Securisation of implementations of cryptographic algorithms in the context of embedded systems

Axel Mathieu-Mahias

université PARIS-SACLAY

Sécurisation des implémentations d'algorithmes cryptographiques pour les systèmes embarqués

Securisation of implementations of cryptographic algorithms in the context of embedded systems

Thèse de doctorat de l'université Paris-Saclay

École doctorale n° 580, Sciences et Technologies de l'Information et de la Communication (STIC)
Spécialité de doctorat: Informatique
Unité de recherche: Université Paris-Saclay, UVSQ, CNRS, Laboratoire de mathématiques de Versailles, 78000, Versailles, France.
Référent: Université de Versailles-Saint-Quentin-en-Yvelines

Thèse présentée et soutenue à Paris-Saclay, le 06/12/2021, par

Axel MATHIEU-MAHIAS

Composition du jury

Jean-Sébastien CORON Professeur, Université du Luxembourg	Président
Guénaël RENAULT HDR, Directeur adjoint du laboratoire Sécurité matérielle à l'ANSSI (Agence nationale de la sécurité des systèmes d'information)	Rapporteur & Examinateur
Damien VERGNAUD Professeur, Sorbonne Université	Rapporteur & Examinateur
Sonia BELAÏD Experte en sécurité, CryptoExperts	Examinatrice

Direction de la thèse

Louis GOUBIN Professeur, Université de Versailles Saint-Quentin-en-Yvelines	Directeur
Michaël QUISQUATER Maître de conférences, Université de Versailles Saint-Quentin-en-Yvelines	Co-encadrant

Thèse de doctorat

NNT: 2021UPASG095

Acronyms

ADC Analog to Digital Converter. 17

AES Advanced Encryption Standard. 25, 45, 113, 114, 131, 177

ALU Arithmetic-Logic Unit. 17

ASIC Application-Specific Integrated Circuit. 15, 19, 26

CMOS Complementary Metal Oxide Semi-conductor. 31

CPU Central Processing Unit. 11, 16, 17, 174

DAC Digital to Analog Converter. 17

DPA Differential Power Analysis. 30–32, 39, 40, 44

DSP Digital Signal Processor. 17, 19

EEPROM Electrically-Erasable Programmable Read-Only Memory. 18

EPROM Erasable Programmable Read-Only Memory. 18

FET Field Effect Transistor. 14, 15

FFT Fast Fourier Transform. 153, 154, 168

FPGA Field-Programmable Gate Array. 15, 26

GCD Greatest Common Divisor. 154

IC Integrated Circuit. 10, 12, 13, 15–17, 31

IoT Internet of Things. 11, 16

LAN Local Area Network. 19

MOS Metal Oxide Semi-conductor. 15

NI Non-Interference. 41, 42, 45

PROM Programmable Read Only Memory. 18

RAM Random Access Memory. 17

ROM Read-Only Memory. 17, 18

RTOS Real-Time Operating System. 16

SCA Side-Channel Analysis. 30

SE Secure Element. 23

SNI Strong Non-Interference. 33, 37, 41–43, 45, 177

SPA Simple Power Analysis. 30

SPN Substitution-Permutation Network. 12, 24

WAN Wide Area Network. 19

Contents

List of Tables

List of Algorithms

Part I

General Introduction

Chapter 1

Panorama of this Thesis

Securing information is crucial in our modern and interconnected world. Information security encompasses the security of many aspects of our lives as we rely more and more on information and communications technology. The security issues related to the way information is created, stored and shared via such technologies can be mitigated by using cryptography, a science solely dedicated to ensuring information security. The correct deployment of cryptography is however challenging. A particular attention has to be paid when it comes to securing physical devices that store and process sensitive information.

In what follows, we briefly go back to the electronic roots of all information and communications technologies, *i.e.* the inventions of the *transistor* and the *Integrated Circuit* (IC). Then we focus on *embedded systems*, major components of the modern electronic landscape. We give examples of their applications and explain why their security is highly required. We introduce cryptography as one of the possible means to secure embedded systems and exhibit the main physical security issues behind the deployment of cryptography on such systems. Finally, we give the main focus of this thesis, that is the security of the so-called *cryptographic devices*, onto which cryptography is deployed, against devastating attacks exploiting physical access.

The extensive use we make today of information and communications technology has been possible by progresses in microelectronics since the invention of the *transistors*. The miniaturization of electronic components played a central role since the smaller they got, the more they could be combined. Basically, integrated circuits are electronic chips integrating a very large number of transistors. Examples of ICs are microprocessors, memory, communication interfaces and electronic peripherals of all kinds. Furthermore, the combination of various different ICs enabled the manufacturing of various information and communications technologies

we use today, such as embedded systems.

An embedded system is a small-scale computing system (with a CPU, memory, communication interfaces and peripherals of all kinds) within a system and that is designed to perform a few specific tasks (e.g. control, monitor, actionate). Embedded systems are typically much smaller and much cheaper than common personnal computers which facilitates their physical deployment for many applications.

Today, most of our industries, businesses and safety-critical sectors rely on embedded systems. We find embedded systems in automotive, telecommunications, health care, banking, military, aerospace, consumer electronics, to name just a few. Embedded systems in smart cards or biometric passports can also be used to store and process sensitive or private information. Furthermore, embedded systems are more and more part of the *Internet of Things* (IoT), the next generation of the Internet. They are embedded into the various "things" that are connected to the Internet, only increasing the amount of information that is created, stored and shared. We are now building smart homes and smart cities, which is only possible via all information that is gathered by the IoT devices containing embedded systems and then analyzed into the Cloud to provide many new functionalities.

The reliance on embedded systems is undeniably increasing, also leading to many new security challenges. The information handled by embedded systems can be stored, used or being transmitted. In any case, it has to be secured. Guaranteeing the security of information handled by a device that is physically accessible is even more challenging. Cryptography can help in securing embedded information which security is at great risk.

Cryptography tackles information security issues. It provides secure building blocks called *cryptographic primitives* that can be implemented in software or hardware into physical devices through their embedded systems. A well-known example of such primitives is a *block cipher* which can be used to render the information inintelligible to unauthorized entities and therefore protecting its disclosure. However, the security of block ciphers themselves rely on the protection of cryptographic assets called *cryptographic keys* which are used during execution and are typically stored in the physical devices. In this context, the purpose of an attacker is to gain information in order to determine the cryptographic keys. With a physical access to a device, an attacker can guess these keys by exploiting key-dependent information in the measurements of the physical properties of the targeted device. A common example of a physical property is typically the instantaneous power consumption of a device while it is executing the implementation of

a block cipher. The problem is that physical properties vary according to the data being processed and therefore also according to the cryptographic key being used. Retrieving such a key can then be done by first recording the power measurements, and then by performing a statistical analysis of the measurements. If the attack is successful, an attacker breaks the security of the block cipher and consequently also breaks the cryptographic security measure that has been deployed in the physical device.

Attacks exploiting a physical access to a device are called *physical attacks*. Not only they can exploit the power consumption but also electromagnetic emanations or running times. Some other physical attacks would try to make the targeted physical device malfunction and exploit an erratic behaviour to retrieve the cryptographic keys. The scope of physical attacks is large and hard to mitigate. One countermeasure that widely deployed in the literature to thwart some of them is called *masking*. This countermeasure is the main focus of this thesis.

In a few words, masking can be deployed to randomize all data processed during the execution of a block cipher by a device. The purpose of masking is to render the physical properties of the running device as independent as possible of the data being processed. This would make the measurements harder to analyze with statistical tools and consequently also protect the keys from being extracted via physical attacks.

In the rest of this introduction, we review the major aspects of the landscape just described above. We also build progressively the focus of this thesis. Chapter 2 goes back to the inventions of the transistor and the IC. We also give details about one of the major application of ICs, *i.e.* embedded systems and their possible classifications. In chapter 3, we address the information security principle underlying block ciphers and give details about one of the most common structure used for their design, the *Substitution-Permutation Network* (SPN). Chapter 4 then describes typical examples of physical attacks against the implementations of block ciphers in embedded systems. It also narrows the focus of this thesis on specific subclasses of physical attacks. Chapter 5 is probably the most important one as it brings together all necessary material to fully understand the focus of this thesis. It describes the masking countermeasure and how it can be deployed to secure any implementation of block ciphers in physical devices against some of the attacks described in chapter 4. Finally, chapter 6 gives the contributions of this thesis with respect to our focus.

Chapter 2

Embedded Systems

Embedded systems are computing systems that are typically small, little demanding in terms of power and are low cost. These features make them perfectly suited for a wide range of applications. Namely, they can be embedded as part of larger systems in many fields for achieving a few specific tasks that are usually low demanding in terms of computing resources.

Several major breakthroughs in electronics were needed to be able to build the various small-scale computing systems we use today. The history of embedded system is inherently tied to the inventions of the transistor and the IC because transistors are part of any integrated circuit and any embedded system is built upon some specific ICs.

In this chapter, we go back to the roots of embedded systems. We start by giving a brief history of the transistor that led to the invention of the IC. We also briefly address the problem of integrating several transistors and other electronic components to build an IC, a circuit on a single small piece of semiconductor that is usually silicon. Then, we give details about typical integrated circuits such as microprocessors, Digital Signal Processors, memories and in particular about microcontrollers, commonly used to build embedded systems. Finally we give classifications of such systems depending on their hardware performances or on their functional requirements.

Contents

2.1 Invention of the Integrated Circuit.

An integrated circuit connects together electrically many individual electronic components such as resistors, capacitors, transistors to form an electronic circuit on one small so-called *chip* of semiconductor material that is usually silicon. The idea of building an integrated circuit was first pointed out by G. W. A. Dummer. His idea was the following : "With the advent of the transistor and the work in semiconductors generally, it seems now possible to envisage electronic equipment in a solid block with no connecting wires. The block may consist of layers of insulating, conducting, rectifying and amplifying materials, the electrical functions being connected directly by cutting out areas of the various layers."

Transistors are one of the most important components of integrated circuits, they are the amplifying materials G. W. A. Dummer mentionned. However, the first transistors were not necessarily highly reliable, and their manufacturing process was too complicated to be massively produced. In what follows, we give a brief history of transistors.

The history we describe starts in 1926 with a patent from Julius Edgar Lilienfeld titled "Method and Apparatus" for "Controlling Electric Currents". In this patent was described what we call today the *Field-effect transistor* (FET). However, there is no proof that Lilienfeld actually constructed a functionning device. Twenty years later, some unsuccessful attempts were made to build a FET out of semiconductors to replace the large and not highly reliable vacuum tubes then used for amplifying signals.

Altough no one succeeded in building a FET at that time, other directions led to the invention of the first (germanium) transistor in 1947 : the *point-contact transistor*. It was created by John Bardeen and Walter Brattain whose device amplified the input current up to 100 times. The term transistor was given by electrical engineer John Pierce. It is worth noting that in 1948, six months after Bardeen and Brattain, the point-contact transistor was independently invented by two German physicists working in Paris : Herbert Mataré and Heinrich Welker. However, both independent inventions of point-contact transistors were superseded the same year by William Shockley's *junction transistor*. This new type of transistor was more reliable than its predecessors which were also noisy.

At that time, the processes for manufacturing transistors was not developped enough. Two more years were needed before Bell Labs scientists and engineers came with solutions. After manufacturing processes were improved, the first applications of transistors were transistor radios. Music and information suddenly became portable.

There was however a limit on how small you could make each transistor, since after it was made it had to be connected to wires and other electronics. The transistors were already at the limit of what was possible to achieve by hand. So, the idea to make a whole circuit (*i.e.* an IC) at once became popular. Building circuits in just one step would allow to manufacture all the parts much smaller.

It was in 1958 that Jack Kilby created the first integrated germanium chip at Texas Instruments. He managed to create simple transitor, capacitor, resistor elements out of semiconductor material. However, the components were still wired by hand using fine gold "flying wires", protruding out of the chip. No other practical production technique was available at that time.

Another approach followed by Robert Noyce in 1959 led to the development of a silicon IC and without "flying wires". He interconnected the electronic components by depositing aluminium metal lines via photolithography. One could therefore configure complete electronic circuits on a single silicon chip quite easily.

The same year, a breakthrough in FET came with the work of Mohamed Atalla and Dawon Kahng who achieved to build the first working FET, more than 30 years after the pioneer work of Lilienfeld. The main problem for building FETs was that surface effects blocked electric field from penetrating into the semiconductor material. Investigating thermally grown silicon-dioxide layers, they found these states could be markedly reduced at the interface between the silicon and its oxide in a sandwich comprising layers of metal (M - gate), oxide (O - insulation), and silicon (S - semiconductor) - thus the name *MOSFET*, popularly known as MOS.

The IC were soon all based on MOS. Millions of MOS transistors were later combined together onto single chips, following the so-called *Complementary* MOS production technique invented by Chih-Tang Sah and Frank Wanlass. The advantage of this technique is that logic circuits combining MOS transistors in a complementary symmetry configuration draw close to zero power in standby mode. In the 1970s, MOS chips were widely adopted, enabling complex semiconductor and telecommunication technologies to be developed, such as microprocessors. The transistor and the integrated circuit revolutionized electronics. Today, they are

part of every computing devices we all use on a daily basis. Between 1960 and 2018, an estimated total of 13 sextillion MOS transistors have been manufactured, accounting for at least 99.9% of all transistors[1].

Microprocessors, microcontrollers, Field Programmable Gate Arrays (FPGA) and Application Specific Integrated Circuits (ASIC) are typical examples of ICs that integrate a very large number of transistors.

2.2 Ubiquity of Embedded Systems.

The term *embedded system* refers to a (sub)system that is enclosed or embedded in a larger system. An embedded system combines hardware and software, as well as other components. In other words, it is an electronic or electro-mechanical system that can be embedded inside any product.

An embedded system is designed to accomplish a few specific tasks only (as opposed to a computer system). It interacts with its surrounding environment, typically by monitoring and controlling some processes. On the contrary to a computer system consisting of a general-purpose operating system, an embedded system is usually a *Real-Time Operating System* (RTOS).

The history of embedded systems began in space. Following the invention of the integrated circuits, silicon chips were used to build the first embedded computer in the 1960's, as part of the Apollo program. The digital Apollo Guidance (embedded) Computer was also the first computer to use silicon IC chips. It was designed by Charles Stark Draper at the Massachussetts Institute of Technology. At that time, NASA's Apollo Program was the largest single consumer of ICs in the world.

