g^y \bmod p$. She can then compute the exponentiation of z by x, and get $z^x = (g^y)^x \bmod p = g^{yx} \bmod p$. Bob knows $w = g^x \bmod p$ and y, and can in the same way compute $w^y = g^{xy} = g^{xy} = z^x \bmod p$. They agreed on a common value. The eavesdropper only knows $g^x \bmod p$ and $g^y \bmod p$. From this, there is no known efficient way allowing him to get the common secret, which can therefor be used to derive a symmetric key.

1.3.2 RSA

The most used public key scheme is RSA, an acronym standing for Rivest, Shamir, Adleman [RSA78a]. RSA is based on a very natural one-way function. If one publishes the product $n = pq$ of two large primes p and q, it will be very difficult to find p and q. It was published in 1978 shortly after Diffie and Hellman's seminal paper and allows in addition to key exchange to construct public key encryption and digital signatures (Section 1.3.3).

The Mathematics of RSA

Uses of RSA's one-way function The knowledge of p and q is used in a more indirect way than in Diffie Hellman. In the finite ring of integer modulo n, it is known by Bézout's identity that an integer x is invertible if and only if it is co-prime with n. A quick computation then shows that there are $(p-1)(q-1)$ such integers with $n = pq$. Working in the set of invertible equipped with the multiplication we are in a group of order $(p-1)(q-1)$. One can note that assuming it is hard to discover $p+q$ from n, the group's order cannot be computed from n. Now by Lagrange's theorem, for all element x of the invertible group we have $x^{k(p-1)(q-1)} = 1 \bmod n$ for any integer k.

RSA's specifications We can now explain how RSA works. Pick an RSA modulus $n = pq$ and an integer e co-prime with $(p-1)(q-1)$ and publish $\{e, n\}$. Keep $d = e{-}1 \bmod (p-1)(q-1)$ secret. To encrypt a message m compute $c = m^e \bmod n$. Recall the $ed = 1 + k(p-1)(q-1)$. Therefore with the secret information d, computing $c^d = (m^e)^d \bmod n = mm^{k(p-1)(q-1)} \bmod n = m \bmod n$, the receiver can recover n. Note that Recovering m from $m^e \bmod n$ might be done ways different than getting d, or factoring n, but no one could find such way to date, and therefore RSA is still in use. Cryptography had to wait one more year to have a cryptosystem by Rabin [Rab79] that cannot be broken unless the adversary can factor n.

1.3.3 Beyond Public Key Encryption

As for the symmetric case, a lot of other primitives are used and explored than mere key exchange or public key encryption. We recall two just for the sake of example.

Digital Signatures A digital signature is the MAC's (Section 1.2.4) public key equivalent. It allows one to prove that he and no one else sent this message. To see the usefulness of it consider the Diffie Hellman key exchange. Assume that an attacker can modify messages sent over the public channel (which is possible in an open network such as the Internet). The attacker can impersonate Alice when talking with Bob (i.e. generate a random x', and replace $g^x \bmod p$ by $g^{x'} \bmod p$) and vice versa. In the end, Alice and Bob will have a shared secret with the attacker, but not with each other, and the attacker will eavesdrop their communications. To prevent that Alice and Bob can sign their messages. A digital signature scheme has a public key and a private key. The private key allows signing a message, and the public key allows verifying that a signature corresponds to a message. Note that unlike physical signatures, digital signatures differ with each message. Otherwise, one could copy a signature on an old message and forge a signed message. An easy way to create signatures is from the inverse of RSA. The signer 'decrypts' a message m to get a signature (i.e. computes $s = m^d \bmod n$). To verify this, we take the signature $s = m^d \bmod n$ and encrypt it. We get $s^e = (m^d)^e = m \bmod n$, and check that the decrypted signatures equals the original message.

Key Encapsulation Mechanisms (KEMs) Beside signatures, an important use of public key cryptography is to exchange (or wrap) symmetric keys. A naive solution would be to encrypt a key using a public key encryption scheme. However most public key schemes taken in their textbook version are insecure for some specific messages, and symmetric keys can fall in this category. For example, symmetric keys are particularly shorts. Typically we find today 3000 bits RSA modulus and 128 bits for AES keys. If k is a 128 bits number, encrypting it with RSA and $e = 3$ (as it is done in many real life applications), we get $k^3 \bmod n = k^3$. In this case, it is easy to get k from k^3 with a simple cubic root computation. One way to prevent this kind of mistake (which can easily be done by real life developers that are not cryptographers) is to use a modification of RSA that is designed to send keys, a Key Encapsulation Mechanism. RSA is secure if a message is chosen at random (i.e. it has an negligible chance not to be secure). One can check that a 128 bits message has a negligible chance to be picked at random, since it would require 3000 - 128 bits chosen at random to be 0's. Therefore we will send a random message m using RSA, this is the encapsulation. Then the receiver decrypts it, uses this m to derive a short symmetric key (e.g. using a hash function Section 1.2.4), this is the decapsulation. The sender then applies himself the same function on the m he generated, and they a have a shared secret.

1.4 Formalisation of Security

In this section we will formalise several notions that will allow us to define security notions in cryptography from the very beginning.

1.4.1 Complexity of an algorithm

Turing Machine A Turing machine is an abstract representation of an algorithm. It is a very straightforward model, but is still relevant regarding complexity and calculability. It was introduced in 1936 by Alan Turing [Tur36].

Definition 1.1 *A Turing machine is a tuple* $\{Q, \Gamma, B \in \Gamma, \Sigma \subset \Gamma \setminus \{B\}, q_0 \in Q, F \subset Q, \delta : Q \setminus F \times \Gamma \mapsto Q \times \Gamma \times \{\leftarrow, \rightarrow\},\}$. *$Q$ is the finite set of states, Γ is the finite alphabet (it can be $\{0, 1, B\}$ without loss of generality), Σ is the set of input letters, q_0 is the initial state, δ is a partial function called the transition function, and F is the set of final states.*

A Turing machine consists of:

- An infinite reading tape divided into cells one next to another, starting with a special symbol. Each cell contains a letter from Γ. At the start the Turing machine contains its input (in Σ^*) right after the special symbol, followed by an infinite number of B.
- A head, which is at some position on the tape. In the beginning, the head is at the first letter of the input. The tape can read the letter at its position write and move left or right.

Given a state q, a tape, and a position for the head reading α, the Turing machine will apply if it can the transition function, i.e. if $\delta(q, \alpha) = (q', \alpha', a \in \{\leftarrow, \rightarrow\})$ is defined then the states changes to q', the α is replaced by α', and the head goes on cell left or right depending on a. If δ is undefined the Turing machine stops and does not return anything, if q' is in F then the Turing machine stops and returns the content of its tape (we can add a special return tape to ease things). Another formalism to return things focuses on acceptance or reject. We can partition the set of final state in the accepting states, and rejecting states. Then the Turing machine can answer a decision problem, a problem which answer is 'yes' or 'no'. If on one input the machine ends in an accepting states, it answered 'yes', otherwise it answered no. We then say that the set of accepted inputs forms the language recognised by the Turing machine. Actually, lots of problems can be decision problems. For example, computing a function f from X to Y is the same as recognising the language $\{(x, f(x)); x \in X\}$, or deciding if (x, y) is in X for all (x, y)[2].

There are many equivalent definition of the Turing machine, and also other formalism (e.g. circuit, λ-calculus, etc...) that do express the same calculability power, but we simply aim at giving a glimpse at the way things are formalised to the bottom. In practice in cryptography, we do not use as it the Turing machine formalism, but we would be able to write every algorithm in this model, and their complexity would not be fundamentally changed[3].

Probabilistic Turing Machines Many algorithms that we will consider are probabilistic. Therefore we will use probabilistic Turing machines, which are Turing machines that have a special random tape of fixed length, from which they will read to make random choices. Before launching the probabilistic Turing, every cell of the random tape is filled independently with a uniformly randomly chosen letter of the alphabet. This tape is often called the random coins of the algorithm, and a particular word written on the random tape is called a coin toss. For example, if we want a Turing Machine on the binary alphabet to choose a random number uniformly in $\{0, \cdots, 2^n - 1\}$, we will get a random tape of length n, and read this tape as a n-bits number written in binary. When we formalise the notion of advantage Section 1.4.2, we will talk about probability of events involving a certain number of probabilistic Turing machines. Theses probabilities will be 'taken over the random coins of the algorithm', meaning that they are computed as $\frac{\text{number of coin tosses such that the evenement happens}}{\text{total number of possible coin tosses}}$.

[2]Some problem complexities can show differences when switching from one formalism to another, but this is outside the scope of this thesis

[3]Even tough that would be extremely painful.

Interactive Turing Machines Another type of Turing machines that we do not formalise here are interactive Turing machines. They are used to formalise protocols and therefore are naturally used in cryptography. We need to add communication tapes so that Turing machines can write on each others' communication tapes (i.e. send messages).

Polynomial Time, Non deterministic Polynomial Time The running time of an algorithm on an input is the number of application to the transition function before it reaches a final state. We say that an algorithm is in polynomial time if there exists a polynomial P such that for all $n \in \mathbb{N}$, and for all input of length n then its running time is less than $P(n)$. The set of problems solved by this defines the complexity class P The polynomial time algorithm can be seen as the efficient ones, the one we can run in practice. Of course this is an approximation and it depends on the length of the input, but it is good enough for theoretical work, which has the great pros of being independent of the computer we use and the technological evolutions. However, to evaluate the efficiency of an algorithm we need either a precise polynomial, or even better, to implement it in real life (which allow to take into account practical details of the machine like the number of processors, or the amount of cache memory). On the other hand we can define the class of non-practical algorithms (with the same precision limitation). This is the famous NP (for non deterministic polynomial) class. The only change is that the transition function is non-deterministic, i.e. $\delta(q, \alpha)$ is now a subset of $(q', \alpha', a \in \{\leftarrow, \rightarrow\})$. Each input then gives several results. We then say that an input is accepted if and only if at least one of the computation accepts it. We can now define NP, which is the class of languages (or decision problems) that are recognised in polynomial time by a non-deterministic Turing machine. Another way to see NP is the class of problems that are difficult (possibly exponential time), but checking that a solution is indeed a solution is easy. To see this, you can in a polynomial amount of steps make an exponential amount of 'trials' by giving two choices to the transition function, and then you check if one of them is a solution in polynomial time. NP problems are usually considered non-practical (there are some counter-examples). It has not been proven yet that $P \neq NP$ (nor $P = NP$), meaning that we do not know if an NP machine is strictly more powerful than a regular one. In turn it means that our definition of practical and non practical, besides its few 'practicalities' issues, might be unfunded theoretically. The fact that this is not answered yet prevents any formally proven cryptography, since we would like our honest parties to be efficient (in P), and the dishonest ones that try to break our systems without keys to be impractical (in NP). This is why we make use of assumptions, of the form 'this problem is not in P' .

1.4.2 Security Model and Games

In this section, we give a few security definitions. These and more specific one can be found when relevant in the following chapter of this thesis. Our definitions will be game based. This is one of the two paradigms used in cryptography, the other one being simulation based (also called real-world/ideal-world paradigm). It is often argue that simulation based definitions are easier to use when using basic primitives (e.g. encryption, signatures etc...) to form more complex protocols. for example the universal composability (UC) framework of Ran Canetti [Can00] makes heavy use of simulation. However, since we focus on primitives, it is not a problem for us to use game based definitions. In fact, our game based definitions have well-known equivalents in the real-world/ideal world paradigm. One could also argue that game based definitions are easier to understand intuitively when talking about primitives such as encryption or signatures. A game will specify the rules, i.e. what the adversary can do, and on what condition he wins. We then say our scheme is secure if the adversary almost (the almost is formalised in Section 1.4.2) never

wins. In figure Figure 1.2 we give a game based description of the notion of semantic security [4] defined by Goldwasser and Micali in [GM82].

Advantage All our experiments are parametrised with the security parameter λ. The same security parameter is given in unary to the key generation algorithm. Since we require algorithms to be polynomial, this sets the allowed time to generate the keys. First we define the advantage of an adversary against a security game as the probability (as a function of λ) that he wins minus 0.5. Indeed a trivial adversary that answers at random has a probability of 0.5 to win, so we want to measure how the adversary performs compared to this trivial adversary. We now define a negligible functionDefinition 1.2.

Definition 1.2 *We say that a function f from $\mathbb{N}$ to $\mathbb{R}$ is negligible if for any polynomial P, there exists an integer N such that for all $n \geq N$:*

$$f(n) \leq \frac{1}{P(n)}$$

We need this definition because an adversary could try for example to guess the secret key λ times. If he succeeds then he can easily win the game. Otherwise, he tries at random. Then his advantage would be higher than 0, but we still want to say that the scheme is secure, since this attack is unavoidable, and has very few chances of success. We can now say if a scheme is secure by choosing a game and say that the advantage of any adversary is negligible.

$SEMSEC_{\mathcal{E}}^{\mathsf{Adv}}(1^\lambda)$:
$(\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathcal{E}.\mathsf{KeyGen}(1^\lambda)$
$(m_0, m_1) \leftarrow \mathsf{Adv}(\mathsf{pk})$
$b \xleftarrow{\$} \{0,1\}$
$\mathsf{Adv} \leftarrow \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, m_b)$
return $b == b'$

Figure 1.2: The $SEMSEC$ experiment represents the semantic security game for a (public key) encryption scheme. It is parametrised by the adversary Adv, the encryption scheme $\mathcal{E}$ and, the security parameter λ. $\mathcal{E}.\mathsf{KeyGen}$ is the key generation algorithm of $\mathcal{E}$, and $\mathcal{E}.\mathsf{Enc}$ is the encryption algorithm of $\mathcal{E}$. The adversary gets the public key, then he chooses a pair of messages m_0, m_1. One of this message is chosen at random and its encryption is given to the adversary. Then he tries to guess which one it was. He wins if the game returns true, i.e. if he guessed correctly which message was encrypted. This captures a very strong notion of security, indeed even if the adversary has very little knowledge about the encryption, he can win it easily. Imagine if the adversary knows that the odd messages are encrypted as odd ciphertext, and even messages are encrypted as even ciphertext. This information does not allow decryption of any messages, but by choosing m_0 odd and m_1 even, the adversary wins the game every time. This also means that in the public key setting a randomisation is needed to be semantically secure. This is primordial as the adversary can otherwise encrypt both messages, and see which encryption match the challenge he is given.

This is the most basic security game for cryptography. In the following, we will introduce different games that give more power to the adversary Section 1.4.2 and to define the security of signatures Section 1.4.2.

[4] To be precise we actually define here the most used, and equivalent notion of indistinguishability.

Definition 1.3 (Semantic security) *We say that a (secret key) encryption scheme $\mathcal{E}$ is semantically secure if for all polynomial time adversary (Turing machines)* Adv*, its advantage for the* $SEMSEC^{\mathsf{Adv}}_{\mathcal{E}}$ *game Figure 1.2 is a negligible function.*

CCA, CPA In this paragraph, we introduce oracles, which will model how the adversary can access information about the cryptosystem, such as getting access to decrypted ciphertext. First, we define in figure Figure 1.3 the semantic security under a chosen plaintext attack, meaning that the adversary can choose some plaintexts to encrypt. Note that this notion only makes sense in a symmetric setting, in an asymmetric setting the adversary can encrypt anyway since he has access to the public key.

$\underline{CPA^{\mathsf{Adv}}_{\mathcal{E}}(1^\lambda)}$:
$L \leftarrow \emptyset$
$(\mathsf{sk}) \xleftarrow{\$} \mathcal{E}.\mathsf{KeyGen}(1^\lambda)$
$(m_0, m_1) \leftarrow \mathsf{Adv}^{\mathsf{Enc}(\cdot)}(1^\lambda)$
if $m_0 \notin L$ $m_1 \notin L$
$b \xleftarrow{\$} \{0,1\}$
$\mathsf{Adv} \leftarrow \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, m_b)$
return $b == b'$
return 0

$\underline{\mathsf{Enc}(m)}$:
$c \xleftarrow{\$} \mathcal{E}.\mathsf{Enc}(\mathsf{sk}, m)$
$L \leftarrow L \cup \{m\}$
return c

Figure 1.3: The chosen plaintext attack semantic security experiment. The $\mathsf{Enc}(\cdot)$ in the exponent of the adversary defines an oracle. The adversary can call it at anytime, and gets written the oracle's answer on a special tape. This is a convenient way to give to the adversary 'limited' access to the secret key, in the sense that he can call the oracle to encrypt a message, which uses the secret key, but he cannot do arbitrary computation with the secret key. One could see this as an equivalent of an API. We can also note the use of a global variable L, which stores some information about the call made to the oracle. This is needed so that the adversary cannot challenge on some m_0 or m_1 which has been given to the oracle. We could remove this condition, but by doing so we would need randomisation to get semantic security, as in the public key setting without chosen messages. Since often symmetric scheme, such as block cipher, are deterministic, we choose to present this experiment.

Definition 1.4 (Semantic security under chosen plaintext attack) *We say that a (secret key) encryption scheme $\mathcal{E}$ is semantically under chosen plaintext attack is secure if for all polynomial time adversary (Turing machines)* Adv*, its advantage for the* $CPA^{\mathsf{Adv}}_{\mathcal{E}}$ *game Figure 1.3 is a negligible function.*

Unforgeability under Chosen Message Attack As a last example of security notion, we define Section 1.4.2 the security of a signatures scheme Σ with the strong Existantially UnForgeability under Chosen Message Attack (EUFCMA) game, meaning that the adversary can get signatures on messages of his choice, and then he tries to produce a signature on a message of his choice (that he did not queried).

As illustration we briefly state several (weaker) variations of security experiment for digital signatures:

- weak EUFCMA : the adversary can produce a signature σ for m even if (m, σ') is in L

$\underline{\mathsf{EUF\text{-}CMA}^{\mathsf{Adv}}_{\Sigma}(1^\lambda)}$:
$L \leftarrow \emptyset$
$(\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \Sigma.\mathsf{KeyGen}(1^\lambda)$
$(m^*, \sigma^*) \leftarrow \mathsf{Adv}^{\mathsf{Sign}(\cdot),\mathsf{Verify}(\cdot,\cdot),H(\cdot)}(1^\lambda)$
if $(m^*, \sigma^*) \notin L$
    return $\Sigma.\mathsf{Verify}(\mathsf{pk}, m^*)$
return 0

$\underline{\mathsf{Sign}(m)}$:
$\sigma \xleftarrow{\$} \Sigma.\mathsf{Sign}(\mathsf{sk}, m)$
$L \leftarrow L \cup \{m, \sigma\}$
return σ

$\underline{\mathsf{Verify}(m, \sigma)}$:
return $\Sigma.\mathsf{Verify}(\mathsf{pk}, m, \sigma)$

Figure 1.4: The strong EUF-CMA experiment for digital signature schemes.

- EUF : the adversary does not have access to the signing algorithm
- Universal Unforgeability : The adversary does not choose the message for which he tries to forge a signature

1.5 Current Research Trends in Cryptography

In this section, we briefly expose some topics with which the cryptography community is interested nowadays so that the reader can have a broader view of where our results are in the field, and why those fields are of interest.

1.5.1 More Adversaries

As exposed in Section 1.4.2 different types of adversaries are considered in cryptography. Knowing that the communicants start their messages by 'hello' gives you access to known plaintext/ciphertext pairs. Cryptography being built on top of Internet protocols allows chosen plaintext/ciphertext attacks, and one could think that this is the worst possible case. This section will expose unintuitive adversaries that are considered in cryptography.

1.5.1.1 Computers are not Black Box

A hidden assumption in the models presented in Section 1.4.2 is that computations are done in a black box, and the adversary only has access to the public channel (e.g. the Internet). However this doesn't hold when cryptographic devices are used 'in the field' (e.g. IoT devices, smart cards etc...) and computers do leak information others than by the intended Input/Output interface. These unintentional leakage channels are called side channels.

Side Channel Analysis In its seminal paper [Koc96] Paul Kocher demonstrated that the time of computation of a computer (or any computing device) is linked to the data it is processing. This allowed to obtain secret key just by measuring how long an RSA encryption took. He later shown in [KJJ99a] that a similar attack could be done with the power consumption. This was then extended in many ways, using, for example, electromagnetic emission, and more advanced statistical methods.

Fault Injection A somewhat similar method to side channel analysis is fault injection. Instead of passively listening to the targeted device, it is possible to perturbate it, so that computations are faulty. This sometimes allows extremely powerful cryptanalysis, as we demonstrate in Section 5.2. Side channels, as well as fault injection models, are sometimes referred as gray box models, since there is some leakage but in a noisy and particular form.

White Box Cryptography What can be thought of as natural extension of gray box cryptography is white box cryptography. It was introduced in [CEJvO02]. Its purpose is to model an almighty adversary, that can see and modify your source codes. Under this model, one could carry cryptographic computations in an untrusted environment such as an infected smart-phone. However, as of today no secure implementation of a white box cipher are known.

1.5.1.2 Post Quantum Cryptography

Another hidden assumption in the classical models is the fact that we model our adversary by Turing machines. However, a new type of computer is under development: quantum computers. We will not explain how they work nor what they can do, but it is sufficient for our purpose to know that Schorr's algorithm [Sho97a] can leverage them to efficiently solve factorisation and discrete logarithm (or more generally the hidden subgroup problem). This prevent classical and efficient public key algorithm such as RSA and Diffie-Hellman key exchange to be used, and the community is now looking at cryptosystem based on other assumptions.

NIST Competition As it is now usual, the NIST launched the first competition for standarisation of post quantum public key algorithms. More information can be found at `https://csrc.nist.gov/Projects/Post-Quantum-Cryptography`.

Type of Post Quantum Cryptography We give as references some paradigms that are considered quantum safe:

- Lattice based: this is the most widely studied. 'A new hope' [ADPS15] based on the Ring-LWE assumption and 'Frodo' [BCD+16] based on the LWE assumption are two representatives of it.
- Code based: This is a very old type of cryptography (1978), but was left aside due to efficiency reasons, until it was find out that we needed quantum resistance. The McElice crpytosystem [McE78] is its principal representative.
- Isogeny based: This is very recent (2011) in its post quantum version [FJP11], and is based on the assumption that it is hard to find isogeny between supersingular curves. The assusmption does not hold with non supersingular curves against quantum computers.
- Mersenne prime arithmetic based: This is the most recent type of post quantum cryptography (and the newest public key cryptosystem). We discuss it in Chapter 4, where we present an attack on an early version, as well as a variation of the 'fixed' version. The 'fixed' version is meant to be presented at the NIST competition at the time of the writing of this thesis.

1.5.2 Quantum Cryptography

Like many topics in cryptography (such as elliptic curves or lattice), quantum technologies have raised interest in cryptanalysis, but can also be used to derive new schemes. Even if quantum computers might be found in the market in a foreseeable future, because of cost issues, it is

hard to believe that one wants to base a cryptographic scheme on them. However, quantum exchange of information could solve the key exchange problem in a different way than encrypting the key. Indeed exchanging information with quantum means, allows removing the eavesdropper assumption, since unlike in classical physics, observing quantum information changes it. This is the basic idea for quantum key exchange. We investigate the security of such schemes from an information theoretic point of view in Section 5.3.

1.5.3 Optimisations and Usability

While it might not be intuitive, cryptography's most recurrent problems for real world applications do not lie in security proofs or assumptions. Most of the time cryptography is misused or not used at all when it is needed. There are lots of different reasons for that. We give three of them which are performances, usability and lack of awareness. We did work on performances, and give usability and lack of awareness to place our work in a greater context. We give them in decreasing order regarding the general amount of work the community put in these.

Performances Since cryptography is a part of computer science, it was natural for the community to look for algorithmic 'tricks' to speed-up computations. This was further justified by the fact that cryptographic algorithms (especially public keys' one) are costly, and had to be executed in constrained environment such as smart cards. Nowadays processors are much more powerful, and efficiency problems are not as present as they used to be. However, there still are incentives for optimisations, such as:

- Cryptography is used more and more, only because more and more sensitive information is being sent trough various networks. So cryptography needs to be faster.
- This information can be treated by even more constrained environments than smart cards, typically IoT devices.
- Some companies that do not manipulate extremely sensible data can easily choose not to add any cryptographic layer, to improve customer experience thanks to faster reaction from their devices.

Due to these reasons, we became interested in two 'batch' optimisations[5] in Chapter 3. The first, Section 3.2 verifies several digital signatures at the same time, and the second Section 3.3 optimizes the creation of several digital signatures at the same time.

Usability Usability is naturally a less scientific topic and therefore has attracted less interest in the community. However, due to numerous mistakes from general purpose developers, some scientific communities got interested in this topic. [ABF+17] is an example of a comparative study on the usability of several cryptographic libraries study.

Lack of Awareness This is not a scientific topic either but is nonetheless of crucial importance for the digital world security and privacy. For example [PRRR15] studies real life privacy breaches by governments. This allows to point out breaches in real life defense, as well as advertise the importance of cryptography and computer security so that Snowden's revelations are not forgotten.

[5] optimisations that works when several cryptographic primitives have to be executed

Chapter 2

Results and Contributions

Contents

2.1 Organisation

- Chapter 3 presents two optimisations for digital signature schemes. Section 3.3 improves Schnorr signatures: we show how to securely reuse the nonce, which furthermore enables signing in fewer operations. This paper was published at ESORICS 2017. Section 3.2 introduces the question of optimal batch signature verification with a priori probabilities, for which we provide an analysis, algorithms, and heuristics. This paper is currently under review.
- Chapter 4 presents a practical cryptanalysis of a recent public key cryptosystem proposed by Aggarwal et al. This paper was published at LATINCRYPT 2017. We also present an unpublished variation of this cryptosystem.
- Section 5.2 presents a differential fault analysis of the lightweight cipher PRINCE. This paper was published at CRiSIS 2016.
- Section 5.3 studies the entropy loss when being eavesdropped during a quantum key exchange. This paper is currently under review.

2.2 Additional Work

The following research works published during the thesis will not be presented here for thematic alignment reasons.

- In [BBBK16] we study a new probabilistic approach to timed language inclusion (an undecidable problem) based on volumetry. This work was published at QUEST 2016.
- In [BFGN16] we study honey encryption, a technique that prevents adversaries from checking the correctness in an attempted decryption, and in turn, achieves security beyond brute force bound. This work was published at MyCrypt 2016.

- In [BCGN17a] we argue why system commands should be encrypted despite Kerckhoffs' principle, and give guidelines to measure their (in)security. This was an invited talk at AsisaCCS 2017.
- [BCGN16b, BCGN16a, BCGN16c] are three popular science articles on cryptography for IEEE Security and Privacy Magazine. The three articles are about Fully Homomorphic Encryption, Obfuscation, and White-Box Cryptography.
- In [ABGN16] we argue for modifications of the rules to participate in the Nijmeegse Vierdaagse (a famous walking event), where the rules were subsequently changed. This was published in the The New Codebreakers 2016

In addition to scientific publications the author filed 6 patents and participated in the writing of an accepted funded proposal for the Analysis oN BLind Cloud (ANBLIC) project which aim at using Fully Homomorphic encryption and Functional Encryption at an industrial scale (http://competitivite.gouv.fr/le-24e-appel-a-projets-fui-regions/les-resultats-du-24e-appel-a-projet html).

Chapter 3

Batch Processing in Cryptography

Contents

3.1 Introduction

Cryptographic operations can be costly, and this is often the case that these costs induce insecurities. Indeed in practice non-critical industries tend to prefer efficiency over security. This can translate into weaker keys or no cryptographic layer at all. In the 70's a number of optimisations were proposed to enable cryptographic applications on smart cards. Nowadays it is argued that the Internet of things will make use of another wave of optimisations, such as lightweight ciphers. We actually leverage two old techniques proposed around the 70's. In

Section 3.2 we tackle the issue of verifying a large number of signatures at the same, which is suited for large-scale industrial applications. This has already been done resulting in being able to say if at least one signature is incorrect in a large group. We add a priori probabilities of correctness of signatures and propose batch signatures algorithm leveraging this additional assumption. In Section 3.3 we propose a modification of the Schnorr signature scheme [Sch90] which is well suited with pre computations techniques for exponentiation. We then study how our new scheme compares to the original scheme with different existing pre computations schemes.

3.2 Optimal Batch Signatures

Abstract

Batch cryptography started with the observation that RSA's homomorphic properties allow checking many signatures at once.

Therefore several verification algorithms were designed to check a batch of RSA or DSA signatures simultaneously. If all the signatures are correct, batch verification succeeds after a few operations. However, if a single signature is incorrect, failure does not indicate *which* signatures are wrong.

This paper describes how to optimally detect incorrect signatures in batches, i.e. in a minimum expected number of tests, given a list indicating the a priori probabilities with which each of the signatures in the batch is correct. The resulting algorithms are non-intuitive and quite surprising.

This is joint work with Éric Brier, Noémie Cartier, Simon Cogliani, Aisling Connolly, Nathanaël Courant, Rémi Géraud and David Naccache. This work is under review at the Algorithmica journal.

3.2.1 Introduction and motivation

Batch cryptography, introduced by Fiat in [Fia90, Fia97], leverages RSA's homomorphic properties [RSA78b] to speed-up signature schemes. On the verification side, the product of individual RSA signatures can be checked in a single operation as explained in [BGR98]. This idea can be applied to many other schemes enjoying homomorphic properties.

In [NMVR95], Naccache et al. described the first batch verifier for DSA signatures. Laih and Yen [YL95] proposed a batch verification method of DSA and RSA signatures, later broken by [BP00]. Similarly, another construction of Harn for RSA and DSA was soon proven insecure and retracted [HLH00, HLT01]. This called for a more systematic approach, where security of batch verification could be modelled and proved.

This question was answered when Bellare, Garay and Rabin [BGR98] presented three generic methods for batching modular exponentiations: the random subset test, the small exponents test, and the bucket test. [BGR98] showed how to apply these methods to batch verification of DSA signatures.

The problem of bad signature identification arises when at least one signature in the batch is incorrect, in which case the batch test fails[1]. The naive approach is then to test individually each signature, which can be costly.

For this reason several solutions were proposed to sieve out bad signatures quickly: At Eurocrypt 1998, Bellare et al. introduced RSA screening [BGR98], soon broken and fixed by Coron and Naccache [CN99]; at PKC 2000 Pastuszak et al. described a simple "divide-and-conquer" algorithm to identify one incorrect signature in a batch [PMPS00]. Another approach by Law and Matt [LM07], using identity-based signature schemes, also allows identifying invalid signatures in a batch.

Our contribution: This paper departs from the above approaches by assuming the availability of extra information: the a priori probability that each given signature is correct. In practice, we may either assume that such probabilities are given, estimated from signer trust metrics, or are learned from past verifications. We assume in this work that these probabilities are known.

[1] Actually testing two incorrect signatures might answers true due to cancellation: if σ_1 and σ_2 are correct signatures for m_1 and m_2, for any α testing $\frac{\sigma_1}{\alpha} \times \alpha\sigma_2$ for $m_1 \times m_2$ will yield true. We will ignore this issue since it can only happen either with negligible probabilities or from manipulation from legitimate signatures.

In this paper, we show that it is possible to find incorrect signatures in an optimal way — i.e. by performing on average the minimum number of tests — by exploring the combinatorial and algebraic properties of verification algorithms. This turns out to be faster than RSA screening or divide-and-conquer verifiers in the majority of settings.

On top of cryptographic applications, we note that optimal batch testing can improve the time, cost and reliability of other tests, such as medical screening, traitor-tracing or fraud control in large networks.

3.2.2 Intuition

Before introducing models and general formulae, let us provide the intuition behind our algorithms.

Let us begin by considering the basic case of two signatures. These can be verified individually or together, in a batch. Individual verification claims a minimum of two units of work—check one signature, then check the other. Batch-checking them requires a minimum of one verification. If it is highly probable that both signatures are correct, then batch verification is interesting: If both signatures are indeed correct, we can conclude after one test and halve the verification cost. However, if that fails, we are nearly back to square one: One of these signatures (at least) is incorrect, and we don't know which one.

In this paper, we identify *when* to check signatures individually, and when to batch-check them instead—including all possible generalizations when there are more than 2 signatures. We assume that the probability of a signature being incorrect is known to us in advance. The result is a testing 'metaprocedure' that offers *the best alternative to sequential and individual testing.*

To demonstrate: the testing procedure that always works is to verify every signature individually, one after the other: This gives the 'naive procedure', which always performs 2 verifications, as illustrated in Figure 3.1. In this representation, the numbers in parentheses indicate which signatures are being tested at any given point. The leaves indicate which signatures are correct (denoted `1`) or incorrect (denoted `0`), for instance, the leaf `01` indicates that only the second signature is valid. Note that the order in which each element is tested does not matter: There are thus 2 equivalent naive procedures, namely the one represented in Figure 3.1, and the procedure obtained by switching the testing order of (1) and (2).

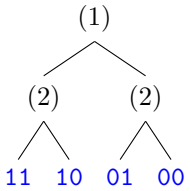

Figure 3.1: The "naive procedure" for $n = 2$ consists of testing each entity separately and sequentially.

Alternatively, we can leverage the possibility to test both signatures together as the set $\{1, 2\}$. In this case, batching the pair $\{1, 2\}$ must be the first step: Indeed, testing $\{1, 2\}$ after any other test would be redundant, and the definition of testing procedures prevents this from happening. If the test on $\{1, 2\}$ is correct, both signatures are correct and the procedure immediately yields the outcome `11`. Otherwise, we must identify which of the signatures 1 or 2 (or both) is responsible for the test's incorrectness. There are thus two possible procedures, illustrated in Figure 3.2.

Intuitively, the possibility that this procedure terminates early indicates that, in some situations at least, only one test is performed, and is thus less costly than the naive procedure. However, in

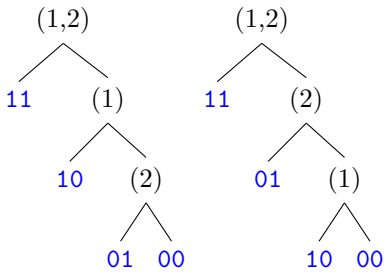

Figure 3.2: Two batching testing procedures having $(1, 2)$ as root.

some situations up to three tests can be performed, in which case it is more costly than the naive procedure.

Concretely, we can compute how many verifications are performed on average by each approach, depending on the probability x_1 that the first signature is incorrect, and x_2 that the second is incorrect. To each procedure, naive, batch-*left*, batch-*right*, we associate the following polynomials representing the expected stopping time:

- $L_{\text{naive}} = 2$
- $L_{\text{batch-}left} = (1 - x_1)(1 - x_2) + 2(1 - x_1)x_2 + 3x_1(1 - x_2) + 3x_1x_2$
- $L_{\text{batch-}right} = (1 - x_1)(1 - x_2) + 3(1 - x_1)x_2 + 2x_1(1 - x_2) + 3x_1x_2$

It is possible to see analytically which of these polynomials evaluates to the smallest value as a function of (x_1, x_2). Looking at Figure 3.3, we use these expectations to define zones in $[0, 1]^2$ where each algorithm is optimal (i.e. the fastest on average). More precisely, the frontier between zones C and B has equation $x_1 = x_2$, the frontier between A and B has equation $x_2 = (x_1 - 1)/(x_1 - 2)$, the frontier between A and C has equation $x_2 = (2x_1 - 1)/(x_1 - 1)$, and the three zones meet at $\overline{x}_1 = \overline{x}_2 = (3 - \sqrt{5})/2$.

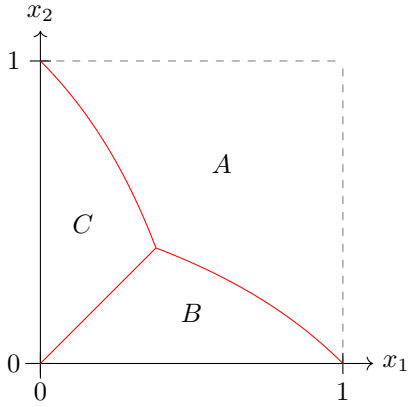

Figure 3.3: Optimality zones for $n = 2$. A: naive procedure; B: batching procedure (right); C: batching procedure (left).

Having identified the zones, we can write an algorithm which, given x_1 and x_2, identifies in which zone of Figure 3.3 (x_1, x_2) lies, and then apply the corresponding optimal verification

sequence. In the specific case illustrated above, three algorithms out of three were needed to define the zones; however, for any larger scenario, we will see that only a very small portion of the potential algorithms will be considered.

Our objective is to determine the zones, and the corresponding verification algorithms, for arbitrary n, to identify which signatures in a set are correct and which are not, while minimizing the expected number of verification operations.

3.2.3 Preliminaries

This section will formalize the notion of a testing procedure, and the cost thereof so that the problem at hand can be mathematically described. We aim at the greatest generality, which leads us to introduce 'and-tests', a particular case of which are signatures that can be batch verified.

3.2.3.1 Testing procedures

We consider a collection of n signatures. Let $[n]$ denote $\{1, \ldots, n\}$, and $\Omega = \mathcal{P}([n])\backslash\{\emptyset\}$, where $\mathcal{P}$ is the power set (ie. $\mathcal{P}(X)$ is the set of subsets of X).

Definition 3.1 (Test) *A* test *is a function $\phi : \Omega \to \{0, 1\}$, that associates a bit to each subset of Ω.*

We are mainly interested in tests satisfying homomorphic properties. We focus in this work on the following:

Definition 3.2 (And-Tests) *An* and-test $\phi : \Omega \to \{0, 1\}$ *is a test satisfying the following property:*

$$\forall T \in \Omega, \quad \phi(T) = \bigwedge_{t \in T} \phi(\{t\}).$$

In other terms, the result of an and-test on a set is exactly the logical and of the test results on individual members of that set.

Example 3.1 *Let* WasSigned *be a function that returns True if and only if all messages were signed at some point by the legitimate signer. Consider a set of RSA signatures $T = \{\sigma_1, \ldots, \sigma_n\}$ on a (respective) set of messages $M = \{m_1, \ldots, m_n\}$, then we have*

$$\mathsf{WasSigned}(M) = \mathsf{Verify}\left(\prod_{m \in M} m, \prod_{\sigma \in T} \sigma\right).$$

Hence the test $\phi(T) = \mathsf{WasSigned}(T)$ is an and-test, that returns False if at least one signature was not generated by the legitimate owner, and True otherwise.

Remark *We have to introduce the* WasSigned *primitive, because if one signature is multiplied by any α and another divided by the same α, then both are incorrect and* Verify *applied on the product will return True. However, an attacker without forgery capabilities cannot generate signatures for which* WasSigned *returns True and are not computed from signatures generated by the legitimate signer with more than negligible probability.*

Remark *Note that 'or-tests', where $\wedge$ is replaced by $\vee$ in the definition, are exactly dual to our setting. 'xor-tests' can be defined as well but are not investigated here. Although theoretically interesting by their own right, we do not address the situation where both and-tests and or-tests are available, since we know of no concrete application where this is the case.*

Elements of Ω can be interpreted as n-bit strings, with the natural interpretation where the i-th bit indicates whether i belongs to the subset. We call *selection* an element of Ω.

Definition 3.3 (Outcome) *The* outcome $F_\phi(T)$ *of a test* ϕ *on* $T \in \Omega$ *is the string of individual test results:*

$$F_\phi(T) = \{\phi(x), x \in T\} \in \{0,1\}^n.$$

When $T = [n]$, F_ϕ *will concisely denote* $F_\phi([n])$.

Our purpose is to determine the outcome of a given test ϕ, by minimizing in the expected number of queries to ϕ. Note that this minimal expectation is trivially upper bounded by n.

Definition 3.4 (Splitting) *Let* $T \in \Omega$ *be a selection and* ϕ *be a test. Let* $\mathcal{S}$ *be a subset of* Ω. *The* positive part of $\mathcal{S}$ with respect to T, *denoted* $\mathcal{S}_T^\top$, *is defined as the set*

$$\mathcal{S}_T^\top = \{S | S \in \mathcal{S}, S \wedge T = T\}.$$

where the operation $\wedge$ *is performed element-wise. This splits* $\mathcal{S}$ *into two. Similarly the complement* $\mathcal{S}_T^\bot = \mathcal{S} - \mathcal{S}_T^\top$ *is called the* negative part of $\mathcal{S}$ with respect to T.

We are interested in algorithms that find F_ϕ. More precisely, we focus our attention on the following:

Definition 3.5 (Testing procedure) *A* testing procedure *is a binary tree* $\mathcal{T}$ *with labelled nodes and leaves, such that:*

1. *The leaves of* $\mathcal{T}$ *are in one-to-one correspondence with* Ω *in string representation;*

2. *Each node of* $\mathcal{T}$ *which is not a leaf has exactly two children,* $(S_\bot, S_\top)$, *and is labelled* (S, T) *where* $S \subseteq \Omega$ *and* $T \in \Omega$, *such that*

   a) $S_\bot \cap S_\top = \emptyset$

   b) $S_\bot \sqcup S_\top = S$

   c) $S_\bot = S_T^\bot$ *and* $S_\top = S_T^\top$.

Remark *It follows from the Definition 3.5 that a testing procedure is always a* finite *binary tree and that no useless calls to* ϕ *are performed. Indeed, doing so would result in an empty* S *for one of the children nodes. Furthermore, the root node has* $S = \Omega$.

3.2.3.2 Interpreting and representing testing procedures

Consider a testing procedure $\mathcal{T}$, defined as above. $\mathcal{T}$ describes the following algorithm. At each node (S, T), perform the test ϕ on the selection T of signatures. If $\phi(T) = 0$, go to the left child; otherwise go to the right child. Note that at each node of a testing procedure, only one invocation of ϕ is performed.

The tree is finite and thus this algorithm reaches a leaf S_{final} in a finite number of steps. By design, $S_{\text{final}} = F_\phi$.

Remark *From now on, we will fix* ϕ *and assume it implicitly.*

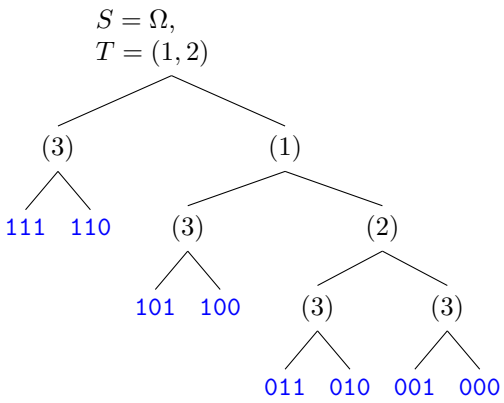

Figure 3.4: Graphical representation of a testing procedure. The collection is $[3] = \{1, 2, 3\}$, $\Omega = \{\{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$, the initial set of selections is $S = \Omega$. Only the T labels are written on nodes. Only the S labels are written for leaves.

Remark *We represent a testing procedure graphically as follows: Nodes (in black) are labelled with T, whereas leaves (in blue) are labelled with S written as a binary string. This is illustrated in Figure 3.4 for $n = 3$.*

This representation makes it easy to understand how the algorithm unfolds and what are the outcomes: Starting from the root, each node tells us which entity is tested. If the test is positive, the right branch is taken. Otherwise, the left branch is taken. Leaves indicate which signatures tested positive and which signatures tested negative from now on.

Remark *The successive steps of a testing procedure can be seen as imposing new logical constraints. These constraints ought to be satisfiable (otherwise one set S is empty in the tree, which cannot happen). The formula at a leaf is maximal in the sense that any additional constraint would make the formula unsatisfiable. This alternative description in terms of satisfiability of Boolean clauses is in fact strictly equivalent to the one that we gave.*

In that case, T is understood as a conjunction $\bigwedge_{T[i]=1} t_i$, S is a proposition formed by a combination of terms t_i, connectors $\vee$ and $\wedge$, and possibly $\neg$. The root has $S = \top$. The left child of a node labelled (T, S) is labelled $S_T^\perp = S \wedge (\neg T)$; while the right child is labelled $S_T^\top = S \wedge T$. At each node and leaf, S must be satisfiable.

3.2.3.3 Probabilities on trees

To determine how efficient any given testing procedure is, we need to introduce a probability measure and a metric that counts how many calls to ϕ are performed.

We consider the discrete probability space $(\Omega, \Pr)$. The *expected value* of a random variable X is classically defined as:

$$\mathrm{E}[X] = \sum_{\omega \in \Omega} X(\omega) \Pr(\omega)$$

Let $\mathcal{T}$ a testing procedure, and let $S \in \Omega$ be one of its leaves. The *length* $\ell_{\mathcal{T}}(S)$ of $\mathcal{T}$ over S is the distance on the tree from the root of $\mathcal{T}$ to the leaf S. This corresponds to the number of tests required to find S if S is the outcome of ϕ. The *expected length* of a testing procedure $\mathcal{T}$ is defined naturally as:

$$L_{\mathcal{T}} = \mathrm{E}\left[\ell_{\mathcal{T}}\right] = \sum_{\omega \in \Omega} \ell_{\mathcal{T}}(\omega) \Pr(\omega)$$

It remains to specify the probabilities $\Pr(\omega)$, i.e. for any given binary string ω, the probability that ω is the outcome.

If the different tests are independent, we can answer this question directly with the following result:

Lemma 3.1 *Assume that the events '$\phi(\{i\}) = 1$' and '$\phi(\{j\}) = 1$' are independent for $i \neq j$. Then, $\forall \omega \in \Omega$, $\Pr(\omega)$ can be written as a product of monomials of degree 1 in $x_1, \ldots, x_n$, where*

$$x_i = \Pr(\phi(\{i\}) = 1) = \Pr(i\text{-th bit of } \omega = 1).$$

Thus $L_{\mathcal{T}}$ is a multivariate polynomial of degree n with integer coefficients.

In fact, or-tests provide inherently independent tests. Therefore we will safely assume that the independence assumption holds.

Example 3.2 *Let $n = 5$ and $\omega =$ `11101`, then $\Pr(\omega) = x_1 x_2 x_3 (1 - x_4) x_5$.*

Remark *$L_{\mathcal{T}}$ is uniquely determined as a polynomial by the integer vector of length 2^n defined by all its lengths: $\ell(\mathcal{T}) = (\ell_{\mathcal{T}}(\texttt{0...0}), \ldots, \ell_{\mathcal{T}}(\texttt{1...1}))$.*

3.2.4 Optimal batch verification

We have now introduced everything necessary to state our goal mathematically. Our objective is to identify the best performing testing procedure $\mathcal{T}$ (i.e. having the smallest $L_{\mathcal{T}}$) in a given situation, i.e. knowing $\Pr(\omega)$ for all $\omega \in \Omega$.

3.2.4.1 Generating all procedures

We can now explain how to generate all the testing procedures for a given $n \geq 2$.

One straightforward method is to implement a generation algorithm based on the definition of a testing procedure. Algorithm 1 does so recursively by using a coroutine. The complete list of testing procedures is recovered by calling `FindProcedure`$(\Omega, \Omega \setminus \{\emptyset\})$.

Algorithm 1: FindProcedure

Input: $S \in \Omega$, $C \in \Omega$.
Output: A binary tree.

1. if $|S| == 1$ then return S
2. $S'_\perp = S'_\top = C' = \emptyset$
3. for each $T \in C$
4. $\quad S_\perp = S_T^\perp$
5. $\quad S_\top = S_T^\top$
6. $\quad$ if $S_\perp \notin S'_\perp$ and $S_\top \notin S'_\top$
7. $\quad\quad S'_\perp = S'_\perp \cup \{S_\perp\}$
8. $\quad\quad S'_\top = S'_\top \cup \{S_\top\}$
9. $\quad\quad C' = C' \cup \{T\}$
10. for $i \in \{1, \ldots, |C'|\}$
11. $\quad \overline{C} = C - C'[i]$
12. $\quad$ for each $\mathcal{T}_\perp \in$ FindProcedure$(S'_\perp[i], \overline{C})$
13. $\quad\quad$ for each $\mathcal{T}_\top \in$ FindProcedure$(S'_\top[i], \overline{C})$
14. $\quad\quad\quad$ yield $(C'[i], \mathcal{T}_\perp, \mathcal{T}_\top)$

We implemented this algorithm in Python. The result of the testing procedure generations for small values of n is summarized in Table 3.1. The number of possible testing procedures grows very quickly with n.

Table 3.1: Generation results for some small n

n	Number of procedures	Time
1	1	0
2	4	~ 0
3	312	~ 0
4	36585024	~ 30 mn

An informal description of Algorithm 1 is the following. Assuming that one has an unfinished procedure (i.e. nodes at the end of branches are not all leaves). For those nodes S, compute for each T the sets $S_T^\top$ and $S_T^\perp$. If either is empty, abort. Otherwise, create a new (unfinished) procedure, and launch recursively on nodes (not on leaves, which are such that S has size 1).

Algorithm 1 terminates because it only calls itself with strictly smaller arguments. We will discuss this algorithm further after describing some properties of the problem at hand.

3.2.4.2 Metaprocedures

Once the optimality zones, and the corresponding testing procedures, have been identified, it is easy to write an algorithm which calls the best testing procedure in every scenario. At first sight, it may seem that nothing is gained from doing so — but as it turns out that only a handful of procedures need to be implemented.

This construction is captured by the following definition:

Definition 3.6 (Metaprocedure) *A* metaprocedure $\mathcal{M}$ *is a collection of pairs* $(Z_i, \mathcal{T}_i)$ *such that:*

1. $Z_i \subseteq [0,1]^n$, $Z_i \cap Z_j = \emptyset$ *whenever* $i \neq j$ *and* $\bigsqcup_i Z_i = [0,1]^n$.
2. $\mathcal{T}_i$ *is a testing procedure and for any testing procedure* $\mathcal{T}$,

$$\forall x \in Z_i, \quad L_{\mathcal{T}_i}(x) \leq L_{\mathcal{T}}(x).$$

A metaprocedure is interpreted as follows: Given $x \in [0,1]^n$ *find the unique* Z_i *that contains* x *and run the corresponding testing procedure* $\mathcal{T}_i$. *We extend the notion of expected length accordingly:*

$$L_{\mathcal{M}} = \min_i L_{\mathcal{T}_i} \leq n$$

One way to find the metaprocedure for n, is to enumerate all the testing procedures using Algorithm 1, compute all expected lengths $L_{\mathcal{T}}$ from the tree structure, and solve polynomial inequalities.

Surprisingly, a vast majority of the procedures generated are nowhere optimal: This is illustrated in Table 3.2. Furthermore, amongst the remaining procedures, there is a high level of symmetry. For instance, in the case $n = 3$, 8 procedures appear 6 times, 1 a procedure appears 3 times, and 1 procedure appears once. The only difference between the occurrences of these procedures — which explains why we count them several times — is the action of the symmetric group S_6 on the cube (see Section 3.2.7 for a complete description).

The metaprocedure for $n = 3$ cuts the unit cube into 52 zones, which correspond to a highly symmetric and intricate partition, as illustrated in Figures 3.5 to 3.7. An STL model was constructed and is available upon request.

The large number of suboptimal procedures shows that the generate-then-eliminate approach quickly runs out of steam: Generating all procedures for $n = 6$ seems out of reach with Algorithm 1. The number of zones, which corresponds to the number of procedures that are optimal in some situation, is on the contrary very reasonable.

Lemma 3.2 (Number of naive procedures) *Let* $n \geq 1$, *then there are*

$$P(n) = \prod_{k=1}^{n} k^{2^{n-k}}$$

equivalent naive procedures.

Proof: By induction on n: There are $(n+1)$ choices of a root node, $P(n)$ choices for the left child, and $P(n)$ choices for the right child. This gives the recurrence $P(n+1) = (n+1)P(n)^2$, hence the result. □

This number grows rapidly and constitutes a lower bound for the total number of procedures (e.g. for $n = 8$ we have $P(n) > 2^{184}$). On the other hand, the naive procedure is the one with maximal multiplicity, which yields a crude upper bound $\alpha P(n)$ on the number of procedures, where α is the 2^k-th Catalan number.

The zones can be determined by sampling precisely enough the probability space. Simple arguments about the regularity of polynomials guarantee that this procedure succeeds when working with infinite numerical precision. In practice, although working with infinite precision is feasible (using rationals), we opted for floating-point numbers, which are faster. The consequence is that sometimes this lack of precision results in incorrect results on the zone borders — however, this is easily improved by increasing the precision or checking manually that there is no subzone near the borders.