The integration of transistors has greatly increased since the design of the first embedded system. Today, we find embedded systems everywhere. They encompass almost all domains of our modern world : Automotive industry (airbag control system, anti-lock braking system), telecommunications (telephone switches, smart phones), household (washing machine, refrigerators), Home automation and security systems (air conditioners, fire alarms), health care (MRI scans, pacemakers), Banking industry (ATMs), military (surveillance systems, defense and aerospace), consumer electronics (digital cameras, smart phones). They are also expected to find more and more applications in the future, especially with the development of the IoT.

[1]Source : Computer History Museum

Microprocessors are an integrated implementation of the central processing unit (CPU) portion on a single chip. They are usually designed to be used in general-purpose systems, like personal computers. However, to implement a complete (embedded) computer system, the input/output subsystems and the external memory system still have to be included. Because the computational capacity of microprocessors is usually high compared to microcontrollers, they tend to consume also more power and therefore also require external cooling system. There are embedded systems built on microprocessors, but modern embedded systems are largely based on microcontrollers.

A microcontroller brings together the CPU, some memory, Input/Output (I/0) Ports, Timers, into a single IC. They are designed to perform a few specific tasks with regards to some specific control applications. Therefore, a typical microcontroller have relatively low computational and memory capacity, consume little power and therefore do not necessarily require a cooling system, which makes it low cost in comparison to microprocessors.

It is worth mentionning that more and more embedded systems are built upon *Digital Signal Processors* (DSPs), particularly of interest for high-data-rate computations. They are usually combined with an *Analog-to-Digital* and Digital-to-Analog Converters (ADC and DAC). DSPs also implement algorithms in hardware and offer high performance in repetitive and numerically intensive tasks (especially for signal processing applications), including audio, video and communication applications. They are however expensive.

2.2.1 Basics of Microcontrollers.

CPU. A CPU is where all computations are performed. It is handled by four main components : the registers, the Arithmetic Logic Unit (ALU), the control unit and the internal CPU buses. The registers provide operands to the ALU, they are a type of fast memory. The ALU performs arithmetic and bitwise operations on integer binary numbers. The control unit manages the entire fetching (instruction, operands from registers) and execution cycle. The internal CPU buses interconnect the ALU, registers and the control unit.

Memory. Microcontrollers integrate RAM (Random Access Memory) and ROM (Read-Only Memory) on the same chip as the CPU. Data can be stored in RAM by writing to it and stored data can be accessed by reading from it. Most RAMs are volatile - the content will be lost when the power is switched off. ROM is used for

storing long term non volatile information. The content of on-chip ROM is usually only accessible by the system it is used in. Such content is typically *the firmware*. The latter provides the low-level control of the device's specific hardware. If the firmware is erased, this could cause the embedded system to malfunction.
CACHE MEMORY. It is a type of RAM that is used to store program instructions that are frequently re-referenced by software during operation. The efficiency of a processor is greatly dependent on its cache memory.

PROM, EPROM, EEPROM (FLASH). PROM is similar to ROM except that it is programmable. However, once it is programmed, it cannot be modified. EPROM solves the previous issue. Even programmed, it can be erased by exposing it to strong ultraviolet light source, and then a new program can be written into it. Finally EEPROM can be written, erased, rewritten electronically. A typical widely used EEPROM memory is the *Flash* memory that can erase electronically all the data fast.

Interfaces. Embedded processor communicate with the external world through input and output interfaces. An Input/Output interface is an electronic device which has one side connecting to the processor and the other side connecting to the Input/Output devices. Input/Output ports (collections of pins) are components of interfaces from which the processor reads/sends the information coming/going to/from the devices.

2.2.2 Embedded Systems with Sensors and Actuators.

Sensors and actuators are essential input and output devices respectively of embedded systems. A sensor (or detector) is an input device of an embedded system. It captures a physical stimulus (detects events or changes in its environment) such as heat, light, sound, pressure or other mechanical motion (e.g. acceleration) and generates a proportional electrical current.

An actuator is an output device of an embedded system. It typically accepts a control input (mostly an electrical signal) it to a proportional physical stimulus such as heat, light, sound, pressure or other mechanical motion.

If the embedded system is designed for any controlling purpose, actuators are connected to the output port of the embedded system. If the embedded system is designed for any monitoring purpose, actuators are not needed, but sensors may be.

2.2.3 A Classification of Embedded Systems.

Embedded systems are usually classified based on two factors depending on their functional requirements or on the performances of the microcontroller.

Functionnal requirements :

1. **Real-time.** It is a system in which the resulting performance depends not only on the correctness of any control action but also on the time at which the actions are produced. A wide variety of applications for embedded systems fall into this category. They include industrial plants control, automotive, flight control systems, industrial automation, space missions, telecommunications, consumer electronics. The most important property of a real-time system is not high speed, but predictability : it should be possible to determine in advance whether all computational activities can be completed within their timing constraints. Automotive airbag control system, flight-control system are examples of this category.

2. **Stand alone.** The system is independent on a host system. It can work by itself, takes the input either in analog or digital form, processes and produces the output. It may either control or drive the connected devices. Examples are digital cameras, microwave ovens or temperature measurement systems.

3. **Networked.** They are connected to a network to access the ressources. The network can be LAN, WAN, or the internet. The connection can be wired or wireless. The embedded web server is a type of system wherein all embedded devices are connected to a web server and accessed and controlled by a web browser. One example of the LAN networked embedded system is a home security system.

4. **Mobile.** They are compact, easy to use and require fewer resources. They are used in portable embedded devices such as mobile phones or digital cameras.

Hardware performances :

1. **Small scale.** Designed with a single 8 or a 16-bit microprocessor/controller. It may or may not contain an operating system. An electronic toy is an example for this type of embedded system.

2. **Medium scale.** Typically designed with a 16 or 32-bit microprocessor/controller, ASICs or DSPs.

3. **Sophisticated or complex.** They have highly complex hardware and software, built around 32 or 64-bit processors/controllers, scalable and configurable processors. They also contain high-performance real-time operating system for task scheduling, prioritization and management.

Chapter 3

Cryptography for Securing Information

Historically, *cryptography* was the science addressing secret ways to communicate and store information. In other words, cryptography delt with *confidentiality* issues related to information. In its early uses, the science of secrets was mainly applied for military purposes. Cryptography has grown over the years to encompass more security aspects of information then just confidentiality. Usually, all modern security concerns about information are regrouped under the terminology *information security*.

In its modern applications, cryptography deals with digital information. An increasing volume of digital information is manipulated by embedded systems every day. Digital information has to be secured whether it is stored, being used or being shared between digital computing devices. Cryptography provides different algorithmic ways to ensure one or several security services related to digital information. Eventually, cryptographic algorithms are implemented into embedded systems to actually provide in practice the related security services. Typically, cryptographic *encryption* algorithms, which can be implemented into embedded systems, can prevent or mitigate information security risks related to confidentiality, still one of the primary concern of information security.

In this chapter, we start by giving a high level description of two complementary paragdims to perform encryption, symmetrically or asymmetrically. We also discuss the security paradigm of encryption algorithms following a well-known cryptographic principle dictated by Auguste Kerchkoff. The focus of this thesis is on *symmetric encryption* and in particular on a subclass of primitives called *block ciphers*. Therefore, we then give more details about such primitives and in particular about the structure underlying the design of many block ciphers : the

so-called *Substitution-Permutation* networks.

Contents

3.1 Encryption for Confidentiality.

Information can be in different states : *at rest*, stored for a later use or *in motion* - being moved from one location to another. Confidentiality is the property that ensures a given information is not accessible by unauthorized entities. This property should hold whether information is at rest or in motion.

In our context, we usually consider that data is stored in an untrusted environment (e.g. personnal computers) and sent on an untrusted channel (e.g. Wi-Fi). This brings us to the notion of *unauthorized entity/party*, which refers to something (typically an application) or someone (typically a user or an external malicious person) that should not gain access to some restricted piece of information if not explicitely authorized.

In what follows, we focus on the cryptographic primitives related to confidentiality, that is *encryption* primitives. For the sake of clarity, we start by distinguishing between *symmetric* and *asymmetric* encryption. We also discuss the possibility to perform *hybrid* encryption by combining encryption primitives of the two worlds. However, asymmetric cryptography is out of the scope of this thesis.

3.1.1 Symmetric, Asymmetric and Hybrid Encryption.

Encryption is the historical principal application of cryptography as a mean to provide confidentiality. It scrambles a given information into incomprehensible data. A specific encryption process is described by an algorithm (a set of successive instructions), that is parameterized by a *cryptographic key*. The original information can be retrieved by a decryption process that also makes use of a cryptographic key. However, symmetric and asymmetric encryption do not proceed similarly for the application of the cryptographic keys between the encryption an decryption processes.

Symmetric encryption make use of a single identical key to perform encryption and decryption of a given message. Symmetric encryption is also called *secret-key* encryption. On the contrary, asymmetric encryption uses a pair of keys that are mathematically related. One key is public knowledge (for encryption) and the other is secret information (for decryption). Asymmetric encryption is also called *public-key* encryption.

Encryption primitives imply that the secret key is only known by authenticated parties beforehand the confidential communication itself. Sharing such a sensitive information on an unsecure channel is not a secure option. For symmetric schemes, sharing the secret-keys is known as the key distribution problem. It was an open problem until new directions in cryptography were found in 1976, i.e. the public-key cryptography.

The aforementionned key distribution problem of symmetric cryptosystems on an unsecure channel can be solved by using an asymmetric encryption scheme. Asymmetric schemes can inherently derived their secret keys securely from the knowledge of the public key (common information shared by the two parties wishing to communicate confidentially). It is therefore possible to use asymmetric cryptosystems to distribute the symmetric keys. The simple idea is to first encrypt the symmetric key with an asymmetric scheme. Then, the encrypted key can be sent on an unsecure channel to the other entity. Finally, the symmetric key is decrypted with the asymmetric scheme. The two parties involved will have the same symmetric key in their possession.

Such a process is usually performed in practice because symmetric encryption is very fast compared to its asymmetric counterpart. It requires less extensive power from the device that is executing the algorithm. Therefore, even small embedded systems can perform fast symmetric encryption. In practice, symmetric encryption is the preferred choice when it comes to send long messages. Asymmetric encryption is reserved to send very short messages such as the symmetric keys beforehand the communication itself. Then, symmetric encryption takes over.

Note that when it comes to encrypt data for guaranteeing the confidentiality during storage, there is no need for hybrid encryption. This setting only requires one secret key and not two identical keys as for communications.

Without discussing security aspects in details, it is quite obvious that secret keys should be kept secret since the cryptographic keys allow decryption. The following

aims at emphasizing that efforts should always be made to safeguard cryptographic secret keys.

3.1.2 Safeguarding the Secret Keys.

Whether one relies on symmetric or asymmetric encryption to provide confidentiality, the underlying security encryption schemes is not guaranteed (or completely broken in the worst cases) if any unauthorized entity can somehow retrieve the cryptographic secret keys. Ways to retrieve cryptographic keys are numerous but some ways are sometimes much easier than others. As examples, one can try to break the targeted primitive by exhibiting some predictible behaviours during its execution and then deduce the keys that have been employed or one can simply steal the cryptographic keys if they are easily accessible.

The security of any encryption primitive should never rely on its secrecy, but only on the secrecy of the secret keys. This principle is attributed to Auguste Kerckhoff, and goes back to the ninetieth century. It transfers the security of any cryptographic primitive solely on the security of the cryptographic keys. Efforts should always be made to safeguard cryptographic keys whether they are being generated, used, transmitted or stored. This can never be enough emphasized. However, encryption algorithms usually run in untrusted environments such as personnal computers, thus putting at risk the secrecy of the secret keys. The generation, use and storage of secret keys should be (if possible) relegated to trusted environments, usually referred to as *secure elements* (SE).

An SE is a microprocessor chip which can store sensitive data and also run secure applications such as payment. It acts as a vault, protecting what is inside the SE (applications and data) from malware attacks that are typical in the host (i.e. the device operating system). A secure element is specifically designed to protect unauthorized access and is only used to run a limited set of applications, as well as store confidential and cryptographic data.

SEs are hardware security solutions that can take various forms such as UICCs (universal integrated circuit cards) and microSD hardware cards. Additionally, SE is available with an embedding option that enables it to be pinned on a device's motherboard. This category of secure elements can either be embedded UICCs or embedded SEs.

In what follows, we focus on one particular type of symmetric encryption primitives, *block ciphers*.

3.2 Symmetric Block Ciphers for Encryption

Block ciphers are designed for a fixed-input size called *blocks* and are iteratively applied to encrypt arbitrary length messages. The original block of information is called *plaintext* or simply *message* and the resulting block of same size is called *ciphertext*. The encryption of a plaintext into a ciphertext is performed under the action of a secret key. The inverse transformation, decryption, enables to retrieve the original plaintext from the corresponding ciphertext provided that the same secret key is used.

More formally, a block cipher can be defined as a bijective function parameterized by a cryptographic key usually chosen at random. It maps n-bit plaintexts to n-bit ciphertexts. For a fixed key, this defines a permutation on n-bit vectors and therefore, each key potentially defines a different permutation. If all $(2^n)!$ bijections on 2^n elements can be reached (with one key per permutation), the n-bit block cipher is said to be a *random cipher*. However, this would require too much memory to represent the keys for practical applications.

Block ciphers are typically designed for practical key lengths providing an acceptable level of computational security. The lower bound for the bit length of a key is given by the computationnal efforts required to perform an *exhaustive search* of this key. The exhaustive search of a n-bit key requires to try in the worst case all the n-bit possible keys. They are 2^n of such keys and therefore it is commonly accepted that $n = 128$ provide enough resistance against exhaustive searches. However, block cipher designs with practical key lengths are not "truly" random ciphers. It is nonetheless accepted that a permutation corresponding to a randomly selected key should appear to have been selected at random from the $(2^n)!$ possible bijections on 2^n elements.

Block ciphers have an iterated structure. They repeat the same function several times (with different inputs), therefore referred to as the *round function*. One advantage of an iterated structure is that it yields small program codes in software and compact circuits in hardware.

Block ciphers are usually categorized following two different designs : the Feistel structure and the substitution-permutation network (SPN) structure. We give a high level description of the latter in the following, the former being out of the scope of this thesis.