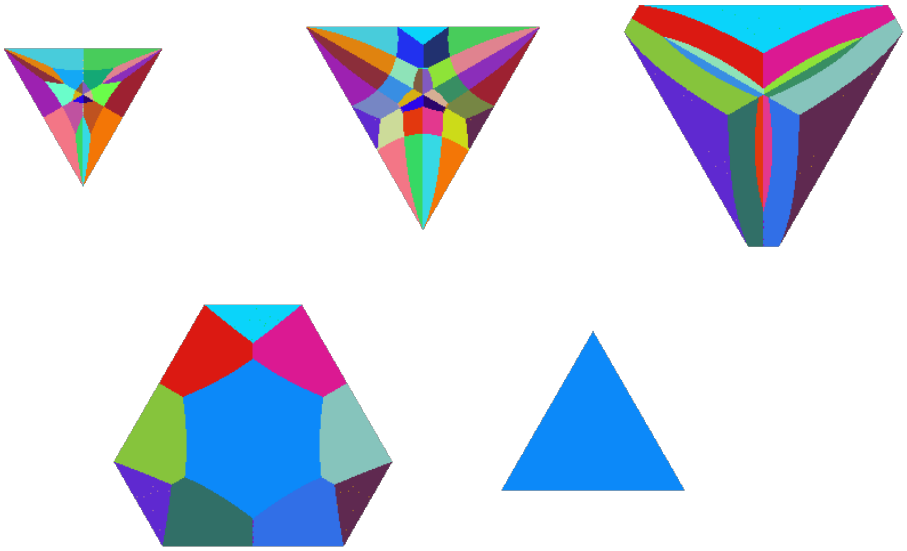

Figure 3.5: Slices of the cube decomposition for the $n = 3$ metaprocedure. The slices are taken orthogonally to the cube's main diagonal, with the origin at the center of each picture. Each color corresponds to a procedure. The symmetries are particularly visible.

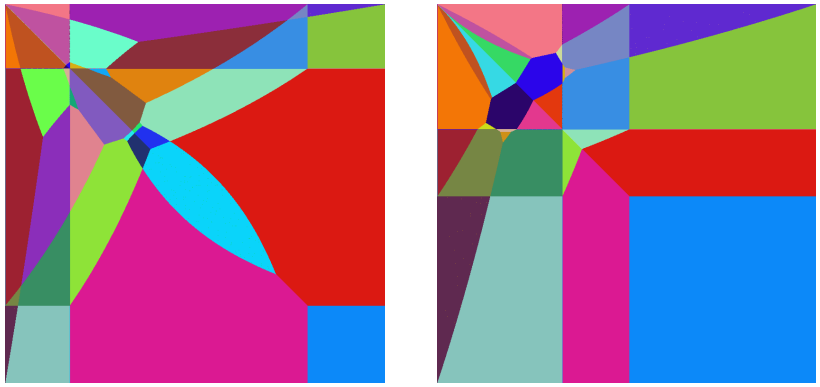

Figure 3.6: Slices through the cube at the $z = 0.17$ (left) and the $z = 0.33$ (right) planes, showing the metaprocedure's rich structure. The origin is at the top left.

n	Number of procedures	Zones
1	1	1
2	4	3
3	312	52
4	36585024	181
5	$8.926 \cdot 10^{20}$	?
6	$2.242 \cdot 10^{55}$	?

Table 3.2: Procedures and metaprocedures for some values of n. The number of zones for $n = 5$ and 6 cannot be determined in a reasonable time with the generate-then-eliminate approach.

3.2.5 Pruning the generation tree

We now focus on some of the properties exhibited by testing procedures, which allows a better understanding of the problem and interesting optimizations. This in effect can be used to prune early the generation of procedures and write them in more compactly by leveraging symmetries. We consider in this section a testing procedure $\mathcal{T}$.

Lemma 3.3 *Let B_0 and B_1 be two binary strings of size n, that only differ by one bit (i.e. $B_0[i] = 0$ and $B_1[i] = 1$ for some i). Then $\ell_{\mathcal{T}}(B_0) \leq \ell_{\mathcal{T}}(B_1)$.*

Proof: First notice that for all T, T', and $b, b' \in \{\top, \bot\}$ we have $(S_T^b)_{T'}^{b'} = (S_{T'}^{b'})_T^b$. We will denote both by $S_{TT'}^{bb'}$.

We have the following : If there exists k, $T_1, \ldots, T_k$, and $\beta_1, \ldots, \beta_k$ such that

$$(\Omega)_{T_1 \cdots T_k}^{\beta_1 \cdots \beta_k} = \{B_1\}$$

then there exists $i \leq k$ such that

$$(\Omega)_{T_1 \cdots T_i \cdots T_k}^{\beta_1 \cdots \neg\beta_i \cdots \beta_k} = \{B_0\}$$

Indeed there exists $i \leq k$ such that $\beta_i = \top$ and $T_i = \{i_0\} \cup E$ where for all j in E, $B_0[j] = B_1[j] = 0$. This yields

$$(\Omega)_{T_1 \cdots T_{i-1} T_{i+1} \cdots T_k}^{\beta_1 \cdots \beta_{i-1} \beta_{i+1} \cdots \beta_k} = \{B_0, B_1\}$$

and the result follows. □

Remark *Lemma 3.3 indicates that testing procedures are, in general, unbalanced binary trees: The only balanced procedure being the naive one.*

Lemma 3.4 *If $\mathcal{N}$ is the naive procedure, then for any testing procedure $\mathcal{T}$ and for all $x_1, \ldots, x_n$ such that $x_i > \frac{1}{2}$,*

$$L_{\mathcal{N}}(x_1, \ldots, x_n) \leq L_{\mathcal{T}}(x_1, \ldots, x_n).$$

In other terms $\{\forall i \in [n], \frac{1}{2} \leq x_i \leq 1\}$ is contained in the naive procedure's optimality zone.

Proof: An immediate corollary of Lemma 3.3 is that for all $i \in [n]$, we have $\partial_{x_i} L_{\mathcal{T}}(x_1, \ldots, x_n) \geq 0$, where ∂_{x_i} indicates the derivative with respect to the variable x_i. Since the native procedure has a constant length, it suffices to show that it is optimal at the point $\{\frac{1}{2}, \ldots, \frac{1}{2}\}$. Evaluating the length polynomials at this point gives

$$L_{\mathcal{T}}\left(\frac{1}{2}, \ldots, \frac{1}{2}\right) = \frac{1}{2^n} \sum_{\omega \in \Omega} \ell_{\mathcal{T}}(\omega) = \int_{[0,1]^n} L_{\mathcal{T}} \, \mathrm{d}x.$$

Now, remember that the naive procedure gives the only perfect tree. It suffices to show that unbalancing this tree in any way results in a longer sum in the equation above. Indeed, to unbalance the tree one needs to:

- Remove two bottom-level leaves, turning their root node into a leaf
- Turn one bottom-level leaf into a node
- Attach two nodes to this newly-created leaf

The total impact on the sum of lengths is $+1$. Hence the naive algorithm is minimal at $\{\frac{1}{2}, \ldots, \frac{1}{2}\}$, and therefore, in the region $\{\forall i \in [n], \frac{1}{2} \leq x_i \leq 1\}$. □

Remark *This also shows that if we assume that the probabilities are supposed uniform (i.e. we assume no a priori knowledge), the optimal procedure is the naive one. Therefore we can see that the gain for* $n = 3$ *is approximately* $0,34$ *since the optimal procedure in average gives* $2,66$*. In percentage the gain is* 15%*. If the probabilities are very low, we have a gain of almost* 2 *which is* 3 *times faster. As expected it is much more interesting if we think that the signatures have a good chance to be correct, which is the case in most real-life scenarii.*

Lemma 3.5 *If the root has a test of cardinality one, then the same algorithms starting at both sons have same expected stopping time. This applies if the next test is also of cardinal one.*

Proof: Without loss of generality we can assume that the test is $\{1\}$. We have $\{0,1\}^{n^{\top}}_{\{1\}} = \{0b_2 \cdots b_n | b_2 \cdots b_n \in \{0,1\}^{n-1}\}$ and $\{0,1\}^{n^{\perp}}_{\{1\}} = \{1b_2 \cdots b_n | b_2 \cdots b_n \in \{0,1\}^{n-1}\}$. A test T that doesn't test 1 applied on those sets will give the same split for both, and the probability that the test answers yes or no is the same. This is also true for the sets and the tests T such that i is not in T for i in $\{1, \ldots, k\}$. $\{0^k b_2 \cdots b_n | b_2 \cdots b_n \in \{0,1\}^{n-k}\}$ and $\{01^k b_2 \cdots b_n | b_2 \cdots b_n \in \{0,1\}^{n-k}\}$. A test T such that there exists i in T $\{1, \ldots, k\}$ brings no information for the set of possibilities $\{01^k b_2 \cdots b_n | b_2 \cdots b_n \in \{0,1\}^{n-k}\}$, but testing this i is useless for the set $\{0^k b_2 \cdots b_n | b_2 \cdots b_n \in \{0,1\}^{n-k}\}$. So we can apply the test $T - \{1, \ldots, k\}$. □

Corollary 3.6 *If the root has a test of cardinal one, then an optimal algorithm can always apply the same test for the right and the left child. If this test is also of cardinal one then the property is still true.*

This result helps in identifying redundant descriptions of testing procedures, and can be used to narrow down the generation, by skipping over obvious symmetries as they appear in the naive procedure (see Figure 3.8).

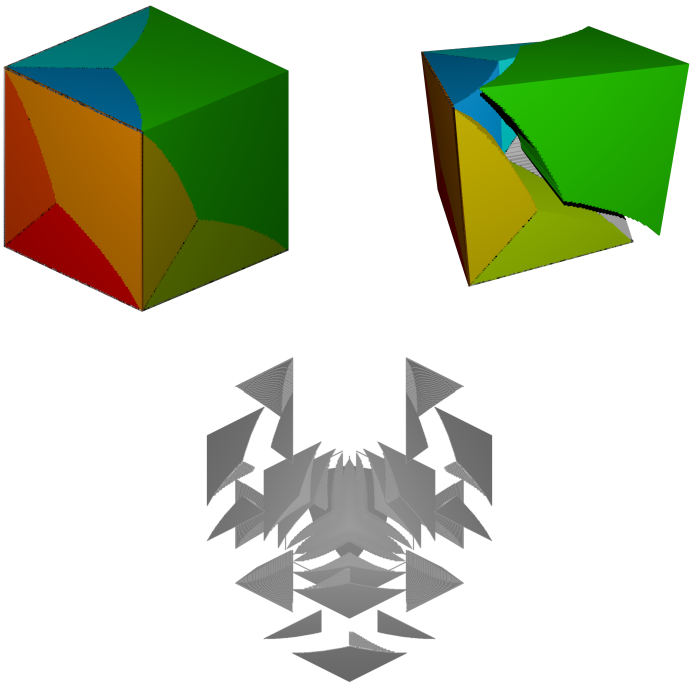

Figure 3.7: A 3D visualisation of the cube. Left: exterior, where it is visible that each face has the same decomposition as the 2D problem; Middle: with the naive algorithm region slightly removed, showing that it accounts for slightly less than half of the total volume; Right: exploded view of the 52 substructures (looking from $(-1, -1, -1)$).

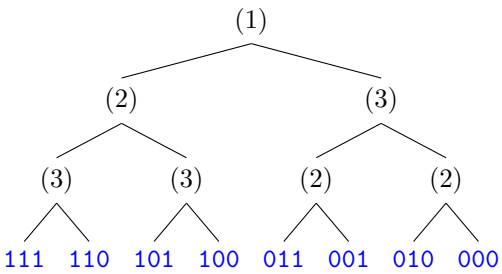

Figure 3.8: Naive algorithm, where the order of tests are unimportant in the left and right branches.

Lemma 3.7 *If a node labelled T_1 has two children that are* both *labelled T_2, then we can interchange T_1 and T_2 without changing the testing procedure's expected length.*

Yet another simple observation allows to reduce the set of subsets T at each step:

Lemma 3.8 *Consider a node labelled $(T, \mathcal{S})$. Assume that there is $i \in [n]$ such that, for all S in $\mathcal{S}$, $i \notin S$. Then we can replace T by $T \cup \{i\}$.*

Proof: We can easily see that $S_T^\top = S_{T\cup\{i\}}^\top$ and $S_T^\perp = S_{T\cup\{i\}}^\perp$. □

Finally we can leverage the fact that the solutions exhibit symmetries, which provides both a compact encoding of testing procedures and an appreciable reduction in problem size.

Lemma 3.9 *Let $\sigma \in \mathfrak{S}_n$ be a permutation on n elements. If we apply σ to each node and leaf of $\mathcal{T}$, which we can write $\sigma(\mathcal{T})$, then*

$$L_{\sigma(\mathcal{T})}(x_1, \ldots, x_n) = L_{\mathcal{T}}\left(\sigma\left(x_1, \ldots, x_n\right)\right).$$

Proof: Note that for any $S \in \Omega$ and $T \in \Omega \setminus \{\emptyset\}$ we have $\sigma\left(S_T^\top\right) = S_{\sigma(T)}^\top$ and $\sigma\left(S_T^\perp\right) = S_{\sigma(T)}^\perp$, where σ operates on each binary string. It follows that for any leaf S, $\ell_{\mathcal{T}}(S)$ becomes $\ell_{\mathcal{T}}(\sigma(S))$ under the action of σ, hence the result. □

Lemma 3.10 *Let S be a simplex of the hypercube, $\mathcal{T}$ a procedure, $E = \{\sigma(\mathcal{T}) | \sigma \in \mathfrak{S}_n\}$, then there exists $\mathcal{T}_0$ in E, such that for all x in S, $\mathcal{T}_1$ in E we have*

$$L_{\mathcal{T}_0}(x) \leq L_{\mathcal{T}_1}(x).$$

Moreover we have for all σ in $\mathfrak{S}_n$, x in $\sigma(S)$, $\mathcal{T}_1$ in E

$$L_{\sigma(\mathcal{T}_0)}(x) \leq L_{\mathcal{T}_1}(x).$$

Remark *The last two propositions allow us to solve the problem on a simplex of the hypercube (of volume $1/n!$) such as $\{p_1, \ldots, p_n \mid 1 \geq p_1 \cdots \geq p_n \geq 0\}$.*

3.2.6 Approximation heuristics

The approach consisting in generating many candidates, only to select a few, is wasteful. In fact, for large values of n (even from 10), generating all the candidates is beyond reach, despite the optimizations we described.

Instead, one would like to obtain the optimal testing procedure *directly*. It is a somewhat simpler problem, and we can find the solution by improving on our generation-then-selection algorithm (see Section 3.2.8). However, if we wish to address larger values of n, we must relax the constraints and use the heuristic algorithms described below, which achieve near-optimal results. This would be useful in real life scenarii for signatures verifications since we would like to verify hundreds or more signatures to have real gain.

3.2.6.1 Information-Based Heuristic

We first associate a 'cost' to each outcome S, and set of outcomes $\mathcal{S}$:

$$\begin{aligned}\text{cost}(S,\mathcal{S}) &= f(S,\mathcal{S}) + g(S,\mathcal{S})\\ f(S,\mathcal{S}) &= \#\{i \in [n] \text{ s.t. } s[i]=1 \text{ and } \exists S \in \mathcal{S}, S'[i]=0\}\\ g(S,\mathcal{S}) &= \begin{cases}1 & \text{if } \exists i \in \{i \in [n] \text{ s.t. } S[i]=0\}, \exists S' \in \mathcal{S}, S'[i]=1\\ 0 & \text{otherwise}\end{cases}\end{aligned}$$

This function approximates the smallest integer n such that there exists n calls to ϕ with arguments $T_1,\ldots,T_n$, and $\beta_1,\ldots,\beta_n$ in $\{\bot,\top\}$ with $\mathcal{S}_{T_1\ldots T_n}^{\beta_1,\ldots,\beta_n} = \{S\}$. This function is used to define a 'gain' function evaluating how much information is gathered when performing a test knowing the set of outcomes:

$$\text{gain}(T,\mathcal{S}) = \sum_{S\in\mathcal{S}_T^\top}\left(1-\frac{\text{cost}(S,\mathcal{S}_T^\top)}{\text{cost}(S,\mathcal{S})}\right)\Pr(S) + \sum_{S\in\mathcal{S}_T^\bot}\left(1-\frac{\text{cost}(S,\mathcal{S}_T^\bot)}{\text{cost}(S,S)}\right)\Pr(S)$$

Intuitively, we give higher gains to subsets T on which testing gives more information. Note that, if a call to ϕ doesn't give any information (i.e. $S_T^\top$ or $S_T^\bot$ is empty), then $\text{gain}(T,S)=0$.

This heuristic provides us with a greedy algorithm that is straightforward to implement. For given values $x_1,\ldots,x_n$ we thus obtain a testing procedure $\mathcal{T}_H$.

Testing the heuristic. We compared numerically $\mathcal{T}_H$ to the metaprocedure found by exhaustion in the case $n=3$. The comparison consists of sampling points at random and computing the sample mean of each algorithm's length on this input. The heuristic procedure gives a mean of 2.666, which underperform the optimal procedure (2.661) by only 1%.

Counter-example to optimality. In some cases, the heuristic procedure behaves very differently from the metaprocedure. For instance, for $n=3$, $x_1=0.01$, $x_2=0.17$, $x_3=0.51$, the metaprocedure yields a tree which has an expected length of 1.889. The heuristic however, produces a tree which has expected length 1.96. Both trees are represented in Figure 3.9.

Beyond their different lengths, the main difference between the two procedures of Figure 3.9 begin at the third node. At that node the set S is the same, namely $\{010,011,100,101,110,111\}$, but the two procedures settle for a different T: The metaprocedure splits S, with $T=\{1,3\}$, into $S_T^\bot=\{010\}$ and $S_T^\top=\{011,100,101,110,111\}$; while the heuristic chooses $T=\{1\}$ instead, and gets $S_T^\bot=\{010,011\}$ and $S_T^\top=\{100,101,110,111\}$.

To understand this difference, first notice that besides 010 and 011, all leaves are associated with a very low probability. The heuristic fails to capture that by choosing $T=\{1,3\}$ early; it could later rule out the leaf 010 in one step and 011 in two. There does not seem to be a simple greedy way to detect this early on.

3.2.6.2 Pairing heuristic

Another approach is to use small metaprocedures on subsets of the complete problem. Concretely, given n objects to test, place them at random into k-tuples (from some small value k, e.g. 5). Then apply the k-metaprocedure on these tuples. While sub-optimal, this approach does not yield worst results than the naive procedure.

In cases where it makes sense to assume that all the x_i are equal, then we may even recursively use the metaprocedures, i.e. the metaprocedures to be run are themselves places into k-tuples, etc. By using lazy evaluation, only the necessary tests are performed.

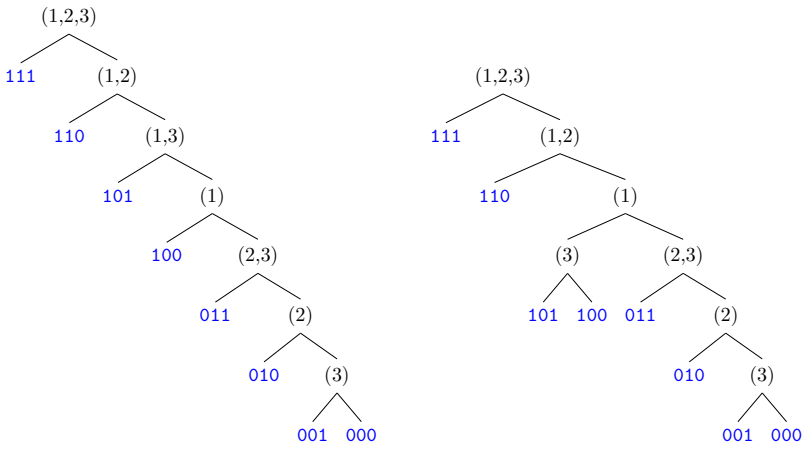

Figure 3.9: The optimal metaprocedure tree (left), and heuristic metaprocedure (right) for the same point $x = (0.01, 0.17, 0.51)$. The optimal procedure has expected length 1.889, as compared to 1.96 for the heuristic procedure.

3.2.7 Equivalences and symmetries for $n = 3$

A procedure can undergo a transformation that leaves its expected length unchanged. Such transformations are called *equivalences*. On the other hand, Lemma 3.9 shows that some transformations operate a permutation σ on the variables x_i — such transformations are called *symmetries*.

Equivalences and symmetries are responsible for a large part of the combinatorial explosion observed when generating all procedures. By focusing on procedures up to symmetry, we can thus describe the complete set in a more compact way and attempt a first classification.

In the following representations (Figures 3.10 to 3.12), blue indicates fixed parts, and red indicates parts undergoing some permutation. Double-headed arrows indicate that swapping nodes is possible. The number of symmetries obtained by such an operation is indicated under the curly brace below.

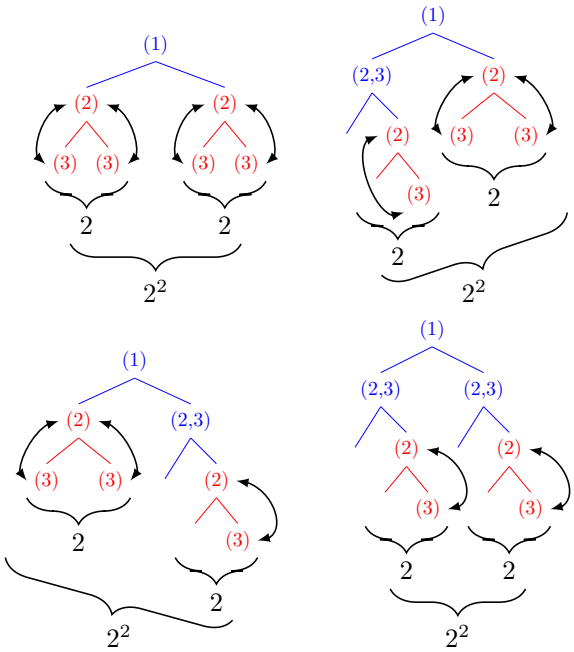

Figure 3.10: Trees representation with a grouping by one element on the root. For a fixed element, we have 2^2 possible permutations. Since we have 4 patterns, we get $2^2 \times 4$ possible permutations for one grouping. Hence, we finally have $2^2 \times 4 \times 3$ for all possible groupings by one element.

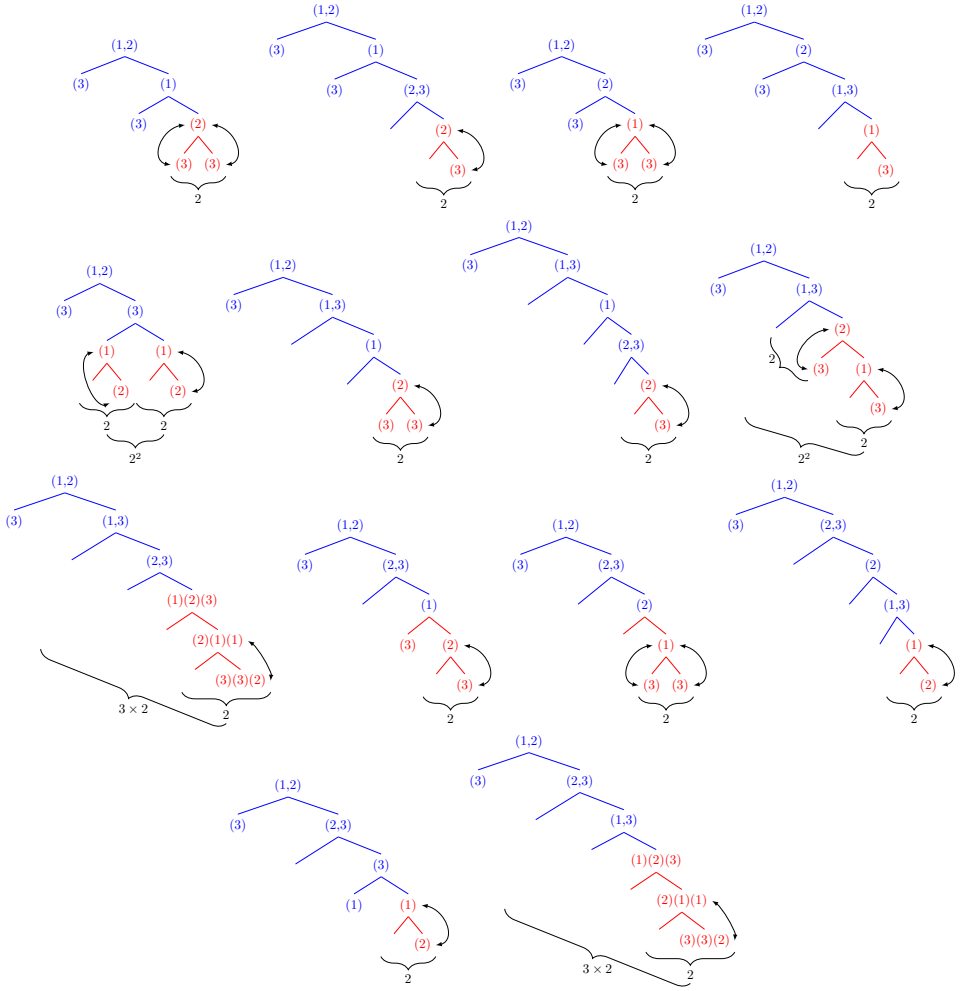

Figure 3.11: Tree representations with a grouping by two elements on the root. For 10 fixed elements, we have 2 possible permutations, for 2 fixed elements, we have 2 possible permutations, and for 2 possible permutations, we have 6 possible permutations. Hence, we finally have $2 \times 10 + 4 \times 2 + 6 \times 2$ for all possible groupings by two elements.

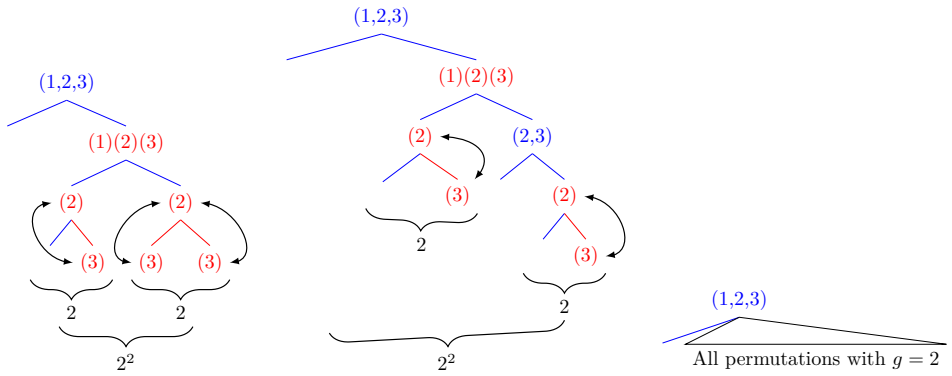

Figure 3.12: Trees representation with a grouping by three elements on the root. For a fixed element at the upper left corner side, we have 2^2 possible permutations. For the upper right corner side, we get 2^2. We replace the sub-root of the fixed trees and get $(2^2 + 2^2) \times 3$. We also have the 40×3 trees from the grouping of two. Hence, we have $40 \times 3 + (2^2 + 2^2) \times 3$

3.2.8 Best testing procedure at a point

We examine the following problem: Find the testing procedure $\mathcal{T}$ for a given $k \leq n$, $(p_{i_1}, \ldots, p_{i_k}) \in [0,1]^n$, and a selection $P \subseteq 2^{[k]}$ that satisfies:

- $\mathcal{S}_\mathcal{T} = P$,
- $\mathcal{T}$ is optimal at point $(p_{i_1}, \ldots, p_{i_k})$

This can be computed using a dynamic programming technique, by examining the outcome of each possible test that is the root node of the testing procedure $\mathcal{T}$. This approach gives Algorithm 2.

The same dynamic programming algorithm can also be used to compute the number of testing procedures (including those leading to duplicate polynomials) that exist in a given dimension. It is actually even easier (meaning that we can apply the algorithm to an even higher dimension than our solution to the given point problem) since there is a huge number of symmetries that can be exploited to count.

We will introduce the following definition, in use in our algorithm:

Definition 3.7 (Decided point) *We say that x is a* decided point *for $\mathcal{S}$ a set of selections if either of the following is true:*

- *$x \in S$ for all $S \in \mathcal{S}$*
- *$x \notin S$ for all $S \in \mathcal{S}$*

In the first case, we will say that x is a positive decided point *and a* negative decided point *in the second case.*

We denote by $\mathcal{D}^+_\mathcal{S}$ the set of positive decided points of $\mathcal{S}$, $\mathcal{D}^-_\mathcal{S}$ its set of negative decided points, and $\mathcal{D}_\mathcal{S} = \mathcal{D}^+_\mathcal{S} \cup \mathcal{D}^-_\mathcal{S}$ its set of decided points.

Algorithm 2: FindOptimal

Input: $k \geq 0$, $(p_1, \ldots, p_k) \in [0,1]^k$, $\mathcal{S} \subset 2^{[k]}$.
Output: The optimal testing procedure $\mathcal{T}$ at point $(p_1, \ldots, p_k)$ which satisfies $\mathcal{S}_\mathcal{T} = \mathcal{S}$.

1. if $k == 0$ then return the naive algorithm
2. if $|\mathcal{D}_\mathcal{S}| > 0$
3. $\quad U \leftarrow \{u_1, \ldots, u_\ell\} = [k] \setminus \mathcal{D}_\mathcal{S}$
4. $\quad \mathcal{R} \leftarrow \{\{r_1, \ldots, r_p\} \mid \{u_{r_1}, \ldots, u_{r_p}\} \cup \mathcal{D}^+_\mathcal{S}\}$
5. $\quad \mathcal{T} \leftarrow \text{FindOptimal}\,(\ell, (p_{u_1}, \ldots, p_{u_\ell}), \mathcal{R})$
6. $\quad$ replace $\{t_1, \ldots, t_r\}$ by $\{u_{t_1}, \ldots, u_{t_r}\}$ in $\mathcal{T}$
7. $\quad$ replace $\{\ell_1, \ldots, \ell_r\}$ by $\{u_{\ell_1}, \ldots, u_{\ell_r}\} \cup \mathcal{D}^+_\mathcal{S}$ in $\mathcal{T}$
8. else
9. $\quad W \leftarrow \emptyset$
10. $\quad$ for each $T \subseteq [k]$
11. $\quad\quad \mathcal{S}_\perp \leftarrow \mathcal{S}^\perp_T$
12. $\quad\quad \mathcal{S}_\top \leftarrow \mathcal{S}^\top_T$
13. $\quad\quad$ if $S_\perp = \emptyset$ or $S_\top = \emptyset$ then continue
14. $\quad\quad \mathcal{T}_\perp \leftarrow \text{FindOptimal}(k, (p_1, \ldots, p_k), \mathcal{S}_\perp)$
15. $\quad\quad \mathcal{T}_\top \leftarrow \text{FindOptimal}(k, (p_1, \ldots, p_k), \mathcal{S}_\top)$
16. $\quad\quad W \leftarrow W \cup \{(\mathcal{T}, \mathcal{T}_\perp, \mathcal{T}_\top)\}$
17. $\quad$ return the best algorithm in W at point $(p_1, \ldots, p_n)$

Counting the number of algorithms in a given dimension works the same way; the only difference is that there is no need to look at the probabilities, and thus, the resulting Algorithm 3 does fewer recursive calls and is faster. We are not aware of a closed-form formula providing the same values as this algorithm.

Algorithm 3: CountAlgorithms

Input: $k \geq 0$, $\mathcal{S} \subset 2^{[k]}$.
Output: The number of testing procedures which satisfy $\mathcal{S}_{\mathcal{T}} = \mathcal{S}$.

1. if $k == 0$ then return 1
2. if $|\mathcal{D}_{\mathcal{S}}| > 0$
3. $\quad U \leftarrow \{u_1, \ldots, u_\ell\} = [k] \setminus \mathcal{D}_{\mathcal{S}}$
4. $\quad \mathcal{R} = \{\{r_1, \ldots, r_p\} \mid \{u_{r_1}, \ldots, u_{r_p}\} \cup \mathcal{D}_{\mathcal{S}}^{+}\}$
5. $\quad$ return CountAlgorithms$(\ell, \mathcal{R})$
6. $c \leftarrow 0$
7. for each $T \subseteq [k]$
8. $\quad \mathcal{S}_{\perp} \leftarrow \mathcal{S}_T^{\perp}$
9. $\quad \mathcal{S}_{\top} \leftarrow \mathcal{S}_T^{\top}$
10. $\quad$ if $S_{\perp} = \emptyset$ or $S_{\top} = \emptyset$ then continue
11. $\quad c_{\perp} \leftarrow$ CountAlgorithms$(k, (p_1, \ldots, p_k), \mathcal{S}_{\perp})$
12. $\quad c_{\top} \leftarrow$ CountAlgorithms$(k, (p_1, \ldots, p_k), \mathcal{S}_{\top})$
13. $\quad c \leftarrow c + c_{\top} c_{\perp}$
14. return c

3.2.9 Enumerating procedures for $n = 3$

All the procedures for $n = 3$ that are optimal at some point, up to symmetries, are represented in Figure 3.13.

3.2.10 Conclusion and open questions

We have introduced the question of optimal batch verification with a priori probabilities, where one is given a set of signatures and must determine in the least average number of operations which signatures are correct, and which are not. We formalized this problem and pointed out several interesting combinatorial and algebraic properties that speed up the computation of an optimal sequence of operations — which we call a *metaprocedure*. We determined the exact solution for up to 4 objects.

For larger values, our approach requires too many computations to be tractable, and thus an exact solution is out of reach; however, we gave several heuristic algorithms that scale well. We showed that these heuristics are sub-optimal in all cases, but they always do better than standard screening. The existence of a polynomial-time algorithm that finds optimal metaprocedures for large values of n is an open question — although there is probably more hope in finding better heuristics. An alternative would be to modify our generation algorithm to kill branches when the resulting expected lengths are all worse than some already-known procedure.

Once the metaprocedure for a given n is known, which only needs to be computed once, implementation is straightforward and only invokes a handful of (automatically generated) cases. Besides the performance gain resulting from implementing metaprocedures for signature verification, the very general framework allows for applications in medical and engineering tests.

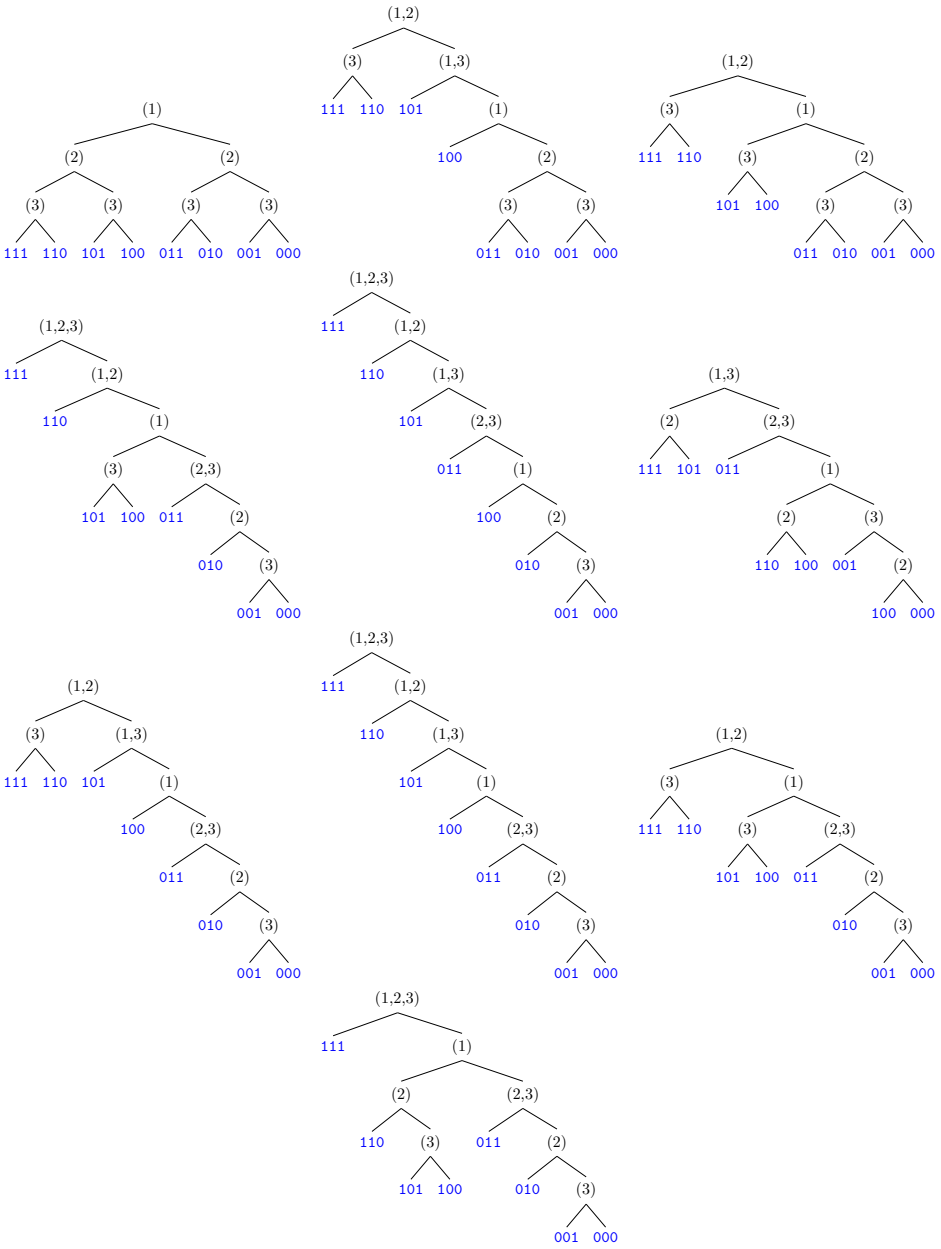

Figure 3.13: Optimal procedures (without permutations) for each zone when $n = 3$.

3.3 Reusing Nonces in Schnorr Signatures

Abstract

The provably secure Schnorr signature scheme is popular and efficient. However, each signature requires a fresh modular exponentiation, which is typically a costly operation. As the increased uptake in connected devices revives the interest in resource-constrained signature algorithms, we introduce a variant of Schnorr signatures that mutualises exponentiation efforts.

Combined with precomputation techniques (which would not yield as interesting results for the original Schnorr algorithm), we can amortise the cost of exponentiation over several signatures: these signatures share the same nonce. Sharing a nonce is a deadly blow to Schnorr signatures, but is not a security concern for our variant.

Our scheme is provably secure, asymptotically-faster than Schnorr when combined with efficient precomputation techniques, and experimentally 2 to 6 times faster than Schnorr for the same number of signatures when using 1 MB of static storage.

This is joint work with Aisling Connolly, Rémi Géraud, David Naccache, and Damien Vergnaud. This work was presented at the 22nd European Symposium on Research in Computer Security, ESORICS 2017, Oslo (Norway) and published as [BCF+17].

3.3.1 Introduction

The increased popularity of lightweight implementations invigorates the interest in resource-preserving protocols. Interestingly, this line of research was popular in the late 1980's, when smart-cards started performing public-key cryptographic operations (e.g. [FS87]). Back then, cryptoprocessors were expensive and cumbersome, and the research community started looking for astute ways to identify and sign with scarce resources.

In this work, we revisit a popular signature algorithm published by Schnorr in 1989 [Sch90] and seek to lower its computational requirements assuming that the signer is permitted to maintain some read-only memory. This storage allows for time-memory trade-offs, which are usually not very profitable for typical Schnorr parameters.

We introduce a new signature scheme, which is provably secure in the random oracle model (ROM) under the assumption that the *partial discrete logarithm problem* (see below) is intractable. This scheme can benefit much more from precomputation techniques, which results in faster signatures.

Implementation results confirm the benefits of this approach when combined with efficient precomputation techniques and enough static memory is available (of the order of 250 couples of the form (x, g^x)). We provide comparisons with Schnorr for several parameters and pre-computation schemes.

3.3.1.1 Intuition and general outline of the idea

Schnorr's signature algorithm uses a large prime modulus p and a smaller prime modulus q dividing $p-1$. The security of the signature scheme relies on the discrete logarithm problem in a subgroup of order q of the multiplicative group of the finite field $\mathbb{Z}_p$ (with $q \mid p-1$). Usually, the prime p is chosen to be large enough to resist index-calculus methods for solving the discrete-log problem (*e.g.* 3072 bits for a 128-bit security level), while q is large enough to resist the square-root algorithms [Sha71] (e.g. 256 bits for 128-bit security level).

The intuition behind our construction is to consider a prime p such that $p-1$ has *several* different factors q_i large enough to resist these birthday attacks, i.e.

$$p = 1 + 2\prod_{i=1}^{\ell} q_i$$

then several "orthogonal" Schnorr signatures can share the same commitment component $r = g^k \bmod p$. This is not the case with standard Schnorr signatures where, if a k is reused then the secret signing key is revealed.

It remains to find how r can be computed quickly. In the original Schnorr protocol k is picked uniformly at random in $\mathbb{Z}_q$. However, to be secure, our construction requires that k is picked in the larger set $\mathbb{Z}_{p-1}$. which means that a much higher effort is required to compute r. Here we cut corners by generating an r with precomputation techniques, which allow an exponentiation to be sub-linear. The trick is that once the exponentiation is sub-linear, we are more effective in our setting than in the original Schnorr setting.

We start by reminding how the original Schnorr signature scheme works and explain how we extend it assuming that k is randomly drawn from $\mathbb{Z}_{p-1}$. We then present applications of our construction, by comparing several pre-processing schemes.

3.3.2 Preliminaries

We denote the security parameter by $\kappa \in \mathbb{N}$ which is given to all algorithms in the unary form 1^κ. Algorithms are randomized unless otherwise stated, and PPT stands for "probabilistic polynomial-time," in the security parameter. We denote random sampling from a finite set X according to the uniform distribution with $x \xleftarrow{\$} X$. We also use the symbol $\xleftarrow{\$}$ for assignments from randomized algorithms, while we denote assignment from deterministic algorithms and calculations with the symbol $\leftarrow$. If n is an integer, we write $\mathbb{Z}_n$ for the ring $\mathbb{Z}/n\mathbb{Z}$. We let $\mathbb{Z}_n^*$ the invertible elements of $\mathbb{Z}_n$. As is usual, $f \in \mathsf{negl}(\kappa)$ denotes a function that decreases faster than the inverse of any polynomial in κ; such functions are called negligible. The set of numbers $1, 2, \ldots, k$ is denoted $[k]$. Most of our security definitions and proofs use code-based games. A game G consists of an initializing procedure Init, one or more procedures to respond to oracle queries, and a finalizing procedure Fin.

3.3.2.1 Schnorr's signature scheme

Schnorr signatures [Sch90] are an offspring ElGamal signatures [ElG86] which are provably secure in the Random Oracle Model under the assumed hardness of solving generic instances of the Discrete Logarithm Problem (DLP) [PS96]. The Schnorr signature scheme is a tuple of algorithms defined as follows:

- $\mathsf{Setup}(1^\kappa)$: Large primes p, q are chosen, such that $q \geq 2^\kappa$ and $p - 1 = 0 \bmod q$. A cyclic group $\mathbb{G} \subset \mathbb{Z}_p$ of prime order q is chosen, in which it is assumed that the DLP is hard, along with a generator $g \in \mathbb{G}$. A hash function $H : \{0,1\}^* \to \mathbb{G}$ is chosen. Public parameters are $\mathsf{pp} = (p, q, g, \mathbb{G}, H)$.

- $\mathsf{KeyGen}(\mathsf{pp})$: Pick an integer x uniformly at random from $[2, q-1]$ as the signing key sk, and publish $y \leftarrow g^x$ as the public key pk.

- $\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, m)$: Pick k uniformly at random in $\mathbb{Z}_q^*$, compute $r \leftarrow g^k \bmod q$, $e \leftarrow H(m, r)$, and $s \leftarrow k - ex \bmod q$. Output $\sigma \leftarrow \{r, s\}$ as a signature.

- $\mathsf{Verify}(\mathsf{pp}, \mathsf{pk}, m, \sigma)$: Let $(r, s) \leftarrow \sigma$, compute $e \leftarrow H(m, r)$ and return True if $g^s y^e = r$, and False otherwise.

3.3.2.2 Security model

We recall the strong[2] EF-CMA security notion:

Definition 3.8 (Strong EF-CMA Security) *A signature scheme Σ is* secure against existential forgeries in a chosen-message attack *(strongly* EF-CMA*-secure) if the advantage of any PPT adversary $\mathcal{A}$ against the* EF-CMA *game defined in Figure 3.14 is negligible:* $\mathsf{Adv}^{\mathsf{EF}}_{\mathcal{A},\Sigma}(\kappa) = \Pr\left[\mathsf{EF}^{\mathcal{A}}_{\Sigma}(\kappa) = 1\right] \in \mathsf{negl}(\kappa)$.

$\underline{\mathsf{EF}^{\mathcal{A}}_{\Sigma}(\kappa)}$:
$L \leftarrow \emptyset$
$(\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \Sigma.\mathsf{KeyGen}(1^\kappa)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot),\mathsf{Verify}(\cdot,\cdot),H(\cdot)}(1^\kappa)$
if $(m^*, \sigma^*) \notin L$
  return $\Sigma.\mathsf{Verify}(\mathsf{pk}, m^*)$
return 0

$\underline{\mathsf{Sign}(m)}$:
$\sigma \xleftarrow{\$} \Sigma.\mathsf{Sign}(\mathsf{sk}, m)$
$L \leftarrow L \cup \{m, \sigma\}$
return σ

$\underline{\mathsf{Verify}(m, \sigma)}$:
return $\Sigma.\mathsf{Verify}(\mathsf{pk}, m, \sigma)$

Figure 3.14: The strong EF-CMA experiment for digital signature schemes.

3.3.3 Using multiple q's

Our construction relies on using a prime p of the form mentioned in the introduction. This is not a trivial change and requires care as we discuss below.

Technically, our construction is a *stateful* signature scheme (see e.g. [KL07, Chapter 12]), in which we simultaneously sign only one message and keep a state corresponding to the values k, g^k and the index i for the current prime number. However, it is more compact and convenient to describe it as a signature for ℓ simultaneous messages.

3.3.3.1 Our scheme

Similar to the Schnorr signature scheme, our scheme is a tuple of algorithms (Setup, KeyGen, Sign, and Verify), which we define as follows:

- $\mathsf{Setup}(1^\kappa)$: Generate ℓ primes $q_1, \ldots, q_\ell$ of size $\geq 2^\kappa$ and ℓ groups $\mathbb{G}_1, \ldots, \mathbb{G}_\ell$ respectively of order $q_1, \ldots q_\ell$ such that the DLP is hard in the respective $\mathbb{G}_i$, and such that $p = 1 + 2\prod q_i$ is prime. This is easily achieved by selecting $(\ell - 1)$ primes q_i and varying the last one until p is prime.[3] Choose a cryptographic hash function $H : \{0,1\}^* \to \{0,1\}^{q_1}$. The hash function will be used to produce elements of $\mathbb{Z}_{q_i}$. For this we will denote by H_i the composition of H and a conversion function from $\{0,1\}^{q_1}$ to $\mathbb{Z}_{q_i}$.[4] Finally, choose g a generator of the group $\mathbb{Z}_p^*$ of order $p - 1$. The public parameters are therefore

$$\mathsf{pp} = \left(p, \{q_i\}_{i=1}^{\ell}, H, g, \{G_i\}_{i=1}^{\ell}\right).$$

[2] In contrast to the *weak* version, the adversary is allowed to forge for a message that they have queried before, provided that their forgery is *not* an oracle response.

[3] See Section 3.3.8 for a discussion on some particularly interesting moduli.

[4] This conversion function can read the string as a binary number and reduce it $\bmod q_i$ for example.

- $\mathsf{KeyGen}(\mathsf{pp})$: The signer chooses $x \xleftarrow{\$} \mathbb{Z}^*_{p-1}$ and computes $y \leftarrow g^x \bmod p$. The key $\mathsf{sk} = x$ is kept private to the signer, while the verification key $\mathsf{pk} = y$ is made public.

- $\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, m_1, \ldots, m_\ell)$: The signer chooses $k \xleftarrow{\$} \mathbb{Z}_p$, such that $k \neq 0 \bmod q_i$ for all i, and computes $r \leftarrow g^k \bmod p$.
  The signer can now sign the ℓ messages m_i as:

$$\rho_i \xleftarrow{\$} \{0,1\}^\kappa, \quad e_i \leftarrow H_i(m_i, r, \rho_i), \quad \text{and} \quad s_i \leftarrow k - e_i x \bmod q_i$$

  outputting the ℓ signatures $\sigma_i = \{r, s_i, \rho_i\}$—or, in a more compact form,

$$\sigma = \{r, s_1, \ldots, s_\ell, \rho_1, \ldots, \rho_\ell\}.$$

- $\mathsf{Verify}(\mathsf{pp}, \mathsf{pk}, m_i, (r, s_i, \rho_i), i)$: Verifying a signature is achieved by slightly modifying the original Schnorr scheme: First check that $s_i \in \{0, \ldots q_i - 1\}$ and compute $e_i \leftarrow H_i(m_i, r, \rho_i)$, then observe that for a correct signature[5]:

$$(g^{s_i} y^{e_i})^{\frac{p-1}{q_i}} = r^{\frac{p-1}{q_i}} \bmod p.$$

  The signature is valid if and only if this equality holds, otherwise the signature is invalid (see Lemma 3.11).

Remark *Note that unlike Schnorr, in the* Sign *algorithm we add a random ρ_i for a signature to make the argument of the hash function unpredictable. This will be useful for the proof of Theorem 3.12 in the ROM.*

Remark *Note also that one almost recovers the original Schnorr construction for $\ell = 1$—the only differences being in the verification formula, where both sides are squared in our version, and the addition of a fresh random to hash.*

Lemma 3.11 (Correctness) *Our scheme is correct.*

Proof: Let g, y, r, s_i, and ρ_i be as generated by the KeyGen and Sign algorithms for a given message m_i. We check that,

$$\left(\frac{(g^{s_i} y^{e_i})^{s_i}}{r}\right)^{\frac{p-1}{q_i}} = 1 \bmod p.$$

By the definition of s_i, there exists $\lambda \in \mathbb{Z}$ such that $g^{s_i} = g^{k - e_i x + \lambda q_i}$, hence

$$g^{s_i} y^{e_i} g^{-k} = g^{\lambda q_i} \bmod p.$$

Raising this to the power of $\frac{p-1}{q_i}$ we get $g^{\lambda(p-1)} = 1$ since the order the multiplicative group $\mathbb{Z}^*_p$ is $p - 1$. □

[5] One can note, $\frac{p-1}{q_i} = 2q_1 \cdots q_{i-1} q_{i+1} \cdots q_\ell$.

3.3.3.2 Security

To aid in the proof of security, we introduce the following problem which we call the partial discrete logarithm problem (PDLP). Intuitively it corresponds to solving a discrete logarithm problem in the subgroup of our choice.

Definition 3.9 (PDLP) *Let $\ell \geq 2$ be an integer, $q_1, \ldots, q_\ell$ distinct prime numbers and $q = q_1 \ldots q_\ell$. Let $\mathbb{G}$ be a group of order q and g a generator of $\mathbb{G}$. Given $g, q, q_1, \ldots, q_l$, and $y = g^x$, the* partial discrete logarithm problem *(PDLP) consists in finding $i \in [\ell]$ and $x_i \in \mathbb{Z}_{q_i}$ such that $x_i = x \bmod q_i$.*

In our context, we are chiefly interested in a subgroup of order q of a multiplicative group of a finite field $\mathbb{Z}_p^*$, where q divides $p-1$—ideally, $q = (p-1)/2$. The best known algorithms to solve the PDLP are index-calculus based methods in $\mathbb{Z}_p^*$ and square-root algorithms in subgroups of prime order q_i for some $i \in [\ell]$. With p of bit-size 3072, $q = (p-1)/2$, $\ell = 12$ and $q_1, \ldots, q_\ell$ of bit-size 256, we conjecture that solving the PDLP requires about 2^{128} elementary operations. In Section 3.3.4, we provide security argument in the generic group model on the intractability of the PDLP for large enough prime numbers $q_1, \ldots, q_\ell$.

Theorem 3.12 (Existential unforgeability) *Our scheme is provably* EF-CMA*-secure assuming the hardness of solving the PDLP, in the ROM.*

To prove this result, we will exhibit a reduction from an efficient EF-CMA forger to an efficient PDLP solver. To that end, we first show a sequence of indistinguishability results between the output distributions of

- Our signature algorithm $\mathsf{Sign} = \mathsf{Sign}_0$ on user inputs.
- A modified algorithm Sign_1 (see Figure 3.15), where the hash of user inputs is replaced by a random value. This situation is computationally indistinguishable from the previous one in the ROM.
- A modified algorithm Sign_2 (see Figure 3.15), that has no access to the signing key x. The output distribution of this algorithm is identical to the output of Sign_1 (Theorem 3.13).

Then we use the forking lemma [PS00, BN06] to show that an efficient EF-CMA-adversary against Sign_2 can be used to construct an efficient PDLP solver. Finally, we leverage the above series of indistinguishability results to use an adversary against Sign_0. Let CRT (for Chinese Remainder Theorem) be the isomorphism that maps $\mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_\ell} \times \mathbb{Z}_2$ to $\mathbb{Z}_{p-1}$.

Theorem 3.13 *The output distributions of Sign_1 and Sign_2 are identical.*

Proof: This theorem builds on several intermediate results described in Lemmata 3.14 to 3.18. We denote δ the output distribution of Sign_1 and δ' the output distribution of Sign_2. The structure of the proof is the following :

- In Lemma 3.14 we show that the output of Sign_2 is a subset of the output of Sign_1.
- Lemma 3.15 shows that in Sign_1 there is a unique random tape per output.
- Lemma 3.16 shows that in Sign_2 there are exactly two random tapes per output.
- Lemma 3.18 shows that there are twice as many random tapes possible for Sign_2 than for Sign_1

$\underline{\mathsf{Sign}_1}$:
$\rho \xleftarrow{\$} \{0,1\}^\kappa$
$k \xleftarrow{\$} \mathbb{Z}_p \setminus \left(\bigcup_{i=1}^{\ell} \{q_i, 2q_i, \ldots, p-1\} \right)$
$r \leftarrow g^k \bmod p$
for $i = 1$ to ℓ
    $e_i \xleftarrow{\$} \mathbb{Z}_{q_i}$
    $s_i \leftarrow k - e_i x \bmod q_i$
    $\rho_i \xleftarrow{\$} \{0,1\}^\kappa$
end for
return $(r, e_1, \ldots, e_\ell, s_1, \ldots, s_\ell, \rho_1 \ldots, \rho_\ell)$

$\underline{\mathsf{Sign}_2}$:
for $i = 1$ to ℓ
    $e_i \xleftarrow{\$} \mathbb{Z}_{q_i}$
    $s_i \xleftarrow{\$} \mathbb{Z}_{q_i}$
    $\rho_i \xleftarrow{\$} \{0,1\}^\kappa$
end for
$a \xleftarrow{\$} \{0,1\}$
$b \xleftarrow{\$} \{0,1\}$
$S \leftarrow \mathsf{CRT}(s_1, \ldots, s_\ell, a)$
$E \leftarrow \mathsf{CRT}(e_1, \ldots, e_\ell, b)$
$r \leftarrow g^S y^E$
for $i = 1$ to ℓ
    check that $r \neq 1 \bmod q_i$,
      otherwise abort
end for
return $(r, e_1, \ldots, e_\ell, s_1, \ldots, s_\ell, \rho_1 \ldots, \rho_\ell)$

Figure 3.15: The algorithms used in Theorem 3.13, as part of the proof of Theorem 3.12.

This demonstrates that by uniformly choosing the random tape, the resulting distributions for Sign_1 and Sign_2 are identical, which is the uniform distribution on the set of valid signatures.

Lemma 3.14 *Every tuple of δ' is a valid signature tuple. Therefore $\delta' \subseteq \delta$.*

Proof: [of Lemma 3.14] Let $(r, e_1, \ldots, e_\ell, s_1, \ldots, s_\ell, \rho_1, \ldots, \rho_\ell) \in \delta'$. Let $i \in [\ell]$. By the Chinese Remainder Theorem we have:

$$S = s_i \bmod q_i \quad \text{and} \quad E = e_i \bmod q_i.$$

So there exists $\lambda, \mu \in \mathbb{Z}$ such that

$$S = s_i + \lambda q_i \quad \text{and} \quad E = e_i + \mu q_i.$$

Hence:

$$\begin{aligned} r^{\frac{p-1}{q_i}} &= \left(g^S y^E\right)^{\frac{p-1}{q_i}} \\ &= \left(g^{s_i + \lambda q_i} y^{e_i + \mu q_i}\right)^{\frac{p-1}{q_i}} \\ &= \left(g^{s_i} y^{e_i}\right)^{\frac{p-1}{q_i}} g^{\lambda(p-1)} y^{\mu(p-1)} \\ &= \left(g^{s_i} y^{e_i}\right)^{\frac{p-1}{q_i}} \end{aligned}$$

The last equality holds since the order of the multiplicative group $\mathbb{Z}_p^*$ is $p-1$, and this concludes the proof with the fact that $r \neq 1 \bmod q_i$. □

Lemma 3.15 *There is exactly one random tape upon which Sign_1 can run to yield each particular tuple of δ.*

Proof: [of Lemma 3.15] Let $k, e_1, \ldots, e_\ell, \rho_1, \ldots, \rho_\ell$ and $k', e'_1, \ldots, e'_\ell, \rho'_1, \ldots, \rho'_\ell$ be random choices of δ that both yield $(r, e_1, \ldots, e_\ell, s_1, \ldots, s_\ell, \rho_1, \ldots, \rho_\ell)$. It is immediate that $e_i = e'_i$ and $\rho_i = \rho'_i$ for all $i \in [\ell]$. Also since $g^k = g^{k'}$, g is of order $p-1$ and since k and k' are in $[p]$ then $k = k'$. □

Lemma 3.16 *There are exactly two random tapes over k, $\rho_1, \ldots, \rho_\ell$, $e_1, \ldots, e_\ell$ that output each tuple of δ'.*

Proof: [of Lemma 3.16] Let $e_1, \ldots, e_\ell$, $s_1, \ldots, s_\ell$, a, b, $\rho_1, \ldots, \rho_\ell$ and $e'_1, \ldots, e'_\ell$, $s'_1, \ldots, s'_\ell$, a', b', $\rho'_1, \ldots, \rho'_\ell$ be random choices that both give $(r, e_1, \ldots, e_\ell, s_1, \ldots, s_\ell, \rho_1, \ldots, \rho_\ell)$. It is immediate that $e_i = e'_i$, $s_i = s'_i$, and $\rho_i = \rho'_i$ for all $i \in [\ell]$. Let S, S', E, and E' be the corresponding CRT images. We have $g^S y^E = g^{S'} y^{E'}$, which is $g^{S+xE} = g^{S'+xE'}$, and $S + xE = S' + xE' \bmod (p-1)$. Since x is odd (it is invertible mod $p-1$), it follows that $S+E$ and $S'+E'$ have the same parity. Therefore $a + b = a' + b' \bmod 2$ and we have two choices: $a = b$, or $a = 1 - b$, both of which are correct.

□

Lemma 3.17 $\#\left(\mathbb{Z}_p \setminus \left(\bigcup_{i=1}^{\ell} \{q_i, 2q_i, \ldots, p-1\}\right)\right) = 2\prod_{i=1}^{\ell}(q_i - 1)$.

Proof: [of Lemma 3.17] The number of invertible elements mod$p$ is $\prod_{i=1}^{\ell}(q_i - 1) \times (2-1)$ so the number of invertible mod q_i for all i (and not necessarily for 2) is $2\prod_{i=1}^{\ell}(q_i - 1)$. This is exactly the cardinality of the set

$$\left(\mathbb{Z}_p \setminus \left(\bigcup_{i=1}^{\ell} \{q_i, 2q_i, \ldots, p-1\}\right)\right),$$

□

Lemma 3.18 *There are twice as many possible random choices in δ' than in δ.*

Proof: [of Lemma 3.18] For the number of random choices in δ we use Lemma 3.17 to count the number of k and then count the number of e_i and get $2\prod_{i=1}^{\ell}(q_i - 1) \times \prod_{i=1}^{\ell} q_i$. For δ', having $r \neq 1 \bmod q_i$ is equivalent to having $s_i \neq -e_i x$. Therefore it has the same number of random choices as a distribution picking the s_i from $\mathbb{Z}_{q_i} \setminus \{e_i x\}$ which is $\prod_{i=1}^{\ell} q_i \times \prod_{i=1}^{\ell}(q_i - 1) \times 2 \times 2$. □

It follows from the above results that the two distributions are the same, i.e. the uniform distribution over the set of valid signatures.
This concludes the proof of Theorem 3.13. □

Theorem 3.19 (Security under Chosen Message Attack) *An efficient attacker against Sign_2 can be turned into an efficient PDLP solver in the ROM.*

Proof: Let $\mathcal{A}$ be an attacker that wins the EF-CMA game for our scheme, illustrated in Figure 3.16. We construct in Figures 3.17 and 3.18 an algorithm $\mathcal{R}$ that uses $\mathcal{A}$ to solve the PDLP. $\mathcal{A}'$ is equivalent to $\mathcal{A}$ (with the same random tape which we omit in the notation), the difference being that it interacts with different oracles. Abusing notation we denote by $\mathcal{R}.H_i$ the composition of the hash function and the conversion function. If L is a list of pairs, we denote by $L^{-1}[e]$ the index of the element e in the list, and by $L[i]$ the i-th element of the list. If they cannot (i.e. if e is not in the list, or the list does not have an i-th element) they abort.

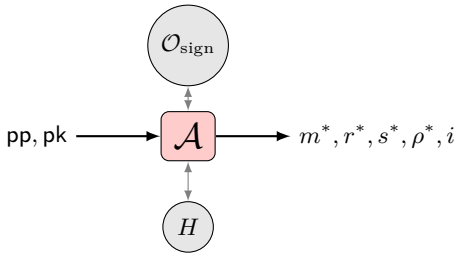

Figure 3.16: An efficient EF-CMA adversary $\mathcal{A}$ against our scheme, with random oracle H and a signing oracle $\mathcal{O}$.

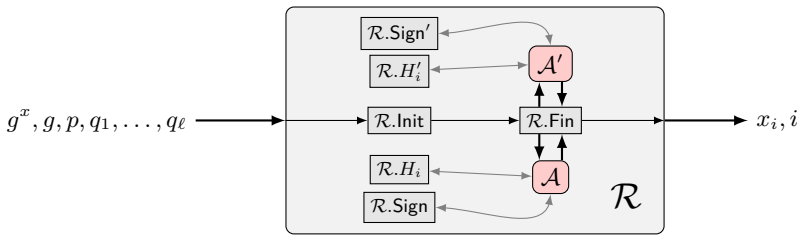

Figure 3.17: An efficient solver $\mathcal{R}$ for the PDLP, using a polynomial number of queries to $\mathcal{A}$. $\mathcal{R}$ implements the random oracle as $\mathcal{R}.H$ and the signing oracle as $\mathcal{R}.\text{Sign}$. The rewinded adversary and oracles are indicated with a prime symbol.

The algorithm $\mathcal{R}$ aborts in four possible ways during the simulation (denoted ($\star$), (†), (‡) and (§)) in Figures 3.17 and 3.18. We upper-bound the probability of these events in the following list:

- ($\star$) This occurs with negligible probability since the ρ is a fresh random which is unpredictable by the adversary.
- (†) This occurs with non overwhelming probability since the adversary is efficient.
- (‡) The element is in the list with non negligible probability because if the adversary forges on an unqueried hash in the ROM, it has a negligible chance to succeed.
- (§) This happens with non overwhelming probability due to the forking lemma [PS00].

If $\mathcal{R}$ does not abort, then $\left(g^{s^*} y^{e^*}\right)^{\frac{p-1}{q_{i^*}}} = (r^*)^{\frac{p-1}{q_{i^*}}} = \left(g^{\tilde{s}^*} y^{\tilde{e}^*}\right)^{\frac{p-1}{q_{i^*}}} \bmod p$. Then $s^* + e^* x = \tilde{s}^* + \tilde{e}^* \bmod q_{i^*}$. It follows that the value returned by $\mathcal{R}$ is equal to $x \bmod q_{i^*}$.
$\mathcal{R}$ succeeds with non negligible probability, as explained earlier. The probability of forking is polynomial in the number of queries to the random oracle, the number of queries to the signature

$\underline{\mathcal{R}.\mathsf{Init}(y = g^x, g, p, q_1, \ldots, q_\ell) :}$
set $L \leftarrow \emptyset$
$\Sigma \leftarrow \emptyset$
$j \leftarrow 1$
$k \leftarrow 0$
$l \leftarrow 0$
$\mathsf{pk} \leftarrow y$
$\mathsf{pp} \leftarrow \{p, \{q_i\}_{i=1}^{\ell}, g\}$
return $(\mathsf{pk}, \mathsf{pp})$

$\underline{\mathcal{R}.\mathsf{Fin}(\mathsf{pk}, \mathsf{pp}) :}$
$(m^*, r^*, s^*, \rho^*, i^*) \xleftarrow{\$} \mathcal{A}(\mathsf{pp}, \mathsf{pk})$
$e^* \leftarrow \mathcal{R}.H_{i^*}(m^*, r^* \bmod q_{i^*}, \rho^*)$
$a \leftarrow L^{-1}[((m^*, r^* \bmod q_{i^*}, \rho^*), e^*)]\ddagger$
if not $\mathsf{Verify}_{\mathsf{pp},\mathsf{pk}}(m^*, r^*, s^*, i^*)$
  abort†
$(m'^*, r'^*, s'^*, \rho'^*, i'^*) \xleftarrow{\$} \mathcal{A}'(\mathsf{pp}, \mathsf{pk})$
if $i^* \neq i'^*$ then abort§
if $r^* \neq r'^*$ then abort§
$e'^* \leftarrow \mathcal{R}.H_{i^*}(m'^*, r^* \bmod q_{i^*}, \rho'^*)$
if $e^* = e'^*$ then abort§
if not $\mathsf{Verify}_{\mathsf{pp},\mathsf{pk}}(m'^*, r^*, s'^*, i^*)$
  abort†
$\Delta s \leftarrow s^* - s'^*$
$\Delta e \leftarrow e'^* - e^*$
return $(i^*, \Delta s/\Delta e)$

$\underline{\mathcal{R}.\mathsf{Sign}'(m) :}$
$l \leftarrow 0$
return $\Sigma.[i]$
$l \leftarrow l + 1$

$\underline{\mathcal{R}.H(x) :}$
if $\exists (x', h') \in L$ s.t. $x' = x$
  return h'
else
  $h \xleftarrow{\$} \mathbb{Z}_p$
  $L \leftarrow L \cup \{(x, h)\}$
  return h

$\underline{\mathcal{R}.H'(x) :}$
$k \leftarrow 0$
$L' \leftarrow \emptyset$
if $\exists (x', h') \in L'$ s.t. $x' = x$
  return h'
else
if $i \leq a$
  $(x', h') \leftarrow L.[i]$
  return h'
  $k \leftarrow k + 1$
  $L' \leftarrow L' \cup \{(x, h)\}$
 else
  $h \xleftarrow{\$} \mathbb{Z}_p$
  $L' \leftarrow L' \cup \{(x, h)\}$
  return h

$\underline{\mathcal{R}.\mathsf{Sign}(m) :}$
if $j = 1$
  $(r, e_1, \ldots, e_\ell, s_1, \ldots, s_\ell, \rho_1, \ldots, \rho_\ell) \xleftarrow{\$} \delta'$
  if $\exists h$ s.t. $((m, r \bmod q_1, \rho_1), h) \in L$
    abort⋆
  $L \leftarrow L \cup \{((m, r \bmod q_1, \rho_1), e_1)\}$
  $j \leftarrow j + 1 \bmod \ell$
  return $(s_1, r, \rho_1, 1)$
  $\Sigma \leftarrow \Sigma \cup \{(s_1, r, \rho_1, 1)\}$
else
  if $\exists h$ s.t. $((m, r \bmod q_j, \rho_j), h) \in L$
    abort⋆
  $L \leftarrow L \cup \{(m, r \bmod q_j, \rho_j), e_j\}$
  $j \leftarrow j + 1 \bmod \ell$
  return (s_j, r, ρ_j, j)
  $\Sigma \leftarrow \Sigma \cup \{(s_j, r, \rho_j, j)\}$

Figure 3.18: An efficient solver for the PDLP, constructed from an efficient EF-CMA adversary against our scheme.

oracle, and ℓ. Note that the reduction is ℓ times looser than [PS00]. This concludes the proof of Theorem 3.19. □

Proof: [of Theorem 3.12] Using Theorem 3.13, we can use Sign_0 instead of Sign_2 as a target for the attacker in Theorem 3.19. □

3.3.4 Generic security of the partial discrete logarithm problem

In this section, we prove that the partial discrete logarithm problem introduced in Section 3.3.3.2 is intractable in the generic group model. This model was introduced by Shoup [Sho97b] for measuring the exact difficulty of solving classical discrete logarithm problems. Algorithms in generic groups do not exploit any properties of the encodings of group elements. They can access group elements only via a random encoding algorithm that encodes group elements as random bit-strings.

Proofs in the generic group model provide heuristic evidence of some problem hardness when an attacker does not take advantage of group elements' encoding. However, they do not necessarily say anything about the difficulty of specific problems in a concrete group.

Let ℓ be some non-negative integers, let $q_1, \ldots, q_\ell$ be some distinct prime numbers and let $q = q_1 \cdots q_\ell$. We consider a cyclic group $\mathbb{G}$ of (composite) order q generated by g. We assume without loss of generality that $q_1 = \max(q_1, \ldots, q_\ell)$. A classical method [PH78] to solve the partial discrete logarithm problem in $\mathbb{G}$ given $h = g^x \in \mathbb{G}$ is to compute $h^{q_2 \cdots q_\ell}$, an element of order diving q_1 (that belongs to the subgroup generated by $g^{q_2 \cdots q_\ell}$) and to compute its discrete logarithm x_1 in base $g^{q_2 \cdots q_\ell}$ using a square root method such as Shanks "baby-step giant-step" algorithm [Sha71]. It is easy to see that x_1 is equal to $x \bmod q_1$ and is obtained within time complexity $O(\sqrt{q_1} + \log(q_2 \cdots q_\ell))$ group operations.

Our goal is to prove that this time complexity is essentially optimal in the generic group model. Let $\mathcal{A}$ be a generic group adversary that solves the partial discrete logarithm problem in $\mathbb{G}$. As usual, the generic group model is implemented by choosing a random encoding $\sigma : \mathbb{G} \longrightarrow \{0,1\}^m$. Instead of working directly with group elements, $\mathcal{A}$ takes as input their image under σ. This way, all $\mathcal{A}$ can test is string equality. $\mathcal{A}$ is also given access to an oracle computing group multiplication and division: taking $\sigma(g_1)$ and $\sigma(g_2)$ and returning $\sigma(g_1 \cdot g_2)$ and $\sigma(g_1/g_2)$ respectively. Finally, we can assume that $\mathcal{A}$ submits to the oracle only encodings of elements it had previously received. This is because we can choose m large enough so that the probability of choosing a string that is also in the image of σ is negligible.

Theorem 3.20 *Let $\mathcal{A}$ be a generic algorithm that takes as input two encodings $\sigma(g)$ and $\sigma(h)$ (where g is a generator of $\mathbb{G}$ and $h = g^x \in \mathbb{G}$) and makes at most τ group oracle queries, then $\mathcal{A}$'s advantage in outputting a partial discrete logarithm (i, x_i) with $i \in \{1, \ldots, \ell\}$ and $x_i = x \bmod q_i$ is upper-bounded by $O(\tau^2/q_1)$.*

Proof: We consider an algorithm $\mathcal{B}$ playing the following game with $\mathcal{A}$. Algorithm $\mathcal{B}$ picks two bit strings σ_1, σ_2 uniformly at random in $\{0,1\}^m$. Internally, $\mathcal{B}$ keeps track of the encoded elements using elements in the ring $\mathbb{Z}_{q_1}[X_1] \times \cdots \times \mathbb{Z}_{q_\ell}[X_\ell]$. To maintain consistency with the bit strings given to $\mathcal{A}$, $\mathcal{B}$ creates a lists $\mathcal{L}$ of pairs (F, σ) where F is a polynomial vector in the ring $\mathbb{Z}_{q_1}[X_1] \times \cdots \times \mathbb{Z}_{q_\ell}[X_\ell]$ and $\sigma \in \{0,1\}^m$ is the encoding of a group element. The polynomial vector F represents the exponent of the encoded element in the group $\mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_\ell}$. Initially, $\mathcal{L}$ is set to

$$\{((1, 1, \ldots, 1), \sigma_1), ((X_1, \ldots, X_n), \sigma_2)\}$$

Algorithm $\mathcal{B}$ starts the game providing $\mathcal{A}$ with σ_1 and σ_2. The simulation of the group operations oracle goes as follows:

Group operation: Given two encodings σ_i and σ_j in $\mathcal{L}$, $\mathcal{B}$ recovers the corresponding vectors F_i and F_j and computes $F_i + F_j$ for multiplication (or $F_i - F_j$ for division) termwise. If $F_i + F_j$ (or $F_i - F_j$) is already in $\mathcal{L}$, $\mathcal{B}$ returns to $\mathcal{A}$ the corresponding bit string; otherwise it returns a uniform element $\sigma \xleftarrow{R} \{0,1\}^m$ and stores $(F_i + F_j, \sigma)$ (or $(F_i - F_j, \sigma)$) in $\mathcal{L}$.

After $\mathcal{A}$ queried the oracles, it outputs a pair $(i^*, x_i^*) \in \{1 \ldots, \ell\} \times \mathbb{Z}_{q_{i^*}}$ as a candidate for the partial discrete logarithm of h in base g. At this point, $\mathcal{B}$ chooses uniform random values $x_1, \ldots, x_n \in \mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_\ell}$. The algorithm $\mathcal{B}$ sets $X_i = x_i$ for $i \in \{1, \ldots, n\}$.

If the simulation provided by $\mathcal{B}$ is consistent, it reveals nothing about $(x_1, \ldots, x_\ell)$. This means that the probability of $\mathcal{A}$ guessing the correct value for $(i^*, x_i^*) \in \{1, \ldots, \ell\} \times \mathbb{Z}_{q_{i^*}}$ is $1/q_{i^*}$. The only way in which the simulation could be inconsistent is if, after we choose values for $x_1, \ldots, x_n$, two different polynomial vectors in $\mathcal{L}$ happen to produce the same value.

It remains to compute the probability of a collision happening due to a unlucky choice of values. In other words, we have to bound the probability that two distinct vectors F_i, F_j in $\mathcal{L}$ evaluate to the same value after the substitution, namely $F_i(x_1, \ldots, x_n) - F_j(x_1, \ldots, x_n) = 0$. This reduces to bound the probability of hitting a zero of $F_i - F_j$. By the simulation, this happens only if $F_i - F_j$ is a vector of polynomials where at least one coordinate — say the k-th — is a non-constant polynomial (and thus of degree one) denoted $(F_i - F_j)^{(k)}$.

Recall that the Schwartz-Zippel lemma says that, if F is a degree d polynomial in $\mathbb{Z}_{q_k}[X_k]$ and $S \subseteq \mathbb{Z}_{q_k}$ then

$$\Pr[F(x_k) = 0 \bmod q_k] \leq \frac{d}{|S|}$$

where x_k is chosen uniformly from S. Going back to our case, we obtain by applying the Schwartz-Zippel lemma :

$$\Pr[(F_i - F_j)^{(k)}(x_k) = 0 \in \mathbb{Z}_{q_k}] \leq 1/q_k \leq 1/q_1.$$

Therefore, the probability that the simulation provided by $\mathcal{B}$ is inconsistent is upper-bounded by $\tau(\tau - 1)/q_1$ (by the union bound) and the result follows. □

3.3.5 Provably secure pre-computations

Often the bottleneck in implementations centers around modular exponentiation. In this section, we briefly outline several proposed *pre-computation* techniques, as well as presenting in more detail two pre-computation schemes which were used in our implementation to compare timings between classical Schnorr and our scheme.

3.3.5.1 Brief overview

The problem of computing modular exponentiations is well-known to implementers of both DLP-based and RSA-based cryptosystems. In the specific case that we want to compute $g^x \bmod p$, the following strategies have been proposed but their security is often heuristic:

- Use signed expansions (only applicable to groups where inversion is efficient);
- Use Frobenius expansions or the GLV/GLS method (only applicable to certain elliptic curves);
- Batch exponentiations together, as suggested by M'Raïhi and Naccache [MN96].

The above approaches work for arbitrary values of x. Alternatively, one may choose a particular value of x with certain properties which make computation faster; however, there is a possibility that doing so weakens the DLP:

- Choose x with low Hamming weight as proposed by Agnew et al. [AMOV91];
- Choose x to be a random Frobenius expansion of low Hamming weight, as discussed by Galbraith [Gal12, Sec. 11.3];
- Choose x to be given by a random addition chain, as proposed by Schroeppel et al. [SOOS95];
- Choose x to be a product of low Hamming weight integers as suggested by Hoffstein and Silverman [HS03]—broken by Cheon and Kim [CK08];
- Choose x to be a small random element in GLV representation—broken by Aranha et al. [AFG+14];

Finally, a third branch of research uses large amounts of pre-computation to generate random pairs $(x, g^x \bmod p)$. The first effort in this direction was Schnorr's [Sch90], quickly broken by de Rooij [de 97]. Other constructions are due to Brickell et al. [BGMW93], Lim and Lee [LL94], and de Rooij [de 95]. The first provably secure solution is due to Boyko et al. [BPV98], henceforth BPV, which was extended and made more precise by [NSS01, CMT01, NS99]. This refined algorithm is called E-BPV (extended BPV).

3.3.5.2 The E-BPV pre-computation scheme

E-BPV[6] relies on pre-computing and storing a set of pairs $(k_i, g^{k_i} \bmod p)$; then a "random" pair $(r, g^r \bmod p)$ is generated by choosing a subset of the k_i, setting r to be their sum, and computing the corresponding exponential by multiplying the $g^{k_i} \bmod p$.

To guarantee an acceptable level of security, and resist lattice reduction attacks, the number n of precomputed pairs must be sufficiently large; and enough pairs must be used to generate a new couple.

(E-)BPV.Preprocessing:

$k_1, \ldots, k_n \xleftarrow{\$} \mathbb{Z}_p^*$
$L \leftarrow \emptyset$
for $j \in [n]$
  $L \leftarrow L \cup \{(k_j, K_j = g^{k_j} \bmod p)\}$
return L

E-BPV.GetRandomPair:

pick $S \subseteq [n]$ s.t. $|S| = k$
$(d_i, D_i) \xleftarrow{\$} D$
$r \leftarrow 0$
$R \leftarrow 1$
for $j \in S$
  $x_j \xleftarrow{\$} [h-1]$
  $r \leftarrow r + k_j x_j \bmod \phi(p)$
  $R \leftarrow R \cdot K_j^{x_j} \bmod p$
return(r, R)

Figure 3.19: The E-BPV algorithm for generating random pairs $(x, g^x \bmod p)$. The BPV algorithm is a special case of E-BPV for $h = 2$.

[6] BPV is a special case of E-BPV where $h = 2$. As such they share the same precomputing step.

Nguyen et al. [NSS01] showed that using E-BPV instead of standard exponentiation gives an adversary an advantage bounded by

$$m\sqrt{\frac{K}{\binom{n}{k}(h-1)^k}}$$

with m the number of signature queries by the adversary, (k, n, h) E-BPV parameters, and K the exponent's size.[7]

We fix conservatively $m = 2^{128}$. For our scheme, at 128-bit security, we have $K = P = 3072$. As suggested in [NSS01] we set $n = k$, and constrain our memory:

$$h^k \geq 2^{3400}$$

Optimizing $2k + h$ under this constraint, we find $(h, k) = (176, 455)$. This corresponds to 1087 modular multiplications, i.e., an amortized cost of 90 multiplications per signature, for about 170 kB of storage.

Alternatively, we can satisfy the security constraints by setting $n = 2048$, $h = 100$, $k = 320$, which corresponds to about 770 kB of storage, giving an amortized cost of 62 modular multiplications per signature.

In the implementation (Section 3.3.6), we solve the constrained optimisation problem to find the best coefficients (i.e., the least number of multiplications) for a given memory capacity.

Remark *To achieve the claimed bounds on modular multiplications, one should not compute* $R \leftarrow K_j^{x_j} \bmod p$ *directly; instead, an efficient speedup due to Brickell et al. [BGMW93] (BGMW) must be used. To illustrate the importance of this remark, we also give timings for a "naive" implementation in Table 3.5.*

Remark (Halving storage cost) *The following idea can halve the amount of storage required for the couples* (x, g^x)*: instead of drawing the values* x *at random, we draw a master secret* s *once, and compute* $x_{i+1} \leftarrow g^{x_i} \oplus s$ *(or, more generally/securely, a PRF with low complexity* $x_{i+1} = \mathrm{PRF}_s(g^{x_i})$*). Only* s*,* x_0*, and the values* g^{x_i} *need to be stored; instead of all the couples* (x_i, g^{x_i})*. This remark applies to both BPV and E-BPV.*

3.3.5.3 Lim and Lee precomputation scheme

We also consider a variation on Lim and Lee's fast exponentiation algorithm [LL94]. Their scheme originally computes g^r for r known in advance, but it is easily adapted to the setting where r is constructed on the fly. The speed-up is only linear, however, which ultimately means we cannot expect a sizable advantage over Schnorr. Nevertheless, Lim and Lee's algorithm is less resource-intensive and can be used in situations where no secure E-BPV parameters can be found (e.g., in ultra-low memory settings).

The Lim-Lee scheme (LL) has two parameters, h and v. In the original LL algorithm, the exponent is known in advance, but it is easily modified to generate an exponent on the fly. Intuitively, it consists in splitting the exponent into a "blocks" of size h, and dividing further each block in b sub-blocks of size v. The number of modular multiplications (in the worst case) is $a + b - 2$, and we have to store $(2^h - 1)v$ pairs. The algorithms are given in Figure 3.20.

For a given amount of memory M, it is easy to solve the constrained optimization problem, and we find

$$h_{\mathrm{opt}} = \frac{1}{\ln(2)}\left(1 + W\left(\frac{1+M}{e}\right)\right)$$

[7] For Schnorr, the exponent's size is Q; for our scheme, it is P.

where W is the Lambert function. For a memory M of 750 kB, this gives $h \approx 8.6$. The optimal parameters for integers are $h = 9$ and $v = 4$.[8]

Remark *For LL, Remark 14 on halving storage requirements does not apply, as x need not be stored.*

LimLee.Preprocessing(h, v):

$g_0 \leftarrow g$
$L = \emptyset$
for $i = 0$ to $h - 1$
  $g_i \leftarrow g_{i-1}^{2^a}$
for $i = 0$ to $2^h - 1$
  let $i = e_{h-1} \dots e_1$ in binary
  $g_{0,i} = g_{h-1}^{e_{h-1}} \dots g_1^{e_1}$
for $i = 0$ to $2^h - 1$
  for $j = 0$ to $v - 1$
    $g_{j,i} \leftarrow g_{j-1,i}^{2^b}$
    $L \leftarrow L \cup \{g_{j,i}\}$
return L

LimLee.GetRandomPair:

$R \leftarrow 1$
$r \leftarrow 0$
for $i = b - 1$ to 0
  $R \leftarrow R^2$
  $r \leftarrow r + r$
for $j = v - 1$ to 0
  $r_{i,j} \xleftarrow{\$} \{0, \dots, 2^h - 1\}$
  $R \leftarrow R \times g_{j,r_{i,j}}$
  $r \leftarrow r + r_{i,j}$
return (r, R)

Figure 3.20: The LL algorithm for generating random pairs $(x, g^x \bmod p)$.

A summary of the properties for the pre-computations techniques E-PBV and LL can be found in Table 3.3.

Algorithm	Storage	Multiplications	Security
Square-and-multiply	0	$1.5 \log P$	Always
BPV [BPV98]	nP	$k - 1$	$m\sqrt{\frac{P}{\binom{n}{k}}} < 2^{-\kappa}$
E-BPV [NSS01]	nP	$2k + h - 3$	$m\sqrt{\frac{P}{\binom{n}{k}(h-1)^k}} < 2^{-\kappa}$
Lim and Lee [LL94]	$2^h \times v \times P$	$\frac{\log P}{h}(1 + \frac{1}{v}) - 3$	Always

Table 3.3: Precomputation/online computation trade-offs.

3.3.6 Implementation results

Reschnorr, using the algorithms described in Sections 3.3.3 and 3.3.5, has been implemented in C using the GMP library. In the interest of timing comparison, we have also implemented the classical Schnorr scheme. The results for several scenarios are outlined in Table 3.4 (at 128-bit security) and Table 3.5 (at 192-bit security). Complete source code and timing framework are available upon request from the authors.

These experiments show that our scheme is faster than Schnorr when at least 250 pairs (i.e., 750 kB at 128-bit security) have been precomputed. This effect is even more markedly visible at higher security levels: our scheme benefits more, and more effectively, from the E-BPV+BGMW

[8] In practice, it turns out that $h = v = 8$ performs slightly better, due to various implementation speed-ups possible in this situation

optimisation as compared to Schnorr. The importance of combining E-BPV and BGMW is also visible: E-BPV using naive exponentiation does not provide any speed-up.

Schnorr and our scheme achieve identical performance when using Lim and Lee's optimisation, confirming the theoretical analysis. When less than 1 MB of memory is allocated, this is the better choice.

Table 3.4: Timing results for Schnorr and our scheme, at 128-bit security ($P = 3072$, $Q = 256$). Computation was performed on an ArchLinux single-core 32-bit virtual machine with 128 MB RAM. Averaged over 256 runs.

Scheme	Storage	Precomp.	Time (per sig.)
Schnorr	–	–	6.14 ms
Schnorr + [NSS01]	170 kB	33 s	105 ms
Schnorr + [NSS01] + [BGMW93]	170 kB	33 s	2.80 ms
Schnorr + [NSS01] + [BGMW93]	750 kB	33 s	2.03 ms
Schnorr + [NSS01] + [BGMW93]	1 MB	34 s	2.00 ms
Schnorr + [NSS01] + [BGMW93]	2 MB	37 s	2.85 ms
Schnorr + [LL94]	165 kB	3 s	949 ns
Schnorr + [LL94]	750 kB	3 s	644 ns
Schnorr + [LL94]	958 kB	3 s	630 ns
Schnorr + [LL94]	1.91 MB	3 s	★ 472 ns
Our scheme	–	–	5.94 ms
Our scheme + [NSS01]	170 kB	33 s	9.2 ms
Our scheme + [NSS01] + [BGMW93]	170 kB	33 s	1.23 ms
Our scheme + [NSS01] + [BGMW93]	750 kB	33 s	426 ns
Our scheme + [NSS01] + [BGMW93]	1 MB	34 s	371 ns
Our scheme + [NSS01] + [BGMW93]	2 MB	37 s	★ 327 ns
Our scheme + [LL94]	165 kB	3 s	918 ns
Our scheme + [LL94]	750 kB	3 s	709 ns
Our scheme + [LL94]	958 kB	3 s	650 ns
Our scheme + [LL94]	1.91 MB	3 s	757 ns

Table 3.5: Timing results for Schnorr and our scheme, at 192-bit security ($P = 7680$, $Q = 384$). Computation was performed on an ArchLinux single-core 32-bit virtual machine with 128 MB RAM. Averaged over 256 runs.

Scheme	Storage	Time (/sig.)
Schnorr	–	35.2 ms
Schnorr + [LL94]	715 kB	508 ns
Schnorr + [NSS01] + [BGMW93]	750 kB	2.08 ms
Schnorr + [NSS01] + [BGMW93]	1.87 MB	1.62 ms
Schnorr + [LL94]	1.87 MB	★ 476 ns
Our scheme	–	33.0 ms
Our scheme + [LL94]	715 kB	486 ns
Our scheme + [LL94]	1.87 MB	467 ns
Our scheme + [NSS01] + [BGMW93]	1.87 MB	★ 263 ns

3.3.7 Heuristic security

Several papers describe server-aided precomputation techniques (e.g., [KU16]), which perform exponentiations with the help of a (possibly untrusted) server, i.e., such techniques allow for outsourcing the computation of $g^x \bmod n$, with public g and n, without revealing x to the server.

Interestingly, the most efficient algorithms in that scenario (which of course we could leverage) use parameters provided by Hohenberger and Lysyanskaya [HL05] for E-BPV. A series of papers took these parameters for granted (including [KU16]), but we should point out that *these are not covered* by the security proof found in [NSS01].

Despite this remark, it seems that no practical attack is known either; therefore if we are willing to relax our security expectations somewhat, it is possible to compute the modular exponentiation faster. Namely, a Q-bit exponent can be computed in $O(\log Q^2)$ modular multiplications.

Our scheme uses an exponent that is ℓ times bigger than Schnorr, which is amortized over ℓ signatures. Comparing our scheme to Schnorr, the ratio is $\frac{\ell \log(Q)^2}{(\log \ell Q)^2}$. With $Q = 256$ we get a ratio of approximately 5.7.

Note that as Q increases, so does ℓ, and therefore so does the advantage of Our scheme over Schnorr in that regime.

3.3.8 Reduction-friendly moduli

As part of computing $g^k \bmod p$, a very costly operation is the reduction mod p. An interesting question is whether some particular moduli p can be found, for which reduction is particularly easy.

An example of such moduli are those that start with a `1` followed by many `0`.

Example 3.3 *For $P = 3072$ and $Q = 256$, using (in hexadecimal notation)*

$$\Delta_i = \{\texttt{12d}, \texttt{165}, \texttt{1e7}, \texttt{247}, \texttt{2f5}, \texttt{31b}, \texttt{327}, \texttt{34f}, \texttt{3a3}, \texttt{439}, \texttt{56b}, \texttt{4fe7}\}$$

and $q_i = 2^Q + \Delta_i$, we have that p equals:

2[60]e0e8[56]18058164[53]1479d1e16e8[51]aa09581f139be[48]3a9dc2e99b080dd[47]dfe705c4e9b3a45678[43]25a378c4e6b62835f401[42]471d330fbde56ef2c80281e[39]5c5388621a308a5425f007648[37]4e506ba1a5b68dc5faca1155e64[35]270051399124b193e6716e08b4408[34]8a07b85ed815e7eac1135861bd67e3

where `[x]` denotes a sequence of `x` hexadecimal zeros.

3.3.9 Conclusion

We have introduced a new digital signature scheme variant of Schnorr signatures, that reuses the nonce component for several signatures. Doing so does not jeopardise the scheme's security; attempting to do the same with classical Schnorr signatures would immediately reveal the signing key. However, the main appeal of our approach is that precomputation techniques, whose benefits can only be seen for large enough problems, become applicable and interesting. As a result, without loss of security, it becomes possible to sign messages using fewer modular multiplications. Our technique is general and can be applied to several signature schemes using several speed-up techniques.

Chapter 4

Post Quantum Public-key based on Mersenne primes

Contents

4.1 Introduction

Diffie and Hellman introduced Public key cryptography in [DH06] in 1976. In 1979 Rabin published the first cryptosystem reducible to a known hard problem [Rab79]. He showed that if an opponent could decrypt randomly chosen ciphertexts, then he could factorise large composite integers. Since it is believed that no one can efficiently factorise large composite integers, then it follows that no one can computationally break this cryptosystem. Since then it is good practice to reduce any proposed system to well defined hard problems. This practice has become almost mandatory in public key cryptography. Over the past decades a number of hard problems were proposed, and a distinction was made between standard assumptions (the ones that have been studied for a long time and are widely believed to hold) and non standard ones (younger ones whose veracity is douted). In Section 4.2 we deal with a new public key cryptosystem that is

reduced to a newly proposed problem. We experimentally showed that this new assumption does not hold. A more precise analysis of our attack followed in [dBDJdW17]. A modified version with adjusted parameters of the original cryptosysytem was consequently published in [AJPS17a], and proposed to the post-quantum NIST competition. In Section 4.3 we propose an unpublished variation of this cryptosystem. This version reduces to a similar problem. Unfortunately this new problem does not seem to better resist to the attack we proposed in Section 4.2. We expose it in our thesis as we believe that our variant is of independant conceptual interest.

4.2 On the Hardness of the Mersenne Low Hamming Ratio Assumption

Abstract

In a recent paper [AJPS17b], Aggarwal, Joux, Prakash, and Santha (AJPS) describe an ingenious public-key cryptosystem mimicking NTRU over the integers. This algorithm relies on the properties of Mersenne primes instead of polynomial rings. The security of the AJPS cryptosystem relies on the conjectured hardness of the Mersenne Low Hamming Ratio Assumption, defined in [AJPS17b].

This work shows that AJPS' security estimates are too optimistic and describes an algorithm allowing to recover the secret key from the public key much faster than foreseen in [AJPS17b].

In particular, our algorithm is *experimentally practical* (within reach of the computational capabilities of a large organization), at least for the parameter choice $\{n = 1279, h = 17\}$ conjectured in [AJPS17b] as corresponding to a 2^{120} security level. The algorithm is fully parallelizable.

This is joint work with Aisling Connolly, Rémi Géraud and David Naccache. The corresponding paper was presented at the 5^{th} International Conference on Cryptology and Information Security in Latin America, Latincrypt 2017, in La Havana, Cuba; and has been published as [BCGN17b].

4.2.1 Introduction

A Mersenne prime is a prime of form $2^n - 1$, where $n > 1$ is itself prime.

In a recent paper [AJPS17b], Aggarwal, Joux, Prakash, and Santha (AJPS) describe an ingenious public-key cryptosystem mimicking NTRU over the integers. This algorithm relies on the properties of Mersenne numbers instead of polynomial rings. This scheme is defined by the following algorithms:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$, which chooses the public parameters $\mathsf{pp} = (n, h)$ so that $p = 2^n - 1$ is prime and so as to achieve a λ-bit security level. In [AJPS17b] the following lower bound is derived
$$\binom{n-1}{h-1} > 2^\lambda$$
which for instance is satisfied by $\lambda = 120, \mathsf{pp} = (n = 1279, h = 17)$.

- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{pk})$, which picks F, G two n-bit strings chosen independently and uniformly at random from all n-bit strings of Hamming weight h, and returns $\mathsf{sk} \leftarrow G$ and $\mathsf{pk} \leftarrow H = F/G \bmod (2^n - 1)$.

- $\mathsf{Encrypt}(\mathsf{pp}, \mathsf{pk}, b \in \{0, 1\}) \to c$, which picks A, B two n-bit strings chosen independently and uniformly at random from all n-bit strings of Hamming weight h, then computes
$$c \leftarrow (-1)^b (AH + B) \bmod (2^n - 1).$$

- $\mathsf{Decrypt}(\mathsf{pp}, \mathsf{sk}, c) \to \{\perp, 0, 1\}$, which computes $D = \|Gc \bmod (2^n - 1)\|$ and returns
$$\begin{cases} 0 & \text{if } D \leq 2h^2, \\ 1 & \text{if } D \geq n - 2h^2, \\ \perp & \text{otherwise} \end{cases}$$

We refer the reader to [AJPS17b] for more details on this cryptosystem which does not require further overview because we directly attack the public key to infer the secret key.

In particular, security rests upon the conjectured intractability of the following problem:

Definition 4.1 *The* Mersenne Low Hamming Ratio Assumption *states that given an n-bit Mersenne prime $p = 2^n - 1$ and an integer h, the advantage of any probabilistic polynomial time adversary attempting to distinguish between $F/G \bmod p$ and R is at most $\frac{\text{poly}(n)}{2^\lambda}$, where R is a uniformly random n-bit strings, and (F, G) are independently chosen n-bit strings each having Hamming weight h.*

As we will see, we argue that (F, G) can be *experimentally* computed from H, at least for the parameter choice $\{n = 1279, h = 17\}$ conjectured in [AJPS17b] as corresponding to a 2^{120} security level.

The full code (Python for partition sampling and Mathematica for lattice reduction) is available from the authors upon request.

4.2.2 Outline of the Analysis

The analysis uses the Lenstra–Lenstra–Lovász lattice basis reduction algorithm (LLL, [LLL82]). We do not recall here any internal details of LLL but just the way in which it can be used to solve a linear equation with k unknowns when the total size of the unknowns is properly bounded.

4.2.2.1 Using LLL to Spread Information

Let $x_1, \ldots, x_k \in \mathbb{N}^*$ be k unknowns. Let $p \in \mathbb{N}$ be a modulus and $a_0, \ldots, a_k \in \mathbb{N}$. Consider the equation:

$$a_0 = \sum_{i=1}^{k} a_i x_i \bmod p.$$

All the reader needs to know is that the LLL algorithm will find $x_1, \ldots, x_k$ if $\prod_{i=1}^{k} x_i < p$.

In particular, LLL can be adapted to provide any uneven split of sizes between the x_i as long as the sum of those sizes does not exceed the size of p. More details on the theoretical analysis of LLL in that setting and variants are given in [NS01, Sec. 3.2] and [Jou09, Chap. 13], in the context of generalised knapsack problems.

4.2.2.2 Partition and Try

The first observation that attracted our attention is that the size[1] of F (and G) has an unusually small expectation $\sigma(n, h)$:

$$\sigma(n, h) = n \left(1 + \frac{(1 - \frac{h}{n})^{n+1} - 1}{\frac{h}{n}(n + 1)} \right)$$

The difference in size between $n = 1279$ and $\sigma(1279, 17)$ is not huge[2] and cannot be immediately exploited. However, the same phenomenon also occurs at the least significant bits and further shortens the expected nonzero parts of F and G by 70 bits.

Similarly, assume that in the key generation procedure, both F and G happen to have bits set to 1 only in their lower halves. When this (rare event) happens, we can directly apply LLL to H to recover F and G. We call this event T.

[1] That is, the length of a number, once its leading zeros are discarded.
[2] $1279 - \sigma(1279, 17) \approx 75$ bits.

Is that event rare? Since F and G are chosen at random, T happens with probability at least 2^{-2h}. While T's probability is not cryptographically negligible, this pre-attack only allows to target one key out of 2^{2h}. For the first suggested parameter set ($\lambda = 120$), one public key out of 67 million can be attacked in this fashion, and its F and G recovered, i.e., a total break. The question is hence, can this phenomenon be extended to any key? And if so, at what cost? In particular, can we sacrifice work to increase the size of the vulnerable key space? The answers to these questions turn out to be positive, as we will explain hereafter.

Random partitions. Instead of a fixed partition of $\{0, \ldots, n-1\}$, we can sample random partitions, for instance by sampling (without replacement) m positions, which are interpreted as boundaries between regions of zeros and regions that possibly contain a `1`. The total number of regions, $m+1$, determines the dimension of the lattice being reduced.

For the sake of simplicity we consider *balanced* partitions:

Definition 4.2 *A partition of $\{0, \ldots, n-1\}$ into $m/2$ type 1 blocks and $m/2+1$ type 2 blocks is* balanced *if the total size of the type 1 blocks and the total size of the type 2 blocks differ by at most one.*[3]

A randomly sampled partition is not necessarily a balanced partition: we use rejection sampling to ensure the balancing property.[4] The sought-after property of these partitions is the following:

Definition 4.3 *Let X be a binary string of length n. A partition of X into $m/2$ type 1 blocks and $m/2+1$ type 2 blocks is* correct for X *if the type 2 blocks are completely made of zeros.*

Figure 4.1 illustrates the partitions that we are interested in on a simple example. Also note that the definition above does not put any constraint on type 1 blocks, which may contain zeros or not; since they are not guaranteed to be zero, we refer to them as "non-zero" blocks. Accordingly, blocks of type 2 in a correct partition is referred to as "zero" blocks.

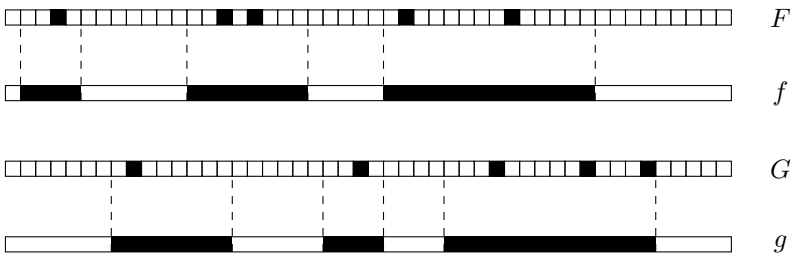

Figure 4.1: An illustration of the partitions that we are interested in: in these diagrams, a black square in F or G represents a `1`, while white squares represent `0`s. The partitions f and g are balanced and correct for F and G respectively, with "zero" blocks coloured white, and "non-zero" blocks coloured black. The vertical dashed lines show how F and G align with their respective partitions.

The observation at the beginning of this section is that using a balanced partition that is correct for F and another one that is correct for G and we can recover F and G from H.

[3] Since n is odd, we must accept a $\pm$ 1 excess.

[4] There is room for improvement here as well since rejection sampling is a very inefficient approach. Nevertheless, it will be sufficient for our discussion, and any approach to generating such partitions would work without impacting the analysis.

Since F and G are unknown, we cannot construct a correct partition from them directly; but the probability that a random balanced partition is correct for F (resp. G) is lower bounded[5] by 2^{-h}. Assuming that F and G are independent, which they should be according to the key generation procedure, we found a correct partition for *both* F and G with a probability of 2^{-2h}.

Remark *We may also consider imbalanced partitions which allow an extra speed-up for a subtle reason: Given that the unknowns found by LLL have a low Hamming density, the odds that these numbers naturally begin by a sequence of zeros (and are hence shorter than expected) is high. The interesting point is that the total length of such natural gains sums up and allows to unbalance the partition in favor of type 1 blocks. Consider the analogy of a fishing boat that can carry up to 1000 kilograms of fish. The fishermen fishes with 3 nets having maximal capacities of 200, 300 and 500 kilograms each. Because waters are sparse in fish, the nets are expected to catch only 70% of their maximal capacity. Hence, we see that larger nets (285, 428, 714) can be used to optimize the boat's fishing capacity. However, unlike the boat, with LLL fish cannot be thrown back to the water and... excess weight sinks the boat (the attack fails). Hence if this speed-up strategy is used, we need to catch more than normal but not be too greedy. Note as well that if all variables end by at least ℓ trailing (LSB) zeros then these $m\ell$ zeros add-up to the gain as well (because there is no constant term in the equation a division of all variables by 2 has no effect on the solution's correctness). We did not exploit nor analyze these tricks in detail.*

Trying partitions. The attack then consists in sampling a balanced partition, running LLL, and checking whether the values of F and G obtained from the reduction have the correct Hamming weight and yield H by division. Concretely, the matrix to be reduced is obtained as follows from the partitions f of F and g of G:

1. Compute the size of the each non-zero blocks in f and g, we call these sizes $\mathbf{u} = \{u_i\}$ and $\mathbf{v} = \{v_i\}$ respectively, with $i = 0, \ldots, m/2 - 1$. Let $w = \max_i\{u_i, v_i\}$.

2. Construct the vector $\mathbf{s} = s_i$ as follows:

$$s_i = \begin{cases} 2^{w-v_i} & \text{if } i < m/2 \\ 2^{w-u_i} & \text{if } m/2 \leq i < m \end{cases}$$

3. Construct the vector $\mathbf{a} = \{a_j\}$ as follows: let f_i (resp. g_i) denote the starting position of the non-zero blocks in F (rep. G), and set

$$a_j = \begin{cases} H \times 2^{g_i} \bmod p & \text{if } j < m/2 \\ p - 2^{f_i} & \text{if } m/2 \leq j < m \end{cases}$$

4. Choose an integer K, and assemble the matrix $\mathbf{M}$ as follows:

$$\mathbf{M} = \begin{pmatrix} \mathrm{diag}(\mathbf{s}) & K\mathbf{a} \\ 0 & Kp \end{pmatrix}$$

   where $\mathrm{diag}(\mathbf{x})$ is the diagonal matrix whose diagonal entries are given by $\mathbf{x}$. The coefficient K is a tuning parameter, which we set to 2^{1200}.

[5] We ignore the fact that we sample without replacement here, as $h \ll n$. Under this approximation, all the bits are sampled uniformly and independently and may fall with probably 1/2 either in a type 1 or a type 2 block.

5. Finally, we use LLL on $\mathbf{M}$ (using the Mathematica command `LatticeReduce`) and recover the reduced matrix's row that complies with the Hamming density of F and G. This row is expected to give the values of the non-zero blocks of F and G, and we can check its correctness by computing its Hamming weight, and checking that the ratio of the candidate values modulo p give H.

By the above analysis, a given partition is correct with probability 2^{-2h}, which for $\lambda = 120$ is only 2^{-34}; if we can run LLL reasonably fast, which is the case for $m = 16$, an efficient attack happens to be within reach of a well-equipped organization. Experimental evidence indeed suggests the feasibility of the attack, see Section 4.2.3.