Logical operations are classified into linear operations and nonlinear operations. Also, the composition of linear operations is also linear. Nonlinear operations

break the linearity brought by the application of linear layers. The design of block ciphers consists in applying alternatively nonlinear and linear layers in the round function, which is in turn executed several times. When the design approach mixes a fixed-length input data by iteratively applying nonlinear layers called substitution layers (S-layers) and linear layers called permutation layers (P-layers), the resulting structure is said to be an SPN.

When the hardware of the device into which the cipher is implemented has limited resources as for microcontrollers, the balance of the implementation memory requirements and the computation efficiency is important. As an example, instead of computing the small nonlinear operations (typically less than 8 bits) of block ciphers, a popular approach consists in using a *substitution table* (or *S-box*), stored in the RAM segment. The table stores the results of the pre-specified mapping in memory. An 8-bit to 8-bit S-box such as the one in the *Advanced Encryption Standard* (AES) can be implemented with 256 bytes of memory and therefore accessing the table (which is quite fast in practice) replaces the computation of the mapping.

Chapter 4

Physical Attacks against Block Ciphers Implementations in Embedded Systems

There are two ways to implement cryptographic primitives into embedded systems, in software or in hardware. A software implementation is a program that is typically handled by a microcontroller. The program is stored in the ROM of the microcontroller which executes sequentially the instructions described in the source code of the program (usually in assembly language). On the contrary, a hardware implementation is a dedicated cicuit that implements an algorithm as an ASIC or with FPGA. Hardware implementations of block ciphers are out of the scope of this thesis.

When a block cipher is implemented in a device, its security has to be evaluated. In cryptography, security often refers to *computational security* which states that security is achieved if breaking it with best currently-known methods requires computing power that exceeds the computationnal abilities of an adversary by a comfortable margin.

The computational security of block ciphers implemented in a device is much harder to achieve than its inherent security making abstraction of the physical device. When abstracted, the device is considered as an opaque module from which an adversary does not gain information to perform attacks. However, embedded systems are usually present in small portable physical devices that are physically accessible by an adversary. From a physical access, an adversary can render the device less opaque in many ways and gain information about any cryptographic key being used. These attacks are called *physical attacks*.

In this chapter, we are interested in security issues of block ciphers implementations in embedded systems that arise from an adversary having a physical access to the device. We emphasize that such a scenario is highly realistic in practice. We start by giving some standard classes of attacks that abstract the device as an opaque module. Then we give more realistic classes of attacks that consider an adversary who has a physical access to the device. Finally, we give details about a subclass of physical attacks called *power analysis*.

Contents

4.1 Adversaries with Physical Access

The purpose of a block cipher is to provide confidentiality which prevents the disclosure of some information that has been encrypted. To quantify the security of block ciphers, we usually refer to more or less realistic security models. In our context, the purpose of security models is to reflect some abilities of an hypothetic adversary that can be used in order to jeopardize the confidentiality provided by a block cipher implemented in an embedded systems.

Usually, information flows in a unsecured channel which allows an adversary to access this information. If encrypted, an adversary has therefore only access to unintelligible information and retrieving the original data would require (*computational*) efforts. Depending on how much information material and the nature of the information the adversary has access to, computational efforts can be more or less substantial. Security models should encompass realistic scenarios and assert the feasibility (in terms of computational efforts and memory requirements) of breaking security. It is said that a block cipher is totally broken if a key can be found without requiring an extensive computational efforts.

To evaluate the security of block ciphers, attacks on block ciphers have been classified : *ciphertext-only* attacks rely on the knowledge of ciphertexts only ; *known-plaintext* attacks manipulate pairs of plaintexts/ciphertexts ; *chosen-plaintext* attacks consider adversaries with access to ciphertexts corresponding to chosen plaintexts. The latter are stronger, security against such attacks provide security

against the two former classes. Some other classes of attacks exist (see [52] for more details).

The previous classes do not fully reflect most modern uses of cryptography. As already explained in this introduction, a tremendous amount of mobile devices are active users of cryptography and they are physically accessible. Thefts of small portable secure elements, such as smart cards, are common and can lead to desastreous security issues. It is likely that in a second step, the thief, (here considered as an adversary), will try to break the security of the stolen device as it typically holds valuable cryptographic assets such as cryptographic keys. The bottom line is that many more problems arise when devices are physically accessible. We detail common abilities of an adversary that somehow gain access to a device.

A simple concrete example of attack via physical access is an adversary that would establish an electrical contact with the bus lines connected to the hardware modules of a microcontroller. Using micro-probe needles for estasblishing the contact, the adversary can monitor the bus signals and therefore has access to internal computations of a device. These attacks are called *probing attacks* [3, 40].

On one side, the running time of some hardware or software implementation of a cryptographic algorithm being executed by a device depends of the device hardware. The better the hardware, the faster the execution. On the other side, the running time of an implementation also depends on the implementation itself, namely, on the choices that have been made during the implementation process. It turns out that implementation choices not only have an impact on running times but also on other measurable physical characteristics, such as the instantaneous power consumption or the amount of electromagnetic radiations emitted of a running device. With an adapted equipment and physical access, an adversary can measure various implementation-dependent physical characteristics of a device.

Furthermore, other and more abstract characteristics of an implementation depend on the device. As an example, consider the *correctness* of an implementation. This property states that the result of an execution is as expected. It is usually proved under the assumption the device runs in standard conditions and without errors. Such errors can however be provoked by tampering with the device and thus break the correctness of the implementation.

In summary, physical access to an electronic device allows monitoring internal signals, measuring implementation-dependent physical characteristics and tampering

with a device. The attacks that exploit such enhanced abilities through physical access are referred to as physical attacks. Physical implementations are not inherently opaque, secure modules as assumed by the standard classes of attacks. The practical security of block ciphers cannot be asserted by assuming the adversary has no physical access to the device. Physical attacks exploiting physical accesses should be taken into considerations for asserting the practical security of block ciphers in embedded systems. This discussion extends Kerchkoff's principle from algorithms to implementations since the security of a cryptographic primitive should also not rely on the secrecy of its implementation.

4.2 Physical Attacks

The purpose of physical attacks is to extract the embedded cryptographic keys of an electronic device whether they are at rest or being used. Indeed, a portable device can also act as a secret store for cryptographic keys. If so, an adversary with physical access to an unprotected device could extract the keys stored from the memory easily. Cryptographic keys should be themselves encrypted which raises the problem of securing the one key that was used to secure the others.

Even if we assume the device implements some security measures to protect the keys at rest, this device can also be used for encryption purposes. Therefore, during encryption, the device will make use of such keys, leaving them unprotected during the time they are used. As explained above, various physical characteristics of a running device depend on the implementations. Therefore, these physical attributes would also partly depend on the key being used. This dependency can therefore be exploited by an adversary to extract the key that is being used for encryption. In what follows, we give a common classification of physical attacks.

4.2.1 A Classification.

Physical attacks are usually classified following two factors whether the adversary is considered *active* or not and whether the adversary uses *invasive* methods or not.

Passive attacks. The adversary measures physical properties of the device under control passively, namely without attempting to make it dysfunction.

Active attacks. On the contrary to passive attacks, active ones consider an adversary that intentionally tamper with the device under control in order to make it function abnormally.

Invasive attacks. Also called *hardware attacks*, they would typically try to access different hardware components of the device. This is done by physically deteriorating the package of the chip to reveal the hardware modules. Once access is gained, an adversary can monitor data signals flowing in various parts of the IC with a probing station. An adversary can also tamper with any accessible hardware component with various equipment such as laser cutters or focused ion beams. These attacks can be quite expensive to mount depending on the equipment.

Non-invasive attacks. On the contrary to invasive attacks, non-invasive are less expensive to mount. They would typically only require a measuring equipment (e.g. an oscilloscope) and only use the device interfaces that are directly accessible to perform the measurements. Such attacks are very hard to detect as well since the chip is not physically altered. They are therefore a big threat against embedded systems in general.

It is worth mentionning that we can further divide these attacks with respect to the standard classes of attack previously presented. In this thesis, we are interested into passive attacks that can be invasive such as the probing attacks previously mentionned or non-invasive such as *Side-Channel Analysis* (SCA) initiated by [42].

Side-Channel Analysis would typically be performed in two steps. At first, these attacks would typically consist in measuring some physical property of the device under control that is performing several encryptions with a fixed key. In a second step, some statistical analysis would typically be used to exploit the dependency between the secret key that has been used during encryptions and the measurements collected in order to reveal the key. Sometimes, only a single encryption is needed and no statistical analysis is required because the dependency is directly visible on the measurements. In what follows we focus on *power analysis attacks* that would exploit the instantaneous power consumption of a device. The other attacks are out of the scope of this thesis.

4.2.2 Power Analysis Attacks.

Power analysis is a subclass of physical attacks and can be divided into two different categories : *Simple Power Analysis* (SPA) and *Differential Power Analysis* (DPA) [43]. The following explains the main differences between the two.

The goal of SPA is to reveil the key without statistical analysis on a small number of measurements collected. Sometimes, a single measurement can reveil the entire key. These attacks exploit key-dependent patterns that are visually observable on

the measurements. Namely, it is sometimes possible to distinguish the sequence of instructions that have been performed within the measurement. If the sequence of instructions depend on the values of the key bits, one can retrieve these key bits by combining the power sample with the knowledge of the algorithm, which is assumed to be known.

DPA typically require a large number of measurements and always perform a post execution statistical analysis step on the power samples that have been acquired. Statistical analysis aims at creating two different statistical distributions on power samples corresponding to known plaintext/ciphertext and guesses of a few key bits manipulated at the chosen point. If these distributions can be distinguished, then the attacker can verify the key guesses. When several points in the circuits are considered, the attacks are referred to as *higher-order* DPA.

Note that DPA can be performed against software and hardware implementations. Hardware implementations are usually CMOS circuits where the overall execution break down to combining logic gates (e.g. AND, OR, NOT, etc) in a specific way, which map the input(s) logic value(s) to their corresponding output(s) logic value(s). During an execution, the logic gates switch values depending on the data that is being processed, yielding to variations of the instantaneous power consumption of the device. DPA would therefore exploit the dependency between the instantenous power consumption of the device under control and key-dependent data at a fixed point in the MOS circuit. In software, implementations break down to a sequence of instructions that are performed sequentially by the microcontroller of the device. These attacks exploit the dependency between the instantenous power consumption of the device under control and a chosen instruction executed at a fixed moment in time, processing some key-dependent data for a known plaintext/ciphertext pair.

The objective of power analysis is to measure the instantaneous power consumption of a device performing encryption. The measurement setup of the instantaneous power consumption of the device consists of a serial circuit involving a power supply, the device itself and a resistor ; A digital sampling oscilloscope connected to a computer is typically used for measuring and recording the voltage difference across the resistor and this voltage difference is proportionnal to the current by Ohm's law. The computer is also connected to the IC.

Chapter 5

Protecting Implementations Against Power Analysis and Probing Attacks

In this chapter, we address the main focus of this thesis. Namely, we describe how it is possible to protect efficiently software implementations of any block cipher against DPA and probing attacks with the well-known *masking* countermeasure introduced in [16, 33].

The problem tackled by masking can be described as follows : all data, whether they are provided to an algorithm as inputs, manipulated or created during execution by the different layers of computations have to be protected until the expected output is computed. Informally, masking is a randomizing countermeasure which protects input data by splitting them into *sharings*, tuples of several randomized pieces called *shares*, and performs all computations on sharings only, via *masked transformations*. All masked data and masked designs are parameterized by a *masking order*, *i.e.* the number of random shares present in sharings. When the masking order is considered to be arbitrarily large, the masking is said to be of *higher-order*. In this thesis we are interested in proving the security of higher-order masked designs following some security model.

The soundness of the masking countermeasure depends on the masking order that acts as a security parameter against both DPA and probing attacks. This was formalized in two security models, *i.e.* the *noisy leakage* and the *probing models* respectively. Each of the two were originally built to model one attack without considering the other, but the probing model has been proved to be also relevant for proving the resistance against DPA adversaries thanks to a security reduction that can be found in [23]. Throughout the rest of this thesis, we are interested in the higher-order security of masked designs in the probing model, *i.e.* that are

proved to ensure security for arbitrary masking orders.

In the probing model, *global security*, *i.e.* the overall security of an implementation, is achieved by proving that any set of shares whose cardinality is upper bounded by the masking order is independent of any sensitive, *i.e.* key-dependent, variables. Directly tackling this problem on all computations is not trivial. This is why a masked block cipher is usually viewed as a sequence of masked elementary field operations or transformations for which one can prove more or less strong security properties, allowing compositionnal reasonings, very useful for proving global security. The strongest security property in the probing model has been introduced in [6] and is referred to as SNI, standing for *Strong Non-Interference*. In this thesis, we aim at proving the SNI property of the masked designs we provide.

Masking schemes are intended to provide the secure masked designs of all operations and transformations of a block cipher, that have been proven to satisfy some security property in the probing model. In masking, the most of the effort is usually spent for proving the higher-order SNI property of the nonlinear masked designs. This usually requires to use additionnal randomness, either directly on some specific shares at a time or via an SNI *refreshing* procedure, which re-randomize at once all the shares of a given sharing during execution. Because randomness generation is usually costly and since the secure computation of nonlinear transformations typically involve the most randomness, the execution of SNI nonlinear masked transformations greatly contributes to the overall performances in practice, especially at high orders. Therefore, a particular attention has to be paid to optimize the secure designs of the refreshing procedure and of the nonlinear masked transformations.

In the following, we start by giving a more precise description of the masking countermeasure. We also describe a method called *the polynomial evaluation method* for building higher-order masking schemes that are generic, *i.e.* that can be applied to protect the implementations of any block ciphers. Then, we address the soundness of the masking countermeasure by giving details about the noisy leakage model and in particular about the probing model, on which we focus in this thesis for proving the security of our constructions. We also describe the security properties in the probing model that secure masked designs can achieve and that are important for proving formally the global security provided by the application of masking schemes.

Contents

5.1 Masking

The randomization of the masking countermeasure is done by applying different layers of security on top of the algorithm that has to be protected, with respect to a pre-defined and fixed masking order. A masking scheme can be divided into three layers. First, an *encoding or splitting layer* randomizes data beforehand actual computations according to the chosen masking order. Secondly, a *secure computation layer* allows to process the encoded data securely by maintaining the previous established level of randomization. Finally, a *decoding layer* allows to retrieve the underlying genuine output of the execution from its encoded counterpart.