Remark *For larger security parameters λ, the ratio h/n deduced from the analysis in [AJPS17b] asymptotically vanishes. It should be checked if this influences imbalanced partition finding to the attacker's relative advantage for larger values of λ. We did not explore this avenue left to the reader as a potential research question.*

4.2.3 Putting it Together

To illustrate the attack feasibility, we fix a random tape in a deterministically verifiable way and implement our algorithm (see Figure 4.2).

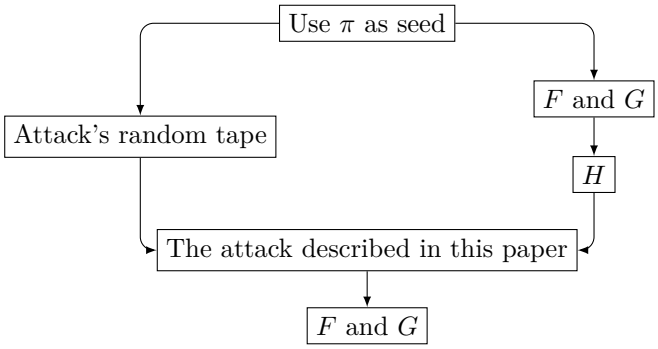

Figure 4.2: The feasibility demonstration consists in deriving the attack's random tape from a verifiable source in a deterministic way, as well as the keys.

We generated a nothing-up-our-sleeves key with the procedure of Figure 4.3. The sample(S, h)

1. $n, h \leftarrow \mathsf{pp}$
2. $I_1 = \{i_1, \ldots, i_h\} \leftarrow \text{sample}(\{0, \ldots, n-1\}, h)$
3. $I_2 = \{i_1, \ldots, i_h\} \leftarrow \text{sample}(\{0, \ldots, n-1\}, h)$
4. $F \leftarrow \sum_{i \in I_1}^{h} 2^i$
5. $G \leftarrow \sum_{i \in I_2}^{h} 2^i$
6. return $(\mathsf{sk} = G, \mathsf{pk} = F \cdot G^{-1} \bmod p)$

Figure 4.3: The KeyGen(pp) procedure.

procedure selects h indices without replacement in the range S. It is implemented[6] by returning the h first entries of a deterministic Fisher–Yates shuffle of S. The randomness in sample(S, h) is simulated by iterating the SHA256 function, starting with the seed given by the ASCII representation of the 100 first decimals of π:

```
31415926535897932384626433832795028841971693993751
05820974944592307816406286208998628034825342117068
```

In a real attack, we would simply use a fast non-cryptographic random number generator, but the above choice serves the purpose of reproducibility.

This gives the following (in hexadecimal notation, the zero MSBs have not been written):

$$I_1 = \{33, 47, 8e, 95, a1, 134, 19f, 1ab, 1ac, 1ce, 25d, 301, 30a, 3ee, 444, 46b, 471\}$$
$$I_2 = \{89, b5, de, 116, 141, 1dd, 1de, 2ae, 322, 37a, 388, 38a, 3f9, 48c, 48d, 4e9, 4f2\}$$

$$F = 208000000001000000000000000000040000000000000000000000000000000$$
$$000000000000000000000000040200000000000000000000000000000000000$$
$$000002000000000000000000000000000000000004000000180080000000000$$
$$000000000000001000000000000000000000000000000000200204000000$$
$$000000000080000800000000000$$

$$G = 402000000000000000000000300000000000000000000000000000000020$$
$$000000000000000000000000050004000000000000000000004000000000000$$
$$000000000000000040000000000000000000000000000000000000000000000$$
$$000006000000000000000000000000000000000000020000000000400000000$$
$$000004000000002000000000200000000000000000000000000000000$$

$$H = 1610fecf11dbd70f5d09da1244a85c3aa7aed7de75a6d1fe4e988b5f66d66e1b$$
$$c27d46afd96800ff8b2b67316dff1046b88d205e620ba78a813c15f47ab8a7d2$$
$$a8f7eb12fe0fcff882307d92d4c0f9296a7cf4390ce3140e11e4b7c802fa67d3$$
$$a8517d30b00980380bdf8992ed6a2d3f74e25f14bae21786672bddae4f2bf897$$
$$f38741cdc10b319f8272d42f738cd296d4907331518c3439621aefad5c3d1a7c$$

4.2.3.1 Recovering F and G from H

Finding a Winning Partition. At this step, we generate random balanced partitions and try LLL on the resulting decomposition. Doing so we quickly find the following partitions

$$f = \{2a, bf, 134, 1ec, 233, 253, 25a, 270, 2ee, 32d, 3e4, 41e, 42b, 4a7, 4f6, 4fd\}$$
$$g = \{7c, 142, 1d0, 22a, 289, 2c8, 2de, 2e7, 2eb, 33c, 372, 3a0, 3da, 3ff, 48a, 4fd\}$$

[6] Other implementations are of course possible and do not affect the analysis. For other classical sampling without replacement algorithms, the reader may consult [SW12].

respectively for F and G, which upon lattice reduction yield candidates of the correct Hamming weight. Their ratio indeed gives H; however one may debate our claim that this partition was found at random and argue that we constructed it from our prior knowledge of F and G.

To counter this argument and insist that finding partitions is reasonably easy, we derived them *deterministically from the same seed as the key.* To achieve this, we proceed as follows: we draw two independent sets of $m/2-1$ indices in the range $[0, n/2]$, which gives the sizes of the zero blocks and the non-zero blocks. This guarantees that the partitions are balanced. The randomness used for this sampling is obtained by iterating SHA256 as for key generation.

As in the example above, we construct partitions for $m = 16$ — this choice is not dictated by probability (as the likelihood to find a correct partition is in theory independent of m), but rather by a trade-off between the cost of LLL and the number of partitions explored. It is possible for instance to start with $m = 2$ partitions, then $m = 3$, and so forth, but we settled for a random search which is easier to implement.

We found the following partition for F at run #1,152,006 (in 116 s):

$$f = \{\texttt{27}, \texttt{b2}, \texttt{10e}, \texttt{13c}, \texttt{198}, \texttt{1cf}, \texttt{24b}, \texttt{27b}, \texttt{2ac}, \texttt{30f}, \texttt{3e1}, \texttt{456}, \texttt{45a}, \texttt{4ba}, \texttt{4d6}, \texttt{4fd}\}$$

Recovering F alone took about two minutes.[7] Given that we have a totally deterministic random tape, we regard our experiment as legitimately reflecting reality. Because F and G are independent, this brings the total effort to about the square of this number, i.e. about 2^{34} attempts to get both partitions with certainty. Each of these attempts must also involve one LLL, which is the main cost factor.

Using the same sequence, #64,249 gave a partition for G too (in 7.6 s):

$$g = \{\texttt{7b}, \texttt{11c}, \texttt{13b}, \texttt{181}, \texttt{1cc}, \texttt{1e1}, \texttt{284}, \texttt{2e6}, \texttt{318}, \texttt{329}, \texttt{36f}, \texttt{3e5}, \texttt{3f1}, \texttt{404}, \texttt{476}, \texttt{4fd}\}$$

Finally, note that the task is fully parallelizable and would benefit from running on several independent computers, a remark that we will later use in our final work factor estimates.

Computing the Secret Key By running our program as explained in Section 4.2.2, we recover F, G, and confirm that $H = F/G \bmod p$.

4.2.3.2 Predicting the Total Execution Time

Putting all the above figures together and assuming no further algorithmic improvements, the total expected effort is:

$$\frac{(\text{LLL\_Time} + 2 \times \text{Partition\_Time}) \times \text{Average\_Partition\_Tries}^2}{\text{Number\_of\_Processors}}$$

Where in our basic scenario $\text{Average\_Partition\_Tries} = 2^h$.

We performed LLL on Mathematica using the `LatticeReduce` function, which took less than a second in the worst case on a simple laptop. We safely assume that this figure can be divided by 10 using a dedicated and optimized code. We also assume that a credible attacker can, for example, very easily afford buying or renting 150 TILE-Gx72 multicore processors.

$$\frac{\frac{1}{10} \times 1{,}152{,}006 \times 64{,}249}{150 \times 72} \times \frac{1}{60 \times 60 \times 24} \approx 7 \text{ days } 22 \text{ hours}$$

Hence, according to the evidence exhibited in this paper, breaking a 1279 bit key takes a week using 150 currently available multicore processors (e.g. TILE-Gx72).

[7] Experiments with random partitions show that this number is quite variable and follows a Poisson distribution, with a correct partition being typically found earlier, with an average of 2^{17} tries.

4.2.4 Conclusion

While we did not formally evaluate efficiency nor asymptotic complexities, our quick and dirty experiments clearly suffice to show that key recovery is fast and within reach. An obvious countermeasure consists of increasing parameter sizes. Hence a precise re-evaluation of parameter sizes and safety margins of the Mersenne Low Hamming Ratio Assumption seems in order.

More systemic protections may consist in modifying the definition of H (and possibly the underlying cryptosystem), which is a very interesting open problem.

Nonetheless, the beautiful idea of Aggarwal, Joux, Prakash, and Santha exploiting the fact that arithmetics modulo Mersenne numbers is (somewhat) Hamming-weight preserving, is very elegant and seems very rich in possibilities and potential cryptographic applications.

4.3 Public-Key Cryptosystems Based on a New Complexity Assumption

Abstract

In 2017, Aggarwal, Joux, Prakash, and Santha introduced a new public-key cryptosystem relying on the conjectured hardness of an *ad hoc* but credible indistinguishability game [AJPS17c]. Subsequent work by Beunardeau et al. [BCGN17c], and de Boer et al. [dBDJdW17], led to a revision of the effective hardness and led Aggarwal et al. to amend substantially their original cryptosystem substantially [AJPS17d]. A bit later Ferradi and Naccache suggested slightly improved variants [FN17] along with several research directions.

In this paper we introduce a cryptosystem similar in spirit to the original Aggarwal–Joux–Prakash–Santha cryptosystem (AJPS-1) but relying on a *different* hardness assumption. Unfortunately, lattice reduction (*à la* Beunardeau et al.) experimentally applies in the same way as it does to AJPS. The resulting construction is conceptually simpler than the "fixed" AJPS cryptosystem (AJPS-ECC) and than Ferradi and Naccache's "high-bandwidth" variant (AJPS-FN-BT).
This is joint work with Aisling Connolly, Rémi Géraud and David Naccache.

4.3.1 Introduction

In 2017, Aggarwal, Joux, Prakash, and Santha [AJPS17c, AJPS17d] introduced a new public-key cryptosystem, inspired by NTRU [HPS98] but conceptually much simpler, and tentatively immune to some of the most classical attacks against NTRU. Since public-key cryptosystems are relatively rare, Aggarwal et al.'s construction (henceforth AJPS-1, following [FN17]) garnered much attention from the cryptographic community. In a matter of weeks, it was found that AJPS-1's initial security estimates were too optimistic, and a modified scheme with larger parameters was proposed [AJPS17d]. Section 4.3.2 recalls in further details the construction and history of these cryptosystems, which we refer to as Mersenne-based cryptosystems.

In this paper, we suggest a further modification of the underlying hardness assumption, which enables the construction of similarly-elegant encryption schemes. The new assumption, dubbed "Projected Mersenne", and a corresponding public-key encryption scheme are introduced in Section 4.3.4.

4.3.2 Preliminaries

Notations We denote by $\|x\|$ the Hamming weight of x, and by $\mathfrak{H}_{n,w}$ the set of all n-bit strings of Hamming weight h. The notation $x \xleftarrow{\$} X$ means that x is the result of uniformly sampling from the set X. Unless stated otherwise, log refers to the natural logarithm, whereas $\log_2$ to the base 2 logarithm. The symbols $\oplus$ and $\wedge$ stand for the binary XOR and AND operations, respectively. We denote the concatenation of x and y by $x\|y$. A q-ary error correcting code with block length d, dimension k, and minimum Hamming distance δ will be denoted $[d, k, \delta]_q$. Algorithms are given as input the (unary) representation of the security parameter λ. PPT stands for probabilistic polynomial time

4.3.3 Prior Work

4.3.3.1 The Mersenne Low Hamming Ratio Assumption

Recall that a Mersenne number is an integer of the form $2^n - 1$ for some n, and that a Mersenne prime is a Mersenne number which is prime.[8]

Definition 4.4 (Mersenne Low Hamming Ratio Search Problem) *Let $p = 2^n - 1$ be a Mersenne prime.[9] Given $n, w \in \mathbb{N}$ and $h \in \mathbb{Z}_p$, find $f, g \in \mathbb{Z}_p$ such that $\|f\| = \|g\| = w$ and $f/g = h \bmod p$, under the promise that such a couple exists.*

We will refer to the problem in Definition 4.4 as the MLHR problem. A brute-force attack on MLHR tries all possible couples $\{f, g\}$, which corresponds to a security level of

$$\lambda = \binom{n-1}{w-1} \approx w \cdot \log n \text{ bits.}$$

A quantum variant of this search, exploiting the generic speed-ups provided by Grover's algorithm, correspondingly halves λ. Should these attacks be optimal — as initially suggested by Aggarwal et al. — the MLHR would enable the construction of conceptually-simple and computationally-efficient post-quantum secure public-key cryptosystems.

4.3.3.2 The Aggarwal–Joux–Prakash–Santha cryptosystem (AJPS-1)

The original AJPS-1 scheme [AJPS17c] is defined by the following algorithms:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$. Outputs the public parameters $\mathsf{pp} = \{n, h\}$, so that in particular $p = 2^n - 1$ is prime. The choice of n and w is such that the cryptosystem achieves some λ-bit security level.

- $\mathsf{KeyGen}(\mathsf{pp}) \to \{\mathsf{sk}, \mathsf{pk}\}$. This algorithm generates the private and public keys. It samples $\{F, G\} \xleftarrow{\$} \mathfrak{H}^2_{n,w}$, and returns:

$$\begin{aligned} \mathsf{sk} &\leftarrow G \\ \mathsf{pk} &\leftarrow H = F/G \bmod p \end{aligned}$$

- $\mathsf{Enc}(\mathsf{pp}, \mathsf{pk}, m) \to C$. This algorithm takes as input the public parameters pp, the public key pk, and a message $m \in \{0, 1\}$. It samples $\{A, B\} \xleftarrow{\$} \mathfrak{H}^2_{n,w}$, and computes:

$$C \leftarrow (-1)^m (AH + B) \bmod p.$$

- $\mathsf{Dec}(\mathsf{pp}, \mathsf{sk}, C) \to \{\perp, 0, 1\}$. This algorithm computes $d \leftarrow \|G \cdot C \bmod p\|$ and returns:

$$\begin{cases} 0 & \text{if } d \leq 2w^2, \\ 1 & \text{if } d \geq n - 2w^2, \\ \perp & \text{otherwise.} \end{cases}$$

[8] In particular, if $2^n - 1$ is prime, then so is n.

[9] The use of a Mersenne *prime* is not necessary for the scheme's correctness, and in fact no attack is currently known if p is a Mersenne composite. The conservative choice of a Mersenne *prime* is recommended to avoid potentially unforeseen attacks exploiting the factorisation of p, cf. [AJPS17d, Section 8].

Table 4.1: Synoptic comparison of NTRU and AJPS-1.

	NTRU [HPS98]	AJPS-1 [AJPS17c]
Ciphertext space	$R = \mathbb{Z}[X]/(X^N - 1)$	$R = \mathbb{F}_2[X]/(X^p - 1)$, $p = 2^n - 1$
Message space	$m \in R_m$	$m \in \{0, 1\}$
Private key	$f \in R_f$	$F \in \mathfrak{H}_{n,w}$
Public key	$h = g/f_q \bmod q$ $(g \in R_g)$	$H = G/F$ $(G \in \mathfrak{H}_{n,w})$
Encryption	$c = prh + m \bmod q$ $(r \in R_r)$	$C = (-1)^m(AH + B)$ $(A, B \in \mathfrak{H}_{n,w})$
Decryption	$m = f_p^{-1}(fc \bmod q) \bmod p$	$m = \begin{cases} 0 & \text{if } \|FC\| \le 2w^2 \\ 1 & \text{if } \|FC\| \ge n - 2w^2 \\ \perp & \text{otherwise} \end{cases}$

Remark (Similarities with the NTRU cryptosystem) *Mersenne-based cryptosystems are reminiscent of NTRU [HPS98], which owes its name to the polynomial ring $R = \mathbb{Z}[X]/(X^N - 1)$ in which operations are performed.*[10] *In comparison, Mersenne-based cryptosystems work in $\mathbb{Z}_p \simeq \mathbb{F}_2[X]/(X^p - 1)$, where $p = 2^n - 1$ is prime.*[11]

Table 4.1 shows the parallels between the two cryptosystems. Notations for NTRU follow [HPS98], except R_f, R_g, R_r, R_m which are subsets of R having a prescribed number of coefficients set to ± 1.

The hard problem underlying NTRU is the Closest-Vector Problem (CVP) in some special convolution modular lattices; namely, f and g form a relatively short vector in a known lattice constructed from q and h. Parameters for NTRU must be chosen to resist lattice reduction attacks (e.g., [CS97, KF17]).

In the original version of their paper [AJPS17c], Aggarwal et al. consider and then dismiss two possible types of attack that could be better than brute force, inspired by the crpytanalysis of NTRU: a combinatorial meet-in-the-middle attack, which is claimed to fail due to the presence of "approximate collisions"; and a lattice-based attack, claimed to fail due to the presence of "parasitic vectors".

4.3.3.3 Beunardeau–Connolly–Géraud–Naccache attack.

The latter claim was rapidly challenged, when a faster experimental attack using lattice reduction was discovered by Beunardeau et al. [BCGN17c], which successfully recovered private keys for the initially suggested $\lambda = 128$ bit security level parameters. This attack runs in time $(2 + \delta + o(1))^{2w}$, for some very small constant $\delta > 0$ [dBDJdW17], thereby collapsing the security of the original AJPS construction to about $2w$ bits.

4.3.3.4 de Boer–Ducas–Jeffery–de Wolf attack.

The former claim was also challenged by de Boer et al. [dBDJdW17], who showed how to circumvent the "approximate collision" problem by leveraging locality-sensitive hashing. This results in a meet-in-the-middle attack, whose complexity is about

$$\binom{n/2}{w/2} \approx \binom{n}{w}^{1/2} \approx \frac{1}{2} w \log n.$$

[10] NTRU stands for N-th Degree Truncated Polynomial Ring Units.
[11] Bernstein et al. [BCLvV16] argue against the use of such rings for NTRU.

A quantum version of this algorithm has a runtime of

$$\binom{n/3}{w/3} \approx \binom{n}{w}^{1/3} \approx \frac{1}{6} w \log n.$$

Pointing out similar work for the related NTRU cryptosystem [Buh98], de Boer et al. conjecture that a combination of the MITM approach with lattice reduction could lead to an even faster attack, reminiscent for instance of Howgrave-Graham's [How07].

4.3.3.5 Aggarwal–Joux–Prakash–Santha with error correction (AJPS-ECC)

To answer these attacks, Aggarwal et al. proposed a new version of their cryptosystem [AJPS17d].

The new version accomodates larger parameters and also improves the cryptosystem's bandwidth. As it makes use of an error correction scheme ECC $= \{\mathcal{D}, \mathcal{E}\}$, we refer to it as AJPS-ECC (following [FN17]). The Setup algorithm is unmodified. The other algorithms are modified as follows:

- $\mathsf{KeyGen}(\mathsf{pp}) \to \{\mathsf{sk}, \mathsf{pk}\}$. Sample $\{F, G\} \xleftarrow{\$} \mathfrak{H}^2_{n,w}$, $R \xleftarrow{\$} \{0,1\}^n$ and return:

$$\begin{aligned} \mathsf{sk} &\leftarrow F \\ \mathsf{pk} &\leftarrow \{R, T\} = \{R, F \cdot R + G \bmod p\} \end{aligned}$$

- $\mathsf{Enc}(\mathsf{pp}, \mathsf{pk}, m) \to C$. Sample $\{A, B_1, B_2\} \xleftarrow{\$} \mathfrak{H}^3_{n,w}$, and compute

$$C \leftarrow \begin{cases} C_1 = A \cdot R + B_1 \bmod p \\ C_2 = (A \cdot T + B_2 \bmod p) \oplus \mathcal{E}(m) \end{cases}$$

- $\mathsf{Dec}(\mathsf{pp}, \mathsf{sk}, C) \to \{\perp, m\}$ is modified accordingly and returns

$$\mathcal{D}((F \cdot C_1 \bmod p) \oplus C_2)$$

An analysis of the parameter choices for ECC and for the cryptosystem, including some additional discussion irrelevant for our purpose, can be found in Aggarwal et al's updated paper[AJPS17d]. AJPS-ECC relies for security on *another assumption* that the hardness of MLHR search; however, Aggarwal et al. point out that slight modifications to Beunardeau et al.'s attack apply to this modified scheme and choose the parameters accordingly.

4.3.3.6 Ferradi–Naccache (AJPS-FN-BT)

An interesting collection of variants is described by Ferradi and Naccache [FN17]. Noticing that some of the random coins used during encryption may be recovered, Ferradi and Naccache suggest turning this into a feature, thereby increasing the cryptosystem's bandwidth. However, the security of most of these variants is left undiscussed. The core idea can be found in Ferradi and Naccache's "bivariate" variant AJPS-FN-BT2.

AJPS-FN-BT2 relies on the availability of an efficient function $\mathsf{Solve}_{x,y}$ which finds a low Hamming weight solution to a given Diophantine equation of the form $\alpha x + \beta y + \gamma = 0$ for given parameters α, β, γ. They suggest implementing this function as a heuristic-based backtracking algorithm. Using this, it becomes possible to recover the values A and B used during encryption, which have low Hamming weight. One possibility is to use A and B to design a key-encapsulation mechanism as follows:

- Setup and KeyGen are identical to those of AJPS-1, except that we additionally agree on a block cipher $\mathcal{F} : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$, a cryptographic hash function $\mathcal{H}_2 : \{0,1\}^* \to \{0,1\}^\lambda$, and a cryptographic hash function $\mathcal{H}_1 : \{0,1\}^* \to \mathfrak{H}^2_{n,w}$.

- $\mathsf{Enc}(\mathsf{pp}, \mathsf{pk}, m) \to C$ is modified as follows. Sample $r \xleftarrow{\$} \{0,1\}^\lambda$ and compute $\{A, B\} \leftarrow \mathcal{H}_1(r\|m)$. Then compute $k \leftarrow \mathcal{H}_2(A\|B)$. Finally, output

$$C = \{C_1, C_2\} = \{AH + B \bmod p,\ \mathcal{F}_k(r\|m)\}$$

- $\mathsf{Dec}(\mathsf{pp}, \mathsf{sk}, C) \to \{\perp, m\}$ is modified as follows: first recover

$$\{A, B\} \leftarrow \mathsf{Solve}_{x,y}\left[GC_1 = Fx + Gy \bmod p\right].$$

  In case of failure, return $\perp$. Otherwise, compute $k \leftarrow \mathcal{H}_2(A\|B)$, and recover $u \leftarrow \mathcal{F}_k^{-1}(C_2)$. If $\mathcal{H}_1(u) \neq \{A, B\}$ then return $\perp$. Otherwise return m.

The correctness of this scheme (and other variants in [FN17]) is not formally analysed but is backed by numerical simulations.

4.3.4 The Projected-Mersenne Cryptosystem

4.3.4.1 The Projected-Mersenne Assumption

We introduce the following problem:

Definition 4.5 (Projected-Mersenne Low Hamming Ratio Search Problem) *Let $p = 2^n - 1$ be a Mersenne prime. Given $n, w, d \in \mathbb{N}$, $M = 2^d - 1$, and $h \in \mathbb{Z}_p$, find $f, g \in \mathbb{Z}_p$ such that*

1. $\|g\| = w$
2. $\|f \wedge M\| = 1$
3. $f/g = h \bmod p$

under the promise that such a couple exists.

This search problem can be solved by brute-force enumeration much like MLHR, as it suffices to find g, i.e., find one in $\binom{n-1}{w-1} \approx 2^{w \log n}$ possibilities. We introduce the following assumption:

Definition 4.6 (α-Projected Mersenne Assumption) *The* α-projected Mersenne assumption[12] *states that given a Mersenne prime $p = 2^n - 1$, an integer a in $\mathsf{poly}(\lambda)$, any PPT distinguisher has a negligible chance to distinguish between R/G and R', where $R \xleftarrow{\$} \{0, \ldots, 2^\alpha - 1\}$, $G \xleftarrow{\$} \mathfrak{H}_{n,w}$, and $R' \xleftarrow{\$} \mathbb{Z}_p$.*

where the distinguishing advantage is defined as usual:

Definition 4.7 *For a PPT distinguisher $\mathcal{D}$ that outputs a bit $b \in \{0,1\}$, the distinguishing advantage to distinguish between two random variables X and Y is defined as:*

$$\Delta^{\mathcal{D}}(X;Y) = |\Pr[\mathcal{D}(X) = 1] - \Pr[\mathcal{D}(Y) = 1]|$$

[12] We choose to explicit only one parameter, namely the random numerator's size. The other parameter is the denominator's Hamming weight, which will be the same throughout our different variants, and therefore will not be explicitly noted.

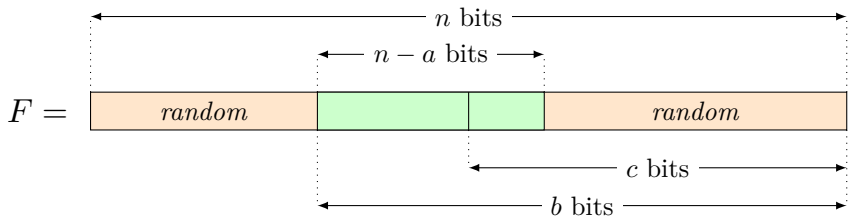

Figure 4.4: An illustration of the structure in F, as used in our KeyGen algorithm: a central region of size $n - a$ contains only a single set bit. Note that this figure and the following are not to scale, we give concrete parameters later.

4.3.5 Projected-Mersenne Encryption

We now describe our encyption scheme.

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$. Choose $p = 2^n - 1$ a Mersenne prime, and parameters w, a, b, c, d so as to achieve a λ-bit security level. Additional constraints on a, b, c, d to ensure correctness are discussed below. We also agree on an error-correcting code[13] $\mathrm{ECC} = (\mathcal{E}, \mathcal{D})$ with codewords of size d bits.

- $\mathsf{KeyGen}(\mathsf{pp}) \to \{\mathsf{sk}, \mathsf{pk}\}$. Sample $G \xleftarrow{\$} \mathfrak{H}_{n,w}$ and a random a-bit number R. Let $F \leftarrow R \cdot 2^b + 2^c \bmod p$. Note that, in general, $F \notin \mathfrak{H}_{n,w}$. An illustration of F's structure is given in Figure 4.4. The KeyGen algorithm returns

$$\begin{aligned} \mathsf{sk} &\leftarrow G \\ \mathsf{pk} &\leftarrow F/G \bmod p \end{aligned}$$

- $\mathsf{Enc}(\mathsf{pp}, \mathsf{pk}, m) \to C$. Sample $B \xleftarrow{\$} \mathfrak{H}_{n,w}$ and return $C \leftarrow \mathcal{E}(m) \cdot \mathsf{pk} + B \bmod p$.

- $\mathsf{Dec}(\mathsf{pp}, \mathsf{sk}, C) \to \{m, \perp\}$. First compute $D \leftarrow 2^{n-c} C \cdot \mathsf{sk} \bmod p$. This should be

$$\begin{aligned} D &= 2^{n-c} C \mathsf{sk} \\ &= 2^{n-c} \left(\mathcal{E}(m) \cdot H + B\right) \cdot G \bmod p \\ &= 2^{n-c} \mathcal{E}(m) F + 2^{n-c} BG \bmod p \\ &= 2^{n-c} \mathcal{E}(m) \cdot \left(2^b R + 2^c\right) + 2^{n-c} BG \bmod p \\ &= \mathcal{E}(m) + 2^{n+b-c} R \mathcal{E}(m) + 2^{n-c} BG \bmod p \end{aligned}$$

  Let $M = 2^d - 1$, then the algorithm outputs $\mathcal{D}(M \wedge D)$. Figure 4.5 illustrates this process.

4.3.5.1 Correctness

The correctness of this scheme is based upon two facts. The first is that we can appropriately choose a, b, c, d so that the first and second terms in the expanded expression of D are disjoint (as

[13] A possibility is to use BCH codes [Hoc59, BRC60] which are efficient and give fine control over the code's parameters, or Reed–Solomon codes [RS60] which are MDS.

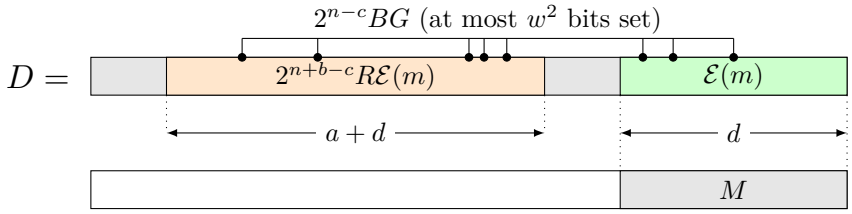

Figure 4.5: An illustration of the structure in D, as used in our Dec algorithm. For appropriately chosen parameters, projecting by M only retains a noisy version of $\mathcal{E}(m)$.

depicted in Figure 4.5). When this is the case, masking by M removes the $2^{n+b-c}R\mathcal{E}(m)$ term. The conditions for this to happen are easily found by inspecting Figures 4.4 and 4.5:

$$n \leq a + 2d \qquad \text{and} \qquad a + b - n < c < b$$

Assuming an ECC $\mathcal{E}$ is given with block length d, we can choose the following parameters, which correspond to maximising a:

$$a = n - 2d \qquad \text{and} \qquad b \xleftarrow{\$} \{0, \ldots, n\} \qquad \text{and} \qquad c = b - d \bmod p. \tag{4.1}$$

The second key fact is that BG has low Hamming weight, namely at most w^2. This is not affected by multiplication by a power of 2, and therefore $(2^{n-c}BD) \wedge M$ has Hamming weight at most $\eta = \min(d, w^2)$.[14] If $\mathcal{E}$ can correct at least η errors, then the decoding algorithm succeeds.[15,16]

4.3.5.2 Semantic Security

As described above, the cryptosystem is vulnerable to a trivial chosen ciphertext attack. Assume the attacker gets a challenge encryption C^* of m_b with $b \in \{0, 1\}$ being the challenge bit she has to guess. Indeed, we get

$$C^* = \mathcal{E}(m) \cdot \mathsf{pk} + B \bmod p$$

Thanks to the knowledge of the public key she will be able to recover the randomness (and break the semantic security). She computes the encrpytion with the randomness being set to 0.

$$C_0 = \mathcal{E}(m_0) \cdot \mathsf{pk} \bmod p$$
$$C_1 = \mathcal{E}(m_1) \cdot \mathsf{pk} \bmod p$$

She then subtract the ciphertexts:

$$C^* - C_0 = (\mathcal{E}(m_b) - \mathcal{E}(m_0)) \cdot \mathsf{pk} + B \bmod p$$
$$C^* - C_1 = (\mathcal{E}(m_b) - \mathcal{E}(m_1)) \cdot \mathsf{pk} + B \bmod p$$

[14] If $\mathcal{E}$ is a linear code, then **??** implies $w^2 \leq d - k + 1$, or in other terms, $k \leq d - w^2 + 1$. Therefore, $\eta = d$.

[15] Even in the case where $\mathcal{E}$ can only correct $t < \eta$ errors, there is still a non zero probability that Dec successfully recovers m, which corresponds to the events where all $\eta - t$ bits lay between 2^d and $2^n - 1$; this probability is roughly $(1 - 2^{d-n})^{\eta - t}$.

[16] In fact, the noise considered here is *additive*, and may result in more than w^2 bits being affected due to carry propagation. We may choose a stronger error correction capacity to account for such unlikely events.

One of these equations is equal to B and has low hamming weight, which allows the adversary to distinguish. This phenomenon is common to other cryptosystems, such as the McEliece code-based encryption scheme. Therefore our system is inherently non CCA-secure. We can treat this problem by usign a key encapsulation mechanism which encrypts a random message, and derive the randomness used in the encryption by hashing the random message. AJPS uses the same method, but they only need it for chosen ciphertext attacks.

4.3.6 Key Encapsulation Mechanism

We can treat this problem by usign a key encapsulation mechanism which encrypts a random message, and derive the randomness used in the encryption by hashing the random message. Since the same is done in AJPS, we use the same notation to define KEM to ease comparison.

Definition 4.8 (Key encapsulation Mechanism) *A KEM comprises three algorithms: the key generation algorithm* KeyGen*, the encapsulation algorithm* Encaps*, and the decapsulation algorithm Decaps, and a key space* K*. The KeyGen algorithm outputs a public-key* pk*, and a secret key* sk*. The encapsulation algorithm* Encaps *takes as input a public key* pk *to produce a ciphertext* C *and a key* $K \in K$*. The decapsulation algorithm* Decaps *takes as input a ciphertext* C *and* sk*, and outputs a key* K_0 *or a special symbol* $\perp$ *indicating rejection. We say that the KEM is* $(1-\delta)$*-correct if* $Pr[Decaps(\mathsf{sk}, C) = K : (C, K) \xleftarrow{\$} \mathsf{Encaps}(\mathsf{pk})] \geq 1 - \delta$ *, where the probability is over the randomness of* pk, sk *and the encapsulation algorithm. Again, we denote the security parameter by* λ*. All other parameters including key lengths and ciphertext size are given as polynomially bounded functions of* λ*.*

Definition 4.9 (Key Encapsulation Mechanism Semantic Security) *The key-encapsulation mechanism* $(\mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$ *is said to be semantically secure if for any probabilistic polynomial time distinguisher, given the public key* pk*, the advantage for distinguishing* (C, K_0) *and* (C, K_1)*, where* $(C, K_0) \xleftarrow{\$} Encaps(\mathsf{pk})$ *and* K_1 *is uniform and independent of* C *is negligible in* λ*.*

Definition 4.10 (KEM Semantic Security Under Chosen Ciphertext Attack) *The key-encapsulation mechanism KEM =* $(\mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$ *is said to be secure under chosen ciphertext attacks if for any probabilistic polynomial time distinguisher that is given access to the decapsulation oracle and the public key* pk*, the advantage for distinguishing* (C, K_0) *and* (C, K_1)*, where* $(C, K0) \xleftarrow{\$} \mathsf{Encaps}(pk)$ *and* K_1 *is uniform and independent of* C *is negligible in* λ *under the assumption that the distinguisher does not query the oracle with* C*.*

We now describe our KEM. Its purpose is to avoid encrpytion with randomness set to 0 as in the attack against our encyption scheme. To achieve this, we will get a key at random, and use it with random oracle to get randomness to encrpyt the key. Let H be a random oracle from the key space $\{0,1\}^\lambda$ to the random tape of our encryption scheme (ie. low hamming weight numbers).

- KeyGen is the same as the encryption scheme.
- $\mathsf{Encaps}(\mathsf{pk})$ draws uniformly at random a key K, produces the ciphertext $\mathcal{E}(K) \cdot \mathsf{pk} + H(K)$ and the key K.
- $\mathsf{Decaps}(\mathsf{sk}, C)$ produces the key $K' = \mathsf{Dec}(\mathsf{sk}, C)$, reencrypts its own randomness $C' = \mathcal{E}(K') \cdot \mathsf{pk} + H(K')$, and checks $C = C'$. If $C \neq C'$ output $\perp$, else output K.

Our KEM is trivially $1-\delta$-correct with δ negligible in λ from the correctness of the encryption scheme.

4.3.7 Security analysis

In this section we show that the KEM's semantic security (Definition 4.9) relies on the Projected-Mersenne Assumption (Definition 4.6), and discuss the attacks that can apply to this assumption.

4.3.7.1 Semantic Security of the Key Encapsulation Mechanism

Theorem 4.1 (Semantic Security) *Our KEM is semantically secure (Definition 4.9) under the a-Projected-Mersenne Assumption (Definition 4.6), with a as defined in 4.4.*

Before proving Theorem 4.7 we give a few lemmas that we will use later. The main thing to prove is that the public key is indistinguishable from random, which is shown in Lemma 4.5.

Lemma 4.2 *Given a PPT computable function f on two random variables X and Y, if there is no PPT distinguisher D that can distinguish between X and Y with non negligible advantage, then there is no PPTdistinguisher D' that can distinguish between X and Y with non negligible advantage.*

The proof of Lemma Lemma 4.2 is well known and can be found easily.

Lemma 4.3 *If $x \in \mathbb{Z}_p^*$ is of hamming weight 1, then x^{-1} is of hamming weight 1.*

Proof: Since the multiplicative group $\mathbb{Z}_p^*$ is of order $p-1$, we have $x^{-1} = x^{p-2}$. Since we work modulo a Mersenne prime, multipliying by a number of hamming weight 1 is equivalent to shifting. Therefore taking the product of two numbers of hamming weight 1 is of hamming weight 1. Since x^{-1} is the product of numbers of hamming weight 1, it is itself of hamming weight 1. □

Lemma 4.4 [17] *Every bit of the public key is the sum of an average of $a/2$ bits of R omitting the contribution of the carry*[18].

Proof: First we notice that although $1/G$ is not random looking (as shown in [BCGN17c]), it has a random hamming weight ($n/2$ on average). Indeed $1/G = G^{p-2}$. So its low hamming weight increases since we perform approximately n squarings to come to the inverse. Second, since R is of size a, every bit of the public key is influenced by a copy of R if a bit in the a bits preceding it is set to 1 in $1/G$. Combining the two observations gives the result. □

Lemma 4.5 *Assuming the a-Projected-Mersenne Assumption (Definition 4.6), given a Mersenne prime $p = 2^n - 1$, an integer a output by* Setup*, any PPT disinguisher has a negligible chance to distinguish between*

$$\mathsf{pk} \text{ and } R'$$

where pk *is the public key generated from* KeyGen *and* $R' \xleftarrow{\$} \mathbb{Z}_p$.

[17] This lemma also gives an intuition about our security. For example doing the same computation on the public key of AJPS-1, we would get a much smaller number of contributions (17/2 for 128-bits of security).

[18] We omit the the carry's influence to simplify analysis : we only need a bound.

Proof: Letting $\mathsf{pk} = 2^b \cdot R/G + 2^c \cdot 1/G \bmod p$. Aplpying 4.2 with f being the division by 2^b, the adversary tries to distinguish $\text{chal} = R/G + \frac{2^{c-b}}{G}$. By 4.3, 2^{c-b} is of hamming weight 1. So we can rewrite $\text{chal} = \frac{R+2^i}{G}$ for some i. Applying 4.4, with overwhelming probability[19], every additional 1 coming from $2^i/G$ will be in a copy of R. Since there are as many copies of R as there are 1s coming from $2^i/G$ (this number being $||1/G||$, the hamming weight of $1/G$), we can rewrite the challenge:

$$\text{chal} = \sum_{i=0}^{||1/G||} 2^{u_i} R_i$$

for some u_i

with $R_i = R + 2^x$ with $x \xleftarrow{\$} 0, a-1$. [20]We now show that we can replace one R_i by R. Since x is taken at random with overwhelming probability an R_i is indistinguishable from R since their distributions are statistically close[21]. Indeed $R \xleftarrow{\$} [0, 2^a - 1]$ and $R_i \xleftarrow{\$} [2^x, 2^x + 2^a - 1]$. X is on average $a/2$, and the statistical distance is $a/2$. With overwhelming probability the statistical distance is negligible.
Since there are polynomially many R_is, one can replace them one by one to get from chal to R/G while staying indistinguishable. □

Lemma 4.6 *Assuming the a-Projected Mersenne Assumption (Definition 4.6), given a Mersenne prime $p = 2^n - 1$, an integer a, any PPT distinguisher has a negligible chance to distinguish between G/R and R' where $R \xleftarrow{\$} \{0, \ldots, 2^a - 1\}$, $G \xleftarrow{\$} \mathfrak{H}_{n,w}$ and $R' \xleftarrow{\$} \mathbb{Z}_p$.*

Proof: This is shown by applying Lemma 4.2 with f being the modular inversion. □ □

We can now prove our main theorem.

Proof: [of Theorem 4.7] For any PPT distinguisher D, we have by the triangle inequality:

$$\begin{aligned}\Delta^D((\mathsf{pk}, c); (\mathsf{pk}, R)) \leq\ & \Delta^D((\mathsf{pk}, c); (R, \mathcal{E}(m)R + B)) \\ & + \Delta^D((R, \mathcal{E}(m)R + B); (R, R')) \\ & + \Delta^D((R, R'); (\mathsf{pk}, R))\end{aligned}$$

where pk is a public key genrated from KeyGen, m is drawn at random, $B \xleftarrow{\$} \mathfrak{H}_{n,w}$, $c = \mathcal{E}(m)\mathsf{pk} + B$ is a ciphertext, and $R, R' \xleftarrow{\$} \mathbb{Z}_p$. This suffice to show the semantic security.

We now have to show that the three bounding terms are negligible:

- $\Delta^D((\mathsf{pk}, c); (R, \mathcal{E}(m)R + B))$. By Lemma 4.5 we have that pk is indistinguishable from a random, applying Lemma 4.2 with $f(X) = (X, \mathcal{E}(m)X + B)$ with m a message drawn at random and $B \xleftarrow{\$} \mathfrak{H}_{n,w}$.

[19]Here we see that the reduction is not tight : for 128-bits of security the probability is $1 - 2^{69}$

[20]The intuition of the security of our scheme is in this lemma. Indeed the message is written using this 2^c in the public key. We basically show that dividing by G makes that the 2^c that will contain the message covered in randoms R.

[21]Once again the reduction is not tight. One could try to make it tighter by choosing which R goes with which 1, to minimize x

- $\Delta^D((R, \mathcal{E}(m)R + B); (R, R'))$. By Lemma 4.6 we have that B/R is indistinguishable from a random, applying Lemma 4.2 with $f(X) = (R, X \cdot R' + \mathcal{E}(m))$ we get that

$$\Delta^D((R, \mathcal{E}(m)R + B); (R, R \cdot R' + \mathcal{E}(m)))$$

  is negligible. By observing that $R \cdot R' + \mathcal{E}(m)$ is uniformly distributed, $\Delta^D((R, \mathcal{E}(m)R + B); (R, R'))$ is negligible.

- $\Delta^D((R, R'); (\mathsf{pk}, R))$. This is shown to be negligible by Lemma 4.5 and Lemma 4.2 with $f(X) = (X, R)$ where $R \xleftarrow{\$} \mathbb{Z}_p$

This concludes the proof. □

4.3.7.2 Chosen Ciphertext Security

We now show that security holds against chosen ciphertext (Definition 4.10). For this we only need to show that the queries will not help the adversary, and then conclude with semantic security. The key point is that the decapsulation oracle to which the adversary has access will answer with overwhelming probability $\perp$ if the ciphertext are not made 'honnestly'. Since once the key is fixed and the random oracle called with it the encapsulation procedure is deterministic, the adversary can simulate it easily.

Theorem 4.7 (Semantic Security under Chosen Ciphertext Attack) *Our KEM is semantically secure under chosen ciphertext attack (Definition 4.10) under the a-Projected-Mersenne Assumption (Definition 4.6), with a as defined in 4.4.*

Proof: There are two cases, either the random oracle was called on the answer of a query to the decapsluation mechanism, or it was not (ie. the adversary tries 'malicious' queries).

- If the adversary queries Decaps with a ciphertext, and gets K, which he queried to the random oracle, then he can simulate the query easily.

- The probability of the second event is $Pr[K \leftarrow \mathsf{Decaps}(C \xleftarrow{\$} \mathsf{Adv})] = Pr[\mathcal{E}(\mathsf{Dec}(C)) \cdot \mathsf{pk} + H(K) = C \xleftarrow{\$} \mathsf{Adv}] \leq \binom{n}{w}$ since it requires guessing the random oracle response. The second event is therefore negligible.

□

4.3.8 Attacks on the Underlying Assumption

Due to the similarity with AJPS, it is natural to discuss the attacks that are most efficient against it, and to measure to what extent such attacks apply to our new construction.

4.3.8.1 Lattice Attacks.

As for the other versions of Mersenne encryption Beunardeau et al.'s attack [BCGN17c], is also applicable to our scheme and has an experimental cost of finding the right partitions for the low hamming weight. We then set $w = lambda$.

4.3.8.2 Brute Force Attacks.

A brute force exhaustion of sk is always possible, and takes an effort of $\binom{n-1}{w-1}$. Thus the bare minimum requirement for security is that this quantity exceeds 2^λ.

4.3.8.3 Meet-in-the-Middle Attacks.

The key result of de Boer et al. is backed by [dBDJdW17, Lemma 3.1], which assumes that F has constant small Hamming weigth w. Without this assumption, the likelihood that a locality-sensitive hash function $\mathcal{H}$ is "good" for g does not have a lower bound, so that it the meet-in-the-middle attack is no longer guaranteed to succeed. We can compare simulations from [dBDJdW17, Appendix A.1] with the same experiment on our scheme, which shows that de Boer et al.'s Heuristic 3.2, which is reasonable against AJPS-1, does not hold for our scheme.

4.3.9 Conclusion

Altough our scheme is vulnerable to the same attacks as AJPS, it is simpler in the sense that we do not need two ciphertexts. We hoped that the size of the random R would have twart our lattice attack. This is not the case experimentally, but since the analysis of our attack is not complete, there is still hope that our scheme is of interest. We also think that it is simpler to analyse our assumption than AJPS's.

Chapter 5

Physical Security and Information Theory

Contents

5.1 Introduction

In this chapter, we look at non standard physical models.

In the traditional cryptographic model, the adversary called Eve is an eavesdropper, meaning that she stands between the two (or more) honest parties, and is able to hear (or intercept and modify) conversations between those parties. This is called the black box model, since every computation made by the parties are non-observable, and the result of those computations seems to come from a block box to Eve.

In Section 5.2 we look at a stronger adversary, which is able to get and modify some information from the internal computations of the parties. This model is called grey box since the information she gets is noisy.

This model is useful when cryptographic computations are done on the field as for smart cards, or IoT devices.

An even stronger model named white box was proposed more recently in 2002 in [CEJvO02]. In this model the adversary is all mighty, to model insecure execution environment, such as smart phones, but this is outside of the scope of this thesis.

In Section 5.3 we look at parties that are able to communicate using means that obeys to the laws of quantum physic. In traditional communications, the information is physically sent with a lot of redundancy, to average noise and quantum effects. An undesirable consequence is that an adversary can eavesdrop without being noticed since there is enough information to be split between the adversary and the receiver. For example, one can measure the voltage between two points in an electronic circuit, without interfering with the circuit functionality. Therefore, key exchange requires cryptographic techniques to prevent the key to be intercepted. Quantum key exchange sends keys in the clear but has so little information that anyone listening would prevent the recipient from getting the information thanks to observer effects happening in quantum mechanics. This can be achieved using optic fiber and the polarization of photons to represent the key. Then every measurement from the adversary randomly changes the polarization of the photons.

5.2 A New Differential Fault Analysis on PRIDE: from Theory to Practice

Abstract

PRIDE is one of the most efficient lightweight block cipher proposed so far for connected objects with high performance and low-resource constraints. In this paper, we describe the first ever complete Differential Fault Analysis against PRIDE. We describe how fault attacks can be used against implementations of PRIDE to recover the entire encryption key. Our attack has been validated first through simulations, and then in practice on a software implementation of PRIDE running on a device that could typically be used in IoT devices. Faults have been injected using electromagnetic pulses during the PRIDE execution, and the faulty ciphertexts have been used to recover the key bits. We also discuss some countermeasures that could be used to thwart such attacks. This is joint work with Benjamin Lac, Anne Canteaut, Jacques Fournier, Renaud Sirdey. This is an extended version of a work presented at the 11th International Conference Risks and Security of Internet and Systems, CRiSIS 2016 and published as [LBC+16]

5.2.1 Introduction

With the emergence of the Internet of Things (IoT), new cryptographic primitives are needed to suit the high performance, low power and low resource constraints of IoT devices. Ciphers like AES, which are good enough for devices like smart cards, do not satisfy the constraints of devices like RFID tags or nodes in sensor networks. During the past years, several lightweight block ciphers have been proposed, like for example PRESENT [BKL+07], PRINCE [BCG+12], SIMON [BSS+15] or SPECK [BSS+15]. Among those, the NSA proposal SPECK is a highly efficient software-oriented cipher, but it does not have any 'linear diffusion layer' implying that it requires a huge number of rounds to guarantee an appropriate security level. In order to keep a small number of rounds, the PRIDE cipher [ADK+14] exploits an optimal linear layer which provides a high diffusion and has highly efficient implementations. Although hardware implementations are more efficient in terms of clock cycles than software implementations, design and study of software-oriented ciphers is nevertheless important since these implementations are used in practice because they are less expensive and more flexible than hardware implementations. To date, when looking at software implementations, PRIDE is one of the most efficient lightweight cryptographic ciphers as shown the performance comparisons given in [ADK+14, BS15]. This led us to study the security provided by PRIDE and its resistance to malicious attacks. In terms of security, two of the differential attacks proposed so far in the literature do not allow to recover the entire key [YHS+15, ZWWD14], while a third one [DC14] does achieve this but under stringent conditions. Since PRIDE is to be used in IoT devices in pervasive environments, we ought to also look at implementation-related issues. In that respect, we propose in this paper the first Differential Fault Analysis (DFA) on PRIDE. DFA is a particular physical attack, in which we compare the results of a correct computation to one which has been disturbed at a precise time, in order to infer information about the key bits used in the algorithm. It is closely related to differential cryptanalysis, but much more efficient since it exploits differential characteristics on very few rounds only.

In this paper, we first present PRIDE before describing the theoretical DFA using different fault models. We then validate our hypotheses and equations using data onto which fault models have been 'simulated'. In order to validate the practical feasibility of our attack, we used electromagnetic pulses to inject faults during the execution of the PRIDE cipher running on an off-the-shelf chip embedding an ARM Cortex-M3 micro-controller and applied our DFA on the corrupted results

obtained. So as to taking advantage of the 32-bit architecture of the micro-controller, we have implemented PRIDE in ARM assembly language. Thereby, we show the practical feasibility of our attack from 32-bit random faults. Finally, we discuss countermeasures that can be implemented to thwart such attacks before concluding the paper with some perspectives.

5.2.2 Fault attacks against cryptographic algorithms

5.2.2.1 Physical attacks

Unlike mathematical attacks which target the actual definition of a cryptographic cipher, physical attacks target the way the cipher is implemented. Physical attacks can be divided into two categories: invasive and non-invasive ones. In this paper, we further focus on non-invasive techniques which mainly consist either in analysing side-channel information leakages or in injecting faults during a cryptographic computation.

Side-Channel Analyses [KS05], [MOP07] exploit the fact that some physical values or "side channels" such as the power consumption [KJJ99b], the electromagnetic radiation [GMO01], [QS01] or the computation time [DKL+98], [Koc96] of an integrated circuit depends on the operations and data manipulated during a given computation. Information about the internal processes of the chip and the data it is manipulating can be derived by observing such external physical characteristics. Such analyses can be quickly mounted with cheap equipment, without altering the physical integrity of the circuit. This dependency between the side channels and the internal computations can be analysed to infer information about the data manipulated using mathematical tools like correlation [BCO04], mutual information [GBTP08], variance [MDF+09] or entropy [MGDF10] or using architecture-dependant behaviors such as cache accesses [BZB+05], [Pag02, Pag04] or branch predictions [AeKKS07, AKS07].

5.2.2.2 Fault attacks

Fault Attacks, introduced in [BDL97], consist in disturbing the behavior of the circuit in order to alter the correct progress of the cipher. The faults are injected into the device by various means such as light pulses [SA03], laser [Sko05], clock glitches [ADN+10], spikes on the voltage supply [BS03] or electromagnetic (EM) perturbations [DDRT12]. Some of those techniques, like the one using a laser, are invasive requiring the "decapsulation" of the chip using mechanical or chemical means. Lasers allow to target one bit in a given register if well manipulated. However, it is a very costly means of injection. Other techniques are not invasive such as glitches (power, clock, electromagnetic). Clock and voltage glitches disturb the whole component, and many injections have to be made before getting the faults required by theoretical attacks. EM glitches, on the other hand, allow having relatively high spatial and temporal precisions using equipment at "affordable costs" [DDRT12].

One of the objectives of fault attacks, especially when considering cryptographic ciphers, is to perform a Differential Fault Analysis (DFA). DFA, originally described in [BS97], consists in retrieving a cryptographic key by comparing the correct ciphertexts with the faulty ones. DFA techniques have been described and applied to most publicly known cryptographic ciphers going from symmetric-key algorithms like the DES [BS97] or the AES [SLIO12] to asymmetric algorithms like RSA [BDL97] or even more complex schemes like pairing-based cryptography [LFG13]. In the particular field of lightweight cryptography, differential fault attacks have been proposed against ciphers like PRESENT [ZWG11] (used in conjunction with a cube attack), SPECK [TBM14] (although about a hundred faults are needed which is way more than usual), TRIVIUM [MBB11] or PRINCE [SH13]. The latter PRINCE block cipher has an SPN structure similar to PRIDE, and in that respect, the DFA proposed in [SH13] is quite similar to the one proposed hereafter:

in our case, the attack is not only adapted to the PRIDE cipher but has also been validated in practice on an embedded device running PRIDE.

DFA techniques are very efficient in retrieving the keys used during a cryptographic computation, usually requiring a few executions only. It is also quite complex to devise physical countermeasures against such attacks because of the diversity of the possible injection methods and because the usually deployed countermeasures (like masking, redundancy, error-correcting codes etc) have a serious impact on the performance of the targeted cryptographic cipher. For all those reasons, in our approach of analysing the security of implementations of PRIDE, we decided to first focus on its resistance against fault attacks in order to identify possible attack paths and devise the most efficient countermeasures in order to keep the high performance characteristics of the original cipher.

5.2.3 The PRIDE block cipher

PRIDE is an iterative block cipher composed of 20 rounds and introduced by Albrecht & al. [ADK+14] in 2014. It takes as input a 64-bit block and uses a 128-bit key $k = k_0||k_1$. The first 64 bits k_0 are used for pre- and post-whitening. The last 64 bits k_1 are used by a key schedule to produce the subkeys $f_r(k_1)$ for each round r. The key schedule simply adds round-constants to parts of the key.

We denote k_{1_i} the i-th byte of k_1 then

$$f_r(k_1) = k_{1_0}||g_r^{(0)}(k_{1_1})||k_{1_2}||g_r^{(1)}(k_{1_3})||k_{1_4}||g_r^{(2)}(k_{1_5})||k_{1_6}||g_r^{(3)}(k_{1_7})$$

for round r with

$$\begin{aligned} g_r^{(0)}(x) &= (x + 193r) && \mod 256 \\ g_r^{(1)}(x) &= (x + 165r) && \mod 256 \\ g_r^{(2)}(x) &= (x + 81r) && \mod 256 \\ g_r^{(3)}(x) &= (x + 197r) && \mod 256 \end{aligned}$$

In this paper, $X[n]$ denotes the n-th nibble (4 bits) of a binary word X while $X\{b\}$ denotes its b-th bit. Moreover, the bits and nibbles are numbered from left to right starting from 0. The following notation is used for the intermediate values of the state within the round function $\mathcal{R}$ of PRIDE (see Figure 5.2):

I_r	the input of the r-th round
X_r	the state after the key addition layer of the r-th round
Y_r	the state after the substitution layer of the r-th round input
Z_r	the state after the permutation layer of the r-th round
W_r	the state after the matrix layer of the r-th round
O_r	the output of the r-th round

The r-th round, $1 \leq r \leq 19$, of PRIDE is then composed of the following steps (see Figure 5.2).

i. Apply the inverse permutation layer $\mathcal{P}^{-1}$ given in Appendix 5.2.8.1 to $f_r(k_1)$ and XOR the permuted round subkey to the input state: $X_r = I_r \oplus \mathcal{P}^{-1}(f_r(k_1))$,

ii. Apply the S-box $\mathcal{S}$ given in Table 5.1 to each of the 16 nibbles of X_r (i.e. apply the substitution layer $\mathcal{S}-\mathsf{layer}$ to X_r): $Y_r = \mathcal{S}-\mathsf{layer}(X_r)$,

iii. Apply the permutation layer $\mathcal{P}$ to Y_r: $Z_r = \mathcal{P}(Y_r)$,

iv. Multiply vector $\begin{pmatrix} Z_r\{16i\} \\ \vdots \\ Z_r\{16i+15\} \end{pmatrix}$ by $\mathcal{L}_i$ in Appendix 5.2.8.1 for $i \in \{0, \cdots, 3\}$:

$$W_r = \mathcal{L}_0 \begin{pmatrix} Z_r\{0\} \\ \vdots \\ Z_r\{15\} \end{pmatrix} ||\mathcal{L}_1 \begin{pmatrix} Z_r\{16\} \\ \vdots \\ Z_r\{31\} \end{pmatrix} ||\mathcal{L}_2 \begin{pmatrix} Z_r\{32\} \\ \vdots \\ Z_r\{47\} \end{pmatrix} ||\mathcal{L}_3 \begin{pmatrix} Z_r\{48\} \\ \vdots \\ Z_r\{63\} \end{pmatrix},$$

v. Apply the inverse permutation $\mathcal{P}^{-1}$ to W_r: $O_r = \mathcal{P}^{-1}(W_r)$.

Table 5.1: S-box of the block cipher PRIDE

x	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xa	0xb	0xc	0xd	0xe	0xf
$\mathcal{S}(x)$	0x0	0x4	0x8	0xf	0x1	0x5	0xe	0x9	0x2	0x7	0xa	0xc	0xb	0xd	0x6	0x3

For the final round, denoted by $\mathcal{R}'$, only the first two steps are applied.

In order to encrypt a plaintext M, the cipher applies $\mathcal{P}^{-1}$ to M, then performs an XOR between the result and k_0. It then applies the 20 rounds as previously described and performs an XOR with k_0 again. Finally, $\mathcal{P}$ is applied to the result to obtain the ciphertext C. Figure 5.1 shows the general structure of PRIDE.

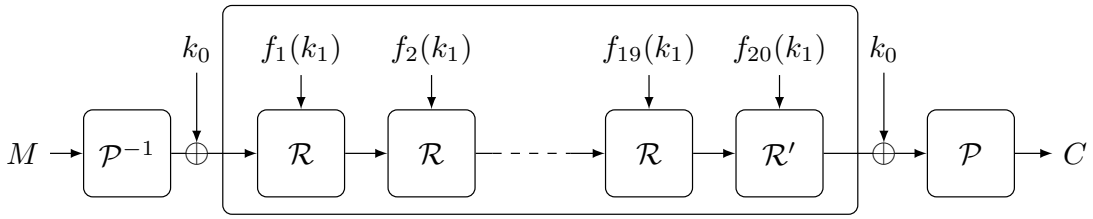

Figure 5.1: The structure of PRIDE

The PRIDE round function $\mathcal{R}$ is depicted on Figure 5.2.

5.2.4 Differential Fault Analysis of PRIDE

In this subsection, we present a technique adapted from the proposed attack in [SH13] to retrieve the secret key using fault injections on PRIDE computations. Our analysis aims at minimizing the number of fault injections needed. We use ideal fault models, and we describe how to exploit them to retrieve the key.

5.2.4.1 General principle

Despite their similarities, a DFA is different from a classical differential analysis. Indeed, for the latter, the differences must be injected into the input of the cipher while for a DFA they can be injected whenever the attacker wants. The DFA that we propose in this paper also differs from most classical DFA since it is not based on statistical methods: it is deterministic.

The attack is composed of two stages, one consists in corrupting data manipulated in the penultimate round to retrieve k_0 and the other in attacking the antepenultimate round to retrieve

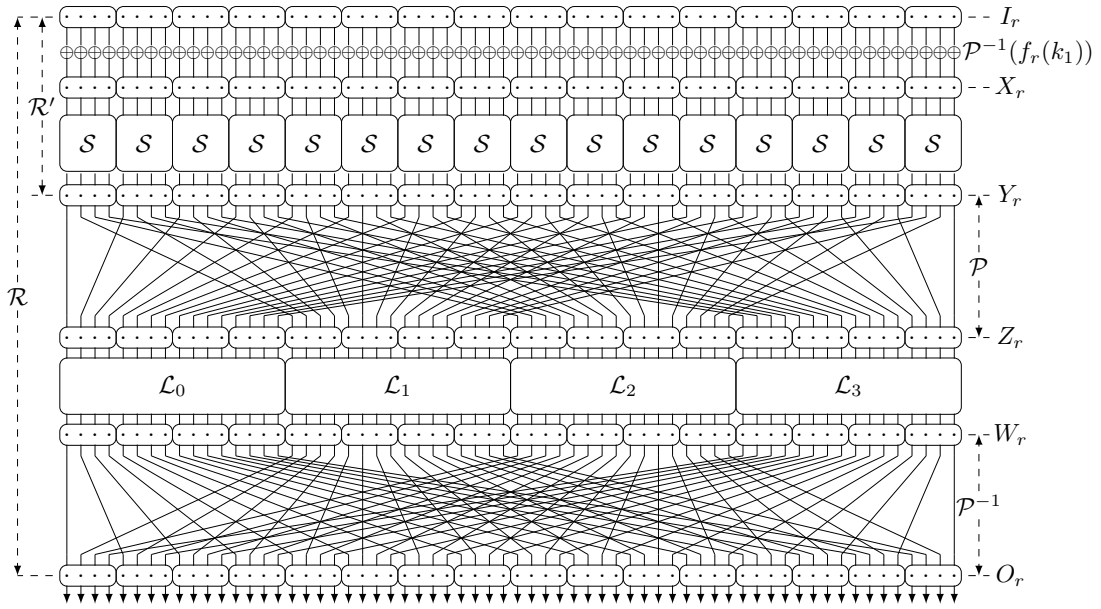

Figure 5.2: The PRIDE round function

k_1. The general structure of the attack is to exploit the diffusion of a 16-bit word within the inverse permutation layer in order to get a known 4-bit difference at the input of each S-box on the following round. Together with the knowledge of the output difference of each S-box, which are derived from the correct and faulty ciphertexts, C and C^*, this allows us to retrieve information about the key. To this end, we exploit the difference distribution table of the PRIDE S-box given in Appendix 5.2.8.2. Indeed, obtaining information on k_0 is possible from the following equation:

$$\Delta X_{20} = \mathcal{S}\text{−}\mathsf{layer}^{-1}(\mathcal{P}^{-1}(C) \oplus k_0) \oplus \mathcal{S}\text{−}\mathsf{layer}^{-1}(\mathcal{P}^{-1}(C^*) \oplus k_0),$$

where $\mathcal{S}\text{−}\mathsf{layer} = \mathcal{S}\text{−}\mathsf{layer}^{-1}$ denotes the substitution layer. We can use this equation for each nibble $0 \leq i \leq 15$:

$$x = \mathcal{P}^{-1}(C)[i] \oplus k_0[i] \text{ and } y = \mathcal{P}^{-1}(C^*)[i] \oplus k_0[i] \text{ satisfy}$$

$$x \oplus y = \Delta Y_{20}[i] = \mathcal{P}^{-1}(\Delta C)[i] \text{ and } \mathcal{S}^{-1}(x) \oplus \mathcal{S}^{-1}(y) = \Delta X_{20}[i].$$

From the knowledge of a nonzero input difference $\Delta Y_{20}[i]$ and of an output difference $\Delta X_{20}[i]$ for $\mathcal{S}^{-1}$, we deduce 2 or 4 candidates for the input value x, because the differential uniformity of $\mathcal{S}^{-1}$ equals 4 (see the difference distribution table in Appendix 5.2.8.2). Moreover, Proposition 5.1 enables us to exhibit pairs of differentials for the S-box which are simultaneously satisfied for a single element. The proof of this proposition is given in Appendix 5.2.8.2.

Proposition 5.1 *Let $\mathcal{S}$ be an n-bit S-box with differential uniformity 4. Let (a_1, b_1) and (a_2, b_2) be two differentials with $a_1 \neq a_2$ such that the system of two equations*

$$\mathcal{S}(x \oplus a_1) \oplus \mathcal{S}(x) = b_1 \tag{5.1}$$

$$\mathcal{S}(x \oplus a_2) \oplus \mathcal{S}(x) = b_2 \tag{5.2}$$

has at least two solutions. Then, each of the three equations (5.1), (5.2) and

$$\mathcal{S}(x \oplus a_1 \oplus a_2) \oplus \mathcal{S}(x) = b_1 \oplus b_2$$

has at least four solutions.

In other words, if we can find two differentials (a_1, b_1) and (a_2, b_2) such that one out of the three entries in the difference distribution table (a_1, b_1), (a_2, b_2) and $(a_1 \oplus a_2, b_1 \oplus b_2)$ equals to 2, then we can guarantee that the input satisfying these two differentials simultaneously is unique.

Note: if one of the three equations does not have any solution, then the system of two equations (5.1) and (5.2) does not have any solution neither.

Once k_0 has been recovered (we will see in the next parts some strategies to achieve this end), X_{20} and X^*_{20} can be computed from the ciphertexts C and C^*. Let $\mathcal{L}$ denote the whole linear layer, i.e.,

$$\mathcal{L} = \mathcal{P}^{-1} \circ \begin{pmatrix} \mathcal{L}_0 & 0 & 0 & 0 \\ 0 & \mathcal{L}_1 & 0 & 0 \\ 0 & 0 & \mathcal{L}_2 & 0 \\ 0 & 0 & 0 & \mathcal{L}_3 \end{pmatrix} \circ \mathcal{P}.$$

Then ΔY_{19} can be computed and the following equation

$$\begin{aligned} \Delta X_{19} = \quad & \mathcal{S}\text{−}\mathsf{layer}^{-1}(\mathcal{L}^{-1}(\mathcal{S}\text{−}\mathsf{layer}^{-1}(\mathcal{P}^{-1}(C) \oplus k_0) \oplus \mathcal{P}^{-1}(f_{20}(k_1)))) \\ & \oplus \mathcal{S}\text{−}\mathsf{layer}^{-1}(\mathcal{L}^{-1}(\mathcal{S}\text{−}\mathsf{layer}^{-1}(\mathcal{P}^{-1}(C^*) \oplus k_0) \oplus \mathcal{P}^{-1}(f_{20}(k_1)))), \end{aligned}$$

allows the attacker to recover $\mathcal{P}^{-1}(f_{20}(k_1))$ and therefore k_1, with the same method but from fault injections in the 18-th round. Indeed, for $0 \leq i \leq 15$:

$$\begin{gathered} x = \mathcal{L}^{-1}(\mathcal{S}\text{−}\mathsf{layer}^{-1}(\mathcal{P}^{-1}(C) \oplus k_0) \oplus \mathcal{P}^{-1}(f_{20}(k_1)))[i] \text{ and} \\ y = \mathcal{L}^{-1}(\mathcal{S}\text{−}\mathsf{layer}^{-1}(\mathcal{P}^{-1}(C^*) \oplus k_0) \oplus \mathcal{P}^{-1}(f_{20}(k_1)))[i] \text{ satisfy} \end{gathered}$$

$$\begin{gathered} x \oplus y = \Delta Y_{19}[i] = \mathcal{L}^{-1}(\mathcal{S}\text{−}\mathsf{layer}^{-1}(\mathcal{P}^{-1}(C \oplus k_0)) \oplus \mathcal{S}\text{−}\mathsf{layer}^{-1}(\mathcal{P}^{-1}(C^* \oplus k_0)))[i] \\ \text{and } \mathcal{S}^{-1}(x) \oplus \mathcal{S}^{-1}(y) = \Delta X_{19}[i]. \end{gathered}$$

5.2.4.2 Ideal fault model

The strategies we propose require at least 2 fault injections for each stage of the attack to retrieve a round key (i.e 4 to retrieve the complete key). For the first stage, whose objective is to find k_0, one of the following approaches can be used:

$$\text{(i.) Flip } Z^0_{19} \text{ then } Z^3_{19} \text{ or (ii.) Flip } W^0_{19} \text{ then } W^3_{19},$$

where Z^i_r (resp. W^i_r) denotes the input (resp. output) of the matrix $\mathcal{L}_i$ at round r. Then, to retrieve the key k_1, and so the complete key, the possible fault injections are the same but are carried out on Z_{18} or W_{18}. A flip of Z^0_r gives us a difference equal to 0xffff on the input of the matrix $\mathcal{L}_0$. The matrix being linear, we know that the output difference is also 0xffff. The latter being the same value than the one obtained with a flip of W^0_r. The other matrices have differences in input and output equal to zero. Then, the inverse permutation layer also being linear, we know the input difference of each S-box of the substitution layer at round $r+1$. These values are equal to 0x8, so we obtain $\Delta X_{r+1}[i] = 0\text{x}8$ for all $i \in \{0, \cdots, 15\}$. Moreover, we recall that the output differences are known from the correct and faulty ciphertexts. Figure 5.3 shows the

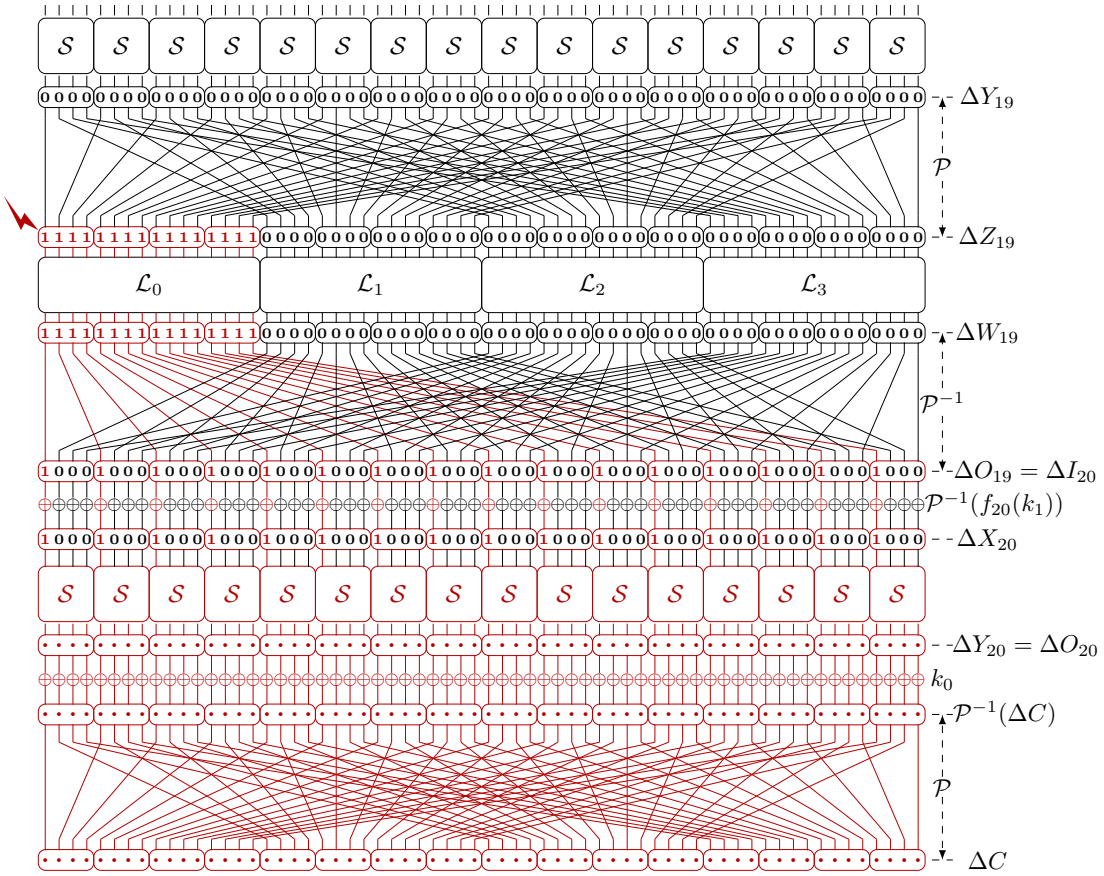

Figure 5.3: Propagation on PRIDE of the difference obtained by a flip of Z^0_{19}

propagation of the difference (displayed in red) obtained by a flip of Z^0_{19}. In the same way, a flip of Z^3_r or W^3_r yields a difference of 0x1 on each S-box at round $r+1$. Finally, with strategy (i.) or (ii.), we obtain pairs of differentials $(\Delta Y_{20}[i], \Delta X_{20}[i])_1 = (a_1, 0x1)$ and $(\Delta Y_{20}[i], \Delta X_{20}[i])_2 = (a_2, 0x8)$ for all $i \in \{0, \cdots, 15\}$ with a_1 and a_2 known. We get the same pairs for $(\Delta Y_{19}[i], \Delta X_{19}[i])$ from faults on the 18-th round. Since $0x1 \oplus 0x8 = 0x9$, from the Proposition 5.1 (and the difference distribution table in Appendix 5.2.8.2), there is only one element in the intersection of the two sets of solutions obtained for each nibble. Therefore, we have shown that we get only one candidate for each nibble of $x = \mathcal{P}^{-1}(C) \oplus k_0$ from faults on the 19-th round and one candidate for each nibble of $x = \mathcal{L}^{-1}(\mathcal{S}-\mathsf{layer}^{-1}(\mathcal{P}^{-1}(C) \oplus k_0) \oplus \mathcal{P}^{-1}(f_{20}(k_1)))[i]$. Finally, from the knowledge of C we retrieve k_0 and from the key schedule, we retrieve k_1.

The strategies we have presented require 4 fault injections to retrieve the complete key. In case the attacker obtains fewer faults, Table 5.2 shows the time complexity, expressed as a number of encryptions, that an attacker can obtain to retrieve the secret key k with 1 to 3 faults following the ideal fault model. A proof of these values is given in Appendix 5.2.8.5.

Table 5.2: Trade-offs between the time complexity, expressed as a number of encryptions, and the number of faults with the ideal fault model.

Number of faults	1	2	3
Time complexity	2^{64}	2^{32}	$2^{27.7}$

5.2.4.3 Random fault model

In order to achieve the attack, we must flip all the bits of four 16-bit words for the ideal fault model used in the preceding part. However, we can see that reversing one bit provides an active S-box, it is, therefore, enough to inverse all the bits of the desired 16-bit words. Indeed, if we flip the bit i of W_{19}^0 from one fault, we obtain 4 candidates for the nibble i of the subkey k_0. Moreover, if we flip the bit i of W_{19}^3 from another fault, we retrieve (by intersection) the value of the nibble i of k_0.

It is easy to target a specific instruction from a simple power (or EM) analysis for example in practice. If the instruction is less than 16 bits, we can then reduce the key space from each active S-box, until it is small enough for an exhaustive search. Finally, we will see in the subsection 5 that the attack is still effective from 32-bit faults, only the exploitation of the faults is different.

5.2.4.4 Properties exploited by our attack

Our attack mainly exploits two properties of the building-blocks of PRIDE:

The design of the linear layer based on the so-called interleaved construction. Indeed, this construction aims at designing a diffusion layer with a high branch number (see Theorem 1 in [ADK+14]). For a SPN whose substitution layer is composed of n S-boxes over $\mathbb{F}_2^k$, the linear layer obtained by the interleaved construction is defined as $L = \mathcal{P}^{-1} \circ \mathcal{L} \circ \mathcal{P}$ where $\mathcal{P}$ is an isomorphism from $(\mathbb{F}_2^k)^n$ into $(\mathbb{F}_2^n)^k$. Then, we deduce from the definition of $\mathcal{P}$ that flipping the n bits of any word at the input of $\mathcal{P}^{-1}$ in $W = (W_1, \cdots, W_k)$ activates all S-boxes in the next round. Indeed, by construction, the n bits of any W_i go to different S-boxes. Hence flipping n consecutive bits in the linear layer of the penultimate round allows the attacker to recover information on all the n nibbles of the subkey used in the last round. The number of candidates for this last-round subkey is upper-bounded by $\delta(\mathcal{S})^n$, where $\delta(\mathcal{S})$ is the differential-uniformity of the S-box ($\delta(\mathcal{S}) = 4$ in the case of PRIDE and of most block ciphers using 4-bit S-boxes).

The differential properties of the S-box, which avoids the existence of differentials with high probability over a large number of rounds. The counterpart of this resistance against classical differential cryptanalysis is that the number of inputs which satisfy two valid differentials simultaneously is usually reduced to a single element. This property enables the attacker to drastically reduce the number of subkey candidates. In the case of PRIDE, two faults, each on n consecutive bits in the linear layer, are enough to obtain a single candidate for the subkey.

5.2.4.5 Simulation of the DFA on PRIDE

In order to validate our theoretical DFA against PRIDE and test the correctness of the proposed equations, we performed a validation by simulation.

In this section we assume that a device executes PRIDE with a key $k = k_0||k_1$ where k_0 = 0xefcdab8967452301 and k_1 = 0x0123456789abcdef. We further assume that an attacker successfully flips all the bits of Z^0_{19}, Z^3_{19}, W^0_{18} and W^3_{18}.

Then, she obtains the following ciphertexts from 5 executions of the same plaintext 0xfedcba9876543210:

i. 0xc40f2551f39c63a9 the correct ciphertext,

ii. 0xe7f325510dc3b7a8, 0xc40fdaaec89376f7 from a flip of Z^0_{19}, Z^3_{19},

iii. 0x2857589433cbdead, 0x461720d9729c1956 from a flip of W^0_{18}, W^3_{18}.

The knowledge of the plaintext is not necessary, it is sufficient to ensure that the same plaintext is used for each execution.[1] The attacker obtains the following differentials for the last substitution layer from the first two faulty ciphertexts:

i. $(\Delta X_{20}, \Delta Y_{20})_1$ = (0x8888888888888888, 0x33a323a88a8aaa23),

ii. $(\Delta X_{20}, \Delta Y_{20})_2$ = (0x1111111111111111, 0x4467656745457776).

From the first differential, she obtains a set of candidates for each nibble of $\mathcal{P}^{-1}(C) \oplus k_0$ where C is the correct ciphertext. She can then sectioned a set of candidates for each nibble of k_0 from $\mathcal{P}^{-1}(C)$ = 0xab720c373416ba8d. Table 5.3 shows the obtained sets of candidates.

Table 5.3: Sets of candidates obtained from $(\Delta X_{20}, \Delta Y_{20})_1$

$k_0[0]$	$k_0[1]$	$k_0[2]$	$k_0[3]$	$k_0[4]$	$k_0[5]$	$k_0[6]$	$k_0[7]$	$k_0[8]$	$k_0[9]$	$k_0[10]$	$k_0[11]$	$k_0[12]$	$k_0[13]$	$k_0[14]$	$k_0[15]$
0x5	0x4	0x4	0x5	0x0	0x0	0x0	0x1	0x5	0x5	0x4	0x5	0x0	0x1	0x0	0x1
0x6	0x7	0x6	0x6	0x2	0x3	0x2	0x2	0x6	0x7	0x7	0x7	0x2	0x3	0x2	0x2
0xd	0xc	0xc	0xd	0x8	0x8	0x8	0x9	0xd	0xd	0xc	0xd	0x8	0x9	0x8	0x9
0xe	0xf	0xe	0xe	0xa	0xb	0xa	0xa	0xe	0xf	0xf	0xf	0xa	0xb	0xa	0xa

From the last differential, the attacker obtains another set of candidates for each nibble of k_0. Table 5.4 shows the resulting candidates.

Table 5.4: Sets of candidates obtained from $(\Delta X_{20}, \Delta Y_{20})_2$

$k_0[0]$	$k_0[1]$	$k_0[2]$	$k_0[3]$	$k_0[4]$	$k_0[5]$	$k_0[6]$	$k_0[7]$	$k_0[8]$	$k_0[9]$	$k_0[10]$	$k_0[11]$	$k_0[12]$	$k_0[13]$	$k_0[14]$	$k_0[15]$
0xa	0xa	0xa	0xa	0xa	0xa	0x8	0x8	0x2	0x2	0x0	0x0	0x2	0x2	0x0	0x0
0xb	0xb	0xb	0xb	0xb	0xb	0x9	0x9	0x3	0x3	0x1	0x1	0x3	0x3	0x1	0x1
0xe	0xe	0xc	0xc	0xc	0xe	0xe	0xe	0x6	0x6	0x4	0x4	0x4	0x4	0x6	0x6
0xf	0xf	0xd	0xd	0xd	0xf	0xf	0xf	0x7	0x7	0x5	0x5	0x5	0x5	0x7	0x7

By doing the intersection of the obtained two sets for each nibble, the attacker gets k_0. Then, with this value of k_0, she obtains the following differences for the antepenultimate substitution layer from the flip of W^0_{18} and W^3_{18}:

i. $(\Delta X_{19}, \Delta Y_{19})_1$ = (0x8888888888888888, 0x23a2288338832828),

ii. $(\Delta X_{19}, \Delta Y_{19})_2$ = (0x1111111111111111, 0x7777456474776476).

From the first differential, she obtains sets of candidates for each nibble Nib_i of $\mathcal{L}^{-1}(\mathcal{S}(\mathcal{P}^{-1}(C) \oplus k_0) \oplus \mathcal{P}^{-1}(f_{20}(k_1)))$ with $i \in \{0, \cdots, 15\}$. Table 5.5 shows the sets of candidates she gets.

[1] If it is not the case, the attacker can mount an attack if she knows, for each faulty ciphertext, the corresponding correct ciphertext - to obtain differentials for the S-boxes. However, the key may not be recovered in this case since the information obtained by the attacker depends on the value of the correct ciphertext.

Table 5.5: Sets of candidates obtained from $(\Delta X_{19}, \Delta Y_{19})_1$

Nib_0	Nib_1	Nib_2	Nib_3	Nib_4	Nib_5	Nib_6	Nib_7	Nib_8	Nib_9	Nib_{10}	Nib_{11}	Nib_{12}	Nib_{13}	Nib_{14}	Nib_{15}
0x0	0x4	0x1	0x0	0x0	0x5	0x5	0x4	0x4	0x5	0x5	0x4	0x0	0x5	0x0	0x5
0x2	0x7	0x3	0x2	0x2	0x6	0x6	0x7	0x7	0x6	0x6	0x7	0x2	0x6	0x2	0x6
0x8	0xc	0x9	0x8	0x8	0xd	0xd	0xc	0xc	0xd	0xd	0xc	0x8	0xd	0x8	0xd
0xa	0xf	0xb	0xa	0xa	0xe	0xe	0xf	0xf	0xe	0xe	0xf	0xa	0xe	0xa	0xe

From the last differential, the attacker obtains other sets of candidates for each nibble Nib_i of $\mathcal{L}^{-1}(\mathcal{S}(\mathcal{P}^{-1}(C) \oplus k_0) \oplus \mathcal{P}^{-1}(f_{20}(k_1)))$ with $i \in \{0, \cdots, 15\}$. Table 5.6 shows the sets of candidates obtained.

Table 5.6: Sets of candidates obtained from $(\Delta X_{19}, \Delta Y_{19})_2$

Nib_0	Nib_1	Nib_2	Nib_3	Nib_4	Nib_5	Nib_6	Nib_7	Nib_8	Nib_9	Nib_{10}	Nib_{11}	Nib_{12}	Nib_{13}	Nib_{14}	Nib_{15}
0x8	0x8	0x8	0x8	0x0	0x2	0xa	0x0	0x8	0x0	0x8	0x8	0xa	0x0	0x8	0xa
0x9	0x9	0x9	0x9	0x1	0x3	0xb	0x1	0x9	0x1	0x9	0x9	0xb	0x1	0x9	0xb
0xe	0xe	0xe	0xe	0x4	0x6	0xc	0x4	0xe	0x4	0xe	0xe	0xc	0x4	0xe	0xc
0xf	0xf	0xf	0xf	0x5	0x7	0xd	0x5	0xf	0x5	0xf	0xf	0xd	0x5	0xf	0xd

By intersecting the obtained two sets for each nibble, the attacker gets

$$\mathcal{L}^{-1}(\mathcal{S}(\mathcal{P}^{-1}(C) \oplus k_0) \oplus \mathcal{P}^{-1}(f_{20}(k_1))) = \text{0x8f9806d4f5efa58d}.$$

Then, she computes

$$\mathcal{S}(\mathcal{P}^{-1}(C) \oplus k_0) \oplus \mathcal{P}^{-1}(f_{20}(k_1)) = \text{0x24c39cc978f41dd4}$$

and from $\mathcal{S}(\mathcal{P}^{-1}(C) \oplus k_0) = \text{0x11c3a9c65f5f772b}$, she retrieves

$$\mathcal{P}^{-1}(f_{20}(k_1)) = \text{0x3500350f27ab6aff}.$$

finally she deduces $f_{20}(k_1) = \text{0x0137454b89ffcd53}$, she gets k_1 from the key scheduling and so she retrieves the complete key.

5.2.5 Practical implementation of the DFA on PRIDE

In order to test the feasibility of our attack against the PRIDE block cipher, we have implemented and run the cipher on an STM32 chip embedding an ARM Cortex-M3 micro-controller. That particular chip was chosen because it is quite representative of the off-the-shelf devices used for IoT applications. Note that the chip does not embed any countermeasures against the kind of the fault attacks implemented in this paper. We validated the attack on an implementation in ARM assembly language taking advantage of the 32-bit architecture of the micro-controller. We present in this section the full analysis conducted on this implementation. The source code is given in Appendix 5.2.8.6, and Table 5.7 compares the performances of this implementation with that of the implementation in AVR assembly language whose source code and performances are given in [ADK+14].

Table 5.7: Comparison between AVR and ARM assembly implementation

	Time (cycle)	Size (bytes)
AVR assembly implementation (given in [ADK+14])	1514	266
ARM assembly implementation (Appendix 5.2.8.6)	2375	490

So as to inject exploitable faults into such a chip, we used EM pulses because with this approach we did not need to decapsulate the chip and we were able to inject faults at precise

enough instants to target specific instructions of the cipher during its execution. The set-up we used is quite similar to the one described in [DDRT12], with the difference that we did not need any motorized X-Y stage: injecting faults 'in the center' of the chip was good enough for having a fault model close to a random fault model (one chance over two to flip a bit). Indeed, it is possible to target a precise 32-bit word (more precisely a specific instruction), but the injected faults follow a random pattern. In order to obtain pairs of differentials $(\Delta X_{20}[i], \Delta Y_{20}[i])$ (resp. $(\Delta X_{19}[i], \Delta Y_{19}[i])$) for $i \in \{0, \cdots, 15\}$, we injected the faults on the first and the second 32-bit word of the state before the inverse permutation in the 19-th (resp. 18-th) round; injecting as many faults as necessary. Each fault on the first word provided us differences on each nibble of ΔX_{20} equal to 0x0, 0x4, 0x8 or 0xc and equal to 0x0, 0x1, 0x2 or 0x3 from each fault on the second word. We validated the attack from these 32-bit faults, we will see that the faults exploitation is different (some pairs of differentials do not allow us a single candidate) but the attack is nevertheless still effective.

In our experiment, we used a key $k = k_0||k_1$ where k_0 = 0xf3f721cb1c882658 and k_1 = 0xe417d148e239ca5d. The plaintext used for all executions was 0x0132546 798badcfe and the correct ciphertext was 0x9aecb37ea45a6c89. We used a simple EM analysis to identify in time the 18-th and 19-th rounds. figure 5.4 shows the curve obtained on the oscilloscope and the 20 rounds are displayed in red.

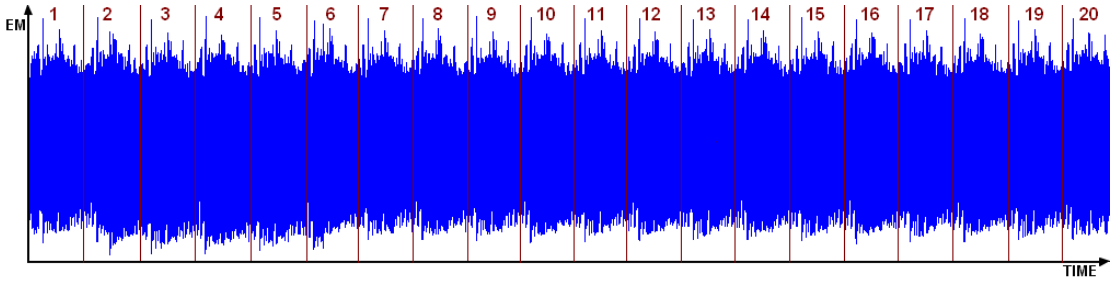

Figure 5.4: EM curve measured of PRIDE cipher

Then we used an electromagnetic pulse generator to disrupt the PRIDE's execution. Table 5.8 (resp. Table 5.9) shows the faults we have obtained from the electromagnetic injection on W_{19} (resp. W_{18}) numbered from 1 to 25. For each fault, Table 5.8 (resp. Table 5.9) provides the value of ΔX_{20} and ΔY_{20} (resp. ΔX_{19} and ΔY_{19}), only obtained from the correct and the faulty ciphertexts. We denote respectively by θ, β, γ, δ the possible pair of values (0x2,0x3), (0x4,0x8), (0x4,0xc), (0x8,0xc). Indeed, some differences in the output of the S-boxes can be obtained from two distinct differences in input. Finally, we give in each table the fault value computed after retrieving the key.

Remark *Out of 2,000 shots, we don't get any cipher for 1,219 cases and we get 247 faulty ciphers including 13 exploitable (i.e. which satisfied the conditions for our DFA). Non exploitable faulty ciphers came from a dysfunction of the UART due to the faults.*

We now give, among the obtained faults, those that give as much information as all faults and all sets of candidates that we can extract from each fault. Table 5.10 shows all sets of candidates obtained for each nibble of k_0 from the differentials (ΔY_{20}, ΔX_{20}) and with $\mathcal{P}^{-1}(C)$ = 0xe17c93c49ec6fc61 where C is the correct ciphertext. Symbol $\emptyset$ means that the fault does not provide any information about the nibble (i.e. the 16 values are possible).

We eventually get 4 possible values for k_0 with $k_0[8] \in \{0x0, 0x1\}$ and $k_0[10] \in \{0x8, 0x9\}$. In order to reduce the number of possible keys, we then used faulty ciphers obtained from fault

No.	Faulty ciphertext	Value of the fault on W_{19}	Value of ΔY_{20}	Value of ΔX_{20}
1	0x1aad3b972c92ec09	0x00000000804108e8	0xf00060007e40600c	0xθ0001000$1\theta$101000θ
2	0x7b4c93dea55a6d89	0x00000000e1a0a0a0	0x88c0000bc0c00000	0x$\theta\theta\theta$0000$\theta\theta\theta$000000
3	0x1b6c733e255aadc9	0x0000000081804040	0xf500000b85000000	0xθ1000000$\theta\theta$1000000
4	0x71ecd27ee55a6d89	0x00000000eb00e900	0x8ec0808f00000000	0x$\theta\theta\theta$0θ0$\theta\theta$00000000
5	0x9aecb324a4426cdb	0x000000000000005a	0x0000000005076050	0x0000000001011010
6	0x9a57b33fa4626cf1	0x0000000000bb005a	0x0000000085bbb08c	0x00000000θ1$\theta\theta\theta$0$\theta\theta$
7	0x9a57b365a4606cb9	0x0000000000bb0000	0x0000000080bfe0ec	0x00000000θ0$\theta\theta\theta$0$\theta\theta$
8	0x77aa24313111ed8c	0x00000000ed461f4d	0xf8868e4f0e006de7	0x$\theta\theta\theta$1$\theta\theta$1θ0θ001$\theta\theta$1
9	0x9ae8b37ac15a6989	0x6500040400000000	0x0220030300000c00	0x0$\delta\delta$00δ0δ00000γ00
10	0x8aecb27e415abc89	0xe400d10000000000	0x3329020600000000	0x$\delta\delta\delta\gamma$0δ0400000000
11	0xa3e692ed909ee688	0x355fab9300000000	0x10ea921c620482c5	0x40c$\beta\gamma\delta$4γ4δ0c8$\delta\gamma$c
12	0x05ecb27e565a7289	0xf3001f0000000000	0xa22b99bc00000000	0x$\beta\delta\delta$c$\gamma\gamma$cγ00000000

Table 5.8: Faults obtained on the 19-th round

No.	Faulty ciphertext	Value of the fault on W_{18}	Value of ΔY_{19}	Value of ΔX_{19}
13	0xf24690de8df8cc89	0x0000000082000000	0xc00000b000000000	0xθ0000θ000000000
14	0x2df93aebf5935009	0x0000000041c0d0d0	0x7807000bd8050000	0x1θ01000$\theta\theta\theta$010000
15	0xa9a4a34f84604dde	0x0000000003010707	0x000004cd0000065c	0x000001$\theta\theta$00000011θ
16	0x52c367c49a9b8786	0x0000000000b55858	0x05077000b6d84808	0x0101100θ1$\theta\theta$1θ0θ
17	0x00632c247f18e99e	0x0000000058580000	0x0e0bb0000d0ef000	0x0$\theta\theta\theta$0000$\theta\theta\theta$000
18	0xecbc98d50864ad3a	0x00000000a7a70000	0xc0f008bbb0d00888	0xθ0θ00$\theta\theta\theta\theta$0θ00$\theta\theta\theta$
19	0x43b733ec34c1ec11	0x0093000000000000	0x00000000300a0022	0x00000000δ00β00$\delta\delta$
20	0xcabdf870ee423736	0x75e5575700000000	0x0c8c0b123baf049e	0x0γ8γ0c4$\delta\delta$cβ40cγc
21	0x46eb59132610ef55	0x01e0c60100000000	0x6f0001133aa00006	0x4400044$\delta\delta\beta\beta$00004
22	0x9d13b57cf2211618	0x13974cd400000000	0x0f036133290c0422	0x040δ44$\delta\delta\delta\gamma$0γ0c$\delta\delta$
23	0x1247352b2400c0ed	0x0000006700000000	0x0000000009900c96	0x00000000$\gamma\gamma$00$\gamma\gamma$4
24	0x770a084c5528c599	0x6363000000000000	0x0a8000330aa00022	0x0β8000$\delta\delta$0$\beta\beta$000$\delta\delta$
25	0xc80ca16eb67b9711	0x3600a90000000000	0x6043623a00000000	0x40cδ4$\delta\delta\beta$00000000

Table 5.9: Faults obtained on the 18-th round

injection on the 18-th round. For this, we compute the difference output ΔY_{19} from the remaining 4 candidates for the key. Then we can observe that some differentials (ΔX_{19}, ΔY_{19}) are not possible and therefore remove the corresponding candidate.

Table 5.10: Sets of candidates obtained from (ΔY_{20}, ΔX_{20})

No.	$k_0[0]$	$k_0[1]$	$k_0[2]$	$k_0[3]$	$k_0[4]$	$k_0[5]$	$k_0[6]$	$k_0[7]$	$k_0[8]$	$k_0[9]$	$k_0[10]$	$k_0[11]$	$k_0[12]$	$k_0[13]$	$k_0[14]$	$k_0[15]$
1	0x0 0x1 0xe 0xf	∅	∅	∅	0x2 0x3 0x4 0x5	∅	∅	∅	0x0 0x1 0x6 0x7	0x2 0x3 0xc 0xd	0x8 0x9 0xc 0xd	∅	0x2 0x3 0x4 0x5	∅	∅	0x4 0x5 0x8 0x9
3	0x0 0x1 0xe 0xf	0x2 0x3 0x6 0x7	∅	∅	∅	∅	∅	0x0 0x1 0x2 0x3 0x8 0x9 0xa 0xb	0x0 0x1 0x2 0x3 0x8 0x9 0xa 0xb	0x8 0x9 0xc 0xd	∅	∅	∅	∅	∅	∅
6	∅	∅	∅	∅	∅	∅	∅	∅	0x0 0x1 0x2 0x3 0x8 0x9 0xa 0xb	0x8 0x9 0xc 0xd	0x0 0x1 0x2 0x3 0x8 0x9 0xa 0xb	0x0 0x1 0x2 0x3 0x8 0x9 0xa 0xb	0x0 0x1 0x2 0x3 0x8 0x9 0xa 0xb	∅	0x4 0x5 0x6 0x7 0xc 0xd 0xe 0xf	0x4 0x5 0x8 0x9
8	0x0 0x1 0xe 0xf	0x0 0x1 0x2 0x3 0x8 0x9 0xa 0xb	0x4 0x5 0x6 0x7 0xc 0xd 0xe 0xf	0x0 0x1 0x6 0x7	0x0 0x1 0x2 0x3 0x8 0x9 0xa 0xb	0x0 0x1 0xe 0xf	0x8 0x9 0xc 0xd	0x4 0x5 0xa 0xb	∅	0x2 0x3 0xc 0xd	∅	∅	0x2 0x3 0x4 0x5	0x6 0x7 0xa 0xb	0x4 0x5 0xa 0xb	0x8 0x9 0xe 0xf
11	0xa 0xb 0xe 0xf	∅	0x1 0xf	0x1 0x5 0x7 0xb 0xd 0xf	0x2 0x4 0xb 0xd	0x1 0x3 0x4 0x6 0x9 0xb	0x8 0x9 0xc 0xd	0x2 0x7 0xb 0xe	0x0 0x1 0x6 0x7	0x4 0x6 0x9 0xb 0xc 0xe	∅	0x8 0xc	0x1 0x2 0x9 0xa	0x4 0x6 0x9 0xb 0xc 0xe	0x0 0x5 0x9 0xc	0x8 0xd
12	0x3 0x5 0x7 0x9 0xd 0xf	0x1 0x3 0x4 0x6 0x9 0xb	0x0 0x2 0x5 0x7 0xd 0xf	0x7 0xc	0x2 0x4 0xb 0xd	0x1 0x7 0x8 0xe	0x7 0xc	0x2 0x7 0xb 0xe	∅	∅	∅	∅	∅	∅	∅	∅

Indeed, from the faulty ciphertext 0xf24690de8df8cc89 obtained from a fault on W_{18}, we obtain the 4 following values for ΔY_{19} for each possible value of k_0:

k_0	ΔY_{19}
f3f721cb0c882658	0xc000009022000000
f3f721cb0c982658	0xe000009022220000
f3f721cb1c882658	0xc00000b000000000
f3f721cb1c982658	0xe00000b000220000

and since we know that we injected faults on the last 32 bits of W_{18}, we know that each nibble of ΔX_{19} is either 0x0, 0x1, 0x2 or 0x3. From the difference distribution table of the S-box, we see that an input difference equal to 0x1, 0x2 or 0x3 can lead to an output difference in {0x4, 0x5, 0x6, 0x7, 0x8, 0xb, 0xc, 0xd, 0xe, 0xf} only. Consequently, we retrieve k_0 (displayed in red).

Then, Table 5.11 shows all sets of candidates obtained for each nibble Nib_i of $\mathcal{L}^{-1}(\mathcal{S}(\mathcal{P}^{-1}(C) \oplus k_0) \oplus \mathcal{P}^{-1}(f_{20}(k_1)))$ with $i \in \{0, \cdots, 15\}$, from differentials (ΔY_{19}, ΔX_{19}). We again denote by $\emptyset$ when the fault does not provide any information about the nibble (i.e. the 16 values are possible).

Finally, by intersecting sets for each nibble, we deduce 8 candidates for k_1 from k_0 and C and we retrieve the correct value of k by testing all. With this we provide, to the best of our knowledge, the first practical validation of a DFA against PRIDE, even against any lightweight SPN-block cipher.

Remark *We observed that injecting 32-bit random faults allows us to have lower complexity than with 16-bit random faults. Indeed, although the differential pairs obtained do not always provide a single candidate in the case of 32-bit faults, the probability to obtain a differential is greater than with 16-bit faults. Finally, we showed that flipping one bit give us a known difference on a nibble, nd so we canlead the attack with faults from 1 to 32 bits.*

5.2.6 Countermeasures

In this section, we present and briefly analyze three possible countermeasures. This list of countermeasures is not exhaustive, and any combination of those three can be used in practice to thwart the DFA proposed in this paper.

Table 5.11: Sets of candidates obtained from (ΔY_{19}, ΔX_{19})

No.	Nib_0	Nib_1	Nib_2	Nib_3	Nib_4	Nib_5	Nib_6	Nib_7	Nib_8	Nib_9	Nib_{10}	Nib_{11}	Nib_{12}	Nib_{13}	Nib_{14}	Nib_{15}
16	∅	0x2 0x3 0x6 0x7	∅	0x8 0x9 0xe 0xf	0x8 0x9 0xe 0xf	∅	∅	∅	0x4 0x5 0x6 0x7 0xc 0xd 0xe 0xf	0xa 0xb 0xc 0xd	0x6 0x7 0xa 0xb	0x0 0x1 0x2 0x3 0x8 0x9 0xa 0xb	0x0 0x1 0x4 0x5	0x0 0x1 0x2 0x3 0x8 0x9 0xa 0xb	∅	0x0 0x1 0x2 0x3 0x8 0x9 0xa 0xb
17	∅	0x2 0x3 0xa 0xb	∅	0x4 0x5 0x6 0x7 0xc 0xd 0xe 0xf	0x4 0x5 0x6 0x7 0xc 0xd 0xe 0xf	∅	∅	∅	∅	0x6 0x7 0xa 0xb	∅	0x2 0x3 0xa 0xb	0x0 0x1 0xe 0xf	∅	∅	∅
18	0x4 0x5 0x8 0x9	∅	0x0 0x1 0xe 0xf	∅	∅	0x0 0x1 0x2 0x3 0x8 0x9 0xa 0xb	0x4 0x5 0x6 0x7 0xc 0xd 0xe 0xf	0x4 0x5 0x6 0x7 0xc 0xd 0xe 0xf	0x4 0x5 0x6 0x7 0xc 0xd 0xe 0xf	∅	0x6 0x7 0xa 0xb	∅	∅	0x0 0x1 0x2 0x3 0x8 0x9 0xa 0xb	0x0 0x1 0x2 0x3 0x8 0x9 0xa 0xb	0x0 0x1 0x2 0x3 0x8 0x9 0xa 0xb
20	∅	0x3 0x6 0xa 0xf	0x5 0x6 0xd 0xe	0x3 0x6 0xa 0xf	∅	0x0 0xb	0x0 0x1 0x4 0x5	0x1 0x2 0x5 0x7 0x8 0xa	0x1 0x2 0x4 0x7 0xc 0xf	0x0 0xb	0x1 0x3 0x7 0x9 0xb 0xd	0x3 0xc	∅	0xa 0xe	0x2 0x4 0xb 0xd	0x6 0x8
22	∅	0x3 0xc	∅	0x1 0x2 0x4 0x7 0xc 0xf	0x8 0x9 0xe 0xf	0x0 0x1 0x4 0x5	0x1 0x2 0x4 0x7 0xc 0xf	0x1 0x2 0x4 0x7 0xc 0xf	0x0 0x2 0x5 0x7 0x8 0xa	0x2 0x4 0xb 0xd	∅	0x3 0x6 0xa 0xf	∅	0xa 0xe	0x0 0x2 0x5 0x7 0x8 0xa	0x0 0x2 0x5 0x7 0x8 0xa
23	∅	∅	∅	∅	∅	∅	∅	∅	∅	0x2 0x4 0xb 0xd	0x2 0x4 0xb 0xd	∅	∅	0x3 0x6 0xa 0xf	0x2 0x4 0xb 0xd	0x8 0x9 0xe 0xf
25	0x8 0x9 0xe 0xf	∅	0xa 0xe	0x1 0x2 0x4 0x7 0xc 0xf	0x8 0x9 0xe 0xf	0x0 0x2 0x5 0x7 0x8 0xa	0x1 0x2 0x4 0x7 0xc 0xf	0x1 0x3 0x7 0x9 0xb 0xd	∅	∅	∅	∅	∅	∅	∅	∅

5.2.6.1 Duplication of computations

Description: A simple countermeasure is to make two computations for the last two rounds. We save the state of the cipher W_{17} in memory, possibly k times for more security - since we are in lightweight cryptography is seems reasonable to take $k = 1$ or $k = 2$. Then we make the computations up to O_{20} and save the state again. We repeat the computation with the saved state (W_{17}) and compare with the first result - possibly k times again. If two different computations give different results we trap the cipher and the system produces no output. Otherwise, the execution performs normally. We can also apply a majority vote by duplicating the computations twice, possibly $2k$ times and give as output one that most appears. 5.5 shows a majority vote using duplication.