There exist different types of masking in the literature. The type of masking is first defined at the encoding layer. It usually consists in applying a refreshing procedure on the unprotected input data. Hovewer, there also exist conversions for switching from one type of masking to another during execution. This can be helpful to process the different operations of an algorithm with a type a masking that is the most adapted. Hereafter, we start by describing two common types of masking called the *Boolean masking* and the *multiplicative masking*. We therefore also introduce the respective refreshings that can be used to achieve the splittings. Then, we address the masking of the computation layer. We illustrate the processing of encoded data through linear and nonlinear functions and also describe the polynomial evaluation method previously mentionned for masking the nonlinear layers of block ciphers. We also discuss the importance of refreshing at the computation layer for guaranteeing security throughout executions.

5.1.1 The Encoding and Decoding Layers

We start by giving some notations that are gonna be useful for the understanding of the rest of the section.

Notations. Operations of block ciphers are usually performed over finite fields

of characteristic 2. Let $\mathbb{F}_{2^k}$ denote Galois fields with 2^k elements. The masking order will be denoted by n and considered to be greater or equal to 1. Also, unprotected data are field elements and will be denoted by lowercase letters while their corresponding sharings will distinguished by using the same lowercase letters but underlined. In our context, sharings are usually considered as vectors of length $n+1$ whose coordinates are elements of $\mathbb{F}_{2^k}$. For any field element $a \in \mathbb{F}_{2^k}$, its corresponding sharing with respect to the masking order $\underline{a} \in (\mathbb{F}_{2^k})^{n+1}$ is therefore defined as $\underline{a} = (a_0, \ldots, a_n)$ where the $a_i's$ are the different shares belonging in $\mathbb{F}_{2^k}$.

Let us now start by introducing the concept of refreshing in masking, from which any unprotected data can be mapped to a sharing of size n. Note that in the context of masking, all sharings are usually defined with respect to a binary operation and such that the combination of all shares by the same binary operation would compute the inverse mapping, *i.e.* map a sharing to its genuine data. This has to hold throughout all computations and therefore operations processing sharings have to provide in result sharings whose shares can also be combined together to the retrieve the correct genuine data. Moreover, all masked operations and by extension masked transformations have to be computed in such a way that they satisfy some security property in the probing model. This is detailed later but one should keep in mind that additional constraints exist for providing masked designs when it comes to proving their security in the probing model. In the following, we are more interested in illustrating the correctness of the computations rather than addressing the specifics about the possible security properties.

Refreshing. A refreshing procedure produces a uniform sharing of the identity element e of some finite Abelian group. This is typically done computing a function on elements generated uniformly at random with the group operation. Therefore if the group $(\mathbb{F}_{2^k}, \oplus)$ is considered, a uniform sharing $\underline{z} = (z_1, \ldots, z_n)$ produced by a refreshing procedure would satisfy $\bigoplus_i z_i = 0$. On the contrary, if the binary operation is $\times$, then it would hold that $\prod_i z_i = 1$ and therefore all random elements that are generated during the refreshing have to be non-zero elements. Clearly, following this definition, the actual mapping of an unprotected field element is still not achieved. We merely produced an encoding of an identity element without taking into consideration the unprotected data that has to be splitted. The Boolean and the multiplicative mappings are addressed in the following. Sharings generated by a refreshing procedure will still be denoted by $\underline{z} = (z_1, \ldots, z_n)$ and are assumed to satisfy $z_1 * \cdots * z_n = e$ where $e = 0$ if $* = \oplus$ and $e = 1$ if $* = \times$.

Boolean masking. In this case, we consider to have a refreshing procedure that generates uniform sharings of 0. We also consider an unprotected field element a

viewed as the sharing $(a, 0, \ldots, 0)$ of size n. The secure sharing $\underline{a}$ of a is defined by

$$\underline{a} = (a \oplus z_1, 0 \oplus z_2, \ldots, 0 \oplus z_n) = (a_1, \ldots, a_n)\,.$$

The first coordinate a_1 is usually called the *masked value* as it involves the original unprotected value. Also, since $\bigoplus_i z_i = 0$, $\underline{a}$ satisfies $\bigoplus_i a_i = a \oplus \bigoplus_i z_i = a$, which enables to retrieve the original data a.

Multiplicative masking. It is similar to the Boolean masking but in this case we consider to have a refreshing procedure that produces uniform sharings of 1. The unprotected field element a is now initially viewed as the sharing $(a, 1, \ldots, 1)$ of size n. Then, the secure sharing $\underline{a}$ of a is defined as

$$\underline{a} = (a \times z_1, 1 \times z_2, \ldots, 1 \times z_n) = (a_1, \ldots, a_n)\,.$$

Since $\prod_i z_i = 1$, $\underline{a}$ satisfies $\prod_i a_i = a \times \prod_i z_i = a$, which enables to retrieve the original data a.

As previously mentionned, there also exist special transformations called *conversions* that allow to convert a Boolean masking into a multiplicative masking and conversely, at any masking order.

Switching between types of masking : the conversions. The conversions can be found in [29]. The complexities of these two conversions are quadratic in terms of the masking order. Note that the switching from a Boolean masking to a multiplicative masking has to deal with the special case where the masked value is zero, for if $a = 0$, $\prod_i a_i = 0$ in any case, which leads to security issues. This problem is referred to as the *zero value problem* (see Chapters 6 and 10 of [49]). In [29], the zero value problem was avoided by only masking non-zero values. For a masked value a that is possibly zero, the idea was to compute securely the Dirac function δ of a. It is defined as $\delta(a) = 1$ if $a = 0$ and $\delta(a) = 0$ otherwise. By doing so, we have $a + \delta(a) \neq 0$, even if $a = 0$.

In what follows, we address the masking of computations on splitted variables. We illustrate this on the Boolean masking.

5.1.2 The Computation Layer (Boolean Masking)

In our context, the computation layer ensures that the computations on sharings are performed securely with respect to the probing model. Also, the different shares have to keep track of the computations so that the final result can be retrieved and be as expected. Namely, if a function f is computed on a sharing $\underline{a}$, the resulting

sharing of $f(\underline{a})$ should satisfy $\bigoplus_{i=1}^{n} f(a_i) = f(a)$. We illustrate this hereafter for the processing of linear and nonlinear functions on sharings.

Computing linear and nonlinear functions. Consider first a function f that is a linear and a Boolean sharing of a, *i.e.* satisying $\bigoplus_i a_i = a$. Computing $f(a)$ with f being linear has the property that

$$f(a) = f(a_1 \oplus \cdots \oplus a_n) = f(a_1) \oplus \cdots \oplus f(a_n)\,.$$

Therefore, defining the encoding of $f(a)$ as $(f(a_1), \ldots, f(a_n))$ has the property that $\bigoplus_i f(a_i) = f(a)$ and is therefore a correct Boolean encoding of $f(a)$.

On the contrary, when considering f being nonlinear, the above equation does not hold anymore, *i.e.* $f(a) \neq f(a_1) \oplus \cdots \oplus f(a_n)$. Therefore, computing the sharing of $f(a)$ is less straightfoward in this case and usually requires to mixing several shares together during computations which can result in reducing the masking order. Consequently, additional randomness is usually required when designing secure designs of nonlinear operations and transformations.

S-boxes are typical examples of nonlinear transformations that have to be computed in block ciphers. In what follows, we describe the polynomial evaluation method, widely used in the literature and which provides a generic way to mask such transformations.

The Polynomial Evaluation Method. For simplicity, we only consider any S-box that are functions from $\mathbb{F}_{2^k}$ to $\mathbb{F}_{2^k}$. In [14], the authors represent any of such S-boxes by a polynomial $S(x) = \sum_{i=0}^{2^k-1} \alpha_i x^i$ over $\mathbb{F}_{2^k}$, following a well-known interpolation result. The polynomial is defined such that for any unprotected field element $a \in \mathbb{F}_{2^k}$, it holds that $S(a) = \text{S-box}(a)$. Therefore, the unprotected computation of S-box(a) for any finite field element a reduces to evaluating its corresponding polynomial at a, *i.e.* computing $S(a)$. This can be done naively by relying on the following field operations : squares and multiplications for evaluating the monomials a^i, scalar multiplications for computing the scalar products $\alpha_i a^i$ and finally additions for computing the sum $\sum_i \alpha_i a^i$.

In the context of Boolean masking, the polynomial evaluation is performed on a Boolean sharing of a. Therefore all field operations involved in the evaluation should be replaced by operations manipulating Boolean sharings only. The processing of secure nonlinear multiplications is the most expensive both in terms of randomness and of field operations. The first secure design for processing nonlinear multiplications over $\mathbb{F}_2$ with Boolean masking in the probing model was proposed

by [40] and is referred to as the ISW multiplication in the rest of the manuscript. It was later extended to $\mathbb{F}_{2^k}$ by [56] and it was further shown to satisfy the SNI property in [7]. Its complexity in terms of the masking order is $\mathcal{O}(n^2)$.

There exist different versions of the original polynomial evaluation method, optimizing the number of secure nonlinear multiplications that have to be performed during the evaluation. One of the best known-method was introduced in [21] and is commonly referred to as the CRV method.

It is worth mentionning again that only replacing unprotected operations by their secure counterparts is not enough for ensuring global security. During the processing of complex sequences of operations such as for polynomial evaluations, the randomization level applied at the splitting step must be maintained. Composing elementary secure operations can sometimes lead to security issues which can be prevented by using the refreshing transformation in between some operations to provide another fresh splitting of the data.

Since the efficiency of polynomial evaluation methods mostly relies on the processing of nonlinear multiplications, we give now more details about their processing.

The nonlinear multiplications. In Boolean masking, the multiplication of $a, b \in \mathbb{F}_{2^k}$ can be computed as

$$a \times b = \left(\bigoplus_{i=0}^{n} a_i \right) \times \left(\bigoplus_{i=0}^{n} b_i \right) = \bigoplus_{0 \leq i,j \leq n} (a_i \times b_j) \,.$$

It is important to note that a field multiplication between sharings would mix up together multiple shares with different indexes and also that the $(n+1)^2$ cross-products $a_i \times b_j$ have to be recombined into $n+1$ shares. A solution to this problem was given by the ISW multiplication previously mentionned, which prevents the mixing of the shares during computations of nonlinear multiplications to introduce security issues in Boolean masking.

Note that nonlinear multiplications between two sharings can be done more straightforwardly if the sharings are multiplicatively masked. To illustrate this, consider now that $a = \prod_i a_i$ and $b = \prod_i b_i$. In this case, it holds that

$$a \times b = \prod_i a_i \times \prod_i b_i = \prod_i (a_i \times b_i) \,.$$

Therefore, the encoded result of $c = a \times b$ can be defined as $\underline{c} = (a_1 \times b_1, \ldots, a_n \times b_n)$ for which it holds that $\prod_i a_i \times b_i = a \times b$.

Theoretically, the correct implementation of masking schemes should render the power consumption of a masked algorithm not directly dependent of the genuine data but rather dependent of all the randomized shares. The power consumption should therefore appear random on the power samples. Also, since input data are masked beforehand actual computations and their processing is protected throughout execution via secure masked operations, a probing adversary should not obtain observations of any genuine sensitive values if not able to observe all corresponding shares to retrieve them.

In what comes next, we address the soundness of masking, namely the difficulty of extracting the sensitive, *i.e.* key-dependent, variables being used by the protected implementations of block ciphers.

5.2 The Soundness of Masking

The methodology for proving global probing security has been extensively analyzed in the literature. We now have various security properties in this model, that have been proven to be very useful for deriving complete and sound security proofs, which makes the probing model very convenient for proving the soundness of the masking countermeasure. Moreover, the relationship between the probing and the noisy leakage models has been made explicit in the literature, as already explained. In particular, it has been showed that probing security implies security in the noisy model. Therefore, deriving proofs in the probing model is very convenient to formally prove the soundness of the masking countermeasure against both probing and DPA attacks.

After introducing the two models, we address the issue of proving global security in the probing model. We describe the common approach followed in the literature that first divides the overall computation into several smaller transformations, called *gadgets*, for which it is easier to first prove strong security properties. Then, global security can be achieved without much more effort by applying compositionnal reasonings between these small highly secure transformations.

5.2.1 The Noisy Leakage Model

Protecting implementations against DPA requires to accurately understand the amount of leakage present in the power samples. The leakage is the exploitable

information contained in the samples that reveals sensitive information through statistical analysis.

The power consumption of a device can be modeled more or less accurately to reflect the global electrical activity generated from various of its components. The difficulty for an adversary to retrieve the sensitive information can be determined from the power model created for the device. Namely, given the power model hypothesized for a device under attack, it is possible to determine a lower bound on the number of power samples that is required to retrieve the secrets via statistical analysis.

The noisy leakage model was first introduced to formally prove the soundness of the masking countermeasure where each original bits were splitted following the Boolean masking at any abritrary order n. In the original leakage model, the adversary was assumed to obtain noisy observations of all shares of the targeted bit. Any noisy observation of any bit share was modeled according to a Gaussian distribution centered around the actual value of the bit. Under this model, it was shown that the number of power samples required to retrieve the secret bit grows exponentially with the masking order. The noisy leakage model was further extended to more general leakage distributions in [54]. They also considered the leakage of sensitive values of any bit size.

5.2.2 The Probing Model

The probing model was introduced in 2003 by Ishai, Sahai and Wagner [40]. Contrary to the noisy leakage model, the probing model considers that an adversary can have access to a limited number of exact values during the computations. This model was first introduced to characterize an adversary who is able to position a limited number of metal needles called probes on a circuit implementing some cryptographic algorithm. It was therefore originally modeling passive invasive adversaries targetting hardware implementations.

Consider a masked hardware implementation at some order n, that is all variables are splitted into n shares during execution. Since the noise free observations of complete sharings would allow an adversary to retrieve the underlying original value, a probing attack with n needles positioned on n relevant wires would theoretically always succeed. However, successfully mounting attacks with a large number of probes can become the bottleneck of the so called higher-order probing attacks. The masking order therefore also acts as a security parameter in the probing model as it forces the adversary to mount more complex attacks.

At first, this model was not considered to be realistic for modeling DPA adversaries as practical DPA observations would typically always be noisy. Nonetheless, the relevance of the probing model for modelling DPA adversaries was made clear and therefore the soundness of the masking countermeasure in the probing model ensures security in the leakage model as well which is very convenient for deriving formal security proofs in the leakage model against DPA adversaries. In what follows, we give more details about the actual strategy for proving probing security that is usually followed in the literature.

Proving global probing security. Probing security consider implementations (hardware and software) represented as a circuit which can in turn be divided into a sequence of gadgets. The latter are probabilistic functions that return all variables manipulated or created during the computations and therefore provide all material that an adversary can have access to.

In the probing model, the purpose is to prove that the complete execution of the sequence of gadgets satisfies the so-called global probing security property. Probing security with respect to the masking order n states that any set of $t < n$ intermediate variables do not reveal any information on the secrets. In other words, any view of an adversary probing up to n wires can be perfectly simulated without knowing the secret variables from the original circuit. The proofs in the probing model therefore aim at showing that any set of t intermediate variables can be simulated from stricly less input shares than the masking order n. It is possible to prove that any subset of at most $n-1$ input shares is uniformly and independently distributed, this means that the adversary learns no sensitive information from the probes.

For proving global probing security of a complete circuit/implementation, stronger properties than probing security can be first proved locally for small gadgets as it is more convenient to reason on a small number of variables. Then, the different gadgets that were proven to be secure under stronger notions can be composed securely to achieve probing security on complete sequences of gadgets. In particular, safe composition can be achieved via the SNI property. There also exist other useful properties such as Non-Interference (NI) and affine properties introduced in [6].

The SNI property implies the affine property and the NI property, which in turn implies probing security. Therefore, the SNI property is the strongest notion in the standard probing model.

5.2.3 Security Definitions in the Probing Model

In the probing model, proofs are based on simulation. Namely, if any adversary observation set (*i.e.* set of probed wires) can be simulated without the knowledge of any input variable then the probes are of no use to an attacker. We remind several security definitions introduced in [6] that are useful to prove the security of a construction in the probing model under the stronger SNI security definition.

An adversary can probe input wires, internal wires or output wires. In what follows, we denote an adversary observation set by Ω and we divide it into two sets $\mathcal{I}$ and $\mathcal{O}$ such that $\mathcal{I}$ is the set of input or internal probed wires while $\mathcal{O}$ is the set of output probed wires. For any set $\Omega = (\mathcal{I}, \mathcal{O})$ of at most $n-1$ probed wires, it is obvious that $|\mathcal{I}| + |\mathcal{O}| \leq n-1$.

The following security definitions rely on whether or not it is possible to simulate Ω. Namely, if Ω can be perfectly simulated without knowledge of any input variable then the $t < n$ probes used by the attacker to build Ω are not dependent on any secret. Indeed, an input variable is a sharing of size n generated such that the knowledge of $n-1$ of its shares does not reveal the original data. Thus, as long as the simulation of Ω only requires strictly less than n shares of each input variable, then Ω can be simulated without knowing any secret. Consequently the $t < n$ probes reveal nothing to the attacker.

The set of input shares that are required for simulating an adversary set of probed wires is denoted by $\mathcal{S}$ in what follows. The latter also indicates which specific shares (*i.e.* which wires) are considered for each input. The upper bounds on the cardinality of $\mathcal{S}$ lead to more or less strong security definitions of [6] which are reminded hereafter. For simplicity we consider a gadget taking as input a single sharing x of size n and that outputs a single sharing y of size n as well.

NI security. Let G be a gadget which takes as input a sharing $(x_1, \ldots, x_n)$ of x and outputs a sharing $(y_1, \ldots, y_n)$ of y. The gadget G is said to be NI secure if for every adversary set of t probed wires $\Omega = (\mathcal{I}, \mathcal{O})$ with $t < n$, there exists a set $\mathcal{S}$ of input shares such that $|\mathcal{S}| \leq t$ and $\mathcal{S}$ is sufficient to simulate the adversary observation set Ω on G.

affine-NI security. Let G be a gadget which takes as input a sharing $(x_1, \ldots, x_n)$ of x and outputs a sharing $(y_1, \ldots, y_n)$ of y. The gadget G is said to be affine-NI secure if for every adversary set of t probed wires $\Omega = (\mathcal{I}, \mathcal{O})$ with $t < n$, there exists a set $\mathcal{S}$ of input shares such that $|\mathcal{S}| \leq |\mathcal{I}| + |\mathcal{O}|$ and $\mathcal{S}$ is sufficient to simulate the adversary observation set Ω on G.

SNI security. Let G be a gadget which takes as input a sharing $(x_1, \ldots, x_n)$ of x and outputs a sharing $(y_1, \ldots, y_n)$ of y. The gadget G is SNI secure if for every adversary set of t probed wires $\Omega = (\mathcal{I}, \mathcal{O})$ with $t < n$, there exists a set $\mathcal{S}$ of input shares such that $|\mathcal{S}| \leq |\mathcal{I}|$ and $\mathcal{S}$ is sufficient to simulate the adversary observation set Ω on G.

SNI security (binary gadgets). Let G be a gadget which takes as inputs a sharing $(x_1, \ldots, x_n)$ of x, a sharing $(y_1, \ldots, y_n)$ of y, and outputs a sharing $(z_1, \ldots, z_n)$ of z. The gadget G is said to be SNI secure if for every adversary set of t probed wires $\Omega = (\mathcal{I}, \mathcal{O})$ with $t < n$, there exist sets $\mathcal{S}^1$ of input shares of x and $\mathcal{S}^2$ of input shares of y such that $|\mathcal{S}^1| \leq |\mathcal{I}|, |\mathcal{S}^2| \leq |\mathcal{I}|$ and $\mathcal{S}^1 \cup \mathcal{S}^2$ is sufficient to simulate the adversary observation set Ω on G.

The probing model is commonly used to address the security of serial implementations. Regarding the security of parallel implementations, the probing model no longer reflects accurately practical observations. In particular, several shares can be manipulated at once during the computations and therefore a single observation leaks more information of these shares. The probing model has been re-interpretated to capture parallel settings which has led to the *bounded moment leakage model* [8]. A convenient theorem in the latter article (see Theorem 1) states that serial security implies parallel security at the same masking order, i.e. that the parallel counterpart of some SNI secure serial algorithm is in fact secure in the bounded moment leakage model.

Chapter 6

Contributions of this Thesis

In the following, we give a high-level description of the contributions of this thesis. All the results presented in this thesis were found and submitted or already published with Michaël Quisquater. The specifics regarding practical performances and the comparisons of the complexities with the state of the art are addressed at the beginning of the different parts of the rest of the manuscript.

Our research mainly focuses on improving the masking countermeasure in the probing model for serial software implementations of block ciphers against DPA and probing attacks. In the previous chapter, we exhibited the main most common components of some existing masking schemes, *i.e.* the refreshing procedure, the polynomial evaluation method and a secure optimized design for nonlinear multiplications. In this thesis, we will detail our results for two out of these three main components of masking schemes.

The first part provides a class of recursive refreshing schemes in $\mathcal{O}(n \cdot \log_2 n)$ for serial implementations that are secure at any masking order n - the security parameter of the masking countermeasure - in the probing model under Strong-Non-Interference (SNI) property. Moreover, our new scheme improves the time and randomness complexities of the state of the art by a factor of about 2. The class of algorithm we provide in this setting are defined for any finite Abelian group while the state of the art is usually more restrictive. We present a method that allows to convert recursive algorithms such as our SNI recursive schemes into computationally-wise equivalent iterative execution and provide the resulting SNI iterative refreshing algorithm. Implementing the latter is particularly relevant for an architecture which has a fast instruction for computing the number of leading zeros of its words, e.g. the CLZ instruction of 32-bit ARM architectures. Finally, our serial SNI refreshing schemes are converted into a bounded moment secure parallel construction, particularized for vector spaces over $\mathbb{F}_2$. Our parallel algorithm

requires a number of iterations and random vectors which is logarithmic in the masking order while their number was linear in this parameter for the best known method so far.

The second part deals with the polynomial evaluation method of the masking countermeasure. We have extended the masking scheme called GPQ [27, 28, 29] that mixes Boolean and multiplicative maskings. The latter was originally only proven to satisfy the NI security property and was only dedicated to the masking of the well-known AES block cipher. We first prove that the original GPQ actually satisfies the stronger SNI security property and we then extend it by using the conversions provided by GPQ to perform some well-chosen parts of polynomial evaluations using multiplicative masking instead of only the Boolean one. Following our approach, the ISW multiplications are not required for executing securely the nonlinear multiplications of well-chosen sections of polynomial evaluations that manipulate multiplicatively masked sharings. This leads to an efficient way to evaluate securely large polynomials over finite fields in the probing model at any masking order. This was enough to provide an extended masking scheme of GPQ for the masking of any block cipher at any masking order, and not only the AES as it was the case originally. However, by itself, our extended version of GPQ is not the most efficient in practice. The CRV polynomial evaluation method, known as the best polynomial evaluation method at the time, required to evaluate several polynomials over binary finite fields. We also use appropriately our extended version of GPQ inside the CRV polynomial evaluation method to evaluate some of its polynomials efficiently, therefore improving its performances. We also derive new sets of parameters for the CRV method that were originally not considered a relevant. Consequently, by extending GPQ to evaluate polynomials over binary finite fields and by combining the extended GPQ masking scheme with the best known polynomial evaluation method, CRV, we provide the most efficient way to mask any block cipher based on polynomial evaluation methods. We also analyze the security of our different constructions for proving they all satisfy the strongest security property in the probing model, *i.e.* the SNI property. Our results are implemented in assembly language for an 8051 based 8-bit architecture with bit-adressable memory. These results are published in the journal Transactions on Cryptographic Hardware and Embedded Systems (TCHES) and were presented at Conference on Cryptographic Hardware and Embedded Systems (CHES) in 2018 [50].

The third and last part of the manuscript presents a fast transform over finite fields of characteristic p. More precisely, it tackles the problem of evaluating/interpolating polynomials written in a particular basis, referred to as the LCH-basis in the fol-

lowing. While the original transform for evaluating the polynomials was presented over $\mathbb{F}_{p^k}$, the problem of interpolation was left unsolved for finite fields of characteristic p [45, 47]. It was only presented for the characteristic 2. We solved the interpolation problem for polynomials written in the LCH-basis for the characteristic p. We also improved the transform for the evaluation problem over $\mathbb{F}_{p^k}$. Our improvement consists in using a matrix representation for the evaluation and interpolation problems using the LCH-basis from which we derive factorized expressions that can be evaluated very efficiently with well-known algorithms for standard multipoint evaluations/interpolations of polynomials in the standard basis [26]. Our approach led to an article presented at the online version of the International Symposium on Symbolic and Algebraic Computation (ISSAC) in 2020 [51]. We also provide the source code, written in C using the FLINT library [38], for the multipoint evaluation/interpolation of polynomials in the LCH-basis over $\mathbb{F}_{p^k}$, on GitHub, at `https://github.com/axelmma/Eval_Interpol_LCH_poly`.

Part II

Refreshing Schemes for Small and Large Masking Orders

Chapter 1

Introduction

In the context of masking, a so-called *masked algorithm* is usually viewed as a sequence of masked elementary operations called *gadgets* that process sharings and which replace the unprotected counterparts of the original algorithm at execution. The security of masking schemes for serial implementations is usually analyzed in the *probing model* [40]. This model assumes that an adversary is only able to observe stricly less variables than the masking order from the complete execution of the sequence of gagdets. In this context, a gadget is a probabilistic function that return all variables manipulated or created during the computations and therefore provide all material that an adversary can have access to.

The goal in the probing model is to prove the *global probing security*, which refers to the security of the complete masked algorithm for serial implementations. Proving global security is however tricky to tackle at once. A common approach for proving global security relies on the thorough analysis of elementary but important gagdets, for which we can prove strong security properties. The strongest security property in the probing model is the *Strong Non-Interference* (SNI for short) that has been introduced in [6]. It is very useful since only SNI gagdets can be composed securely without having resort to other security mecanisms for proving global security at any masking order. This essentially transfers the difficulty of proving global security in the probing model to proving the SNI property of some well-chosen gadgets at any masking order, which however comes at a cost. Compared to less secure gadgets, the existing SNI gadgets typically make use of more randomness, which generation usually greatly contributes to the overhead of the masking countermeasure in practice [34, 41].

Also, when it comes to mask the nonlinear transformations of block ciphers, the sharings are manipulated in complex ways which may also induce flaws. The

security of masking schemes usually relies on a *refreshing* procedure providing new uniform sharings of sensitive variables and which is typically applied in between various elementary operations at several points during execution. However, a refreshing scheme which is not SNI does not necessarily compose securely with other SNI operations [20] and therefore fails to achieve its purpose. This is why the SNI property is particularly relevant for refreshing schemes. In this paper, we tackle the problem of designing serial refreshing procedures with low time and randomness complexities over any finite Abelian group and that is SNI for any masking order. More precisely, we start with the design of a class recursive refreshing schemes for which we aim to prove their security and then tackle the problem of converting such schemes into iterative ones with the same security.

In the next chapter, we also tackle the problem of designing a parallel refreshing algorithm. The security in this context is however no longer reflected accurately by the probing model. This is due to the fact that several shares, possibly an entire sharing, can be contained into a single register and therefore a single observation leaks information on all these shares. Hence, the probing model has been re-interpreted to capture parallel settings which has led to the *bounded moment leakage model* [8]. A convenient theorem in the latter article (see Theorem 1) states that serial security implies parallel security at the same masking order, *i.e.* that the parallel counterpart of some SNI secure serial algorithm is in fact secure in the bounded moment leakage model.

1.1 Related Works.

The following complexities are given in terms of the masking order n. Regarding SNI refreshing algorithms for serial implementations, [23] presents an algorithm based on the so-called ISW secure multiplication [40] and has complexity $\mathcal{O}(n^2)$. Its SNI security proof can be found in [6]. Also, [11] provides an algorithm with time and randomness complexities $\mathcal{O}(n \cdot \log_2 n)$. To the best of our knowledge, this was so far the best known generic SNI refreshing procedure in the serial setting[1].

To the best of our knowledge, there exist only two parallel refreshing algorithms in the bounded moment model that are based on rotations : one that is presented in [8] and its improved parameterized version that can be found in [9]. The generic security of the first algorithm is adressed in [9]. The improved algorithm is not

[1]Note that there exists a $\mathcal{O}(n)$ refreshing construction in the serial setting based on expander graphs [4]. However, this refreshing scheme is not proved to be SNI and therefore cannot be used for composability purposes. So far, no practical construction of this scheme has been exhibited and proved to be efficient in practice.

generic but proved secure for masking orders up to 20 in the same article. They both have linear time and randomness complexities considering that any complete sharing fits inside a register of the underlying architecture. Finally, the improved algorithm also offers an enhanced security against multivariate attacks, *i.e. horizontal side-channel attacks* [10] due to the reduced usage of the sensitive variable during its refreshing.