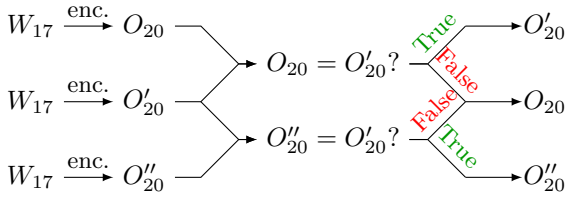

Figure 5.5: Majority vote using duplication

Cost: This countermeasure uses, for encryption and decryption, two additional matrix layer and three additional substitution layers, subkey updates and subkey additions. The cost can be bounded from above by 15% of the total PRIDE cost.

5.2.6.2 Desynchronisation

Description: This countermeasure consists in adding time randomisation during the cipher so that the temporal position of the 18-th and the 19-th round will not be the same for each

execution. For the time randomisation generation, we can use a simple Linear Feedback Shift Register (LFSR) whose value indicates the 'random' delay time. Those random delay functions can be added before the 18-th round. 5.6 illustrates the countermeasure.

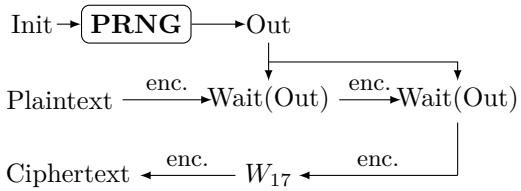

Figure 5.6: Desynchronisation

Cost: The cost depends on the time randomization generation - a simple LFSR implemented in hardware has a low cost with respect to IoT constraints, it also depends on the duration of the 'random delay', and on the time needed to access the random output of the LFSR.

5.2.6.3 Masking

Description: Another countermeasure proposed by Guilley *and al.* in [GSDS10] is to add a random mask to the message to prevent two consecutive executions of the same plaintext. More precisely, in its original description, it consists in generating a 64-bit random mask different at each execution, XOR it with the asked plaintext and the ciphertext obtained is sent with the mask.

In our case, we use a simple LFSR defined by a minimal primitive polynomial of degree 64 ($X^{64} + X^{63} + X^{61} + X^{60} + 1$ for example), and by an initialization made public. The LFSR thus generates $2^{64} - 1$ different masks. It must not be again accessible by the user to prevent its reset. For this, it must be correctly implemented in hardware. We apply the mask by an XOR on the input of the 10-th round. This prevents the adversary from getting two encryptions of the same plaintext, and therefore to make a DFA. For decryption, we apply an XOR between the mask and the output of the 10-th round and get the correct plaintext. We then have two options. The first is to send the mask with the ciphertext. Unfortunately, in this case, this method does not protect against an attack on decryption. Indeed, the attacker can choose the same mask on each decryption. However, in the context of IoT it is common that the card is only used for encryption and decryption is carried out on a protected server. The second is to synchronize the encryption and the decryption. They both use the same LFSR with the same initialization and the decryption must be applied in the same order as ciphertexts received. Therefore, the countermeasure protects both the encryption, and the decryption but with an additional synchronisation constraint. 5.7 illustrates the countermeasure.

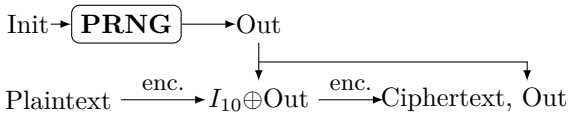

Figure 5.7: Masking based on that of Guilley

Cost: The cost depends on the choice of the random mask generation. A simple LFSR - like the one we cited - implemented in hardware has a low cost with respect to IoT constraints. Moreover, applying the mask requests an additional cost of an XOR for encryption and the same for decryption in the second case.

5.2.7 Conclusion

In this paper, we propose the first differential fault analysis on the block cipher PRIDE. We explain how this attack can be optimized and we demonstrate it, with 4 faults only to retrieve the full secret key. We show that our attack is indeed feasible from 32-bit random faults obtained with electromagnetic injection, which is a low-cost means of injection. We believe that the resistance against DFA is important for a cipher like PRIDE, which is expected to be largely deployed in low-end devices thanks to its lightness. At last we propose some countermeasures which leave the cipher still very efficient for IoT devices. They can be combined to provide more security and are not exhaustive. An optimization of these countermeasures is possible to make them less costly and keep the light side of the cipher. It is also necessary to be careful that the protections to prevent the DFA do not open doors to further attacks. Finally, it appears that our attack applies to any SPN-based block ciphers with a linear layer similar to the one used in PRIDE, like the LS-Designs family introduced by Grosso & al [GLSV15] in 2014. The details of this generalization will be studied in a future work.

5.2.8 Appendices

5.2.8.1 Details of PRIDE

This subsection provides permutations and matrices used by PRIDE. Tables 5.12 and 5.13 respectively describe the permutation $\mathcal{P}$ and its inverse $\mathcal{P}^{-1}$ used in the round function of PRIDE as well as in the beginning and the end of the cipher.

Table 5.12: Permutation $\mathcal{P}$ of PRIDE

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\mathcal{P}(x)$	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
x	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$\mathcal{P}(x)$	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
x	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$\mathcal{P}(x)$	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
x	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$\mathcal{P}(x)$	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

Table 5.13: Permutation $\mathcal{P}^{-1}$ of PRIDE

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\mathcal{P}^{-1}(x)$	0	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60
x	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$\mathcal{P}^{-1}(x)$	1	5	9	13	17	21	25	29	33	37	41	45	49	53	57	61
x	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$\mathcal{P}^{-1}(x)$	2	6	10	14	18	22	26	30	34	38	42	46	50	54	58	62
x	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$\mathcal{P}^{-1}(x)$	3	7	11	15	19	23	27	31	35	39	43	47	51	55	59	63

The matrices defining the linear layer of PRIDE are:

$$\mathcal{L}_0 = \begin{pmatrix}
0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0\\
0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0\\
0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0\\
0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1\\
1&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0\\
0&1&0&0&0&0&0&0&0&1&0&0&0&1&0&0\\
0&0&1&0&0&0&0&0&0&0&1&0&0&0&1&0\\
0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&1\\
1&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0\\
0&1&0&0&0&1&0&0&0&0&0&0&0&1&0&0\\
0&0&1&0&0&0&1&0&0&0&0&0&0&0&1&0\\
0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&1\\
1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0\\
0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0\\
0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0\\
0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0
\end{pmatrix}, \quad \mathcal{L}_1 = \begin{pmatrix}
1&1&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&1&1&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&1&1&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&1&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&1&0&1&0&0&0&0&0&0\\
1&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&1&1&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&1&1&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&1\\
0&0&0&0&0&1&0&0&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&1&1&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&1&1&0&0&0&0
\end{pmatrix}$$

$$\mathcal{L}_2 = \begin{pmatrix}
0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&1&1&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&1&0&1&0&0&0&0&0&0\\
1&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0\\
1&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&1&1&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&1&1&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&1&1&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&1\\
0&0&0&0&0&1&0&0&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&1&1&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&1&1&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&1&1&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&1&1&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&1
\end{pmatrix}, \quad \mathcal{L}_3 = \begin{pmatrix}
1&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0\\
0&1&0&0&0&1&0&0&0&0&0&0&0&1&0&0\\
0&0&1&0&0&0&1&0&0&0&0&0&0&0&1&0\\
0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&1\\
1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0\\
0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0\\
0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0\\
0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0\\
0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0\\
0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0\\
0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0\\
0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1\\
1&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0\\
0&1&0&0&0&0&0&0&0&1&0&0&0&1&0&0\\
0&0&1&0&0&0&0&0&0&0&1&0&0&0&1&0\\
0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&1
\end{pmatrix}$$

5.2.8.2 Differential properties of the PRIDE S-box

5.2.8.3 Difference distribution table of the PRIDE S-box

Table 5.14 shows the difference distribution table T of the PRIDE S-box, which is defined by $T(i,j) = \#\ \{(x,y) \in \{0,1\}^4 \times \{0,1\}^4 \mid x \oplus y = i, \mathcal{S}(x) \oplus \mathcal{S}(y) = j\}$.

Table 5.14: difference distribution table of the PRIDE S-box

T	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xa	0xb	0xc	0xd	0xe	0xf
0x0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x1	0	0	0	0	4	4	4	4	0	0	0	0	0	0	0	0
0x2	0	0	0	0	0	0	0	0	4	0	0	4	2	2	2	2
0x3	0	0	0	0	0	0	0	0	4	0	0	4	2	2	2	2
0x4	0	4	0	0	0	0	4	0	0	2	2	0	2	0	0	2
0x5	0	4	0	0	0	4	0	0	0	2	2	0	2	0	0	2
0x6	0	4	0	0	4	0	0	0	0	2	2	0	0	2	2	0
0x7	0	4	0	0	0	0	0	4	0	2	2	0	0	2	2	0
0x8	0	0	4	4	0	0	0	0	4	0	4	0	0	0	0	0
0x9	0	0	0	0	2	2	2	2	0	0	0	0	2	2	2	2
0xa	0	0	0	0	2	2	2	2	4	0	4	0	0	0	0	0
0xb	0	0	4	4	0	0	0	0	0	0	0	0	2	2	2	2
0xc	0	0	2	2	2	2	0	0	0	2	0	2	2	0	2	0
0xd	0	0	2	2	0	0	2	2	0	2	0	2	0	2	0	2
0xe	0	0	2	2	0	0	2	2	0	2	0	2	2	0	2	0
0xf	0	0	2	2	2	2	0	0	0	2	0	2	0	2	0	2

5.2.8.4 Proof of Proposition 5.1

We can see that, from the knowledge of a nonzero input $(x \oplus y)$ and of an output difference $(\mathcal{S}(x) \oplus \mathcal{S}(y))$ for $\mathcal{S}$ we deduce 0, 2 or 4 candidates for the input value x. Moreover, we can easily find pairs of differentials (a_1, b_1) and (a_2, b_2) which are satisfied by a single input x. For this, we use Proposition 5.1 that we prove here.

Proof: [of Proposition 5.1] Let $\mathcal{D}(a,b)$ denote the set of solutions of the equation

$$\mathcal{S}(x \oplus a) \oplus \mathcal{S}(x) = b.$$

Let us consider (a_1, b_1) and (a_2, b_2) be two differentials with $a_1 \neq a_2$ such that

$$\#\mathcal{D}(a_1, b_1) \cap \mathcal{D}(a_2, b_2) \geq 2.$$

Let us first prove that both $\mathcal{D}(a_1, b_1)$ and $\mathcal{D}(a_2, b_2)$ have at least 4 elements. If these two sets have two elements only, $\mathcal{D}(a_1, b_1) = \{x, x \oplus a_1\}$ and $\mathcal{D}(a_2, b_2) = \{x, x \oplus a_2\}$, implying that they cannot be the same since $a_1 \neq a_2$. Then, at least one of the two sets contains at least four elements. Suppose that $\#\mathcal{D}(a_1, b_1) = 4$ and $\#\mathcal{D}(a_2, b_2) = 2$. Then, $x \oplus a_2 \in \mathcal{D}(a_1, b_1)$, with $\mathcal{D}(a_2, b_2) = \{x, x \oplus a_2\}$. Consequently,

$$\mathcal{S}(x \oplus a_1 \oplus a_2) \oplus \mathcal{S}(x \oplus a_2) = b_1 = \mathcal{S}(x \oplus a_1) \oplus \mathcal{S}(x)$$

implying that

$$\mathcal{S}(x \oplus a_2) \oplus \mathcal{S}(x) = \mathcal{S}(x \oplus a_1 \oplus a_2) \oplus \mathcal{S}(x \oplus a_1).$$

Thus, $x \oplus a_1 \in \mathcal{D}(a_2, b_2)$ is a contradiction. We have proved that $\#\mathcal{D}(a_2, b_2) = 4$.
Now, it is clear that any element x in $\mathcal{D}(a_1, b_1) \cap \mathcal{D}(a_2, b_2)$ is a solution of

$$\mathcal{S}(x \oplus a_2) \oplus \mathcal{S}(x \oplus a_1) = b_1 \oplus b_2,$$

i.e., $x \oplus a_1 \in \mathcal{D}(a_1 \oplus a_2, b_1 \oplus b_2)$ and $x \oplus a_2 \in \mathcal{D}(a_1 \oplus a_2, b_1 \oplus b_2)$.

Suppose now that $\{x, x \oplus a_4\} \subseteq \mathcal{D}(a_1, b_1) \cap \mathcal{D}(a_2, b_2)$ for some $a_4 \neq 0$, we deduce that the four elements $x \oplus a_1$, $x \oplus a_2$, $x \oplus a_1 \oplus a_4$ and $x \oplus a_2 \oplus a_4$ belong to $\mathcal{D}(a_1 \oplus a_2, b_1 \oplus b_2)$. These four elements are either distinct or satisfy $a_4 = a_1 \oplus a_2$ which implies that $x \oplus a_4 \oplus a_2 = x \oplus a_1$ belongs to $\mathcal{D}(a_2, b_2)$, i.e., $x \oplus a_1 \in \mathcal{D}(a_1, b_1) \cap \mathcal{D}(a_2, b_2)$. Therefore, $x \oplus a_1$, $x \oplus a_2$, x and $x \oplus a_1 \oplus a_2$ all belong to $\mathcal{D}(a_1 \oplus a_2, b_1 \oplus b_2)$ and $\#\mathcal{D}(a_1 \oplus a_2, b_1 \oplus b_2) = 4$. □ □

5.2.8.5 Other trade-offs between the number of faults and the time complexity

We have shown that 4 faults with an appropriate strategy enable the attacker to recover the whole key. In this subsection, we evaluate the number of key candidates that an attacker can obtain with fewer faults. This number then corresponds to the time complexity of the complete key recovery. Indeed, if the attacker knows a pair of plaintext-ciphertext, encrypting the known plaintext under each key candidate until the correct ciphertext is recovered leading to a complete key recovery[2]. Firstly, the number of remaining candidates for the subkey k_0 (resp. k_1) that an attacker can obtain with one fault on the 19-th round (resp. 18-th round) is 2^{32}. We now use this result to estimate the cost of the full key recovery from a few faults only.

With a single fault. We want to determine the cost of the key recovery if the attacker can inject a single fault. If this fault is injected in the 19-th round, then the possible values of k_0 is reduced to a list of 2^{32} candidates. This corresponds to a total of 2^{96} candidates for the whole 128-bit key. If the attacker knows two plaintext-ciphertext pairs, he can then encrypt the first known plaintext under each of these 2^{96} key candidates, until the corresponding ciphertext is recovered. Only $2^{96-64} = 2^{32}$ key candidates then remain, and the second plaintext-ciphertext pair can then be exploited for recovering the key. The main part of the time complexity in this attack is the cost of the exhaustive search over the 2^{96} candidates, which corresponds to 2^{96} encryptions.

If the fault is now injected in the 18-th round, then the attack consists in successively examining all 2^{64} possible values for k_0. For each of these 2^{64} candidates, the attacker inverts the last encryption round for both the correct and the faulty ciphertexts C and C^*. He deduces the value of ΔX_{20}, and then of ΔY_{19}. When choosing a random k_0, ΔX_{20} varies in the set of all input differences which can appear when the output difference equals ΔY_{20}. From the difference distribution table of the S-box, the average number of valid input differences corresponding to a fixed output difference is

$$\frac{1}{16}(1 + 4 \times 2 + 6 \times 8 + 8 \times 5) = 6.0625.$$

[2]provided that the number of key candidates is smaller than 2^{64}. Otherwise, two plaintext-ciphertext pairs are needed.

Therefore, ΔX_{20} (and then ΔY_{19}) takes in average $6.0625^{16} = 2^{41.6}$ different values, and each of these differences appears for $2^{22.4}$ values of k_0 in average.

However, the difference ΔX_{20} is not valid if the corresponding value of ΔY_{19} does not have the form expected from the value of the fault. As the fault has been injected on Z_{18}^0 or Z_{18}^3, each nibble of ΔX_{19} is equal either to 0x1, or to 0x8. Then, the corresponding nibble ΔY_{19} can take 4 values only. Therefore, the proportion of valid values for ΔY_{19} is $4^{16} \times 2^{-64} = 2^{-32}$. It follows that, among the $2^{41.6}$ values of ΔY_{19} which are obtained from the partial decryption, only $2^{9.6}$ are valid, implying that only 2^{32} values of k_0 need to be considered. For each of these 2^{32} values of k_0, the value of the fault, and then of ΔX_{19} provides 2^{32} candidates for k_1 as proved in the previous subsection. This step then leads to a list of 2^{64} candidates for the whole 128-bit key, with a time complexity which mainly corresponds to the cost for decrypting one round of PRIDE 2^{64} times. The bottleneck of the attack is then the final key recovery procedure, which consists in testing the 2^{64} remaining keys on two plaintext-ciphertext pairs. The overall cost of the attack is then roughly the cost of 2^{64} encryption.

With two faults. If the two faults are injected in the 18-th round, then the previously described technique which enables the attacker to eliminate some candidates for k_0 is repeated twice. Only a proportion of 2^{-64} values of ΔY_{19} will be valid, implying that only the correct value of ΔY_{19} will remain after this step. As previously explained, each value of ΔY_{19} is obtained for $2^{22.4}$ values of k_0 in average. Therefore, this sieving procedure leads to a list of $2^{22.4}$ candidates for k_0. Now, exploiting the two faults injected in the 18-th round provides one candidate for k_1. Therefore, we get $2^{22.4}$ candidates for the whole key. The total time complexity of the attack then corresponds to 2^{64} decryption of a single round, and to an exhaustive search among the $2^{22.4}$ remaining keys. The first step is then the bottleneck and its cost is less than the cost of $2^{64}/20 = 2^{59.7}$ complete encryptions.

If the first fault is now injected in the 19-th round, then the list of possible values for k_0 is first reduced to a list of size 2^{32} as explained in the previous subsection. The second fault, injected on the 18-th round, then enables to reduce this list to $2^{32-32} = 1$ possible value for k_0. For this value of k_0, a list of 2^{32} candidates for k_1 is obtained from the second fault. The number of candidates for the whole key, which need to tested, is then 2^{32}. The bottleneck of the attack is then the exhaustive search over the 2^{32} remaining key candidates, which corresponds to a time complexity equal to the cost of 2^{32} encryptions.

With three faults. The best strategy with three faults consists in injecting one fault in the 19-th round, and two in the 18-th round. From the fault in the 19-th round, the attacker gets a list of 2^{32} candidates for k_0. By decrypting the last round under these 2^{32} values of k_0, we roughly get 2^{32} pairs of values for ΔY_{19} among which one is expected to be consistent with the two faults injected in the 18-th round. Moreover, these two faults lead to one candidate for k_1, i.e., one candidate for the whole key. The time complexity of the attack then corresponds to the cost of 2^{32} encryptions of a single round, i.e., $2^{27.7}$ full encryptions.

5.2.8.6 ARM source code

L-layer

; $\mathcal{L}_0$ and $\mathcal{L}_1$
; State s_0
; Temporary registers $t_0, \cdots, t_6$
(1) MOV t_0, #0x00F0
(2) MOVT t_0, #0xF0F0
(3) AND t_1, t_0, s_0, LSL#4
(4) LSR t_0, #4
(5) AND t_2, t_0, s_0, LSR#4
(6) AND t_0, s_0, #0xFF000000
(7) AND t_3, s_0, #0XFF0000
(8) EOR t_1, t_1, t_2
(9) AND s_0, s_0, #0xFF00
(10) EOR s_0, s_0, t_1
(11) AND t_1, s_0, #0x8000
(12) AND t_2, s_0, #0x01
(13) AND t_4, s_0, #0xFF00
(14) AND t_5, s_0, #0x00FF
(15) MOV t_6, #0xFF000000
(16) AND t_6, t_6, s_0, LSL#8
(17) EOR s_0, s_0, r10
(18) AND t_6, s_0, #0xFF000000
(19) EOR t_0, t_0, t_6
(20) BIC s_0, s_0, #0xFF0000
(21) EOR s_0, s_0, t_0, LSR#8
(22) EOR s_0, s_0, t_3, LSL#8
(23) MOV t_0, #0xFF00
(24) AND t_0, t_0, t_4, LSL#1
(25) EOR t_0, t_0, t_1, LSR#7
(26) LSR t_3, t_5, #1
(27) EOR t_3, t_3, t_2, LSL#7
(28) EOR s_0, s_0, t_3, LSL#8
(29) AND t_3, s_0, #0xFF00
(30) EOR s_0, s_0, t_0
(31) EOR s_0, s_0, t_3, LSR#8

; $\mathcal{L}_2$ and $\mathcal{L}_3$
; State s_1
; Temporary registers $t_0, \cdots, t_5$
(1) MOV t_0, #0xF0F0
(2) MOVT t_0, #0xF000
(3) AND t_1, t_0, s_1, LSL#4
(4) LSR t_0, #4
(5) AND t_2, t_0, s_1, LSR#4
(6) AND t_0, s_1, #0xFF00
(7) AND t_3, s_1, #0X00FF
(8) AND s_1, s_1, #0xFF0000
(9) EOR t_1, t_1, t_2
(10) EOR s_1, s_1, t_1
(11) AND t_1, s_1, #0x80000000
(12) AND t_2, s_1, #0x00010000
(13) MOV t_4, #0xFF000000
(14) AND t_5, s_1, t_4
(15) AND t_4, t_4, t_5, LSL#1
(16) EOR t_1, t_4, t_1, LSR#7
(17) MOV t_4, #0x00FF0000
(18) AND t_5, s_1, t_4
(19) AND t_4, t_4, t_5, LSR#1
(20) EOR t_2, t_4, t_2, LSL#7
(21) EOR s_1, s_1, t_2, LSL#8
(22) AND t_2, s_1, #0xFF000000
(23) EOR s_1, s_1, t_1
(24) EOR s_1, s_1, t_2, LSR#8
(25) AND t_4, s_1, #0x00FF
(26) EOR s_1, s_1, t_4, LSL#8
(27) AND t_4, s_1, #0xFF00
(28) EOR t_3, t_3, t_4, LSR#8
(29) EOR t_0, t_0, t_3
(30) BIC s_1, s_1, #0x00FF
(31) EOR s_1, s_1, t_0

S-layer ; State s_0, s_1
; Temporary registers t_0, t_1
(1) MOV t_1, s_0
(2) AND t_0, s_0, s_0, LSL#16
(3) EOR t_0, t_0, s_1
(4) AND s_0, s_0, s_1, LSR#16
(5) EOR s_0, s_0, t_0
(6) AND t_0, s_0, s_0, LSL#16
(7) EOR t_0, t_0, t_1
(8) AND s_1, s_0, t_0, LSR#16
(9) EOR s_1, s_1, t_0

5.3 From Clustering Supersequences to Entropy Minimizing Subsequences for Single and Double Deletions

Abstract

A binary string transmitted via memoryless i.i.d. deletion channel is received as a subsequence of the original input. From this, one obtains a posterior distribution on the channel input, corresponding to a set of candidate supersequences weighted by the number of times the received subsequence can be embedded in them. In a previous work it is conjectured on the basis of experimental data that the entropy of the posterior is minimized and maximized by the constant and the alternating strings, respectively. In this work, we present an algorithm for counting the number of subsequence embeddings using a run-length encoding of strings. We then describe two different ways of clustering the space of supersequences and prove that their cardinality depends only on the length of the received subsequence and its Hamming weight, but not its exact form. Then, we consider supersequences that contain a single embedding of a fixed subsequence, referred to as singletons, and provide a closed form expression for enumerating them using the same run-length encoding. We prove an analogous result for the minimization and maximization of the number of singletons, by the alternating and the uniform strings, respectively. Next, we prove the original minimal entropy conjecture for the special cases of single and double deletions using similar clustering techniques and the same run-length encoding, which allow us to characterize the distribution of the number of subsequence embeddings in the space of compatible supersequences to demonstrate the effect of an entropy decreasing operation.

5.3.1 Introduction

The original motivation for this work goes back to an analysis of quantum key distribution (QKD) protocols [RC13], which among other things, suggested some modifications of the quantum bit error rate (QBER) estimations. These modifications led to an information theory problem that was first investigated in [ARR15].

The mathematical problem encountered in the aforementioned analysis is the following. A random bit string y of length n emitted from a memoryless source is transmitted via an i.i.d. deletion channel such that a shorter bit string x of length m ($m \leq n$) is received as a subsequence of y, after having been subject to $n-m$ deletions. Consequently, the order in which the remaining bits are revealed is preserved, but the exact positions of the bits are not known. Given a subsequence x, the question is to find out how much information about y is revealed. More specifically, the quantity we are interested in is the conditional entropy [CT12] over the set of candidate supersequences upon observing x, i.e., $H(Y|X=x)$ where Y is restricted to the set of compatible supersequences as explained below.

The said information leakage is quantified as the drop in entropy [Sha01] for a fixed x according to a weighted set of its compatible supersequences, referred to as the *uncertainty set*. The uncertainty set, denoted by $\Upsilon_{n,x}$, contains all the supersequences that could have given rise to x upon $n-m$ deletions. In [ARR15], an alternative proof shows that this set's cardinality is independent of the details of x and that it is only a function of n and m. The weight distribution used in the computation of entropy is given by the number of occurrences or embeddings of a fixed subsequence in its compatible supersequences, i.e., the number of distinct ways x can be extracted from y upon a fixed number of deletions, denoted by $\omega_x(y)$. Furthermore, in the same work it is conjectured that the constant subsequences consisting of all `1`'s (or all `0`'s), $x = \texttt{11...1}$, and the alternating `1`'s and `0`'s, i.e., $x = \texttt{1010...}$, minimize and maximize the said entropy, respectively.

Despite the specific context in which the problem was first encountered, the underlying mathematical puzzle is a close relative of several well-known challenging problems in formal

languages, DNA sequencing and coding theory. In fact, the distribution of the number of times a string x appears as a subsequence of y, lies at the center of the long-standing problem of determining the capacity of deletion channels. More precisely, knowing this distribution would give us a maximum likelihood decoding algorithm for the deletion channel [Mit08]. In effect, upon receiving x, every set of $n-m$ symbols is equally likely to have been deleted. Thus, for a received sequence, the probability that it arose from a given codeword is proportional to the number of times it is contained as a subsequence in the originally transmitted codeword. More specifically, we have $p(y|x) = p(x|y)\frac{p(y)}{p(x)} = \omega_x(y)d^{n-m}(1-d)^m\frac{p(y)}{p(x)}$, with d denoting the deletion probability. Thus, as inputs are assumed to be a priori equally likely to be sent, we restrict our analysis to $\omega_x(y)$ for simplicity.

In this work, we first study several closely-related counting problems involving (super/sub)-sequences and then we revisit the aforementioned entropy question. It is worth pointing out that while questions on the combinatorics of random subsequences requiring closed-form expressions are already quite challenging, the problem tackled in this work and first raised in [ARR15], is further complicated by the dependence of entropy on the distribution of subsequence embeddings, i.e., the number of supersequences having specific embedding weights. To put this in contrast, in a related work [SF03], a closed-form expression is provided for computing the number of distinct subsequences that can be obtained from a fixed supersequence for the special case of two deletions, whereas here we need to account for the entire space of supersequences and characterize the number of times a given subsequence can be embedded in them in order to address the entropy question. Moreover, one would have to work out how these weights (number of embeddings) get shifted across their compatible supersequences when we move from one subsequence to another.

5.3.1.1 Results and Contributions

We first present an algorithm based on a run-length encoding of strings for counting the number of embeddings of x into y as a subsequence. Similar to how the cardinality of the set of supersequences that can project to a given subsequence, i.e., $|\Upsilon_{n,x}|$, depends only on their respective lengths, we prove that the number of supersequences that admit an initial embedding of a subsequence such that the last index of their initial embedding overlaps with their last bit, also depends only on $|y| = n$ and $|x| = m$. We then describe two clustering techniques that give rise to subspaces in $\Upsilon_{n,x}$ whose sizes depend only on n, m and the Hamming weight of x, but not the exact form of x. We derive analytic expressions, as well as a recurrence, for the cardinality of these sets. The approach and methodology used for deriving our clustering results depend heavily on the notion of initial or canonical embeddings of subsequences in their compatible supersequences, which provide further insight into the importance of initial embeddings.

Next, we consider the problem of enumerating supersequences that admit exactly a single occurrence of a subsequence, referred to as singletons, and give an analytic expression for their count. Furthermore, we prove a similar result for the maximization and minimization of the number of singletons by the constant and alternating strings, respectively.

Finally, we revisit the original entropy extremization question and prove the minimal entropy conjecture for the special cases of single and double deletions, i.e., for $n = m+1$ and $n = m+2$. The entropy result is obtained via a characterization of the number of strings with specific weights, along with an entropy decreasing operation. This is achieved using clustering techniques and a run-length encoding of strings: we identify groupings of supersequences with specific weights by studying how they can be constructed from a given subsequence using different insertion operations, which are in turn based on analyzing how runs of `1`'s and `0`'s can be extended or split. The methods used in the analysis of the underlying combinatorial problems, based on clustering techniques and the run-length encoding of strings may be of interest in their own right.

5.3.1.2 Structure

We begin by providing a survey of related work in Section 5.3.2. In Section 5.3.3, we introduce our notation and describe the main definitions, models, and building blocks used in our study. Next, in Section 5.3.4, we present an algorithm for counting the number of subsequence embeddings, which relies on the run-length encoding of strings. We then explore counting problems and clustering techniques in the space of supersequences including an analysis of a class of supersequences, referred to as singletons, that admit exactly a single embedding of a given subsequence and prove similar extremization results for their count. Finally, we turn to the original entropy question in Section 5.3.5 and prove the minimal entropy conjecture for the special cases of single and double deletions. Finally, we conclude by summarizing our findings and stating some open problems in Section 5.3.6.

5.3.2 Related Work

Studies involving subsequences and supersequences encompass a wide variety of problems that arise in various contexts such as formal languages, coding theory, computer intrusion detection and DNA sequencing to name a few. Despite their prevalence in such a wide range of disciplines, they remain largely unexplored and still present a considerable wealth of unanswered questions. In the realm of stringology and formal languages, the problem of determining the number of distinct subsequences obtainable from a fixed number of deletions, and closely related problems, have been studied extensively in [Cha76, FHS04, Hir99, HR00]. Perhaps it is worth noting that the same entropy minimizing and maximizing strings conjectured in [ARR15] and characterized in the present work, has been shown to lead to the minimum and maximum number of distinct subsequences, respectively. The problems of finding shortest common supersequences (SCS) and longest common subsequences (LCS) represent two well-known NP-hard problems [JL95, Mid95, MM04] that involve subproblems similar to our work. Finally, devising efficient algorithms for subsequence combinatorics based on dynamic programming for counting the number of occurrences of a subsequence in DNA sequencing is yet another important and closely related line of research [Rah06, ERW08].

In coding theory, and more specifically in the context of insertion and deletions channels, similar long-standing problems have been studied extensively, and yet many problems still remain elusive. This includes designing optimal coding schemes and determining the capacity of deletion channels, both of which incorporate the same underlying combinatorial problem addressed in the present work. Considering a finite number of insertions and deletions for designing correcting codes for synchronization errors [Ull67, SF03, KM13] and reconstructing the original string from a fixed subsequence [Gra15] represent two specific and related research areas. More recent work on the characterization of the number of subsequences obtained via the deletion channel [SD13, SGSD15, LL15], e.g., in terms of the number of runs in a string, shows great overlap with our work. A graph-theoretic approach for deletion correcting codes, closely related to our clustering analysis, including an alternative proof for the Hamming weight clustering given in Theorem (5.6) based on a different approach, is given in [CKK12].

An important body of research in this area is dedicated to deriving tight bounds on the capacity of deletion channels [DMP07, KMS10, RD13, CK14] and developing bounding techniques [OS14].

Perhaps rather surprisingly, the problem of determining the number of occurrences of a fixed subsequence in random sequences has not received the same amount and level of attention from the various communities. The state-of-the-art in the finite-length regime remains rather limited in scope. More precisely, the distribution of the number of occurrences constitutes a central problem in coding theory, with a maximum likelihood decoding argument, which represents the holy grail

in the study of deletion channels. A comprehensive survey, which among other things, outlines the significance of figuring out this particular distribution is given by Mitzenmacher in [Mit08].

5.3.3 Framework

We consider a memoryless source that emits symbols of the supersequence, drawn independently from the binary alphabet $\Sigma = \{0, 1\}$. Given an alphabet $\Sigma = \{0, 1\}$, Σ^n denotes the set of all Σ-strings of length n. Let p_α denote the probability of the symbol $\alpha \in \Sigma$ being emitted, which in the binary case simplifies to $p_\alpha = 0.5$. This means that the probability of occurrence of a random supersequence y is given by $P(y) = \prod_{i=1}^{n} p_{y_i}$. The probability of a subsequence of length m is defined in a similar manner. Throughout, we use $h(s)$ to denote the Hamming weight of the binary string s.

Notation We use the notation $[n] = \{1, 2, \ldots, n\}$ and $[n_1, n_2]$ to denote the set of integers between n_1 and n_2; individual bits from a string are indicated by a subscript denoting their position, starting at 1, i.e., $y = (y_i)_{i\in[n]} = (y_1, \ldots, y_n)$. We denote by $|S|$ the size of a set S, which for binary strings also corresponds to their length in bits. We also introduce the following notation: when dealing with binary strings, $[a]^k$ means k consecutive repetitions of $a \in \{0, 1\}$.

Subsequences and Supersequences Given $x \in \Sigma^m$ and $y \in \Sigma^n$, let $x = x_1x_2 \cdots x_m$ denote a subsequence obtained from $y = y_1y_2 \cdots y_n$ with a set of indexes $1 \leq i_1 < i_2 < \cdots < i_m \leq n$ such that $y_{i_1} = x_1, y_{i_2} = x_2, \ldots, y_{i_m} = x_m$. Subsequences are obtained by deleting characters from the original string and thus adjacent characters in a given subsequence are not necessarily adjacent in the original string.

Projection Masks We define $y_\pi = (y_i)_{i\in\pi} = x$ to mean that the string y filtered by the mask π gives the string x. Let π denote a set of indexes $\{j_1, \ldots, j_m\}$ of increasing order that when applied to y, yields x, i.e., $x = y_{j_1}y_{j_2} \cdots y_{j_m}$ and $1 \leq j_1 < j_2 \cdots j_m \leq n$.

Deletion Masks A deletion mask δ represents the set of indexes that are deleted from y to obtain x, i.e., $\delta_i \in [n] \setminus \pi$ and $|\delta| = n - m$, whereas a projection mask π denotes indexes that are preserved. Thus, similarly, δ is a subset of $[n]$ and the result of applying a mask δ on y is denoted by $y_\delta = x$.

Compatible Supersequences We define the *uncertainty set*, $\Upsilon_{n,x}$, as follows. Given x and n, this is the set of y strings that could project to x for some projection mask π.

$$\Upsilon_{n,x} := \{y \in \{0, 1\}^n : (\exists\pi)[y_\pi = x]\} = \{y \in \{0, 1\}^n : (\exists\delta)[y_\delta = x]\}$$

Number of Masks or Embeddings Let $\omega_x(y)$ denote the number of distinct ways that y can project to x:

$$\omega_x(y) := |\{\pi \in \mathcal{P}([n]) : y_\pi = x\}| = |\{\delta \in \mathcal{P}([n]) : y_\delta = x\}|$$

we refer to the number of masks associated with a pair (y, x) as the weight of y, i.e., the number of times x can be embedded in y as a subsequence.

Initial Projection Masks or Canonical Embeddings Given $y_\pi = x$, we define π to be initial if there is no lexicographically earlier mask π' such that $y_{\pi'} = x$. π' is a lexicographically earlier mask than π if, for some r, the smallest r members of π and π' are the same, but the $(r+1)$-th of π' is strictly smaller than that of π. Throughout, we will use $\tilde{\pi}$ to denote an initial projection mask. The first embedding of a subsequence x in y is also often referred to as the *canonical* embedding in the literature. Note that for a fixed mask or embedding π, the members of y up to the last member of π are completely determined if π is initial.

Run-Length Encodings A substring T of a string $Y = y_1 y_2 \dots y_n$ over Σ is called a *run* of Y if T is a consecutive sequence of the same character (i.e., $T \in \alpha^+$ for an $\alpha \in \Sigma$). Let $\mathcal{R}_{x,\alpha}$ denote the set of runs of α in x. The notion of run-length encoding will be central to our analysis. Given an n-bit binary string y, its *run-length encoding* (RLE) is the sequence $r_j = (a_j, b_j)$, $1 \le j \le m$, such that

$$y = [a_1]^{b_1}[a_2]^{b_2} \cdots [a_m]^{b_m}, \qquad m \le n.$$

with $a_j \in \{0,1\}$ and $b_j \in \{1,\dots,n\}$. This encoding is unique if we assume that $a_i \neq a_{i+1}$, at which point we only need to specify a single a_i (e.g., the first one) to deduce all the others.

Thus the RLE[3] for a string y is denoted by

$$y = [a_1; b_1, b_2, \dots, b_m].$$

When the value of a_1 is irrelevant, which will often be the case later on[4], we will drop it from the notation. Consecutive zeros or ones in a binary string will be referred to as *blocks* or *runs*.

Example 5.1 *Let $y =$ `0011010001`; then we have $y = [0; 2, 2, 1, 1, 3, 1]$ as the first bit is zero; and we have 2 zeros, 2 ones, 1 zero, 1 one, 3 zeros, 1 one. Alternatively, $[2, 2, 1, 1, 3, 1]$ designates simultaneously `0011010001` and `1100101110`.*

Entropy For a fixed subsequence x of length m, the underlying weight distribution used in the computation of the entropy is defined as follows. Upon receiving a subsequence x, we consider the set of compatible supersequences y of length n (denoted by $\Upsilon_{n,x}$) that can project to x upon $n-m$ deletions. Every $y \in \Upsilon_{n,x}$ is assigned a weight given by its number of masks $\omega_x(y)$, i.e., the number of times x can be embedded in y as a subsequence. We consider the conditional Shannon entropy $H(Y|X=x)$ where Y is confined to the space of compatible supersequences $\Upsilon_{n,x}$. The total number of masks in $\Upsilon_{n,x}$ is given by

$$\mu_{n,m} = \binom{n}{m} \cdot 2^{n-m} \tag{5.3}$$

Thus, forming the normalized weight distribution

$$P_x = \left\{ \frac{\omega_x(y_1)}{\mu_{n,m}}, \dots, \frac{\omega_x(y_n)}{\mu_{n,m}} \right\}. \tag{5.4}$$

Finally, for simplicity, we use $H_n(x)$ throughout this work to refer to the entropy of a distribution P corresponding to a subsequence x as defined below

$$H_n(x) = -\sum_i p_i \cdot \log_2(p_i) \tag{5.5}$$

[3]The notation $[n]$ is overloaded, but the difference will be clarified when it is not clear from the context.
[4]Indeed, if $x = y_\pi$, then $\overline{x} = \overline{y}_\pi$ and $\omega_x(y) = \omega_{\overline{x}}(\overline{y})$.

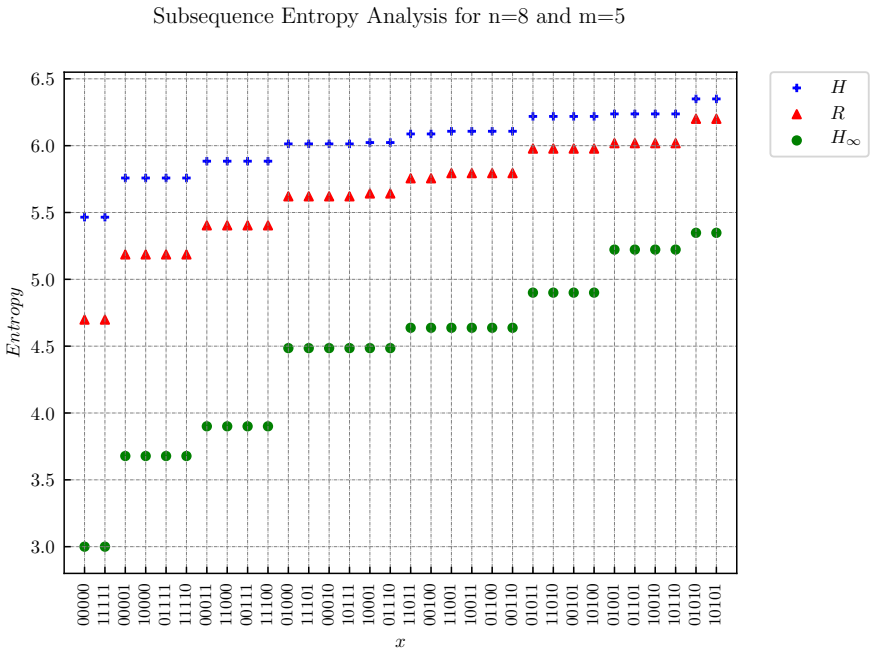

Figure 5.8: Shannon entropy H, second-order Rényi entropy R, and Min-entropy H_∞

where p_i is given by

$$p_i = \frac{\omega_x(y_i)}{\mu_{n,m}}.$$

5.3.4 Clustering Supersequences and Counting Subsequences

We now briefly review the results of the entropy analysis presented in [ARR15], in which it is conjectured that the constant/uniform string consisting of all `1`'s (or all `0`'s), $x = \texttt{11...1}$, and the alternating x string, i.e., $x = \texttt{1010...}$ minimize and maximize the entropy $H_n(x)$, respectively. To illustrate this, the plot given in Figure 5.8 shows the values of the min-entropy (H_∞), the second-order Rényi entropy (R) and the Shannon entropy (H) computed for all x strings of length 5, with $n = 8$.

Counting multisets: Throughout, we use the combinatorics of counting multisets, also referred to as the method of stars and bars, to enumerate all possibilities for placing n indistinguishable objects into bins marked by m distinguishable separators such that the resulting configurations are distinguished only by the number of objects present in each bin, which is given by $\binom{n+m-1}{n}$.

5.3.4.1 Compatible Supersequences and Subsequence Embeddings

Recall that for a fixed subsequence x of length m; we consider the set of y strings of length n ($n \geq m$), referred to as compatible supersequences, that can contain x as a subsequence embedding. The set of compatible supersequences is denoted by $\Upsilon_{n,x}$. It is known that the cardinality of $\Upsilon_{n,x}$

is independent of the form of x and that it is only a function of n and m.

$$|\Upsilon_{n,x}| = \sum_{r=m}^{n} \binom{n}{r} \tag{5.6}$$

We provided an alternative proof for this based on a simple recursion in [ARR15]. The original motivation for the clustering scheme presented here was to have a more fine-grained view of the distribution of masks in the space of supersequences. This approach led to the discovery of similar structures in $\Upsilon_{n,x}$, in that their cardinality does not depend on the form of x, analogous to how $|\Upsilon_{n,x}|$ depends only on n and m.

5.3.4.2 Counting Subsequence Embeddings via Runs

Efficient dynamic programming algorithms for computing the number of subsequence embeddings are known in the literature, e.g., a recursive algorithm requiring $\Theta(n \times m)$ operations [ERW08]. Here we provide an alternative algorithm, which is primarily based on the run-length encoding of strings.

Using the RLE notation, there are a few cases in which this question is easy to answer. For instance, if $y = [a; k_1, \ldots, k_\ell]$ and $x = [a; k'_1, \ldots, k'_\ell]$, with the same value of ℓ, i.e., we have the same number of blocks in x and y, then it is easy to see that there is a one-to-one sequential mapping of blocks between x and y. This allows us to enumerate the different masks depending on how they map the blocks to each other as follows:

$$\omega_x(y) = \prod_{i=1}^{\ell} \binom{k_i}{k'_i}. \tag{5.7}$$

If $y = [a; k_1, \ldots, k_\ell]$ and $x = [\overline{a}; k'_1, \ldots, k'_{\ell'}]$ do not start with the same character, we have to delete the first block to recover the case $y = [a; k_2, \ldots, k_\ell]$, and $x = [a; k'_1, \ldots, k'_{\ell'}]$. We will now suppose that x and y start with the same character. However, in the general case, this property does not hold and the number of blocks in x, and y can also be different.

Here we describe an algorithm wherein for a fixed pair of x and y strings, we structure and enumerate the corresponding space of masks by accounting for the number of different ways we can delete characters in order to merge blocks/runs such that we can recover the simple case given in Equation (5.7). In the more general case, let $y = [k_1, \ldots, k_\ell]$ and $x = [k'_1, \ldots, k'_{\ell'}]$.

Definition 5.1 *Let S be the set of maps $f : [\ell'] \to [\ell]$ that satisfy the following properties: f is strictly increasing and $f(i) \equiv i \bmod 2$. A function f will define a subset of masks, by specifying blocks that will have to be completely deleted. We group the masks according to a set of functions f that map indexes of blocks of x to indexes of blocks of y. Intuitively, f maps the i-th block of x to the block of y that contains the last letter of the i-th block of x. Therefore, all blocks of y between $f(i)+1$ and $f(i)$ that are not composed of the right letter have to be deleted such that we can recover the simple case in Equation (5.7).*

For the subsequent analysis, recall that k_i denotes the length of the run at index i, whereas k_i^* refers to the actual set of indexes of the i-th run.

Definition 5.2 *Let k_i^* denote the set of indexes belonging to the i-th block of y, i.e., $\{\sum_{j=1}^{i-1} k_j, \sum_{j=1}^{i-1} k_j + 1, \ldots, \sum_{j=1}^{i} k_j\}$, and $F(i)^* = \{f(i-1)+2, f(i-1)+4, \ldots, f(i)-1\}$, then a deletion mask δ corresponds to f if:*

- $\forall i : \cup_{j \in F(i)^*} k_j^* \subset \delta$

- $\forall i : k^*_{f(i)} \not\subset \delta$ *(this allows us to have a partition)*

We call ω_f the set of masks corresponding to f.

Theorem 5.2 *The family $(\omega_f)_{f\in S}$ defines a partition on the set of masks from y to x.*

Proof: We first show that for $f \neq f' \in S$, every deletion mask δ corresponding to f is different from every mask δ' associated with f' (i.e., $\omega_f \cap \omega_{f'} = \emptyset$). Since $f \neq f'$, we have a *smallest* integer $i \in [\ell]$ such that $f(i) \neq f'(i)$. We assume without loss of generality that $f'(i-1) = f(i-1) < f(i) < f'(i)$. Due to the condition on parity, $f(i+1) \neq f'(i)$. We distinguish between two cases:

- If $f'(i-1) < f(i+1) < f'(i)$, then $k_{f(i+1)} \not\subset \delta$, and $k_{f(i+1)} \subset \delta'$ since $f(i+1) \in F'^*(i-1)$.
- Conversely if $f(i-1) < f'(i) < f(i+1)$, then $k_{f'(i)} \not\subset \delta'$, and $k_{f'(i)} \subset \delta$ since $f'(i) \in F^*(i+1)$.

Therefore, we have $\delta \neq \delta'$. We now show that $\cup_{f\in S}\omega_f$ is the set of masks from y to x. We will use projection masks here as they are more suitable for this proof. Let π be a projection mask such that $y_\pi = x$. We let $\pi = \{\pi_1, \ldots, \pi_m\}$, where the π_i are in increasing order. Therefore, we have for all i, $y_{\pi_i} = x_i$. We define $\phi : [n] \to [\ell]$ to be the mapping that takes an index of y and returns the index of the block/run it belongs to, i.e., $\phi(a)$ returns the smallest i such that $\sum_{j=1}^{i} k_j \geq a$. We define f such that $f \in S$ and π is in ω_f, by $f(i) = \phi(\pi_{\sum_{j=1}^{i} k'_j})$. To prove that f is in S, note that given i:

- We have that $f(i) \leq f(i+1)$ since the π_i are in increasing order.
- Moreover, $\pi_{\sum_{j=1}^{i} k'_j}$ and $\pi_{\sum_{j=1}^{i+1} k'_j}$ correspond to indexes (of y) of opposite letter (if the first one is a 1, the second is a 0 and vice versa) since $\sum_{j=1}^{i} k'_j$ and $\sum_{j=1}^{i+1} k'_j$ correspond to indexes (of x) of opposite letter.

Therefore, $f(i)$ and $f(i+1)$ are of opposite parity and $f(i) < f(i+1)$.

We now prove that π corresponds to f. For a fixed $i \in [\ell]$, let $k^*_{f(i-1)} = b^{k_{f(i-1)}}$, i.e., the $f(i-1)$-th block of y is made of letters b. Therefore, $k^*_t = b^{k_t}$ for $t \in F(i)^*$, since t has the same parity as $f(i-1)$. Moreover, we have $b = x_{\sum_{j=1}^{i-1} k'_j}$ according to the definition of f. So for every index h between $\sum_{j=1}^{i-1} k_j + 1$ and $\sum_{j=1}^{i} k_j$, $x_h = y_{\pi_h} = \bar{b}$, and for $t \in F(i)^*$, we have $k^*_t \cap \pi = \emptyset$ (equivalently with the deletion mask δ, $k^*_t \subset \delta$). By definition, $\pi_{\sum_{j=1}^{i} k_j} \in k^*_{f(i)}$ so $\pi \cap k^*_{f(i)} \neq \emptyset$ (equivalently with the deletion mask δ, $k^*_{f(i)} \not\subset \delta$). □

Definition 5.3 *We now introduce the quantity Ω_f, which is the number of masks corresponding to f for $f \in S$.*

$$\Omega_f = \prod_{i=0}^{\ell} \binom{\sum_{j\in F(i)} k_j}{k'_i} - \binom{\sum_{j\in F(i)\setminus\{f(i)\}} k_j}{k'_i} \tag{5.8}$$

where

$$F(i) = \{f(i-1)+1, f(i-1)+3, \ldots, f(i)\}$$

Theorem 5.3 $\Omega_f = |\omega_f|$ *for all $f \in S$.*

Proof: Upon the deletion induced by f, we obtain a string of the form $[\sum_{j\in F(1)} k_j, \ldots, \sum_{j\in F(\ell)} k_j]$. Therefore, we have the same number of blocks in both the y string as well as the x string, and the number of masks can be computed easily as shown in Equation (5.7). We first count the number of ways to choose k'_i elements from $k_{F(i)}$ and then subtract the number of combinations not using any of the $k_{f(i)}$. □

Remark *We can note that $F(i)$ and $F(i)^*$ form a partition of $[\ell]$.*

Following from the preceding theorems, the total number of masks can be computed as follows

$$\omega_x(y) = \sum_{f\in S} \Omega_f. \tag{5.9}$$

By summing over all $f \in S$, we get the total number Ω of compatible masks. Note that it may happen that $\Omega_f = 0$; this happens when we try to trace a large block of x from a smaller block of y.

The Set S: We now determine the size of S, as a function of ℓ and ℓ'. Let this size be denoted by $\sigma(\ell', \ell)$. We denote $u = \lfloor(\ell - \ell')/2\rfloor$. If $f(1) = 1$, then we get $\sigma(\ell' - 1, \ell - 1)$; if $f(1) = 3$, we get $\sigma(\ell' - 1, \ell - 3)$, etc. We also know that $\sigma(x, x) = 1$ for all x, and that $\sigma(x, y) = 0$ for all x, y such that $y < x$. We, therefore get the following recurrence:

$$\sigma(\ell', \ell) = \sum_{i=0}^{u} \sigma(\ell' - 1, \ell - 1 - 2i)$$

Iterating this recursion, we get

$$\sigma(\ell', \ell) = \sum_{i=0}^{u} \sum_{j=0}^{u-i} \sigma(\ell' - 2, \ell - 2 - 2i - 2j),$$

and grouping the terms yields

$$\sigma(\ell', \ell) = \sum_{i=0}^{u} (i + 1)\sigma(\ell' - 2, \ell - 2 - 2i).$$

We now describe a direct combinatorial argument which gives a closed form formula for $\sigma(\ell', \ell) = |S|$. First note that if $\ell \not\equiv \ell' \bmod 2$ then ℓ cannot be in the image of f. So let $\tilde{\ell} = \ell$ if $\ell \equiv \ell' \bmod 2$ and $\tilde{\ell} = \ell - 1$ if not. Now the problem is to choose $[\ell']$ elements from $[\tilde{\ell}]$ such that all the gaps have even width. Equivalently, we are interleaving the ℓ' chosen elements with $u = (\tilde{\ell} - \ell')/2$ gap-segments of width 2. The number of ways to do this is plainly

$$|S| = \sigma(\ell', \ell) = \binom{\ell' + u}{u}. \tag{5.10}$$

Example 5.2 *For $y =$ `0000111100001111` and $x =$ `0011`, we obtain $\omega_y(x) = 300$. We now compute the number of embeddings using the run-based algorithm described above. We have $\ell = 4$, $\ell' = 2$ and $u = (\tilde{\ell} - \ell')/2$, which means the size of S is $|S| = \sigma(\ell', \ell) = \binom{l'+u}{u} = \binom{2+1}{1} = 3$. The three deletions $S = \{f_1, f_2, f_3\}$ are computed as follows: $y_{f_1} = [k_1, k_2] =$ `00001111`, which amounts to $\omega_{f_1} = \binom{4}{2}\binom{4}{2} = 36$. Similarly, for f_2 and f_3, we get $y_{f_1} = [k_1 + k_3, k_4] =$ `000000001111` and $y_{f_1} = [k_1, k_2 + k_4] =$ `000011111111`, the two of which add up to $2 \times \left(\binom{8}{2}\binom{4}{2} - \binom{4}{2}\right) = 2 \times 132 = 264$. So the total is $\omega_y(x) = \sum_{f\in S} \Omega_f = 36 + 132 + 132 = 300$.*

5.3.4.3 From Maximal Initials to Hamming Clusters

Definition 5.4 *Let $\Upsilon^c_{n,x}$ be the cluster of supersequences that have c extra `1`'s with respect to x, where $0 \leq c \leq n-m$.*

$$\Upsilon^c_{n,x} = \{y \in \Upsilon_{n,x} \mid h(y) - h(x) = c\}.$$

The set of compatible supersequences is thus broken down into $n-m+1$ disjoint sets indexed from 0 to $n-m$ such that strings in cluster c contain $h(x)+c$ `1`'s:

$$\Upsilon_{n,x} = \bigcup_{c=0}^{n-m} \Upsilon^c_{n,x}.$$

Definition 5.5 *Maximal initials represent y strings for which the largest index of their initial mask, $\tilde{\pi}$, overlaps with the last bit of y. In other words, the last index of the canonical embedding of x in y overlaps with the last bit of y. Recall that we use $\tilde{\pi}$ to denote a mask π that is initial.*

$$\mathcal{M}_{n,x} = \{y \in \Upsilon_{n,x} \mid (\exists\tilde{\pi})[y_{\tilde{\pi}} = x \wedge \max(\tilde{\pi}) = |y| = n]\}.$$

Similarly, we define a clustering for maximal initials based on the Hamming weight of the y strings

$$\mathcal{M}^c_{n,x} = \{y \in \mathcal{M}_{n,x} \mid h(y) = h(x) + c\}.$$

Example 5.3 *For example, the initial embedding of $x =$ `1011` in $y =$ `110011` given by $\tilde{\pi} = \{1,3,5,6\}$ is maximal, whereas its initial embedding in $y' =$ `101011` given by $\tilde{\pi}' = \{1,2,3,5\}$ is not maximal as the last index of $\tilde{\pi}'$ does not overlap with the position of the last bit of y'.*

A more exhaustive example illustrating these concepts is given in Table 5.15. In addition to the distribution of weights, i.e., the number of masks per y, clusters and maximal initials are indicated by horizontal separators and bold font, respectively.

Theorem 5.4 *For given n, the cardinality of $\mathcal{M}_{n,x}$ is independent of the exact x.*

Proof: It is clear that every n-element sequence that has x as an m-element subsequence has a unique initial mask $\tilde{\pi}$ that gives x. Furthermore, if we fix π, then the members of y up to the last member of π are completely determined if π is initial. To see this, consider the case $i \in \tilde{\pi}$, then y_i (the i-th member of y) must correspond to x_j, where i is the j-th smallest member of $\tilde{\pi}$. If $i \notin \tilde{\pi}$, but smaller than $max(\tilde{\pi})$, then the i-th member of y must correspond to x_{j+1}, where j is the number of members of $\tilde{\pi}$ smaller than i. The latter follows because if this bit were x_{j+1}, then the given π would not be initial.

We also need to observe that for a given $\tilde{\pi}$, there always exists a y that has x initially in $\tilde{\pi}$: suppose that x starts with a `0`, we set all the bits of y before $\tilde{\pi}$ to be `1`. For a given value ℓ of $max(\tilde{\pi})$ - which can range from m to n - there are exactly $\binom{\ell-1}{m-1}$ $\tilde{\pi}$'s, one for each selection of the other $m-1$ members of $\tilde{\pi}$ amongst the $\ell-1$ values less than ℓ.

Moreover, here we have an additional constraint, namely that the initial masks should be maximal as well, i.e., $max(\tilde{\pi}) = n$. This means that $\ell = n$ and so we can count the number of distinct initials for the remaining $m-1$ elements of x in the remaining $(n-1)$-long elements of y strings, which is simply given by

$$|\mathcal{M}_{n,x}| = |\mathcal{M}_{n,m}| = \binom{n-1}{m-1} \tag{5.11}$$

Table 5.15: Clusters, maximal initial projection masks and distribution of weights.

$x = \texttt{110}$			$x = \texttt{101}$		
y	$\tilde{\pi}$	ω	y	$\tilde{\pi}$	ω
00110	$\{\mathbf{3},\mathbf{4},\mathbf{5}\}$	1	00101	$\{\mathbf{3},\mathbf{4},\mathbf{5}\}$	1
01010	$\{\mathbf{2},\mathbf{4},\mathbf{5}\}$	1	01001	$\{\mathbf{2},\mathbf{3},\mathbf{5}\}$	2
01100	$\{2,3,4\}$	2	01010	$\{2,3,4\}$	1
10010	$\{\mathbf{1},\mathbf{4},\mathbf{5}\}$	1	10001	$\{\mathbf{1},\mathbf{2},\mathbf{5}\}$	3
10100	$\{1,3,4\}$	2	10010	$\{1,2,4\}$	2
11000	$\{1,2,3\}$	3	10100	$\{1,2,3\}$	1
01101	$\{2,3,4\}$	1	01011	$\{2,3,4\}$	2
01110	$\{\mathbf{2},\mathbf{3},\mathbf{5}\}$	3	01101	$\{\mathbf{2},\mathbf{4},\mathbf{5}\}$	2
10101	$\{1,3,4\}$	1	10011	$\{1,2,4\}$	4
10110	$\{\mathbf{1},\mathbf{3},\mathbf{5}\}$	3	10101	$\{1,2,3\}$	4
11001	$\{1,2,3\}$	2	11001	$\{\mathbf{1},\mathbf{3},\mathbf{5}\}$	4
11100	$\{1,2,4\}$	6	11010	$\{1,3,4\}$	2
11011	$\{1,2,3\}$	1	10111	$\{1,2,3\}$	3
11101	$\{1,2,4\}$	3	11011	$\{1,3,4\}$	4
11110	$\{\mathbf{1},\mathbf{2},\mathbf{5}\}$	6	11101	$\{\mathbf{1},\mathbf{4},\mathbf{5}\}$	3

Clearly the cardinality of the set of maximal initials is independent of the form of x and depends only on n and m. □

Remark *Note that if we extend the analysis in the proof of Theorem 5.4 and let ℓ run over the range $[m, n]$, we can count all the distinct initial embeddings in $\Upsilon_{n,x}$, given by $\sum_{\ell=m}^{n} \binom{\ell-1}{m-1}$.*

Moreover, since the bits beyond $max(\tilde{\pi})$ are completely undetermined, for a given $\tilde{\pi}$, there are exactly $2^{n-max(\tilde{\pi})}$ y's that have $\tilde{\pi}$ in common, which, incidentally, provides yet another proof for the fact that $|\Upsilon_{n,x}|$ is a function of only n and m since $|\Upsilon_{n,x}| = \sum_{\ell=m}^{n} \binom{\ell-1}{m-1} 2^{n-\ell}$. This allows us to choose the x comprising m 0's and the result in Equation (5.6) follows immediately.

Theorem 5.5 *All x strings of length m that have the same Hamming weight, give rise to the same number of maximal initials in each cluster.*

$$\forall x, x' \in \Sigma^m, h(x) = h(x') \implies |\mathcal{M}^c_{n,x}| = |\mathcal{M}^c_{n,x'}|.$$

Proof: We now describe a simple combinatorial argument for counting the number of maximal initials in each cluster indexed by c, i.e., a grouping of all $y \in \Upsilon^c_{n,x}$ such that $h(y) = h(x) + c$. Let p and q denote the number of additional 0's, and 1's contributed by each cluster, respectively. Furthermore, let a and b denote the number of 1's and 0's in x, respectively.

Similar to the method used in the proof of Theorem 5.4, due to maximality we fix the last bit of y and x, and consider $y' = y - tail(y)$ and $x' = x - tail(x)$ where $tail(s)$ denotes the last bit of s. Now the problem amounts to counting distinct initials of length $m-1$ in $(n-1)$-long elements in each cluster by counting the number of ways distinct configurations can be formed as a result of distributing c 1's and $(n-m-c)$ 0's around the bars/separators formed by the b 0's and a 1's in x, respectively.

We now need to observe that to count such strings with distinct initials, we can fix the $m-1$ elements of x' as distinguished elements and count all the unique configurations formed by distributing p indistinguishable `0`'s and q indistinguishable `1`'s among bins formed by the fixed `1`'s and `0`'s of x' such that each such configuration is distinguished by a unique initial.

Equivalently, we are counting the number of ways we can place the members of x' among $n-1$ positions comprising p `0`'s and q `1`'s without changing the relative order of the elements of x' such that these configurations are uniquely distinguished by the positions of the $m-1$ elements.

Intuitively, the arrangements are determined by choosing the positions of the $m-1$ bits of x': by counting all the unique distributions of bits of opposite value around the elements of x', we are simply displacing the elements of x' in the $n-1$ positions, thereby ensuring that each configuration corresponds to a unique initial.

Note that this coincides exactly with the multiset coefficient (computed via the method of stars and bars) as we can consider the elements of the runs of x to be distinguished elements forming bins among which we can distribute indistinguishable bits of opposite value to count the number of configurations that are distinguished only by the number of `1`'s and `0`'s present in the said bins.

Thus we count the number of unique configurations formed by distributing p `0`'s and q `1`'s among the a `1`'s and b `0`'s of x, respectively. The total count for each cluster c is given by: $\binom{p+a-1}{p}\binom{q+b-1}{q}$, which expressed in terms of the Hamming weight of x gives

$$|\mathcal{M}^c_{n,x}| = \binom{(n-m-c)+h(x)-1}{n-m-c}\binom{c+(m-h(x))-1}{c} \tag{5.12}$$

With the total number of maximal initials in $\Upsilon_{n,x}$ given by

$$|\mathcal{M}_{n,x}| = \sum_{c=0}^{n-m} |\mathcal{M}^c_{n,x}| = \binom{n-1}{m-1}.$$

□

Theorem 5.6 *The size of a cluster is purely a function of n, m, c and $h(x)$*

$$\forall x, x' \in \Sigma^m, h(x) = h(x') \implies |\Upsilon^c_{n,x}| = |\Upsilon^c_{n,x'}|$$

Proof: Let ℓ denote the position of the last bit of y ranging from $|x| = m$ to $|y| = n$. Starting from a fixed x string, we enumerate all y strings in cluster c by considering maximal initials within the range of ℓ, i.e., $\ell \in [m, \ldots, n]$.

Let g denote the number of `1`'s belonging to the surplus bits in cluster c constrained within the range of the maximal initial, $[1, \ldots, \ell]$. For each ℓ, compute $|\mathcal{M}^g_{\ell,x}|$ and count the combinations of choosing the remaining $c-g$ additional bits in the remaining $n-\ell$ bits. Let $UB = \min(c, \ell-m)$ and $LB = \max(0, c-(n-\ell))$ and thus we get the following:

$$|\Upsilon^c_{n,x}| = \sum_{\ell=m}^{n} \sum_{g=\max(0,c-(n-\ell))}^{\min(c,\ell-m)} |\mathcal{M}^g_{\ell,m}| \binom{n-\ell}{c-g} \tag{5.13}$$

Finally, inserting Equation (5.12) into Equation (5.13) gives

$$|\Upsilon^c_{n,x}| = \sum_{\ell=m}^{n} \sum_{g=LB}^{UB} \binom{(\ell-m-g)+h(x)-1}{\ell-m-g}\binom{g+(m-h(x))-1}{g}\binom{n-\ell}{c-g}. \tag{5.14}$$

As shown in Equation (5.14), $|\Upsilon^c_{n,x}|$ depends on the length and the Hamming weight of x, but it is independent of the exact form of x. □

5.3.4.4 Simple closed form expression for the size of a cluster

We have shown that $|\Upsilon^c_{n,x}|$ is independent of the form of x. We can now derive a more simplified analytic expression for this count by considering an x string of the following form: $x = \texttt{11...11}_a\texttt{00...0}_m$, i.e., a 1's followed by b 0's, with $a > 0$ and $b = m - a$.

The y strings in each cluster are precisely the strings of length n that have $a + c$ 1's in them (and $n - a - c$ 0's) where the a-th 1 (i.e., the last one in an initial choice for x) occurs before at least b 0's. Clearly, there are $\binom{n}{a+c}$ strings with exactly $a + c$ 1's, but some of these will violate the second principle. To find an expression for counting the valid instances, we sum over the positions of the a-th 1, which must be between a and $a + z$, where $z = n - a - b - c$ is the number of added 0's. Thus we get the following expression

$$|\Upsilon^c_{n,x}| = \sum_{p=h(x)}^{h(x)+z} \binom{p-1}{h(x)-1}\binom{n-p}{c}. \tag{5.15}$$

With $z = n - m - c$ and p denoting the index of the a-th 1, we thus count the number of ways of picking 1's before p and the c 1's after p. Note that for $h(x) = 0$, the cardinality of cluster c is simply given by $\binom{n}{c}$.

5.3.4.5 Recursive expression for the size of a cluster

We present a recurrence for computing the size of a cluster by considering overlaps between the first bits of x and y, respectively. Let $\bullet$ and ε denote concatenation and the empty string, respectively. Moreover, let x' be the tail of x (resp. y' the tail of y).

- $\Upsilon^c_{n,0\bullet x} = \Upsilon^c_{n-1,x} + \Upsilon^{c-1}_{n-1,0\bullet x}$
  - First term: first bit of y is 0, find x' in y'
  - Second term: first bit of y is 1 (part of cluster), so we reduce c and find x in y'
- $\Upsilon^c_{n,1\bullet x} = \Upsilon^c_{n-1,x} + \Upsilon^c_{n-1,1\bullet x}$
  - Same arguments as above, but for x starting with 1
- Base cases:
  - $\Upsilon^0_{n,0\bullet x} = \Upsilon^0_{n-1,x}$
  - $\Upsilon^0_{n,1\bullet x} = \Upsilon^0_{n-1,x} + \Upsilon^0_{n-1,1\bullet x}$
  - $\Upsilon^c_{n,\varepsilon} = \binom{n}{c}$
  - if $c + |x| > n$ then return 0 else $\Upsilon^c_{n,x}$

It is worth pointing out that since this recursion depends on the form of x, i.e., whether or not x starts with a 0 or 1, it does not explicitly capture the bijection between clusters of x strings that have the same Hamming weight, as proved in Theorem 5.6.

5.3.4.6 Enumerating Singletons via Runs

Let *singletons* define supersequences in $\Upsilon_{n,x}$ that admit exactly a single mask for a fixed subsequence x of length m, i.e.; they give rise to exactly a single occurrence of x upon $n-m$ deletions. We use $\mathcal{S}_{n,x}$ to denote this set.

$$\mathcal{S}_{n,x} = \{y \in \Upsilon_{n,x} |\ \omega_x(y) = 1\}.$$

To compute the cardinality of $\mathcal{S}_{n,x}$, we describe a counting technique based on splitting runs of `1`'s and `0`'s in x according to the following observations: (i) inserting bits of opposite value to either side of the framing bits in x, i.e., before the first or after the last bit of x, does not alter the number of masks. (ii) splitting runs of 0's and 1's in x, i.e., insertion of bits of opposite value in between two identical bits, does not modify the count. This amounts to counting the number of ways that singletons can be obtained from a fixed x string via weight preserving insertions.

The number of possible run splittings corresponds to the number of distinct ways that c `1`'s and $(n-m-c)$ `0`'s can be placed in between the bits of the runs of `0`'s and `1`'s in x, respectively. Again, this count is given by the multiset number $\binom{a+b-1}{a}$, where we count the number of ways a indistinguishable objects can be placed into b distinguishable bins. Note that the number of singletons depends heavily on the number of runs in x and their corresponding lengths. The counting is done by summing over all $n-m$ clusters and computing the configurations that lead to singletons as a function of the runs in x and the number of additional `1`'s and `0`'s contributed by each cluster at index c.

In order to do this computation, we first count the number of insertions slots in x as a function of its runs of `1`'s and `0`'s, given by $\rho_0(x)$ and $\rho_1(x)$, respectively. Let r_i^j be a run with i and j denoting its first and last index and let $\rho_\alpha(x)$ denote the number of insertion slots in x as a function of its runs of α. To compute $\rho_\alpha(x)$, we iterate through the runs of α and in x and count the number of indexes at which we can split runs as follows

$$\rho_\alpha(x) = \sum_{r \in \mathcal{R}_{x,\alpha}} f(r) \tag{5.16}$$

where

$$f(r) = \begin{cases} |r_i^j| + 1, & \text{if } i = 1 \wedge j = n \\ |r_i^j|, & \text{if } (i = 1 \wedge j < n) \vee (i > 1 \wedge j = n) \\ |r_i^j| - 1, & \text{otherwise} \end{cases} \tag{5.17}$$

Note that if either the first bit or the last bit of a run overlaps with the first or last bit of x, the number of bars is equal to the length of the run. If the said indexes overlap with neither the first nor the last bit of x, the count is equal to the length of the run minus 1, and finally, if both indexes overlap with the first and last bit of x the count is equal to the length of the run plus 1.

We can now count the total number of singletons for given n and x as follows. Let c and b $(b = n-m-c)$ denote the number of `1`'s and `0`'s contributed by the c-th cluster, and the total number of singletons is given by

$$|\mathcal{S}_{n,x}| = \binom{n-m+\rho_1(x)-1}{n-m} + \sum_{c=1}^{n-m-1} \binom{b+\rho_1(x)-1}{b}\binom{c+\rho_0(x)-1}{c} + \binom{n-m+\rho_0(x)-1}{n-m} \tag{5.18}$$

The first and last terms correspond to the number of singletons in the first and last cluster, respectively, where we insert either `1`'s or `0`'s, but not both. The summation over the remaining clusters counts the configurations that incorporate both additional `1`'s and `0`'s.

Theorem 5.7 *The constant (i.e., $x =$ `11...1` or $x =$ `00...0`) and the alternating x strings maximize and minimize the number of singletons, respectively.*

Proof: This follows immediately from a maximization and minimization of the number of runs in x, i.e., $\rho_\alpha(x)$. In the case of the all 1's x string, which comprises a single run, every index in x can be used for splitting. Conversely, the alternating x has the maximum number of runs $|\mathcal{R}| = m$, where $\forall.r \in \mathcal{R}_x : |r| = 1$, thus splittings are not possible, i.e., no operations of type (ii), and the insertions are confined to pre-pending and appending bits of opposite values to the first and last bit of x, respectively. □

5.3.5 Entropy Minimization

We now prove the minimal entropy conjecture for the special cases of one and two deletions. Our approach incorporates two key steps: first, we work out a characterization of the number of y strings that have specific weights $\omega_x(y)$. We then consider the impact of applying an entropy decreasing transformation to x, denoted by $g(x)$, and prove that this operation shifts the weights in the space of supersequences such that it results in a lowering of the corresponding entropy. This is achieved using clustering techniques and a run-length encoding of strings: we identify groupings of supersequences with specific weights by studying how they can be constructed from a given subsequence using different insertion operations, which are in turn based on analyzing how runs of `1`'s and `0`'s can be extended or split.

Definition 5.6 *We now define the transformation g on strings of length m as follows:*

$$g([k_1, \ldots, k_\ell]) = \begin{cases} [k_1 + k_2, k_3, \ldots, k_\ell] & \text{if } \ell > 1 \\ g([m]) = [m] \end{cases} \tag{5.19}$$

Hence g is a "merging" operation that connects the two first blocks together. As we shall see, g decreases the entropy. Thus, one can start from any subsequence x and apply the transformation g until the string becomes $[m]$, i.e., $[0]^m$ or $[1]^m$. As a result, $[m]$ exhibits minimal entropy and thus the highest amount of leakage in the original key exchange problem. Note that, as indicated implicitly in the definition above, this transformation always reduces the number of runs by one by flipping the first run to its complement.

Thus we avoid cases where merging two runs would lead to connecting to a third neighboring run, thereby resulting in a reduction of runs by two. For example, g transforms the string $x = \texttt{1001110} = [1; 1, 2, 3, 1]$ into $x = \texttt{0001110} = [0; 3, 3, 1]$, as opposed to $x = \texttt{1111110} = [1; 6, 1]$.

The plots shown in Figure 5.9 illustrate the impact of the transformation g on the weight distribution as we move from $x = \texttt{101010}$ to $x' = \texttt{000000}$ ($\texttt{101010} \to \texttt{001010} \to \texttt{111010} \to \texttt{000010} \to \texttt{111110} \to \texttt{000000}$).

5.3.5.1 Single Deletions

In this subsection, we consider the case of a single deletion. Let x be a fixed string of length m. We study the space of y strings of length $n = m + 1$ that can be masked to yield x, i.e., $Y_1 = \{y \in \{0,1\}^n \mid \exists \delta \in \mathcal{P}([n]), y_\delta = x \text{ and } |\delta| = 1\}$. Recall that we associate a weight $\omega_1(x, y)$ to each $y \in Y_1$, defined as the number of ways that y can be masked into x. Finally, we define the entropy associated to x as the Shannon entropy of the variable $Z \in \{0,1\}^n$ having distribution

$$\Pr[Z = y] = \frac{1}{\mu_1} w_x(y).$$

where $\mu_1 = \sum_{y \in \Upsilon_{n,x}}$, which for the case $m = n - 1$ gives $\omega_1(x, y) = \binom{n}{m} 2^{n-m} = \binom{n}{n-1} 2^{n-(n-1)} = 2n$.

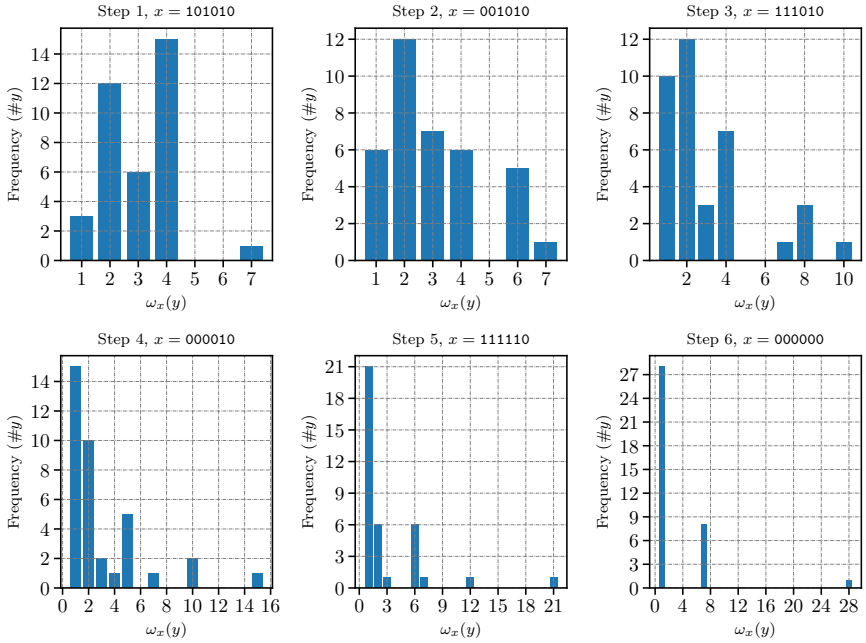

Figure 5.9: Impact of the transformation g on the weight distribution for converting $x = \texttt{101010}$ to $x' = \texttt{000000}$, with $n = 8, m = 6$.

Clustering Supersequences via Single Insertions Let $x = [k_1, \ldots, k_\ell]$. A string $y \in Y_1$ can take only one of the following forms:

1. $y = [k_1, \ldots, k_{i-1}, k_i + 1, k_{i+1}, \ldots k_\ell]$ for some $i \in [\ell]$;
2. $y = [k_1, \ldots, k_{i-1}, k'_i, 1, k''_i, k_{i+1}, \ldots, k_\ell]$, for some $i \in [\ell]$ and where $k'_i + k''_i = k_i$ and $k'_i \neq 0$ and $k''_i \neq 0$;
3. $y = [1, k_1, \ldots k_\ell]$;
4. $y = [k_1, \ldots, k_\ell, 1]$.

The first case will be referred to as a "weight increasing insertion", denoted by $1/0$, which corresponds to extending runs/blocks. The last three cases will be referred to as "weight preserving insertions", and denoted by $0/1$, corresponding to splitting runs or adding a new run of length 1. For the remainder of our discussion, a/b means: "a weight increasing insertions and b weight preserving insertions".

Lemma 5.8 *Y_1 is composed of[5]:*

- *ℓ weight increasing insertions, resulting in strings of respective weights $k_1 + 1, k_2 + 1, k_3 + 1, \ldots, k_\ell + 1$; and*
- *$m - \ell + 2$ weight preserving insertions, i.e., strings of weight 1.*

[5] A sanity check can be done to verify that we do not miss any strings, since $\binom{m+1}{m} + \binom{m+1}{m+1} = \ell + m - \ell + 2$.

Minimal Entropy For Single Deletions

Lemma 5.9 *The transformation g decreases the entropy $H_n(x)$ for single deletions, i.e., $m = n - 1$.*

Proof: The proof consists of computing the difference between the entropy before and after applying g, i.e., $\Delta_1 = H_n(x) - H_n(g(x))$, and showing that this difference is positive. From Lemma 5.8, after applying g,

- The weight increasing insertions give $\ell - 1$ strings of respective weights $k_1 + 1 + k_2 + 1 - 1, k_3 + 1, \ldots, k_\ell + 1$.
- The weight preserving insertions give $m + 2 - (\ell - 1)$ strings of weight 1.

We now compute the difference of the entropies thanks to the analyses of $[k_1, k_2, k_3, \ldots, k_\ell]$ and $[k_1 + k_2, k_3, \ldots, k_\ell]$, which after simplification gives

$$\Delta_1(k_1, \ldots, k_\ell) = (k_1 + 1) \log \frac{1}{k_1 + 1} + (k_2 + 1) \log \frac{1}{k_2 + 1} - (k_1 + k_2 + 1) \log \frac{1}{k_1 + k_2 + 1}$$

To show that this is positive when $k_1 \geq 1$ and $k_2 \geq 1$, it suffices to compute the partial derivatives along each axis, which are positive, and evaluate the function in $k_1 = k_2 = 1$, which is also positive. □

Corollary 5.10 *For all n and any subsequence x of length $m = n - 1$, we have*

$$H_n(x) \geq H_n([m]),$$

with equality only if $x \in \{0^m, 1^m\}$.

Proof: Given any $x \neq [m]$ of length $m = n - 1$, it can be transformed into the string $[m]$ by a series of consecutive g operations, as defined in Definition 5.6. Each such operation can only decrease the entropy, as shown in Lemma 5.9, and thus we get a proof for the fact that $H_n(x) \geq H_n(0^m)$. □

Remark *It is worth pointing out that for the special case of single deletions, the minimization of entropy by the constant string, $x = [m]$, can also be proved using a simple combinatorial argument as follows. For $m = n - 1$, in cluster $c = 1$ we get a single y string with maximum weight, $\omega_y(x) = \binom{n}{m}$, corresponding to $y = [n]$ and $x = [m]$, and the remaining strings in cluster $c = 0$ are all singletons, $\omega_x(y) = 1$. This is clearly the most concentrated distribution and hence the least entropic one.*

5.3.5.2 Double Deletions

In the case of two deletions, there are three types of insertions to consider; using the notation introduced in the previous subsection, these are 2/0, 1/1, and 0/2 insertions. For a fixed string $x = [k_1, \ldots, k_\ell]$, we now analyze each case to account for the corresponding number of supersequences and their respective weights in each cluster. We will then study how this distribution changes when we go from x to $g(x)$ in order to prove the following lemma:

Lemma 5.11 *The transformation g decreases the entropy $H_n(x)$ for single deletions, i.e., $m = n - 2$.*

Note that while this technique could be applied to a higher number of insertions, the complexity of the analysis blows up already for two deletions, as the next subsection will show.

Clustering Supersequences via Double Insertions

Case $2/0$ The case $2/0$ corresponds to the situation where the insertions do not create new blocks. This happens when both bits are added to the same block, or when they are added to two different blocks, as follows.

The former corresponds to

$$y = [k_1, \ldots, k_{i-1}, k_i + 2, k_{i+1}, \ldots, k_\ell]$$

for some $i \in [\ell]$, which has weight $\omega_x(y) = \binom{k_i+2}{2}$. There are ℓ strings of this type.

The latter corresponds to

$$y = [k_1, \ldots, k_{i-1}, k_i + 1, k_{i+1}, \ldots, k_{j-1}, k_j + 1, k_{j+1}, \ldots, k_\ell]$$

for $1 \leq i < j \leq \ell$, and has weight $\omega_x(y) = (k_i + 1)(k_j + 1)$. There are $\frac{\ell(\ell-1)}{2}$ strings with this weight. In total, there are $\frac{\ell(\ell+1)}{2}$ strings for the case $2/0$.

Case $0/2$ In the $0/2$ case, there are only weight preserving insertions, hence all strings have weight 1. Weight preserving insertions may happen in a single block, or in two separate blocks. To ease notation, we introduce

$$\widetilde{k}_i = \begin{cases} k_i - 1 & \text{if } i \in [2, \ell - 1] \\ k_i & \text{if } i = 1 \text{ or } i = \ell \end{cases}$$

The different treatments for "endpoints" 1 and ℓ correspond to cases $[1, k_1, \ldots, k_\ell]$ and $[k_1, \ldots, k_\ell, 1]$, whereas a weight preserving insertion in the i-th block can happen at only $k_i - 1$ places.

- If we insert into the first or the last block, we choose respectively k_1 and k_ℓ positions, i.e., there are respectively k_1 and k_ℓ different strings.
- If we insert into any other block i, we choose amongst $k_i - 1$ positions, which yields $k_i - 1$ different strings.
- If we insert in different blocks, we apply the same analysis twice, independently, which gives $\widetilde{k}_i\widetilde{k}_j$ different strings.
- If we insert twice in the same block, we get $\binom{\widetilde{k}_i+1}{2}$ different strings.

In the end, the total number of $0/2$ insertions is

$$\sum_{1 \leq i < j \leq \ell} \widetilde{k}_i\widetilde{k}_j + \sum_{i=1}^{\ell} \binom{\widetilde{k}_i + 1}{2}$$

Example 5.4 *If $k_1 = \cdots = k_\ell = 1$, so that $\widetilde{k}_1 = \widetilde{k}_\ell = 1$ and $\widetilde{k}_2 = \cdots = \widetilde{k}_{\ell-1} = 0$, we count 3 strings.*

Example 5.5 () *For example, for $1 < i < j < l$ we get for all $a_1, a_2, b_1, b_2 > 0$ such that $a_1 + a_2 = k_i$ and $b_1 + b_2 = k_j$, the string $k_1 \dots k_{i-1} a_1 1 a_2 k_{i+1} \dots k_{j-1} b_1 1 b_2 k_{j+1} \dots k_l$. The number of such strings is $(\widetilde{k}_i)(\widetilde{k}_j)$.*

Another example: for the particular cases $i = j = 1$ we get for all $a_1, a_2, a_3 \in \mathbb{N}$ with a_2 strictly positive such that $a_1 + a_2 + a_3 = k_1$, the string $a_1 1 a_2 1 a_3 k_2 k_l$ or (case $a_2 = 0$) $a_1 2 a_3 k_2 k_l$. The number of such strings is $\binom{\widetilde{k}_1+1}{2}$.

Case 1/1 As in the previous case, we choose a block in which we apply a weight increasing insertion, yielding $k_i + 1$ masks; then we choose a block for a weight preserving insertion, yielding $\widetilde{k}_i$ strings. However, one must be careful: to see why, consider the following string $x = \texttt{000111} = [3, 3]$.

- If we insert a weight increasing $\underline{0}$ in the first block, and then a weight preserving $\texttt{1}$ in the last-but-one position of the first block, we get the string $y = \texttt{00010111} = [3, 1, 1, 3]$. This string is of weight $(3 + 1) + (3 + 1)$, since we can delete the $\underline{0}$ then one of the four 1, or the $\texttt{1}$ then one of the four 0.
- If we insert a weight increasing $\underline{1}$ in the second block, followed by a weight preserving $\texttt{0}$ in the second position of the second block, we obtain the same string $y = \texttt{00010111} = [0; 3, 1, 1, 3]$.

Hence, there are two ways to get each y. We will, therefore, exercise a preference toward the first situation, where we perform a weight increasing insertion in the first block, followed by a weight preserving insertion in the first block's last-but-one position. Let $i \in [\ell]$.

- If $i = 1$, we get $\sum_{j=1}^{\ell} \widetilde{k}_j (= m - \ell + 2)$ strings of weight $k_1 + 1$, as well as a string of weight $k_1 + k_2 + 2$. In total, we get $m - \ell + 3$ strings.
- If $1 < i < \ell$, we perform a weight increasing insertion in the block i, the number of strings we will get is $(\sum_i \widetilde{k}_i)$. Indeed the string $[k_1, \dots, k_{i-1}, 1, 1, k_i, k_{i+1}, \dots, k_\ell]$ will be counted for the case $i - 1$. Each of these strings has weight $k_i + 1$, except one ($[k_1, \dots, k_i, 1, 1, k_{i+1}, \dots, k_\ell]$) which has weight $k_i + 1 + k_{i+1} + 1$ (the string that we will not count for the next example).
- If $i = \ell$, we can keep the same formula by introducing $k_{\ell+1} = 0$ for the weight of the string $[k_1, \dots, k_\ell, 1, 1]$.

Remark (Sanity check for the number of strings) *As in the case of one deletion, we will count the number of strings considered to make sure that we do not miss anything. We have:*

- *case 2/0 :* $\frac{\ell(\ell+1)}{2}$
- *case 0/2 :* $\sum_{1 \le i < j \le \ell} \widetilde{k}_i \widetilde{k}_j + \sum_{i=1}^{\ell} \binom{\widetilde{k}_i+1}{2}$
- *case 1/1 :* $1 + \ell(m - \ell - 2)$

We give an algebraic proof in Section 5.3.7.2 showing that if there exist positive integers $(k_i)_{i \in \{1,\dots,\ell\}}$ such that $m = \sum_{i=1}^{\ell} k_i$, then we have $\frac{\ell(\ell+1)}{2} + \sum_{1 \le i < j \le \ell} \widetilde{k}_i \widetilde{k}_j + \sum_{i=1}^{\ell} \binom{\widetilde{k}_i+1}{2} + 1 + \ell(m - \ell - 2) = \binom{m+2}{m} + \binom{m+2}{m+1} + \binom{m+2}{m+2}$, to make sure we have not missed or double-counted any strings.

Minimal Entropy For Double Deletions As in Section 5.3.5.1, we analyze the effects of the merging operation g on entropy. For this, we consider the impact of $g(x) = [k_1 + k_2, k_3, \ldots, k_\ell]$ on the clustering results developed in Section 5.3.5.2. We will omit the analyses when no insertions are made in the first or second block since we will get the same weight and this will disappear in the difference.

Case $2/0$ For x, we had $\frac{\ell(\ell+1)}{2}$ strings of this type, we now have $\frac{\ell(\ell-1)}{2}$, there are ℓ less strings and $\ell - 1$ that grow bigger. The rest remains the same.

Case $0/2$ Similar to x, we have a certain number of strings with weight 1 counted as before

$$\sum_{3 \leq i \leq j \leq \ell} \widetilde{k_i}\widetilde{k_j} + \sum_{3 \leq i \leq \ell} \binom{\widetilde{k_i} + 1}{2}$$

However, a part of the formula changes:

$$\binom{k_1 + k_2 + 1}{2} + (k_1 + k_2) \sum_{3 \leq i \leq \ell} \widetilde{k_i} \tag{5.20}$$

Then, for the part of the analysis of $g(x)$ equivalent with that of x we get

$$\binom{k_1 + 1}{2} + \binom{k_2}{2} + (k_1 + k_2 - 1) \times \sum_{3 \leq i \leq \ell} \widetilde{k_i} + k_1(k_2 - 1) \tag{5.21}$$

now we take the difference between Equation (5.20) and Equation (5.21)

$$\sum_{3 \leq i \leq \ell} \widetilde{k_i} + \binom{k_1 + k_2 + 1}{2} - \left(\binom{k_1 + 1}{2} + \binom{k_2}{2} + k_1(k_2 - 1) \right)$$

After simplifications, we obtain $\sum_{1 \leq i \leq \ell} \widetilde{k_i} + 1$.

Case $1/1$ In the case of x, we had

$$(\ell - 1) \sum_{1 \leq i \leq \ell} (\widetilde{k_i} - 1) + \sum_{1 \leq i \leq \ell} \widetilde{k_i}.$$

We now have

$$(\ell - 2)\left(\sum_{1 \leq i \leq \ell} (\widetilde{k_i} - 1) + 1 \right) + \sum_{1 \leq i \leq \ell} \widetilde{k_i} + 1.$$

Taking the difference between now and before we get $\sum_{1 \leq i \leq \ell} \widetilde{k_i} + 1 - l$. We have $(\sum_{1 \leq i \leq l} (\widetilde{k_i} - 1) + 1)$ weights (the weight increasing insertion in the first block) that grow bigger, the rest stays the same.

Remark (Sanity check) *We can check that the numbers of string is constant:*

- *Case 0/2:* $\sum_{1 \leq i \leq \ell} \widetilde{k_i} + 1$ *more strings*
- *Case 1/1:* $(\sum_{1 \leq i \leq \ell} \widetilde{k_i} + 1 - \ell)$ *less strings*

- *Case 2/0: ℓ less strings.*

and $\sum_{1\leq i\leq \ell} \widetilde{k_i} + 1 - (\sum_{1\leq i\leq \ell} \widetilde{k_i} + 1 - \ell) - \ell = 0$.

We can now compute the difference of the two entropies. Note that instead of working with the probabilities, we will multiply everything by $4\binom{m+2}{m}$. We can focus on the very few strings that show a change in weight (when an insertion is made in the first or second block).
Case 2/0**:** For x, we have 1 string for each of the weights

$$\begin{aligned}&(k_1+1)(k_2+1), (k_1+1)(k_3+1), \ldots,\\ &(k_1+1)(k_l+1), (k_2+1)(k_3+1), (k_2+1)(k_4+1), \ldots,\\ &(k_2+1)(k_l+1), \binom{k_1+2}{2}\binom{k_2+2}{2}\end{aligned}$$

For $g(x)$, we still have 1 string for each of the following weights:

$$(k_1+k_2+1)(k_3+1), (k_1+k_2+1)(k_4+1), \ldots, (k_1+k_2+1)(k_l+1), \binom{k_1+k_2+2}{2}$$

Case 0/2**:** For $g(x)$, we have $\sum_{1\leq i\leq \ell} \widetilde{k_i} + 1$.
Case 1/1**:** For x, the remaining strings are:

Multiplicity	Weight
$\sum_{i=1}^{\ell} \widetilde{k_i}$	k_1+1
$\sum_{i=1}^{\ell} k_i - 1$	k_2+1
1	k_1+k_2+2
1	k_2+k_3+2

There remains, for $g(x)$, one string for each of the following weights $k_3+1, k_4+1, \ldots, k_l+1$ and $\sum_{1\leq i\leq \ell} \widetilde{k_i} + 1$ strings of weight k_1+k_2+1 along with 1 string of weight $k_1+k_2+k_3+2$. The difference of entropies is equal to the difference between A and B defined in the following

equations:

$$
\begin{aligned}
A = & \sum_{2\le i\le \ell} (k_1+1)(k_i+1)\log\frac{1}{(k_1+1)(k_i+1)} + \binom{k_1+2}{2}\log\frac{1}{\binom{k_1+2}{2}} + \binom{k_2+2}{2}\log\frac{1}{\binom{k_2+2}{2}} \\
& + \sum_{1\le i\le \ell} \widetilde{k_i}(k_1+1)\log\frac{1}{(k_1+1)} + (k_1+k_2+2)\log\frac{1}{(k_1+k_2+2)} \\
& + (\sum_{1\le i\le \ell} \widetilde{k_i} - 1)(k_2+1)\log\frac{1}{(k_2+1)} \\
& + (k_2+k_3+2)\log\frac{1}{(k_2+k_3+2)} \\
B = & \sum_{3\le i\le l} (k_i+1)\log\frac{1}{(k_i+1)} + (\sum_{1\le i\le l} \widetilde{k_i}+1)(k_1+k_2+1)\log\frac{1}{(k_1+k_2+1)} \\
& + (k_1+k_2+k_3+2)\log\frac{1}{(k_1+k_2+k_3+2)} \\
& + \sum_{3\le i\le l} (k_1+k_2+1)(k_i+1)\log\frac{1}{(k_1+k_2+1)(k_i+1)} \\
& + \binom{k_1+k_2+2}{2}\log\frac{1}{\binom{k_1+k_2+2}{2}}
\end{aligned}
$$

where A corresponds to x, and B corresponds to $g(x)$. We are now in a position to conclude the proof of Lemma 5.11.

Lemma 5.12 *The transformation g decreases the entropy $H_n(x)$ for double deletions, i.e., $m = n-2$.*

Proof: To prove this, it suffices to show that for $\ell \ge 2$, $A - B > 0$. The proof mostly consists of computing partial derivatives to show that the function is increasing. We refer the reader to Section 5.3.7.1 for details. □

Corollary 5.13 *For all n and any subsequence x of length $m = n-2$, we have*

$$H_n(x) \ge H_n([m]),$$

with equality only if $x \in \{0^m, 1^m\}$.

Proof: Given any $x \ne [m]$ of length $m = n-2$, it can be transformed into the string $[m]$ by a series of consecutive g operations (cf. Definition 5.6). Similar to the single-bit deletion case, each such operation can only decrease the entropy, as proved in Lemma 5.11, and thus we get a proof for the fact that $H_n(x) \ge H_n(0^m)$. □

5.3.6 Concluding Remarks

From the original cryptographic motivation of the problem, the minimal entropy case corresponding to maximal information leakage is arguably the case that interests us the most. While our results

shed more light on various properties of the space of supersequences and the combinatorial problem of counting the number of embeddings of a given subsequence in the set of its compatible supersequences, the original entropy maximization conjecture remains an open problem. Finally, proving the entropy minimization conjecture for an arbitrary number of deletions as well as a more general characterization of the distribution of the number of subsequence embeddings in supersequences of finite-length present some further open problems.

5.3.7 Appendices

5.3.7.1 Proof of Lemma 5.12

Proof: The proof consists of two steps: first we show that $A - B > 0$ when $k_1 = \cdots = k_\ell = 1$; then we show that $\nabla(A - B)$ is positive along all directions, so that an increase in any of the k_i, results in an increase of $A - B$.

- Step 1. For $k_1 = \cdots = k_\ell = 1$, we have:

$$\begin{aligned} A = & -\sum_{2\le i\le \ell} 4\log 4 - 3\log 3 - 3\log 3 \\ & -\sum_{1\le i\le \ell} 2\widetilde{k_i}\log 2 - 4\log 4 \\ & -2\log 2 \sum_{1\le i\le \ell} (\widetilde{k_i} - 1) \\ & -4\log 4 \\ = & -8(\ell-1) - 6\log 3 - 2\sum_{i=1}^{\ell}\widetilde{k_i} - 8 + 2\ell - 2\sum_{i=1}^{\ell}\widetilde{k_i} - 8 \\ = & -6\ell - 6\log 3 - 16. \end{aligned}$$

  Similarly,

$$\begin{aligned} B = & -\sum_{3\le i\le \ell} 2\log 2 - 3\log 3 \sum_{1\le i\le \ell} (\widetilde{k_i} + 1) \\ & -5\log 5 \\ & -\sum_{3\le i\le \ell} 6\log 6 \\ & -6\log 6 \\ = & -2(\ell-2) - 3\ell\log 3 - 3\log 3\sum_{i=1}^{\ell}\widetilde{k_i} - 5\log 5 - 6(\ell-1)\log 6 \\ = & -2\ell - 6\ell\log 6 - 3\ell\log 3 + 4 - 6\log 3 - 5\log 5 + 6\log 6 \\ = & -(2 + 6 + 6\log 3 + 3\log 3)\ell + 4 - 6\log 3 - 5\log 5 + 6 + 6\log 3 \\ = & -(8 + 9\log 3)\ell + 10 - 5\log 5 \end{aligned}$$

  Thus,

$$\begin{aligned} A - B = & -6\ell - 6\log 3 - 16 - (-(8 + 9\log 3)\ell + 10 - 5\log 5) \\ = & (2 + 9\log(3))\ell - 26 + 5\log(5) - 6\log(3) \end{aligned}$$

Therefore, $A - B > 0$ iif $\ell > (26 + 6\log(3) - 5\log(5))/(2 + 9\log(3)) \approx 1.46$. For $\ell \geq 2$, and $k_1 = \cdots = k_\ell = 1$, we therefore have $A - B > 0$.

- STEP 2. To simplify computations, we introduce the function $e(x) = -x \log_2 x$. We also use the fact that $e(xy) = xe(y) + ye(x)$, and develop the binomial coefficients: $e\left(\binom{a+b}{2}\right) = \binom{a+b}{2} + e((a+b)(a+b-1))$. Then we match the sum indexes. We also introduce the notation $e_i = e(k_i + 1)$.