1.2 Contributions.

This next chapter first presents a new generic, *i.e.* secure at any masking order, SNI recursive refreshing transformation in $\mathcal{O}(n \cdot \log_2 n)$ that improves the time and randomness complexities of the recursive refreshing algorithm RefreshMasks presented in [11] by a factor of about 2. The latter algorithm requires two layers of randomness for each level of the recursion but thanks to a thorough analysis, we prove that only a well-chosen single layer is sufficient, provided also that a modification concerning the insertion of the random values is made. We show that our algorithm benefits of some degree of freedom concerning the insertion of the random values, which is valuable for deriving an efficient parallel algorithm. We prove that our method is SNI secure at any masking order, over any finite Abelian group, while the RefreshMasks algorithm of [11] (algorithm 7) is only SNI secure over $\mathbb{F}_{2^k}$. We also develop a computational method for converting recursive designs such as our recursive refreshing schemes into iterative ones. Our method allows to determine the recursive call parameters "on-the-fly" during an iterative execution, without any additional memory.

Based on our results for the serial case, we then convert our generic SNI recursive algorithm into its parallel counterpart over $\mathbb{F}_2^{\beta n}$, which by theorem 1 of [8] benefits generic parallel security for any masking order under the bounded moment leakage model. One particular step of the resulting parallel algorithm can be performed more efficiently thanks to the degree of freedom that is provided by our SNI recursive algorithm. Since the sensitive variable is used in a similar fashion as [9] during its refreshing, our parallel algorithm also benefits of an enhanced resistance against horizontal side-channel attacks. Our parallel algorithm is presented in the same context as the two other existing parallel refreshing schemes. Our proposal requires $\lfloor \log_2 n \rfloor$ iterations and $\lceil \lceil \log_2 n \rceil / 2 \rceil$ random vectors while these values for the previous generic scheme [8] were both $\lceil n - 1 \rceil / 3$.

In summary, we provide the most efficient refreshing schemes, central basic blocks of any masking scheme, in the contexts of serial and parallel implementations. The security of our serial and parallel schemes are proven under the relevant security

models, for any masking order. The serial case is analyzed under the probing model for which we prove the SNI property, strongest security notion in this context, which allows safe compositions with other masked operations, very important feature for proving the global probing security of any masking scheme. The parallel case is secure under the bounded moment leakage model.

Chapter 2

Serial and Parallel Higher-Order Refreshing Schemes with Low Complexity.

This chapter is organized as follows. Section 1 starts by introducing useful notations and some basics around refreshing. In particular, we give a definition of the SNI property for a refreshing gadget, on which we base one of the main result of this article. It also presents the original scheme of [11] and describes our scheme that improves the time and randomness complexities of [11] by a factor of about 2. In section 2 we conduct the actual analysis of our scheme in order to prove its SNI property. Then, section 3 describes the conversion of our recursive scheme into an SNI iterative refreshing. The secure parallel counterpart in the Bounded Moment Leakage model of our serial refreshing scheme is given in section 4. Eventually, section 5 concludes the chapter.

Contents

2.1 New Serial Recursive Refreshing Scheme with Improved Complexity.

In this section, the preliminaries start by introducing useful notations that we use throughout the article. We also included some basics around the concept of refreshing in the masking context, a description of the SNI property for a refreshing gadget, along with a formal definition.

After the preliminaries, we recall the SNI recursive refreshing scheme with complexity $\mathcal{O}(n \cdot \log_2 n)$ described in [11] (algorithm 7) and present our recursive refreshing algorithm. The section is concluded on a description of the differences between our approach and the original work of [11].

2.1.1 Preliminaries.

Notations. Let $(G, +)$ be any finite Abelian group. Let n with $n \geq 2$ denote the order of the masking. Vectors will be denoted by underlined letters. For a given vector $\underline{x} \in G^n$, we also represent it by its coordinates as the tuple $(x_1, \ldots, x_n)$. To specify one of the coordinates, we indifferently write x_i (the coordinate indexed by i) or $\underline{x}_i$ (the i^{th} coordinate of the vector $\underline{x}$). A sharing of a sensitive value x is a vector $\underline{x} = (x_1, \ldots, x_n)$ for which it holds that $\sum_{i=1}^{n} x_i = x$. Also, $r \xleftarrow{\$} G$ means the element r is drawn uniformly at random over G. Random variables are denoted by capital letters while the use of lowercase letters denote particular values. The probability of the particular event $\{X = x\}$ shall be written $\mathrm{P}[X = x]$. Moreover, when the random variable X follows a uniform distribution over a set S - if for any $x \in S, \mathrm{P}[X = x] = 1/\#S$ - we write $X \sim \mathcal{U}(S)$.

Additive splitting of sensitive data via a refreshing procedure. A common approach for splitting the sensitive data is to make use of the so-called *additive* encoding which indicates that data are encoded over $(G, +)$. More precisely, for a masking order that is equal to n, the additive encoding of some sensitive variable $x \in G$ is a sharing $\underline{x} = (x_1, \ldots, x_n) \in G^n$ satisfying $\sum_{i=1}^{n} x_i = x$. The splitting can be done thanks to a refreshing procedure, this is detailed hereafter.

Consider an unprotected sensitive variable $x \in G$ viewed as the sharing $\underline{x} = (x, 0, \ldots, 0)$ of size n. The secure splitting of x can be done by relying on a refreshing procedure, applied on $\underline{x}$ as defined just above. A natural and common way to perform a refreshing of the sharing $\underline{x}$ of size n is to add a "fresh" random sharing of 0, *i.e.* $\underline{y} = (y_1, \ldots, y_n)$, produced by the refreshing procedure and which satifies $\sum_{i=1}^{n} y_i = 0$, on top of $\underline{x}$. Once the encoding $\underline{y}$ of 0 is generated by the refreshing procedure, the splitting is achieved by simply computing the sharing

$$\underline{z} = (x + y_1, 0 + y_2, \ldots, 0 + y_n),$$

which securely encodes x with random elements. Also, because $\sum_{i=1}^{n} y_i = 0$, it holds that $x + \sum_{i=1}^{n} + y_i = x$, and therefore the sharing $(x + y_1, y_2, \ldots, y_n)$ is an additive sharing of x.

Re-encoding of data via a refreshing procedure. Once encoded, data are then securely processed throughout a masked algorithm via such sharings. The sharings may however still leak information about the original sensitive data they encode when they are being processed through the masked operations. It is therefore sometimes required for security reasons to securely re-encode some sharing with a refreshing scheme in between masked operations, at different points during execution.

As already explained in the introduction, it is important for refreshing procedures to satisfy the SNI property in the probing model. The main results of this article, *i.e.* the SNI property of our refreshing scheme with optimized complexity for serial implementations and its parallel counterpart are based on the following definition.

The SNI property of a refreshing gadget. Consider a gagdet G with a single sharing as input and which produces a single sharing as output as it is the case for a refreshing gadget. In the probing model, an adversary who targets a gadget is allowed to build vectors of input, intermediate and output shares during a given execution. The number of observations is however limited by the pre-determined masking order n. The goal of a probing adversary for breaking the security of

a gadget is to build some sharing with at most $n-1$ shares and show that its distribution can be simulated with more input shares than a specified bound. Such a bound varies depending on the level of security one wishes to achieve for a gadget.

The SNI property gives the tightest bound on the number of input shares from which the distribution of any sharing an adversary can build must be simulated. More precisely, for any possible adversary sharing $\underline{v}$ that can be constructed during an execution of the gadget G, consider its distribution. Simulating such a distribution must be done with least input shares of G than the number of input and intermediate shares that have been observed by an adversary to build $\underline{v}$. In particular, for any simulation of any $\underline{v}$, the bound on the number of input shares of G does not depend on the number of output shares of $\underline{v}$. This is expressed more formally hereafter.

Let $\underline{x}$ (resp. $\underline{z}$) be the input (resp. output) sharing of size n of some refreshing procedure. Also, let $\underline{y}$ denote the vector composed of any other shares that are produced during the refreshing and that are not part of the input or the output sharings. Consider the subsets I, J, K of indexes of coordinates of input, intermediate and output vectors respectively, such that $\#I + \#J + \#K < n$. These subsets are used to indicate which shares are observed by an adversary. The transformation $\underline{z} = \text{refreshing}(\underline{x})$ is SNI if for any such I, J, K, the distribution of the associated adversary sharing $(\underline{x}_I, \underline{y}_J, \underline{z}_K)$ only depends on at most $\#I + \#J$ shares of $\underline{x}$.

In what follows, we recall the original refreshing scheme of [11] with complexity $\mathcal{O}(n \cdot \log_2 n)$ and we then describe our algorithm.

2.1.2 Original Refreshing Scheme and Our Approach.

We start by describing the RefreshMasks algorithm of [11]. This recursive algorithm is SNI. It takes as input a sharing of size n, *i.e.* $\underline{x} = (x_1, \ldots, x_n)$, and recursively splits its input sharing into the two sharings $(x_1, \ldots, x_{\lfloor n/2 \rfloor})$ and $(x_{\lfloor n/2 \rfloor + 1}, \ldots, x_n)$. Also, before and after each recursive call, a freshly generated random encoding of 0 is added to the input sharing $(x_1, \ldots, x_n)$. For the sake of clarity, the algorithm is hereafter presented over $(\mathbb{F}_{2^k}, +)$ as in the original paper.

It can be seen that the input $\underline{x}$ is successively modified throughout computations, mixed at each recursive call with elements generated uniformly at random over $\mathbb{F}_{2^k}$. Also, it can be noted that for each recursive call there correspond two so-called *linear layers* of randomness, a *pre-processing layer* and a *post-processing*

Algorithm 1 RefreshMasks [11] (alg. 7)

Require: $\underline{x} = (x_1, \ldots, x_n)$
Ensure: $\underline{y} = (y_1, \ldots, y_n)$ such that $\sum_{i=1}^{n} y_i = \sum_{i=1}^{n} x_i$

1: **if** $n = 1$ **then**
2: **return** (x_1)
3: **end if**
4: **if** $n = 2$ **then**
5: $r \xleftarrow{\$} \mathbb{F}_{2^k}$
6: **return** $(x_1 + r, x_2 + r)$
7: **end if**
*** Pre-processing layer ***
8: **for** $i = 1$ **to** $\lfloor n/2 \rfloor$ **do**
9: $r_i \xleftarrow{\$} \mathbb{F}_{2^k}$
10: $x_i \leftarrow x_i + r_i$
11: $x_{\lfloor n/2 \rfloor + i} \leftarrow x_{\lfloor n/2 \rfloor + i} + r_i$
12: **end for**
13: $(x_1, \ldots, x_{\lfloor n/2 \rfloor}) \leftarrow \text{RefreshMasks}(x_1, \ldots, x_{\lfloor n/2 \rfloor})$
14: $(x_{\lfloor n/2 \rfloor + 1}, \ldots, x_n) \leftarrow \text{RefreshMasks}(x_{\lfloor n/2 \rfloor + 1}, \ldots, x_n)$
*** Post-processing layer ***
15: **for** $i = 1$ **to** $\lfloor n/2 \rfloor$ **do**
16: $r_i \xleftarrow{\$} \mathbb{F}_{2^k}$
17: $y_i \leftarrow x_i + r_i$
18: $y_{\lfloor n/2 \rfloor + i} \leftarrow x_{\lfloor n/2 \rfloor + i} + r_i$
19: **end for**
20: **return** $(y_1, \ldots y_n)$

layer, respectively computed in steps 8 to 12 and 15 to 19 of the algorithm.

From double-layers to single-layers designs. In the original work [11], the authors claimed (see Remark 3) that neither any of the pre-processing layers or any of the post-processing layers could be removed. Removing even a single layer would break the SNI assumptions of their scheme.

In this article, we show otherwise by presenting a design with single layers of randomness. More precisely, we are able to construct a SNI refreshing scheme without any pre-processing layer of randomness provided that some well-chosen modifications of [11] are made (see below).

The double-layers approach of [11] using two identical single layers at every recursive step allows to make global arguments for proving the security of the scheme. This is due to the fact that half of the layers protects the other half. On the contrary, proving the security of our scheme with single layers only, is less straightforward. In our case, every computations have to be analyzed precisely due to the use of only half of the randomness compared to [11].

This is why, while the security proof of [11] is relatively short, a very thorough analysis is required in our case which leads to a technical proof.
Note that our schemes with single layers reduces the randomness complexity of the original scheme with double layers by a factor of about 2.

Let us now give an algorithmic description of our scheme. Also, remember that our scheme is described for any Abelian group $(G, +)$ while the original approach only deals with computations over $(\mathbb{F}_{2^k}, +)$.

A two-step approach. Let us consider any finite Abelian group $(G, +)$. Consider a sharing $\underline{x}$ of some sensitive variable $x \in G$ with respect to the masking order n. It therefore holds $\sum_{i=1}^{n} \underline{x}_i = x$. Also consider a sharing $\underline{y}$ of zero for which it holds $\sum_{i=1}^{n} \underline{y}_i = 0$ by definition and that has been generated uniformly at random. Our SNI refreshing procedure first consists in generating such an encoding of zero recursively by the algorithm 2 below and then simply in adding it to the input sharing $\underline{x}$ by algorithm 3 below. Note that the resulting vector still encodes the sensitive value x since $\sum_{i=1}^{n} \underline{x}_i + \underline{y}_i = x$.

Algorithm 2 $\mathcal{R}_n$

Require: A sharing $\underline{y}$ of size n.
Ensure: A random sharing $\underline{y}$ of 0.

**** Linear layers for $n = 2$ and $n = 3$ ****
if $n = 2$ **then**
  $r \xleftarrow{\$} G$
  return $(y_1 + r, y_2 - r)$
end if
if $n = 3$ **then**
  $(r_1, r_2) \xleftarrow{\$} G \times G$
  $y_1 \leftarrow y_1 + r_1$
  $y_2 \leftarrow y_2 - r_1$
  return $(y_1, y_2 + r_2, y_3 - r_2)$
end if
$(y_1, \ldots, y_{\lfloor n/2 \rfloor}) \leftarrow \mathcal{R}_{\lfloor n/2 \rfloor}\big(y_1, \ldots, y_{\lfloor n/2 \rfloor}\big)$
$(y_{\lfloor n/2 \rfloor+1}, \ldots, y_n) \leftarrow \mathcal{R}_{\lceil n/2 \rceil}\big(y_{\lfloor n/2 \rfloor+1}, \ldots, y_n\big)$
**** Linear layers for $n \geq 4$ ****
$select$[1] $= 0$ **or** 1
for $i = 1$ **to** $\lfloor n/2 \rfloor$ **do**
  $r_i \xleftarrow{\$} G$
  $y_i \leftarrow y_i + r_i$
  if $select = 0$ **then**
    $y_{\lfloor n/2 \rfloor+i} \leftarrow y_{\lfloor n/2 \rfloor+i} - r_i$
  else
    $y_{\lceil n/2 \rceil+i} \leftarrow y_{\lceil n/2 \rceil+i} - r_i$
  end if
end for
return $\underline{y}$

The $\mathcal{R}_n$ algorithm described above produces an additive encoding of 0. The following algorithm simply consists in doing the actual refreshing, *i.e.* , in adding the freshly generated encoding of 0 onto the sharing of the sensitive data.

[0] At each call, *select* can be indifferently set to 0 or 1 which is the degree of freedom that will allow us to derive an efficient parallel algorithm in section 2.4.

Algorithm 3 Refreshing

Require: A sharing $\underline{x}$ of size n.
Ensure: A sharing $\underline{z}$ of size n such that $\sum_{i=1}^{n} z_i = \sum_{i=1}^{n} x_i$

$\underline{y} \leftarrow 0$
$\underline{y} \leftarrow \mathcal{R}_n(\underline{y})$
1: **for** $i = 1$ **to** n **do**
$z_i \leftarrow x_i + y_i$
2: **end for**
return $\underline{z}$

Modifications. The original recursive refreshing procedure (algorithm 7 of [10]) combines the two above algorithms with different terminal conditions. Its final linear layers are given for $n = 2$ and $n = 1$ which breaks down the recursion a bit further than us. Also, for any of its linear layers, each random element is directly added on a sensitive share while we proceed in two steps. While the original method makes use of two identical linear layers of randomness for any $n \geq 3$, before and after each recursive call, our algorithm $\mathcal{R}_n$ only makes use of a post-processing linear layer of randomness which allows us to reduce the time and randomness complexities by a factor of about 2. Also, our linear layer computed at steps 14 to 22 of the $\mathcal{R}_n$ algorithm provides a degree of freedom (see steps 18 and 20). This degree of freedom is important for the parallel case as it allows us to derive an efficient parallel algorithm. This is detailed in section 4.

2.2 Theoretical Analysis and SNI Property of Our Scheme.

In this section, we conduct the actual analysis of our refreshing scheme in the serial setting. Following the two-step approach of our refreshing scheme previously described, our analysis is also divided into two main steps. The first part deals with the analysis of algorithm 2 ($\mathcal{R}_n$) and the second part deals with algorithm 3 (Refreshing).

The analysis of $\mathcal{R}_n$ is conducted by induction. It follows the recursive definition of algorithm 2. We first analyze the recursive step which corresponds to the merging of two smaller schemes via a single linear layer of randomness (steps 14 to 21 of the algorithm). For the sake of clarity, consider $\mathcal{R}_9((y_1, \ldots, y_9))$. Also consider the outputs of the two smaller recursive calls, *i.e.* $(y_1, \ldots, y_4) \leftarrow \mathcal{R}_4((y_1, \ldots, y_4))$ and $(y_5, \ldots, y_9) \leftarrow \mathcal{R}_5((y_5, \ldots, y_9))$. One merging that can occur during execution

in this example is the computation of $(y_1 + r_1, \ldots, y_4 + r_4, y_5 - r_1, \ldots, y_8 - r_4, y_9)$, where the r_i's are variables generated uniformly at random. The latter computation consists in applying the layer of randomness (for the variable *select* set to 0).

During such mergings, we analyze the shapes of distributions of vectors of all possible adversary observations. Note that the linear layers of the $\mathcal{R}_n$ algorithm provide a degree of freedom regarding the insertion of random elements (for $select = 0$ or $select = 1$). We therefore derived two results,*i.e.* Lemma 2.2.1 and Lemma 2.2.2 which respectively deals with the first type of insertion and the second one during the merging. The general result for $\mathcal{R}_n$ is given in theorem 2.2.3. It proves by induction an inequality related to $\mathcal{R}_n$.

Based on this first part of our analysis on the $\mathcal{R}_n$ algorithm, the second part consists in proving the SNI property of our refreshing scheme given by our Refreshing algorithm. This is the purpose of theorem 2.2.4. Remembering that $\mathcal{R}_n$ is a subroutine of algorithm 3, the latter theorem makes use of the inequality related to $\mathcal{R}_n$ we proved in the first part.

In what follows, we start by the preliminaries in which we define all the symbols and some results that are used throughout our analysis. After the preliminaries, we conduct the actual two-step analysis as described above. The first step of the analysis deals with $\mathcal{R}_n$ and is divided into two parts, *i.e.* its recursive step and then its complete execution. The second step of the analysis proves the SNI property of the Refreshing algorithm, thanks to the results we proved for its subroutine $\mathcal{R}_n$. Note that this section is written for any finite abelian group $(G, +)$.

2.2.1 Preliminaries

Our analysis is conducted on random variables and by extension on vectors of random variables that represent all possible sharings that can be obtained over any finite Abelian group with additive law. The definitions and propositions we provide hereafter are useful to formalize the effect of the different operations on the distribution of vectors performed during the execution of our scheme. These operations basically correspond to adding or substracting random elements onto some specific coordinates. Also, in order to analyze such effects on distributions over the relevant spaces, we also describe the spaces that correspond to the sets of secrets that can be obtained by summing up some specified coordinates of some vectors.

The last result of the preliminaries, *i.e.* Proposition 7, is an abstract result allowing

to prove Lemma 2.2.1.

Operators on vectors. The following definitions are useful to shape some vectors over G^n appropriately either by fixing some coordinates to zero or setting a component of a null vector to a particular value, by selecting a subset of coordinates only or by setting a component of a null vector to a particular value. We start by defining an application for fixing some coordinates of a vector to zero.

Definition 1 (Projection)**.** *For any finite Abelian group G and any subset $V \subseteq \{1, \ldots, n\}$, the application $\pi_V : G^n \mapsto G^n$ is defined for any $\underline{x} = (x_1, \ldots, x_n) \in G^n$ by*

$$\left(\pi_V(\underline{x})\right)_i = x_i \text{ if } i \in V \text{ and } \left(\pi_V(\underline{x})\right)_i = 0 \text{ otherwise.}$$

We now introduce a notation for the selection of some coordinates of a vector.

Definition 2 (Shrinking)**.** *Consider the subset $V \subseteq \{1, \ldots, n\}$ such that $\#V = k$. If $\underline{x} = (x_1, \ldots, x_n) \in G^n$ then we define*

$$\underline{x}_V \triangleq (x_{i_1}, \ldots, x_{i_k}) \text{ with } i_j \in V \text{ and } i_s < i_t \text{ if and only if } s < t.$$

Finally, let us define a notation to set a component of a null vector to a particular value.

Definition 3 (Translation of an element)**.** *For any $s \in \{1, \ldots, n\}$, the application $\tau_s^n : G \mapsto G^n$ is defined for any $r \in G$ by*

$$\left(\tau_s^n(r)\right)_i = r \text{ if } i = s \text{ and } \left(\tau_s^n(r)\right)_i = 0 \text{ otherwise.}$$

Vector spaces and random vectors. We start by defining some spaces and in particular the spaces over which the different vectors we analyze are distributed. Let $S, T \subseteq G^n$. In what follows, the set $\{T(s) \mid s \in S\}$ is denoted by $\mathsf{Im}\, T$ or indifferently by $T(S)$ when the domain has to be specified.

Definition 4. *Consider the subset $V \subseteq \{1, \ldots, n\}$. For any finite Abelian group G, we define*

$$\pi_V(G^n) \triangleq \{\pi_V(x) \mid x \in G^n\} = \left\{(x_1, \ldots, x_n) \in G^n \mid x_s = 0 \quad \forall s \in V^{\mathsf{c}}\right\}$$

The sets defined below are going to be used to thoroughly conduct the probabilistic analysis of our scheme. In particular, they describe the different ways a secret a can be shared into $\#V$ shares.

Definition 5. *For any finite Abelian group G and any subset $V \subseteq \{1,\dots,n\}$, we define for any $a \in G$,*

$$S_V^n(a) \triangleq \begin{cases} \left\{(x_1,\dots,x_n) \in \pi_V(G^n) \;\middle|\; \sum_{i=1}^n x_i = a\right\} & \text{if } V \neq \emptyset\,, \\ \emptyset & \text{otherwise.} \end{cases}$$

Basic operations on vectors and associated distributions. Adding and substracting a uniform random variable to components of a random vector uniformly distributed over $S_V^n(a)$ leaves its distribution unchanged.

Proposition 1 (Invariance)**.** *Consider a finite Abelian group G, a subset $V \subseteq \{1,\dots,n\}$ and $a \in G$. If $\underline{X} \sim \mathcal{U}\big(S_V^n(a)\big)$ and $R \sim \mathcal{U}(G)$ are stochastically independent, then for any $s,t \in V$,*

$$\underline{X} + \tau_s^n(R) - \tau_t^n(R) \;\sim\; \mathcal{U}\left(S_V^n(a)\right)\,.$$

Proof. Let $\underline{x} \in S_V^n(a)$ and $s,t \in V$. We have

$$\mathrm{P}\big[\,\underline{X}+\tau_s^n(R)-\tau_t^n(R) = \underline{x}\,\big] = \sum_{r\in G} \mathrm{P}\big[\,\underline{X}+\tau_s^n(R)-\tau_t^n(R) = \underline{x} \mid R = r\,\big]\cdot\mathrm{P}\big[\,R=r\,\big]\,.$$

Also, by replacing the random variable R by its value into the right-hand side and noting that R and $\underline{X}$ are stochastically independent random variables, it gives

$$\sum_{r\in G} \mathrm{P}\big[\,\underline{X} + \tau_s^n(r) - \tau_t^n(r) = \underline{x}\,\big]\cdot\mathrm{P}\big[\,R=r\,\big]\,,$$

and therefore we have

$$\mathrm{P}\big[\,\underline{X} + \tau_s^n(R) - \tau_t^n(R) = \underline{x}\,\big] = \sum_{r\in G} \mathrm{P}\big[\,\underline{X} = \underline{x} - \tau_s^n(r) + \tau_t^n(r)\,\big]\cdot\mathrm{P}\big[\,R=r\,\big]\,.$$

Noting that $\underline{x} \in S_V^n(a)$ if and only if $\underline{x} - \tau_s^n(r) + \tau_t^n(r) \in S_V^n(a)$ and that by hypothesis $\underline{X}$ is uniform on $S_V^n(a)$, we have

$$\begin{aligned}\mathrm{P}\big[\,\underline{X} + \tau_s^n(R) - \tau_t^n(R) = \underline{x}\,\big] &= \sum_{r\in G} \mathrm{P}\big[\,\underline{X} = \underline{x}\,\big]\cdot\mathrm{P}\big[\,R=r\,\big] \\ &= \mathrm{P}\big[\,\underline{X} = \underline{x}\,\big]\cdot\sum_{r\in G}\mathrm{P}\big[\,R=r\,\big]\,,\end{aligned}$$

which finally gives

$$\mathrm{P}\big[\,\underline{X} + \tau_s^n(R) - \tau_t^n(R) = \underline{x}\,\big] = \mathrm{P}\big[\,\underline{X} = \underline{x}\,\big]\,.$$

The result follows. □

The following proposition evaluates the effect of the addition of a constant to a component of a random vector on its distribution.

Proposition 2 (Translation)**.** *Consider a finite Abelian group G, a subset $V \subseteq \{1,\dots,n\}$ and $a \in G$. If $\underline{X} \sim \mathcal{U}\left(S_V^n(a)\right)$, then for any $r \in G$ and any $s \in V$*

$$\underline{X} + \boldsymbol{\tau}_s^n(r) \;\sim\; \mathcal{U}\left(S_V^n(a+r)\right) .$$

Proof. By hypothesis,

$$\mathrm{P}\big[\,\underline{X} = \underline{x}\,\big] = \begin{cases} 1/\#G^{\#V-1} & \text{if } \underline{x} \in S_V^n(a)\,, \\ 0 & \text{otherwise.} \end{cases}$$

For any $\underline{x} \in G^n$, any $s \in V$ and any $r \in G$, we have

$$\mathrm{P}\big[\,\underline{X} + \boldsymbol{\tau}_s^n(r) = \underline{x}\,\big] = \mathrm{P}\big[\,\underline{X} = \underline{x} - \boldsymbol{\tau}_s^n(r)\,\big]$$

and therefore

$$\mathrm{P}\big[\,\underline{X} + \boldsymbol{\tau}_s^n(r) = \underline{x}\,\big] = \begin{cases} 1/\#G^{\#V-1} & \text{if } \underline{x} - \boldsymbol{\tau}_s^n(r) \in S_V^n(a)\,, \\ 0 & \text{otherwise.} \end{cases}$$

Finally, note that $\underline{x} - \boldsymbol{\tau}_s^n(r) \in S_V^n(a)$ if and only if $\underline{x} \in S_V^n(a+r)$. The result follows. □

Evaluating the effect of adding and substracting a random variable to components of two random vectors on its common distribution is done by the following proposition.