  Thus we can write:

$$
\begin{aligned}
A = & \sum_{2 \leq i \leq l} e((k_1+1)(k_i+1)) + e\left(\binom{k_1+2}{2}\right) + e\left(\binom{k_2+2}{2}\right) \\
& + e(k_1+1) \sum_{1 \leq i \leq l} \widetilde{k_i} + e(k_1+k_2+2) \\
& + e(k_2+1) \sum_{1 \leq i \leq l} (\widetilde{k_i} - 1) \\
& + e(k_2+k_3+2) \\
= & \sum_{3 \leq i \leq \ell-1} e((k_1+1)(k_i+1)) + e((k_1+1)(k_2+1)) + e((k_1+1)(k_\ell+1)) \\
& + \binom{k_1+1}{2} + e((k_1+1)(k_1+2)) \\
& + \binom{k_2+1}{2} + e((k_2+1)(k_2+2)) \\
& + e(k_1+1) \sum_{3 \leq i \leq \ell-1} \widetilde{k_i} + e(k_1+k_2+2) + e(k_1+1)\widetilde{k_1} + e(k_1+1)\widetilde{k_2} + e(k_1+1)\widetilde{k_\ell} \\
& + e(k_2+1) \sum_{3 \leq i \leq \ell-1} \widetilde{k_i} - \ell e(k_2+1) + e(k_2+1)\widetilde{k_1} + e(k_2+1)\widetilde{k_2} + e(k_2+1)\widetilde{k_\ell} \\
& + e(k_2+k_3+2) \\
= & (k_1+1) \sum_{3 \leq i \leq \ell-1} e_i + e_1 \sum_{3 \leq i \leq \ell-1} (k_i+1) \\
& + (k_2+1)e_1 + (k_1+1)e_2 + (k_1+1)e_\ell + (k_\ell+1)e_1 \\
& + \binom{k_1+1}{2} + (k_1+2)e_1 + (k_1+1)e(k_1+2) \\
& + \binom{k_2+1}{2} + (k_2+2)e_2 + (k_2+1)e(k_2+2) \\
& + e_1 \sum_{3 \leq i \leq \ell-1} k_i - (\ell-3)e_1 + e(k_1+k_2+2) + e_1k_1 + e_1k_2 - e_1 + e_1k_\ell \\
& + e_2 \sum_{3 \leq i \leq \ell-1} k_i - (\ell-3)e_2 - \ell e_2 + e_2k_1 + e_2k_2 - e_2 + e_2k_\ell \\
& + e(k_2+k_3+2)
\end{aligned}
$$

At this point we regroup all terms in e_i together:

$$\begin{aligned}
A = &\left(2k_1 + 2k_2 + 2k_\ell - 3 + 2\sum_{3\le i\le \ell-1} k_i\right) e_1 \\
&+ \left(2k_1 + 2k_2 + k_\ell - 2\ell - 1 + \sum_{3\le i\le \ell-1} k_i\right) e_2 \\
&+ (k_1+1)\sum_{3\le i\le \ell-1} e_i \\
&+ (k_1+1)e_\ell \\
&+ e(k_1+k_2+2) + (k_1+1)e(k_1+2) + (k_2+1)e(k_2+2) + e(k_2+k_3+2) \\
&+ \binom{k_1+1}{2} + \binom{k_2+1}{2}
\end{aligned}$$

We simplify the expression for B in the same fashion:

$$\begin{aligned}
B = &\, 2e(k_1+k_2+1)\sum_{3\le i\le \ell-1} k_i + k_1 e(k_1+k_2+1) + k_2 e(k_1+k_2+1) + k_\ell e(k_1+k_2+1) \\
&+ e(k_1+k_2+k_3+2) \\
&+ (\ell-3)e(k_1+k_2+1) + (k_1+k_2+2)\sum_{3\le i\le \ell} e_i \\
&+ \binom{k_1+k_2+2}{2} + e((k_1+k_2+1)(k_1+k_2+2)) \\
= &\, (k_1+k_2+2)\sum_{3\le i\le \ell-1} e_i \\
&+ \left(2k_1 + 2k_2 + k_\ell + \ell - 1 + 2\sum_{3\le i\le \ell-1} k_i\right) e(k_1+k_2+1) \\
&+ e(k_1+k_2+k_3+2) + (k_1+k_2+1)e(k_1+k_2+2) + (k_1+k_2+2)e_\ell \\
&+ \binom{k_1+k_2+2}{2}
\end{aligned}$$

so that we can now compute the difference:

$$
\begin{aligned}
A - B = & \left(-3 + 2 \sum_{1 \le i \le \ell} k_i\right) e_1 \\
& + \left(k_1 + k_2 - 2\ell - 1 + \sum_{1 \le i \le \ell} k_i\right) e_2 \\
& - (k_2 + 1) \sum_{3 \le i \le \ell} e_i \\
& + (k_1 + 1)e(k_1 + 2) + (k_2 + 1)e(k_2 + 2) \\
& + 1 - k_1 k_2 \\
& - \left(-k_\ell + \ell - 1 + 2 \sum_{1 \le i \le \ell} k_i\right) e(k_1 + k_2 + 1) \\
& - (k_1 + k_2)e(k_1 + k_2 + 2) - e(k_1 + k_2 + k_3 + 2) + e(k_2 + k_3 + 2) \\
= & P(\mathbf{k})e_1 + Q(\mathbf{k})e_2 - (k_2 + 1) \sum_{i=3}^{\ell} e_i + (k_1 + 1)e(k_1 + 2) + (k_2 + 1)e(k_2 + 2) \\
& + 1 - k_1 k_2 - R(\mathbf{k})e(k_1 + k_2 + 1) - (k_1 + k_2)e(k_1 + k_2 + 2) - e(k_1 + k_2 + k_3 + 2) \\
& + e(k_2 + k_3 + 2).
\end{aligned}
$$

Where

$$
\begin{aligned}
P(\mathbf{k}) &= -3 + 2 \sum_{1 \le i \le \ell} k_i, \\
Q(\mathbf{k}) &= k_1 + k_2 - 2\ell - 1 + \sum_{1 \le i \le \ell} k_i \\
R(\mathbf{k}) &= -k_\ell + \ell - 1 + 2 \sum_{1 \le i \le \ell} k_i.
\end{aligned}
$$

The gradient can be computed term by term thanks to linearity, observing that for any polynomial $S(\mathbf{k})$,

$$
\begin{aligned}
\partial_i e_i &= -\log_2(k_1 + 1) - \frac{1}{\ln(2)} \\
\partial_i e_j &= 0 \qquad (i \neq j) \\
\nabla S(\mathbf{k}) e_j &= (e_j \partial_i S(\mathbf{k}) + S(\mathbf{k}) \partial_i e_j)_{i=1}^{\ell}
\end{aligned}
$$

Hence, by denoting $\mathbf{u}_1, \dots, \mathbf{u}_\ell$ the canonical basis, we have:

$$\begin{aligned}
\nabla P(\mathbf{k})e_1 &= (e_1\partial_i P(\mathbf{k}) + P(\mathbf{k})\partial_i e_1)_{i=1}^{\ell} = \partial_1 e_1 P(\mathbf{k})\mathbf{u}_1 + e_1(\partial_i P(\mathbf{k}))_{i=1}^{\ell} \\
&= \partial_1 e_1 P(\mathbf{k})\mathbf{u}_1 + 2(\mathbf{u}_1 + \cdots + \mathbf{u}_\ell) \\
&= (2 + \partial_1 e_1 P(\mathbf{k}))\mathbf{u}_1 + 2\mathbf{u}_2 + \cdots + 2\mathbf{u}_\ell \\
\nabla Q(\mathbf{k})e_2 &= (e_2\partial_i Q(\mathbf{k}) + Q(\mathbf{k})\partial_i e_2)_{i=1}^{\ell} = \partial_2 e_2 Q(\mathbf{k})\mathbf{u}_2 + (\partial_i S(\mathbf{k}))_{i=1}^{\ell} \\
&= \partial_2 e_2 Q(\mathbf{k})\mathbf{u}_2 + \mathbf{u}_1 + \mathbf{u}_2 + \mathbf{u}_1 + \cdots + \mathbf{u}_\ell \\
&= 2\mathbf{u}_1 + (2 + \partial_2 e_2 Q(\mathbf{k}))\mathbf{u}_2 + \mathbf{u}_3 + \cdots + \mathbf{u}_\ell \\
-\nabla\left((k_2+1)\sum_{i=3}^{\ell} e_i\right) &= -(k_2+1)\nabla\sum_{i=3}^{\ell} e_i - (\nabla(k_2+1))\sum_{i=3}^{\ell} e_i \\
&= -((k_2+1)\partial_i e_i \mathbf{u}_i)_{i=3}^{\ell} - \left(\sum_{i=3}^{\ell} e_i\right)\mathbf{u}_2 \\
\nabla\left((k_j+1)e(k_j+2)\right) &= -\left(\log_2(k_j+2) + \frac{1}{\ln(2)}\frac{k_j+1}{k_j+2}\right)\mathbf{u}_j \\
\nabla(1-k_1k_2) &= -k_2\mathbf{u}_1 - k_1\mathbf{u}_2 \\
-\nabla\left(R(\mathbf{k})e(k_1+k_2+1)\right) &= -R(\mathbf{k})\nabla e(k_1+k_2+1) - e(k_1+k_2+1)\nabla R(\mathbf{k}) \\
&= R(\mathbf{k})\left(\log_2(k_1+k_2+1) + \frac{1}{\ln(2)}\right)(\mathbf{u}_1+\mathbf{u}_2) \\
&\quad - e(k_1+k_2+1)(\partial_i R(\mathbf{k}))_{i=1}^{\ell} \\
&= R(\mathbf{k})\left(\log_2(k_1+k_2+1) + \frac{1}{\ln(2)}\right)(\mathbf{u}_1+\mathbf{u}_2) \\
&\quad - e(k_1+k_2+1)(2\mathbf{u}_1 + \cdots + 2\mathbf{u}_{\ell-1} + \mathbf{u}_\ell) \\
-\nabla(k_1+k_2)e(k_1+k_2+2) &= -(k_1+k_2)\nabla e(k_1+k_2+2) - e(k_1+k_2+2)\nabla(k_1+k_2) \\
&= (k_1+k_2)\left(\log_2(k_1+k_2+2) + \frac{1}{\ln(2)}\right)(\mathbf{u}_1+\mathbf{u}_2) \\
&\quad - e(k_1+k_2+2)(\mathbf{u}_1+\mathbf{u}_2) \\
&= \left((k_1+k_2)\left(\log_2(k_1+k_2+2) + \frac{1}{\ln(2)}\right) - e(k_1+k_2+2)\right)(\mathbf{u}_1+\mathbf{u}_2) \\
-\nabla e(k_1+k_2+k_3+2) &= \left(\log_2(k_1+k_2+k_3+2) + \frac{1}{\ln(2)}\right)(\mathbf{u}_1+\mathbf{u}_2+\mathbf{u}_3) \\
\nabla e(k_2+k_3+2) &= -\left(\log_2(k_2+k_3+2) + \frac{1}{\ln(2)}\right)(\mathbf{u}_2+\mathbf{u}_3)
\end{aligned}$$

As it is clearly visible from the above equations, we only need to consider the components along $\mathbf{u}_1$, $\mathbf{u}_2$, $\mathbf{u}_3$, $\mathbf{u}_\ell$, and along $\mathbf{u}_i$ for any $3 < i < \ell$. For the latter, we have

$$\begin{aligned}
(\nabla(A-B))_i &= 2 + 1 - (k_2+1)\partial_i e_i - 2e(k_1+k_2+1) \\
&= 3 + 2(k_1+k_2+1)\log_2(k_1+k_2+1) + (k_2+1)\left(\log_2(k_i+1) + \frac{1}{\ln(2)}\right) \\
&> 0.
\end{aligned}$$

Now, along the very similar $\mathbf{u}_\ell$ axis,

$$\begin{aligned}(\nabla(A-B))_\ell &= 2+1-(k_2+1)\partial_\ell e_\ell - e(k_1+k_2+1)\\ &= 3+(k_1+k_2+1)\log_2(k_1+k_2+1)+(k_2+1)\left(\log_2(k_\ell+1)+\frac{1}{\ln(2)}\right)\\ &> 0.\end{aligned}$$

Along $\mathbf{u}_3$,

$$\begin{aligned}(\nabla(A-B))_3 &= 2+1-(k_2+1)\partial_3 e_3 - 2e(k_1+k_2+1)+\log_2(k_1+k_2+k_3+2)+\frac{1}{\ln(2)}\\ &\quad -\log_2(k_2+k_3+2)+\frac{1}{\ln(2)}\\ &= 3+2(k_1+k_2+1)\log_2(k_1+k_2+1)+(k_2+1)\left(\log_2(k_3+1)+\frac{1}{\ln(2)}\right)\\ &\quad +\log_2(k_1+k_2+k_3+2)-\log_2(k_2+k_3+2)\\ &> 0.\end{aligned}$$

Along $\mathbf{u}_2$,

$$\begin{aligned}(\nabla(A-B))_2 &= 2+2+Q(\mathbf{k})\partial_2 e_2-(k_2+1)\partial_2 e_2\\ &\quad -\sum_{i=3}^{\ell} e_i-\left(\log_2(k_2+2)+\frac{1}{\ln(2)}\frac{k_2+1}{k_2+2}\right)-k_1\\ &\quad +R(\mathbf{k})\left(\log_2(k_1+k_2+1)+\frac{1}{\ln(2)}\right)-2e(k_1+k_2+1)\\ &\quad +\left((k_1+k_2)\left(\log_2(k_1+k_2+2)+\frac{1}{\ln(2)}\right)-e(k_1+k_2+2)\right)\\ &\quad +\log_2(k_1+k_2+k_3+2)+\frac{1}{\ln(2)}-\log_2(k_2+k_3+2)-\frac{1}{\ln(2)}\\ &= 4-(Q(\mathbf{k})-k_2-1)\left(\log_2(k_2+1)+\frac{1}{\ln(2)}\right)-\frac{1}{\ln(2)}\frac{k_2+1}{k_2+2}-k_1\\ &\quad +\sum_{i=3}^{\ell}(k_i+1)\log_2(k_i+1)\\ &\quad +R(\mathbf{k})\left(\log_2(k_1+k_2+1)+\frac{1}{\ln(2)}\right)-2e(k_1+k_2+1)\\ &\quad +(k_1+k_2)\left(\log_2(k_1+k_2+2)+\frac{1}{\ln(2)}\right)-e(k_1+k_2+2)\\ &\quad +\log_2(k_1+k_2+k_3+2)-\log_2(k_2+k_3+2)-\log_2(k_2+2)\end{aligned}$$

Finally, along $\mathbf{u}_1$,

$$\begin{aligned}
(\nabla(A-B))_1 &= 2+\partial_1 e_1 P(\mathbf{k})+2-(k_2+1)\partial_1 e_1-\left(\log_2(k_1+2)+\frac{1}{\ln(2)}\frac{k_1+1}{k_1+2}\right)-k_2\\
&\quad+R(\mathbf{k})\left(\log_2(k_1+k_2+1)+\frac{1}{\ln(2)}\right)-2e(k_1+k_2+2)\\
&\quad+(k_1+k_2)\left(\log_2(k_1+k_2+2)+\frac{1}{\ln(2)}\right)-e(k_1+k_2+2)\\
&\quad+\log_2(k_1+k_2+k_3+2)+\frac{1}{\ln(2)}\\
&=4-(P(\mathbf{k})-k_2-1)\left(\log_2(k_1+1)+\frac{1}{\ln(2)}\right)-k_2\\
&\quad+R(\mathbf{k})\left(\log_2(k_1+k_2+1)+\frac{1}{\ln(2)}\right)-2e(k_1+k_2+2)\\
&\quad+(k_1+k_2)\left(\log_2(k_1+k_2+2)+\frac{1}{\ln(2)}\right)-e(k_1+k_2+2)\\
&\quad+\log_2(k_1+k_2+k_3+2)-\log_2(k_1+2)+\frac{1}{\ln(2)}\left(1-\frac{k_1+1}{k_1+2}\right)
\end{aligned}$$

Lemma 5.14 $(\nabla(A-B))_1>0$.

Proof: [Proof of Lemma 5.14] It suffices to check that $(k_1+k_2)(\lambda+\log_2(k_1+k_2+1))-k_2>0$, that $\log_2(k_1+k_2+k_3+2)-\log_2(k_1+2)>0$, and that all the remaining quantities are positive. □

Lemma 5.15 (Sublemma) $(\nabla(A-B))_2>0$.

Proof: [Proof of Lemma 5.15]

$$\begin{aligned}
&-(Q(\mathbf{k})-k_2-1)(\log_2(k_2+1)+\lambda)-\lambda\frac{k_2+1}{k_2+2}-k_1+\sum_{i=3}^{\ell}(k_i+1)\log_2(k_i+1)\\
&+R(\mathbf{k})(\lambda+\log_2(k_1+k_2+1))\\
&=\lambda\left(R(\mathbf{k})-Q(\mathbf{k})+k_2+1-\frac{k_2+1}{k_2+2}\right)+R(\mathbf{k})\log_2(k_1+k_2+1)\\
&+\sum_{i=3}^{\ell}(k_i+1)\log_2(k_i+1)-Q(\mathbf{k})\log_2(k_2+1)\\
&=\lambda\left(\sum_{i=2}^{\ell-1}k_i+3\ell+1-\frac{k_2+1}{k_2+2}\right)+\sum_{i=3}^{\ell}(k_i+1)\log_2(k_i+1)\\
&+R(\mathbf{k})\log_2(k_1+k_2+1)-Q(\mathbf{k})\log_2(k_2+1).
\end{aligned}$$

The last line is positive since in particular $R(\mathbf{k})\log_2(k_1+k_2+1) - Q(\mathbf{k})\log_2(k_2+1) > (R(\mathbf{k}) - Q((k))\log_2(k_2+1)$. □

As a result, we have that $A - B > 0$ for all $\mathbf{k}$ such that $k_i \geq 1$, which establishes the theorem. □

5.3.7.2 Proof of Remark 24

We prove that for all positive integer sequences $(k_i)_{i\in\{1,\dots,\ell\}}$ such that $\sum_{i=1}^{\ell} = m$ we have :

$$\frac{\ell(\ell+1)}{2} + \sum_{1\leq i<j\leq \ell} \widetilde{k_i}\widetilde{k_j} + \sum_{i=1}^{\ell} \binom{\widetilde{k_i}+1}{2} + 1 + \ell(m-\ell-2) = \binom{m+2}{m} + \binom{m+2}{m+1} + \binom{m+2}{m+2}$$

We fix ℓ and m, then proceed by induction on the sequences of $(k_i)_{i\in\{1,\dots,\ell\}}$. We fist show the equality for $k_1 = m-\ell+1$, and $k_i = 1$ for all $i > 1$.

Proof: We have

$$\begin{aligned}
&\frac{\ell(\ell+1)}{2} + \sum_{1\leq i<j\leq \ell} \widetilde{k_i}\widetilde{k_j} + \sum_{i=1}^{\ell} \binom{\widetilde{k_i}+1}{2} + 1 + \ell(m-\ell-2) \\
&= \frac{\ell(\ell+1)}{2} + 1 + \ell(m-\ell-2) + (m-\ell+1)\sum_{j=2}^{\ell} \widetilde{k_j} + \binom{2}{2} + \binom{m-\ell+2}{2} \\
&= \frac{1}{2}\left(\ell(\ell+1) + (m-\ell+2)(m-\ell+1)\right) + (\ell+1)(m-\ell+2) + 1 \\
&= \frac{1}{2}(m^2+3m+2) + m + 3 \\
&= \binom{m+2}{m} + \binom{m+2}{m+1} + \binom{m+2}{m+2}
\end{aligned}$$

which concludes the initialization. □

We now fix a sequence $(k_i)_{i\in\{1,\dots,\ell\}}$, and $i_0 \in \{1,\dots,\ell\}$. We assume that the equality holds for this sequence and show that it is true for the sequence $(k'_i)_{i\in\{1,\dots,\ell\}}$ defined as $k'_i = k_i$ if $i \neq i_0$ and $i \neq i_0+1$, $k'_{i_0} = k_{i_0} - 1$ and $k'_{i_0+1} = k_{i_0+1} + 1$.

Proof: We first note that only a part of the formula on the left hand side depends on $(k_i)_{i\in\{1,\dots,\ell\}}$. Letting $F\left((k_i)_{i\in\{1,\dots,\ell\}}\right) = \sum_{1\leq i<j\leq \ell} \widetilde{k_i}\widetilde{k_j} + \sum_{i=1}^{\ell} \binom{\widetilde{k_i}+1}{2}$, we just have to prove that

$$F\left((k_i)_{i\in\{1,\dots,\ell\}}\right) - F\left((k'_i)_{i\in\{1,\dots,\ell\}}\right) = 0.$$

Expanding the above difference, we have:

$$\begin{aligned}
&(\widetilde{k_{i_0}} - \widetilde{k'_{i_0}})(\sum_{i=1}^{i_0-1} \widetilde{k_i}) + (\widetilde{k_{i_0+1}} - \widetilde{k'_{i_0+1}})(\sum_{i=1}^{i_0-1} \widetilde{k_i}) + \widetilde{k_{i_0}}\widetilde{k_{i_0+1}} - \widetilde{k'_{i_0}}\widetilde{k'_{i_0+1}} \\
&\quad + (\widetilde{k_{i_0}} - \widetilde{k'_{i_0}})(\sum_{i=i_0+1}^{\ell} \widetilde{k_i}) + (\widetilde{k_{i_0+1}} - \widetilde{k'_{i_0+1}})(\sum_{i=i_0+2}^{\ell} \widetilde{k_i}) \\
&\quad + \binom{\widetilde{k_{i_0}}+1}{2} - \binom{\widetilde{k'_{i_0}}+1}{2} + \binom{\widetilde{k_{i_0+1}}+1}{2} - \binom{\widetilde{k_{i_0+1}}+1}{2}
\end{aligned}$$

This is equal to

$$\begin{aligned}
&\widetilde{k_{i_0}}\widetilde{k_{i_0+1}} - (\widetilde{k_{i_0}} - 1)(\widetilde{k_{i_0+1}} + 1) + \widetilde{k_{i_0+1}} + \frac{1}{2}\Big(\widetilde{k_{i_0}}(\widetilde{k_{i_0}} + 1) - (\widetilde{k_{i_0}} + 1)\widetilde{k_{i_0}}\Big) \\
&\qquad - \frac{1}{2}\Big(\widetilde{k_{i_0+1}}(\widetilde{k_{i_0+1}} + 1) - (\widetilde{k_{i_0+1}} + 2)\widetilde{k_{i_0+1}} + 1\Big) \\
&= -\widetilde{k_{i_0}} + \widetilde{k_{i_0+1}} + 1 + \frac{1}{2}(2\widetilde{k_{i_0}} - 2\widetilde{k_{i_0+1}} - 2) \\
&= 0.
\end{aligned}$$

This concludes the proof. □

Bibliography

[ABF+17] Yasemin Acar, Michael Backes, Sascha Fahl, Simson L. Garfinkel, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. Comparing the usability of cryptographic apis. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 154–171, 2017.

[ABGN16] Antoine Amarilli, Marc Beunardeau, Rémi Géraud, and David Naccache. Failure is also an option. In Peter Y. A. Ryan, David Naccache, and Jean-Jacques Quisquater, editors, *The New Codebreakers - Essays Dedicated to David Kahn on the Occasion of His 85th Birthday*, volume 9100 of *Lecture Notes in Computer Science*, pages 161–165. Springer, 2016.

[ADK+14] Martin R. Albrecht, Benedikt Driessen, Elif Bilge Kavun, Gregor Leander, Christof Paar, and Tolga Yalçın. *Block Ciphers – Focus on the Linear Layer (feat. PRIDE)*, pages 57–76. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[ADN+10] Michel Agoyan, Jean-Max Dutertre, David Naccache, Bruno Robisson, and Assia Tria. When clocks fail: On critical paths and clock faults. In Dieter Gollmann, Jean-Louis Lanet, and Julien Iguchi-Cartigny, editors, *CARDIS 2010*, volume 6035 of *Lecture Notes in Computer Science*, pages 182–193, Passau, Germany, April 14-16, 2010. Springer, Heidelberg, Germany.

[ADPS15] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - a new hope. Cryptology ePrint Archive, Report 2015/1092, 2015. https://eprint.iacr.org/2015/1092.

[AeKKS07] Onur Acıiçmez, Çetin Kaya Koç, and Jean-Pierre Seifert. On the power of simple branch prediction analysis. In *2007 ACM SYMPOSIUM ON INFORMATION, COMPUTER AND COMMUNICATIONS SECURITY (ASIACCS'07*, pages 312–320. ACM Press, 2007.

[AFG+14] Diego F. Aranha, Pierre-Alain Fouque, Benoît Gérard, Jean-Gabriel Kammerer, Mehdi Tibouchi, and Jean-Christophe Zapalowicz. GLV/GLS decomposition, power analysis, and attacks on ECDSA signatures with single-bit nonce bias. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 262–281, Kaoshiung, Taiwan, R.O.C., December 7–11, 2014. Springer, Heidelberg, Germany.

[AJPS17a] Divesh Aggarwal, Antoine Joux, Anupam Prakash, and Miklos Santha. A new public-key cryptosystem via mersenne numbers. Cryptology ePrint Archive, Report 2017/481, 2017. https://eprint.iacr.org/2017/481.

[AJPS17b] Divesh Aggarwal, Antoine Joux, Anupam Prakash, and Miklos Santha. A new public-key cryptosystem via Mersenne numbers. Cryptology ePrint Archive, Report 2017/481, 2017. http://eprint.iacr.org/2017/481.

[AJPS17c] Divesh Aggarwal, Antoine Joux, Anupam Prakash, and Miklos Santha. A new public-key cryptosystem via Mersenne numbers, version 20170530:001542. Cryptology ePrint Archive, Report 2017/481, 2017. https://eprint.iacr.org/2017/481.

[AJPS17d] Divesh Aggarwal, Antoine Joux, Anupam Prakash, and Miklos Santha. A new public-key cryptosystem via Mersenne numbers, version 20171206:004924. Cryptology ePrint Archive, Report 2017/481, 2017. https://eprint.iacr.org/2017/481.

[AKS07] Onur Aciiçmez, Çetin Kaya Koç, and Jean-Pierre Seifert. Predicting secret keys via branch prediction. In Masayuki Abe, editor, *Topics in Cryptology – CT-RSA 2007*, volume 4377 of *Lecture Notes in Computer Science*, pages 225–242, San Francisco, CA, USA, February 5–9, 2007. Springer, Heidelberg, Germany.

[AMOV91] Gordon B. Agnew, Ronald C. Mullin, I. M. Onyszchuk, and Scott A. Vanstone. An implementation for a fast public-key cryptosystem. *Journal of Cryptology*, 3(2):63–79, 1991.

[ARR15] Arash Atashpendar, AW Roscoe, and Peter YA Ryan. Information leakage due to revealing randomly selected bits. In *Security Protocols XXIII*, pages 325–341. Springer, 2015.

[BBBK16] Benoît Barbot, Nicolas Basset, Marc Beunardeau, and Marta Kwiatkowska. Uniform sampling for timed automata with application to language inclusion measurement. In Gul Agha and Benny Van Houdt, editors, *Quantitative Evaluation of Systems - 13th International Conference, QEST 2016, Quebec City, QC, Canada, August 23-25, 2016, Proceedings*, volume 9826 of *Lecture Notes in Computer Science*, pages 175–190. Springer, 2016.

[BCD+16] Joppe Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! practical, quantum-secure key exchange from lwe. Cryptology ePrint Archive, Report 2016/659, 2016. https://eprint.iacr.org/2016/659.

[BCF+17] Marc Beunardeau, Aisling Connolly, Houda Ferradi, Rémi Géraud, David Naccache, and Damien Vergnaud. Reusing nonces in schnorr signatures - (and keeping it secure...). In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part I*, volume 10492 of *Lecture Notes in Computer Science*, pages 224–241. Springer, 2017.

[BCG+12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knežević, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract.

In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225, Beijing, China, December 2–6, 2012. Springer, Heidelberg, Germany.

[BCGN16a] Marc Beunardeau, Aisling Connolly, Rémi Géraud, and David Naccache. Cdoe obofsucaitn: Securing software from within. *IEEE Security & Privacy*, 14(3):78–81, 2016.

[BCGN16b] Marc Beunardeau, Aisling Connolly, Rémi Géraud, and David Naccache. Fully homomorphic encryption: Computations with a blindfold. *IEEE Security & Privacy*, 14(1):63–67, 2016.

[BCGN16c] Marc Beunardeau, Aisling Connolly, Rémi Géraud, and David Naccache. White-box cryptography: Security in an insecure environment. *IEEE Security & Privacy*, 14(5):88–92, 2016.

[BCGN17a] Marc Beunardeau, Aisling Connolly, Rémi Géraud, and David Naccache. The case for system command encryption. In Ramesh Karri, Ozgur Sinanoglu, Ahmad-Reza Sadeghi, and Xun Yi, editors, *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2017, Abu Dhabi, United Arab Emirates, April 2-6, 2017*, page 6. ACM, 2017.

[BCGN17b] Marc Beunardeau, Aisling Connolly, Rémi Géraud, and David Naccache. On the hardness of the mersenne low hamming ratio assumption. In *Fifth International Conference on Cryptology and Information Security in Latin America, Latincrypt 2017, La Habana, Cuba. September 20–22*, 2017. To appear.

[BCGN17c] Marc Beunardeau, Aisling Connolly, Rémi Géraud, and David Naccache. On the hardness of the Mersenne low Hamming ratio assumption. Cryptology ePrint Archive, Report 2017/522, 2017. https://eprint.iacr.org/2017/522.

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '96, pages 1–15, London, UK, UK, 1996. Springer-Verlag.

[BCLvV16] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. NTRU prime. *IACR Cryptology ePrint Archive*, 2016:461, 2016.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29, Cambridge, Massachusetts, USA, August 11–13, 2004. Springer, Heidelberg, Germany.

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51, Konstanz, Germany, May 11–15, 1997. Springer, Heidelberg, Germany.

[BDPA09] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak specifications, 2009.

[BFGN16] Marc Beunardeau, Houda Ferradi, Rémi Géraud, and David Naccache. Honey encryption for language - robbing shannon to pay turing? In Raphael C.-W. Phan and Moti Yung, editors, *Paradigms in Cryptology - Mycrypt 2016. Malicious and Exploratory Cryptology - Second International Conference, Mycrypt 2016, Kuala Lumpur, Malaysia, December 1-2, 2016, Revised Selected Papers*, volume 10311 of *Lecture Notes in Computer Science*, pages 127–144. Springer, 2016.

[BGMW93] Ernest F. Brickell, Daniel M. Gordon, Kevin S. McCurley, and David Bruce Wilson. Fast exponentiation with precomputation (extended abstract). In Rainer A. Rueppel, editor, *Advances in Cryptology – EUROCRYPT'92*, volume 658 of *Lecture Notes in Computer Science*, pages 200–207, Balatonfüred, Hungary, May 24–28, 1993. Springer, Heidelberg, Germany.

[BGR98] Mihir Bellare, Juan A. Garay, and Tal Rabin. Fast batch verification for modular exponentiation and digital signatures. In Kaisa Nyberg, editor, *Advances in Cryptology – EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 236–250, Espoo, Finland, May 31 – June 4, 1998. Springer, Heidelberg, Germany.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466, Vienna, Austria, September 10–13, 2007. Springer, Heidelberg, Germany.

[BN06] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 06: 13th Conference on Computer and Communications Security*, pages 390–399, Alexandria, Virginia, USA, October 30 – November 3, 2006. ACM Press.

[BP00] Colin Boyd and Chris Pavlovski. Attacking and repairing batch verification schemes. In Tatsuaki Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 58–71, Kyoto, Japan, December 3–7, 2000. Springer, Heidelberg, Germany.

[BPV98] Victor Boyko, Marcus Peinado, and Ramarathnam Venkatesan. Speeding up discrete log and factoring based schemes via precomputations. In Kaisa Nyberg, editor, *Advances in Cryptology – EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 221–235, Espoo, Finland, May 31 – June 4, 1998. Springer, Heidelberg, Germany.

[BRC60] Raj Chandra Bose and Dwijendra K. Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and control*, 3(1):68–79, 1960.

[BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In Burton S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525, Santa Barbara, CA, USA, August 17–21, 1997. Springer, Heidelberg, Germany.

[BS03] Johannes Blömer and Jean-Pierre Seifert. Fault based cryptanalysis of the advanced encryption standard (AES). In Rebecca Wright, editor, *FC 2003: 7th International Conference on Financial Cryptography*, volume 2742 of *Lecture Notes in Computer Science*, pages 162–181, Guadeloupe, French West Indies, January 27–30, 2003. Springer, Heidelberg, Germany.

[BS15] Adnan Baysal and Sühap Sahin. Roadrunner: A small and fast bitslice block cipher for low cost 8-bit processors. In Tim Güneysu, Gregor Leander, and Amir Moradi, editors, *LightSec 2015*, volume 9065 of *Lecture Notes in Computer Science*, pages 58–76, Bochum, Germany, September 10-11, 2015. Springer, Heidelberg, Germany.

[BSS+15] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. SIMON and SPECK: Block ciphers for the internet of things. Cryptology ePrint Archive, Report 2015/585, 2015. http://eprint.iacr.org/2015/585.

[Buh98] Joe Buhler, editor. *Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings*, volume 1423 of *Lecture Notes in Computer Science*. Springer, 1998.

[BZB+05] Guido Bertoni, Vittorio Zaccaria, Luca Breveglieri, Matteo Monchiero, and Gianluca Palermo. AES power attack based on induced cache miss and countermeasure. In *ITCC 2005, Volume 1*, pages 586–591, Las Vegas, Nevada, USA, April 4-5, 2005. IEEE Computer Society.

[Can00] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. https://eprint.iacr.org/2000/067.

[CCF+16] Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 313–333. Springer, 2016.

[CEJvO02] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. White-box cryptography and an AES implementation. In Kaisa Nyberg and Howard M. Heys, editors, *Selected Areas in Cryptography, 9th Annual International Workshop, SAC 2002, St. John's, Newfoundland, Canada, August 15-16, 2002. Revised Papers*, volume 2595 of *Lecture Notes in Computer Science*, pages 250–270. Springer, 2002.

[Cha76] Phillip J Chase. Subsequence numbers and logarithmic concavity. *Discrete Mathematics*, 16(2):123–140, 1976.

[CK08] Jung Hee Cheon and HongTae Kim. Analysis of low hamming weight products. *Discrete Applied Mathematics*, 156(12):2264–2269, 2008.

[CK14] Daniel Cullina and Negar Kiyavash. An improvement to levenshtein's upper bound on the cardinality of deletion correcting codes. *IEEE Transactions on Information Theory*, 60(7):3862–3870, 2014.

[CKK12] Daniel Cullina, Ankur A. Kulkarni, and Negar Kiyavash. A coloring approach to constructing deletion correcting codes from constant weight subgraphs. In *Proceedings of the 2012 IEEE International Symposium on Information Theory, ISIT 2012, Cambridge, MA, USA, July 1-6, 2012*, pages 513–517. IEEE, 2012.

[CMT01] Jean-Sébastien Coron, David M'Raïhi, and Christophe Tymen. Fast generation of pairs (k, [k]p) for koblitz elliptic curves. In Serge Vaudenay and Amr M. Youssef, editors, *Selected Areas in Cryptography, 8th Annual International Workshop, SAC 2001 Toronto, Ontario, Canada, August 16-17, 2001, Revised Papers*, volume 2259 of *Lecture Notes in Computer Science*, pages 151–164. Springer, 2001.

[CN99] Jean-Sébastien Coron and David Naccache. On the security of RSA screening. In Hideki Imai and Yuliang Zheng, editors, *PKC'99: 2nd International Workshop on Theory and Practice in Public Key Cryptography*, volume 1560 of *Lecture Notes in Computer Science*, pages 197–203, Kamakura, Japan, March 1–3, 1999. Springer, Heidelberg, Germany.

[CS97] Don Coppersmith and Adi Shamir. Lattice attacks on NTRU. In Walter Fumy, editor, *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 52–61. Springer, 1997.

[CT12] Thomas M. Cover and Joy A. Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[dBDJdW17] Koen de Boer, Léo Ducas, Stacey Jeffery, and Ronald de Wolf. Attacks on the ajps mersenne-based cryptosystem. Cryptology ePrint Archive, Report 2017/1171, 2017. https://eprint.iacr.org/2017/1171.

[DC14] Yibin Dai and Shaozhen Chen. Cryptanalysis of full PRIDE block cipher. Cryptology ePrint Archive, Report 2014/987, 2014. http://eprint.iacr.org/2014/987.

[DDRT12] Amine Dehbaoui, Jean-Max Dutertre, Bruno Robisson, and Assia Tria. Electromagnetic transient faults injection on a hardware and a software implementations of AES. In Guido Bertoni and Benedikt Gierlichs, editors, *FDTC 2012*, pages 7–15, Leuven, Belgium, September 9, 2012. IEEE Computer Society.

[de 95] Peter de Rooij. Efficient exponentiation using procomputation and vector addition chains. In Alfredo De Santis, editor, *Advances in Cryptology – EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*, pages 389–399, Perugia, Italy, May 9–12, 1995. Springer, Heidelberg, Germany.

[de 97] Peter de Rooij. On Schnorr's preprocessing for digital signature schemes. *Journal of Cryptology*, 10(1):1–16, 1997.

[DH06] White Diffie and Martin Hellman. New directions in cryptography. *IEEE Trans. Inf. Theor.*, 22(6):644–654, September 2006.

[DKL+98] Jean-François Dhem, François Koeune, Philippe-Alexandre Leroux, Patrick Mestré, Jean-Jacques Quisquater, and Jean-Louis Willems. A practical implementation of the timing attack. In Jean-Jacques Quisquater and Bruce Schneier, editors, *CARDIS '98*, volume 1820 of *Lecture Notes in Computer Science*, pages 167–182, Louvain-la-Neuve, Belgium, September 14-16, 1998. Springer, Heidelberg, Germany.

[DMP07] Suhas Diggavi, Michael Mitzenmacher, and H Pfister. Capacity upper bounds for deletion channels. In *Proceedings of the International Symposium on Information Theory*, pages 1716–1720, 2007.

[DR99] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael, 1999.

[EJ01] D. Eastlake, 3rd and P. Jones. Us secure hash algorithm 1 (sha1), 2001.

[ElG86] Taher ElGamal. On computing logarithms over finite fields. In Hugh C. Williams, editor, *Advances in Cryptology – CRYPTO'85*, volume 218 of *Lecture Notes in Computer Science*, pages 396–402, Santa Barbara, CA, USA, August 18–22, 1986. Springer, Heidelberg, Germany.

[ERW08] Cees Elzinga, Sven Rahmann, and Hui Wang. Algorithms for subsequence combinatorics. *Theoretical Computer Science*, 409(3):394–404, 2008.

[FHS04] Abraham Flaxman, Aram Wettroth Harrow, and Gregory B. Sorkin. Strings with maximally many distinct subsequences and substrings. *Electr. J. Comb.*, 11(1), 2004.

[Fia90] Amos Fiat. Batch RSA. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 175–185, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Heidelberg, Germany.

[Fia97] Amos Fiat. Batch RSA. *Journal of Cryptology*, 10(2):75–88, 1997.

[FJP11] Luca De Feo, David Jao, and Jérôme Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. Cryptology ePrint Archive, Report 2011/506, 2011. https://eprint.iacr.org/2011/506.

[FN17] Houda Ferradi and David Naccache. Integer reconstruction public-key encryption. Cryptology ePrint Archive, Report 2017/1231, 2017. https://eprint.iacr.org/2017/1231.

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

[Gal12] Steven D. Galbraith. *Mathematics of Public Key Cryptography*. Cambridge University Press, 2012.

[GBTP08] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual information analysis. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 426–442, Washington, D.C., USA, August 10–13, 2008. Springer, Heidelberg, Germany.

[GLSV15] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varici. LS-designs: Bitslice encryption for efficient masked software implementations. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption – FSE 2014*, volume 8540 of *Lecture Notes in Computer Science*, pages 18–37, London, UK, March 3–5, 2015. Springer, Heidelberg, Germany.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, pages 365–377, New York, NY, USA, 1982. ACM.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261, Paris, France, May 14–16, 2001. Springer, Heidelberg, Germany.

[Gol06] Oded Goldreich. On post-modern cryptography. *IACR Cryptology ePrint Archive*, 2006:461, 2006.

[Gra15] Benjamin Graham. A binary deletion channel with a fixed number of deletions. *Combinatorics, Probability and Computing*, 24(03):486–489, 2015.

[GSDS10] Sylvain Guilley, Laurent Sauvage, Jean-Luc Danger, and Nidhal Selmane. Fault injection resilience. In Luca Breveglieri, Marc Joye, Israel Koren, David Naccache, and Ingrid Verbauwhede, editors, *FDTC 2010*, pages 51–65, Santa Barbara, California, USA, August 21, 2010. IEEE Computer Society.

[Hir99] Daniel S. Hirschberg. Bounds on the number of string subsequences. In Maxime Crochemore and Mike Paterson, editors, *Combinatorial Pattern Matching, 10th Annual Symposium, CPM 99, Warwick University, UK, July 22-24, 1999, Proceedings*, volume 1645 of *Lecture Notes in Computer Science*, pages 115–122. Springer, 1999.

[HL05] Susan Hohenberger and Anna Lysyanskaya. How to securely outsource cryptographic computations. In Joe Kilian, editor, *TCC 2005: 2nd Theory of Cryptography Conference*, volume 3378 of *Lecture Notes in Computer Science*, pages 264–282, Cambridge, MA, USA, February 10–12, 2005. Springer, Heidelberg, Germany.

[HLH00] Min-Shiang Hwang, Iuon-Chang Lin, and Kuo-Feng Hwang. Cryptanalysis of the batch verifying multiple RSA digital signatures. *Informatica, Lith. Acad. Sci.*, 11(1):15–19, 2000.

[HLT01] Min-Shiang Hwang, Cheng-Chi Lee, and Yuan-Liang Tang. Two simple batch verifying multiple digital signatures. In Sihan Qing, Tatsuaki Okamoto, and Jianying Zhou, editors, *Information and Communications Security, Third International Conference, ICICS 2001, Xian, China, November 13-16, 2001*, volume 2229 of *Lecture Notes in Computer Science*, pages 233–237. Springer, 2001.

[Hoc59] Alexis Hocquenghem. Codes correcteurs d'erreurs. *Chiffres*, 2(2):147–56, 1959.

[How07] Nick Howgrave-Graham. A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. In Alfred Menezes, editor, *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, volume 4622 of *Lecture Notes in Computer Science*, pages 150–169. Springer, 2007.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In Buhler [Buh98], pages 267–288.

[HR00] Daniel S. Hirschberg and Mireille Regnier. Tight bounds on the number of string subsequences. *Journal of Discrete Algorithms*, 1(1):123–132, 2000.

[HS03] Jeffrey Hoffstein and Joseph H. Silverman. Random small hamming weight products with applications to cryptography. *Discrete Applied Mathematics*, 130(1):37–49, 2003.

[JL95] Tao Jiang and Ming Li. On the approximation of shortest common supersequences and longest common subsequences. *SIAM J. Comput.*, 24(5):1122–1139, 1995.

[Jou09] Antoine Joux. *Algorithmic cryptanalysis*. CRC Press, 2009.

[KF17] Paul Kirchner and Pierre-Alain Fouque. Revisiting lattice attacks on overstretched NTRU parameters. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 3–26, 2017.

[KJJ99a] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[KJJ99b] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Heidelberg, Germany.

[KL07] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC Press, 2007.

[KM13] Yashodhan Kanoria and Alessandro Montanari. Optimal coding for the binary deletion channel with small deletion probability. *IEEE Transactions on Information Theory*, 59(10):6192–6219, 2013.

[KMS10] Adam Kalai, Michael Mitzenmacher, and Madhu Sudan. Tight asymptotic bounds for the deletion channel with small deletion probabilities. In *IEEE International Symposium on Information Theory, ISIT 2010, June 13-18, 2010, Austin, Texas, USA, Proceedings*, pages 997–1001. IEEE, 2010.

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113, Santa Barbara, CA, USA, August 18–22, 1996. Springer, Heidelberg, Germany.

[KS05] François Koeune and François-Xavier Standaert. A tutorial on physical security and side-channel attacks. In *Foundations of Security Analysis and Design III, FOSAD 2004/2005 Tutorial Lectures*, volume 3655 of *Lecture Notes in Computer Science*, pages 78–108. Springer, Heidelberg, Germany, 2005.

[KU16] Mehmet Sabir Kiraz and Osmanbey Uzunkol. Efficient and verifiable algorithms for secure outsourcing of cryptographic computations. *Int. J. Inf. Sec.*, 15(5):519–537, 2016.

[LBC+16] Benjamin Lac, Marc Beunardeau, Anne Canteaut, Jacques J. A. Fournier, and Renaud Sirdey. A first DFA on PRIDE: from theory to practice. In Frédéric Cuppens, Nora Cuppens, Jean-Louis Lanet, and Axel Legay, editors, *Risks and Security of Internet and Systems - 11th International Conference, CRiSIS 2016, Roscoff, France, September 5-7, 2016, Revised Selected Papers*, volume 10158 of *Lecture Notes in Computer Science*, pages 214–238. Springer, 2016.

[LFG13] Ronan Lashermes, Jacques Fournier, and Louis Goubin. Inverting the final exponentiation of Tate pairings on ordinary elliptic curves using faults. In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems – CHES 2013*, volume 8086 of *Lecture Notes in Computer Science*, pages 365–382, Santa Barbara, CA, USA, August 20–23, 2013. Springer, Heidelberg, Germany.

[LL94] Chae Hoon Lim and Pil Joong Lee. More flexible exponentiation with precomputation. In Yvo Desmedt, editor, *Advances in Cryptology – CRYPTO'94*, volume 839 of *Lecture Notes in Computer Science*, pages 95–107, Santa Barbara, CA, USA, August 21–25, 1994. Springer, Heidelberg, Germany.

[LL15] Yuvalal Liron and Michael Langberg. A characterization of the number of subsequences obtained via the deletion channel. *IEEE Transactions on Information Theory*, 61(5):2300–2312, 2015.

[LLL82] Arjen Klaas Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[LM07] Laurie Law and Brian J. Matt. Finding invalid signatures in pairing-based batches. In Steven D. Galbraith, editor, *Cryptography and Coding, 11th IMA International Conference, Cirencester, UK, December 18-20, 2007, Proceedings*, volume 4887 of *Lecture Notes in Computer Science*, pages 34–53. Springer, 2007.

[MBB11] Mohamed Saied Emam Mohamed, Stanislav Bulygin, and Johannes A. Buchmann. Using SAT solving to improve differential fault analysis of trivium. In Tai-Hoon Kim, Hojjat Adeli, Rosslin John Robles, and Maricel O. Balitanas, editors, *ISA 2011*, volume 200 of *Communications in Computer and Information Science*, pages 62–71, Brno, Czech Republic, August 15-17, 2011. Springer, Heidelberg, Germany.

[McE78] Robert McEliece. A Public-Key Cryptosystem Based On Algebraic Coding Theory. *Deep Space Network Progress Report*, 44:114–116, January 1978.

[MDF+09] Houssem Maghrebi, Jean-Luc Danger, Florent Flament, Sylvain Guilley, and Laurent Sauvage. Evaluation of countermeasure implementations based on Boolean masking to thwart side-channel attacks. In *International Signals, Circuits and Systems Conference - SCS 2009*, pages 1–6, 2009.

[MGDF10] Houssem Maghrebi, Sylvain Guilley, Jean-Luc Danger, and Florent Flament. Entropy-based power attack. In Jim Plusquellic and Ken Mai, editors, *HOST 2010*, pages 1–6, Anaheim Convention Center, California, USA, June 13-14, 2010. IEEE Computer Society.

[Mid95] Martin Middendorf. On finding minimal, maximal, and consistent sequences over a binary alphabet. *Theoretical Computer Science*, 145(1):317–327, 1995.

[Mit08] Michael Mitzenmacher. A survey of results for deletion channels and related synchronization channels. In Joachim Gudmundsson, editor, *Algorithm Theory - SWAT 2008, 11th Scandinavian Workshop on Algorithm Theory, Gothenburg, Sweden, July 2-4, 2008, Proceedings*, volume 5124 of *Lecture Notes in Computer Science*, pages 1–3. Springer, 2008.

[MM04] Martin Middendorf and David F Manlove. Combined super-/substring and super-/subsequence problems. *Theoretical computer science*, 320(2):247–267, 2004.

[MN96] David M'Raïhi and David Naccache. Batch exponentiation: A fast dlp-based signature generation strategy. In Li Gong and Jacques Stearn, editors, *CCS '96, Proceedings of the 3rd ACM Conference on Computer and Communications Security, New Delhi, India, March 14-16, 1996.*, pages 58–61. ACM, 1996.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.

[MP08] S. Mrdovic and B. Perunicic. Kerckhoffs' principle for intrusion detection. In *Networks 2008 - The 13th International Telecommunications Network Strategy and Planning Symposium*, volume Supplement, pages 1–8, Sept 2008.

[MVM09] Frederic P. Miller, Agnes F. Vandome, and John McBrewster. *Advanced Encryption Standard*. Alpha Press, 2009.

[NMVR95] David Naccache, David M'Raïhi, Serge Vaudenay, and Dan Raphaeli. Can D.S.A. be improved? Complexity trade-offs with the digital signature standard. In Alfredo De Santis, editor, *Advances in Cryptology – EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*, pages 77–85, Perugia, Italy, May 9–12, 1995. Springer, Heidelberg, Germany.

[NS99] Phong Q. Nguyen and Jacques Stern. The hardness of the hidden subset sum problem and its cryptographic implications. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 31–46, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Heidelberg, Germany.

[NS01] Phong Q. Nguyen and Jacques Stern. The two faces of lattices in cryptology. In Joseph H. Silverman, editor, *Cryptography and Lattices, International Conference, CaLC 2001, Providence, RI, USA, March 29-30, 2001, Revised Papers*, volume 2146 of *Lecture Notes in Computer Science*, pages 146–180. Springer, 2001.

[NSS01] Phong Q. Nguyen, Igor E. Shparlinski, and Jacques Stern. Distribution of modular sums and the security of the server aided exponentiation. In *Cryptography and Computational Number Theory*, pages 331–342. Springer, 2001.

[oCoST12] U.S. Department of Commerce, National Institute of Standards, and Technology. *Secure Hash Standard - SHS: Federal Information Processing Standards Publication 180-4*. CreateSpace Independent Publishing Platform, USA, 2012.

[OS14] Or Ordentlich and Ofer Shayevitz. Bounding techniques for the intrinsic uncertainty of channels. In *2014 IEEE International Symposium on Information Theory, Honolulu, HI, USA, June 29 - July 4, 2014*, pages 3082–3086. IEEE, 2014.

[Pag02] Dan Page. Theoretical use of cache memory as a cryptanalytic side-channel. Cryptology ePrint Archive, Report 2002/169, 2002. http://eprint.iacr.org/2002/169.

[Pag04] Daniel Page. Defending against cache based side-channel attacks. *Information Security Technical Report*, 8(1):30–44, April 2004.

[PH78] Stephen C. Pohlig and Martin E. Hellman. An improved algorithm for computing logarithms over gf(p) and its cryptographic significance (corresp.). *IEEE Trans. Information Theory*, 24(1):106–110, 1978.

[PMPS00] Jaroslaw Pastuszak, Dariusz Michatek, Josef Pieprzyk, and Jennifer Seberry. Identification of bad signatures in batches. In Hideki Imai and Yuliang Zheng, editors, *PKC 2000: 3rd International Workshop on Theory and Practice in Public Key Cryptography*, volume 1751 of *Lecture Notes in Computer Science*, pages 28–45, Melbourne, Victoria, Australia, January 18–20, 2000. Springer, Heidelberg, Germany.

[PRRR15] Bart Preneel, Phillip Rogaway, Mark Dermot Ryan, and Peter Y. A. Ryan. Privacy and security in an age of surveillance (dagstuhl perspectives workshop 14401). *Dagstuhl Manifestos*, 5(1):25–37, 2015.

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

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, 2000.

[QS01] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA): measures and counter-measures for smart cards. In *E-smart 2001*, volume 2140 of *Lecture Notes in Computer Science*, pages 200–210, Cannes, France, September 19-21, 2001. Springer, Heidelberg, Germany.

[Rab79] Michael O. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical report, Cambridge, MA, USA, 1979.

[Rah06] Sven Rahmann. Subsequence combinatorics and applications to microarray production, DNA sequencing and chaining algorithms. In Moshe Lewenstein and Gabriel Valiente, editors, *Combinatorial Pattern Matching, 17th Annual Symposium, CPM 2006, Barcelona, Spain, July 5-7, 2006, Proceedings*, volume 4009 of *Lecture Notes in Computer Science*, pages 153–164. Springer, 2006.

[RC13] Peter YA Ryan and Bruce Christianson. Enhancements to prepare-and-measure based qkd protocols. In *Security Protocols XXI*, pages 123–133. Springer, 2013.

[RD13] Mehdi Rahmati and Tolga M Duman. Bounds on the capacity of random insertion and deletion-additive noise channels. *IEEE Transactions on Information Theory*, 59(9):5534–5546, 2013.

[Rog16] Phil Rogaway. Practice-oriented provable security and the social construction of cryptography. *IEEE Security Privacy*, 14(6):10–17, Nov 2016.

[RS60] Irving S. Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

[RSA78a] Ron Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[RSA78b] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signature and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978.

[SA03] Sergei P. Skorobogatov and Ross J. Anderson. Optical fault induction attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 2–12, Redwood Shores, CA, USA, August 13–15, 2003. Springer, Heidelberg, Germany.

[SBK+17] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. The first collision for full SHA-1. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 570–596. Springer, 2017.

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Heidelberg, Germany.

[SD13] Frederic Sala and Lara Dolecek. Counting sequences obtained from the synchronization channel. In *Proceedings of the 2013 IEEE International Symposium on Information Theory, Istanbul, Turkey, July 7-12, 2013*, pages 2925–2929. IEEE, 2013.

[SF03] Theo G. Swart and Hendrik C. Ferreira. A note on double insertion/deletion correcting codes. *IEEE Transactions on Information Theory*, 49(1):269–273, 2003.

[SGSD15] Frederic Sala, Ryan Gabrys, Clayton Schoeny, and Lara Dolecek. Three novel combinatorial theorems for the insertion/deletion channel. In *IEEE International Symposium on Information Theory, ISIT 2015, Hong Kong, China, June 14-19, 2015*, pages 2702–2706. IEEE, 2015.

[SH13] Ling Song and Lei Hu. Differential fault attack on the PRINCE block cipher. Cryptology ePrint Archive, Report 2013/043, 2013. http://eprint.iacr.org/2013/043.

[Sha49] C. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal, Vol 28, pp. 656–715*, Oktober 1949.

[Sha71] Daniel Shanks. Class number, a theory of factorization, and genera. In *Proc. Symp. Pure Math*, volume 20, pages 415–440, 1971.

[Sha01] Claude E Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[Sho97a] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997.

[Sho97b] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266, Konstanz, Germany, May 11–15, 1997. Springer, Heidelberg, Germany.

[Sko05] Sergei Skorobogatov. Semi-invasive attacks - A new approach to hardware security analysis. Technical Report 630, University of Cambridge, April 2005.

[SLIO12] Kazuo Sakiyama, Yang Li, Mitsugu Iwamoto, and Kazuo Ohta. Information-theoretic approach to optimal differential fault analysis. *IEEE Transactions on Information Forensics and Security*, 7(1):109–120, 2012.

[SOOS95] Richard Schroeppel, Hilarie K. Orman, Sean W. O'Malley, and Oliver Spatscheck. Fast key exchange with elliptic curve systems. In Don Coppersmith, editor, *Advances in Cryptology – CRYPTO'95*, volume 963 of *Lecture Notes in Computer Science*, pages 43–56, Santa Barbara, CA, USA, August 27–31, 1995. Springer, Heidelberg, Germany.

[SW12] Dennis Stanton and Dennis White. *Constructive combinatorics*. Springer Science & Business Media, 2012.

[TBM14] Harshal Tupsamudre, Shikha Bisht, and Debdeep Mukhopadhyay. Differential fault analysis on the families of SIMON and SPECK ciphers. Cryptology ePrint Archive, Report 2014/267, 2014. http://eprint.iacr.org/2014/267.

[Tur36] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936.

[Ull67] Jeffrey D. Ullman. On the capabilities of codes to correct synchronization errors. *IEEE Transactions on Information Theory*, 13(1):95–105, 1967.

[YHS+15] Qianqian Yang, Lei Hu, Siwei Sun, Kexin Qiao, Ling Song, Jinyong Shan, and Xiaoshuang Ma. Improved differential analysis of block cipher PRIDE. In Javier Lopez and Yongdong Wu, editors, *ISPEC 2015*, volume 9065 of *Lecture Notes in Computer Science*, pages 209–219, Beijing, China, May 5-8, 2015. Springer, Heidelberg, Germany.

[YL95] Sung-Ming Yen and Chi-Sung Laih. Improved digital signature suitable for batch verification. *IEEE Trans. Computers*, 44(7):957–959, 1995.

[ZWG11] XinJie Zhao, Tao Wang, and ShiZe Guo. Improved side channel cube attacks on PRESENT. Cryptology ePrint Archive, Report 2011/165, 2011. http://eprint.iacr.org/2011/165.

[ZWWD14] Jingyuan Zhao, Xiaoyun Wang, Meiqin Wang, and Xiaoyang Dong. Differential analysis on block cipher PRIDE. Cryptology ePrint Archive, Report 2014/525, 2014. http://eprint.iacr.org/2014/525.

Résumé

Cette thèse, à la frontière entre sécurité de l'information et cryptographie s'intéresse à l'utilisation de cette dernière dans la sécurité informatique. Cette thèse est divisée en trois parties scientifiquement indépendantes, qui partagent la même propriété de résoudre des problèmes auxquels sont ou seront confrontés les industries du digital. Nous étudions ainsi le traitement par batch de signatures, afin de répondre à la future omniprésence d'appareils à faible puissance de calcul étant connecté à des réseaux ouverts ; et devant donc authentifier un grand nombre de messages. Nous nous intéressons ensuite à la menace post-quantique, en examinant un nouveau problème difficile impliquant des ratios de nombre de faible poids de Hamming. Enfin nous regardons la sécurité physique d'algorithme symétrique et d'échange de clé quantique, le premier étant un défi de longue date, et l'autre une possibilité pour la futur distribution de clé cryptographique s'affranchissant des problèmes classique de la cryptographie.

Mots clés

Sécurité de l'information, sécurité physique, cryptographie à clé publique

Abstract

This thesis, on the border between information security and cryptography, focuses on the use of information security in computer security. This thesis is divided into three scientifically independent parts, which share the same property of solving problems that are or will be faced by the digital industries. We study the batch processing of signatures, in order to respond to the future omnipresence of devices with low computing power being connected to open networks; and therefore having to authenticate a large number of messages. We then focus on the post-quantum threat, examining a new challenging problem involving low-weight Hamming number ratios. Finally we look at the physical security of symmetric algorithm and quantum key exchange, the former being a long-standing challenge, and the other a possibility for future cryptographic key distribution free from the classic problems of cryptography

Keywords

Information security, physical security, public key cryptography