Proposition 3 (Merging)**.** *Consider the subsets $V_1, V_2 \subseteq \{1,\dots,n\}$ such that $V_1 \cap V_2 = \emptyset$ and consider $a_1, a_2 \in G$. If $\underline{X} \sim \mathcal{U}\left(S_{V_1}^n(a_1)\right)$, $\underline{Y} \sim \mathcal{U}\left(S_{V_2}^n(a_2)\right)$ and $R \sim \mathcal{U}(G)$ are stochastically independent, then for any $s \in V_1$ and any $t \in V_2$,*

$$\underline{X} + \underline{Y} + \boldsymbol{\tau}_s^n(R) - \boldsymbol{\tau}_t^n(R) \;\sim\; \mathcal{U}\left(S_{V_1 \cup V_2}^n(a_1 + a_2)\right) .$$

Proof. Consider $\underline{z} = (z_1,\dots,z_n) \in S_{V_1\cup V_2}^n(a_1+a_2)$. Let us compute

$$\mathrm{P}\big[\,\underline{X} + \underline{Y} + \boldsymbol{\tau}_s^n(R) - \boldsymbol{\tau}_t^n(R) = \underline{z}\,\big] . \tag{2.1}$$

By the law of total probability, it gives

$$\sum_{r\in G} \mathrm{P}\big[\,\underline{X} + \underline{Y} + \boldsymbol{\tau}_s^n(R) - \boldsymbol{\tau}_t^n(R) = \underline{z} \mid R = r\,\big] \cdot \mathrm{P}\big[\,R = r\,\big]$$

or equivalently

$$\sum_{r\in G} \mathrm{P}\big[\, \underline{X} + \underline{Y} + \boldsymbol{\tau}_s^n(r) - \boldsymbol{\tau}_t^n(r) = \underline{z} \mid R = r \,\big] \cdot \mathrm{P}\big[\, R = r \,\big]\,. \tag{2.2}$$

By hypothesis, R is stochastically independent of $(\underline{X}, \underline{Y})$. Therefore

$$\mathrm{P}\big[\, \underline{X} + \underline{Y} + \boldsymbol{\tau}_s^n(r) - \boldsymbol{\tau}_t^n(r) = \underline{z} \mid R = r \,\big] = \mathrm{P}\big[\, \underline{X} + \underline{Y} + \boldsymbol{\tau}_s^n(r) - \boldsymbol{\tau}_t^n(r) = \underline{z} \,\big] \tag{2.3}$$

Also, $V_1 \cap V_2 = \emptyset$ by hypothesis. Therefore $\underline{z} = \pi_{V_1}(\underline{z}) + \pi_{V_2}(\underline{z})$. Also, $s \in V_1$ and $t \in V_2$. It follows that

$$\begin{aligned}\mathrm{P}\big[\, \underline{X} + \underline{Y} + \boldsymbol{\tau}_s^n(r) - \boldsymbol{\tau}_t^n(r) = \underline{z} \,\big] &= \mathrm{P}\big[\, \underline{X} + \underline{Y} = \pi_{V_1}(\underline{z}) - \boldsymbol{\tau}_s^n(r) + \pi_{V_2}(\underline{z}) + \boldsymbol{\tau}_t^n(r) \,\big] \\ &= \mathrm{P}\big[\, \underline{X} = \pi_{V_1}(\underline{z}) - \boldsymbol{\tau}_s^n(r)\,;\, \underline{Y} = \pi_{V_2}(\underline{z}) + \boldsymbol{\tau}_t^n(r) \,\big]\,.\end{aligned}$$

The random vectors $\underline{X}$ and $\underline{Y}$ being stochastically independent, we have

$$\mathrm{P}\big[\, \underline{X} + \underline{Y} + \boldsymbol{\tau}_s^n(r) - \boldsymbol{\tau}_t^n(r) = \underline{z} \,\big] = \mathrm{P}\big[\, \underline{X} = \pi_{V_1}(\underline{z}) - \boldsymbol{\tau}_s^n(r) \,\big] \cdot \mathrm{P}\big[\, \underline{Y} = \pi_{V_2}(\underline{z}) + \boldsymbol{\tau}_t^n(r) \,\big]\,. \tag{2.4}$$

By hypothesis,

$$\mathrm{P}\big[\, \underline{X} = \pi_{V_1}(\underline{z}) - \boldsymbol{\tau}_s^n(r) \,\big] = \begin{cases} 1/\#G^{\#V_1 - 1} & \text{if } \sum_{i\in V_1} \underline{z}_i - r = a_1\,, \\ 0 & \text{otherwise.} \end{cases}$$

and

$$\mathrm{P}\big[\, \underline{Y} = \pi_{V_2}(\underline{z}) + \boldsymbol{\tau}_t^n(r) \,\big] = \begin{cases} 1/\#G^{\#V_2 - 1} & \text{if } \sum_{i\in V_2} \underline{z}_i + r = a_2\,, \\ 0 & \text{otherwise.} \end{cases}$$

Remember that by hypothesis, $\underline{z} \in S^n_{V_1 \cup V_2}(a_1 + a_2)$ and $V_1 \cap V_2 = \emptyset$. Therefore $\sum_{i\in V_1} \underline{z}_i + \sum_{i\in V_2} \underline{z}_i = a_1 + a_2$. It follows that $\sum_{i\in V_1} \underline{z}_i - r = a_1$ if and only if $\sum_{i\in V_2} \underline{z}_i + r = a_2$. Relationships (2.2), (2.3), (2.4) and the above discussion gives that (2.1) is equal to

$$\mathrm{P}\big[\, \underline{X} = \pi_{V_1}(\underline{z}) - \boldsymbol{\tau}_s^n(r_0) \,\big] \cdot \mathrm{P}\big[\, \underline{Y} = \pi_{V_2}(\underline{z}) + \boldsymbol{\tau}_t^n(r_0) \,\big] \cdot \mathrm{P}\big[\, R = r_0 \,\big]$$

where $r_0 = \sum_{i\in V_1} \underline{z}_i - a_1$. It turns out that

$$\begin{aligned}\mathrm{P}\big[\, \underline{X} + \underline{Y} + \boldsymbol{\tau}_s^n(R) - \boldsymbol{\tau}_t^n(R) = \underline{z} \,\big] &= \frac{1}{\#G^{\#V_1 - 1}} \cdot \frac{1}{\#G^{\#V_2 - 1}} \cdot \frac{1}{\#G} \\ &= \frac{1}{\#G^{\#V_1 + \#V_2 - 1}}\,.\end{aligned}$$

Also, $V_1 \cap V_2 = \emptyset$ by hypothesis. Therefore $\#(V_1 \cup V_2) = \#V_1 + \#V_2$. Consequently we have

$$\mathrm{P}\big[\, \underline{X} + \underline{Y} + \boldsymbol{\tau}_s^n(R) - \boldsymbol{\tau}_t^n(R) = \underline{z} \,\big] = \frac{1}{\#G^{\#(V_1 \cup V_2) - 1}}\,.$$

The result follows. □

Direct sum and conditioning. Our probabilistic analysis requires to analyze in details the distributions over sums of spaces and also conditionnal distributions over sums of spaces. This is formalized in the next three propositions. We start by studying the interaction between the direct sum of sets and the operator of definition 1.

Proposition 4 (Direct sum)**.** *Consider $V_1, V_2 \subseteq \{1, \ldots, n\}$ such that $V_1 \cap V_2 = \emptyset$. Consider the subsets $S \subseteq \pi_{V_1}(G^n)$ and $T \subseteq \pi_{V_2}(G^n)$. Define*

$$S + T = \left\{ \underline{x} + \underline{y} \mid \underline{x} \in S, \underline{y} \in T \right\} .$$

Then,

1. *$\underline{z} = \pi_{V_1}(\underline{z}) + \pi_{V_2}(\underline{z})$ for any $\underline{z} \in S + T$.*
2. *$\pi_{V_1}(S + T) = S$ and $\pi_{V_2}(S + T) = T$.*
3. *If $\underline{Z} \sim \mathcal{U}(S + T)$, then $\pi_{V_1}(\underline{Z}) \sim \mathcal{U}\big(S\big)$ and $\pi_{V_2}(\underline{Z}) \sim \mathcal{U}\big(T\big)$ and $\pi_{V_1}(\underline{Z})$ and $\pi_{V_2}(\underline{Z})$ are stochastically independent.*
4. *If $\underline{X} \sim \mathcal{U}(S)$, $\underline{Y} \sim \mathcal{U}(T)$ and $\underline{X}, \underline{Y}$ are stochastically independent, then $\underline{Z} = \underline{X} + \underline{Y} \sim \mathcal{U}(S + T)$.*

Remark 1. *In our context S and T will be sums of the spaces $S^n_{V_i}(a_i)$.*

Proof. By hypothesis, $V_1 \cap V_2 = \emptyset$. It follows from definition 1 that $\underline{z} = \pi_{V_1}(\underline{z}) + \pi_{V_2}(\underline{z})$ for any $\underline{z} \in S + T$.

Also, $\mathsf{Im}\ \pi = S \times T$ and $\mathsf{Ker}\ \pi = \{0\}$ where

$$\pi : S + T \mapsto G^n \times G^n : x \mapsto (\pi_{V_1}(x), \pi_{V_2}(x)) .$$

Therefore, there exists a bijection between $S + T$ and $S \times T$ and it follows that $\#(S + T) = \#S \cdot \#T$.

Suppose that $\underline{Z} \sim \mathcal{U}(S + T)$. We have

$$\mathrm{P}\big[\, \underline{Z} = \underline{z} \,\big] = \frac{1}{\#(S + T)} = \frac{1}{\#S \cdot \#T} .$$

Also, for any $\underline{x} \in S$,

$$\begin{aligned}
\mathrm{P}\big[\,\pi_{V_1}(\underline{Z}) = \underline{x}\,\big] &= \mathrm{P}\big[\,\pi_{V_1}(\underline{Z}) = \underline{x}\,;\,\pi_{V_2}(\underline{Z}) \in T\,\big]. \\
&= \sum_{\underline{y}\in T} \mathrm{P}\big[\,\pi_{V_1}(\underline{Z}) = \underline{x}\,;\,\pi_{V_2}(\underline{Z}) = \underline{y}\,\big] \\
&= \sum_{\underline{y}\in T} \mathrm{P}\big[\,\pi_{V_1}(\underline{Z}) + \pi_{V_2}(\underline{Z}) = \underline{x} + \underline{y}\,\big] \\
&= \sum_{\underline{y}\in T} \mathrm{P}\big[\,\underline{Z} = \underline{x} + \underline{y}\,\big] \\
&= \sum_{\underline{z}\in \underline{x}+T} \mathrm{P}\big[\,\underline{Z} = \underline{z}\,\big] \\
&= \frac{\#T}{\#S \cdot \#T} = \frac{1}{\#S}\,.
\end{aligned}$$

Similarly, for any $\underline{y} \in T$, $\mathrm{P}\big[\,\pi_{V_2}(\underline{Z}) = \underline{y}\,\big] = 1/\#T$.

Finally, for any $\underline{x} \in S$, and any $\underline{y} \in T$

$$\begin{aligned}
\mathrm{P}\big[\,\pi_{V_1}(\underline{Z}) = \underline{x}\,\big] \cdot \mathrm{P}\big[\,\pi_{V_2}(\underline{Z}) = \underline{y}\,\big] &= \frac{1}{\#S} \cdot \frac{1}{\#T} \\
&= \frac{1}{\#\,(S+T)} \\
&= \mathrm{P}\big[\,\underline{Z} = \underline{x} + \underline{y}\,\big] \\
&= \mathrm{P}\big[\,\pi_{V_1}(\underline{Z}) + \pi_{V_2}(\underline{Z}) = \underline{x} + \underline{y}\,\big] \\
&= \mathrm{P}\big[\,\pi_{V_1}(\underline{Z}) = \underline{x}\,;\,\pi_{V_2}(\underline{Z}) = \underline{y}\,\big]\,.
\end{aligned}$$

It turns out that $\pi_{V_1}(\underline{Z})$ and $\pi_{V_2}(\underline{Z})$ are stochastically independent.

Now, assume that $\underline{X} \sim \mathcal{U}\,(S)$ and $\underline{Y} \sim \mathcal{U}\,(T)$ are stochastically independent. Define $\underline{Z} = \underline{X} + \underline{Y}$. Remember that for any $\underline{z} \in S+T$, there exist $\underline{x} \in S$ and $\underline{y} \in T$ such that $\underline{z} = \underline{x} + \underline{y}$. We have

$$\mathrm{P}\big[\,\underline{Z} = \underline{z}\,\big] = \mathrm{P}\big[\,\underline{X} + \underline{Y} = \underline{x} + \underline{y}\,\big]\,.$$

Remembering that $S \subseteq \pi_{V_1}(G^n)$ and $T \subseteq \pi_{V_2}(G^n)$ and $V_1 \cap V_2 = \emptyset$, we have

$$\mathrm{P}\big[\,\underline{X} + \underline{Y} = \underline{x} + \underline{y}\,\big] = \mathrm{P}\big[\,\underline{X} = \underline{x}\,;\,\underline{Y} = \underline{y}\,\big]\,.$$

Also,

$$\mathrm{P}\big[\,\underline{X} = \underline{x}\,;\,\underline{Y} = \underline{y}\,\big] = \mathrm{P}\big[\,\underline{X} = \underline{x}\,\big] \cdot \mathrm{P}\big[\,\underline{Y} = \underline{y}\,\big] = \frac{1}{\#S} \cdot \frac{1}{\#T}$$

by hypothesis. It follows that

$$\mathrm{P}\big[\,\underline{Z} = \underline{z}\,\big] = \frac{1}{\#(S+T)}\,.$$

The result follows. □

Next proposition is a result related to stochastic independence and conditioning.

Proposition 5. *Consider two stochastically independent random vectors* $(\,\underline{X}, \underline{Y}\,)$ *and* $(\,\underline{X}', \underline{Y}'\,)$. *Denote by* $S = \Big\{\underline{x} \;\Big|\; \mathrm{P}\big[\,\underline{X} = \underline{x}\,\big] \neq 0\Big\}$ *and by* $S' = \Big\{\underline{x} \;\Big|\; \mathrm{P}\big[\,\underline{X}' = \underline{x}\,\big] \neq 0\Big\}$. *Then, for any* $(\underline{x}, \underline{x}') \in S \times S'$,

$$\underline{Y} \;\mid\; \underline{X} = \underline{x} \;\sim\; \underline{Y} \;\mid\; (\underline{X} = \underline{x}\,;\, \underline{X}' = \underline{x}')$$

and

$$\underline{Y}' \;\mid\; \underline{X}' = \underline{x}' \;\sim\; \underline{Y}' \;\mid\; (\underline{X} = \underline{x}\,;\, \underline{X}' = \underline{x}')\,. \tag{2.5}$$

Also, $\underline{Y} \mid (\underline{X} = \underline{x}\,;\, \underline{X}' = \underline{x}')$ *and* $\underline{Y}' \mid (\underline{X} = \underline{x}\,;\, \underline{X}' = \underline{x}')$ *are stochastically independent.*

Proof. For any $\underline{x} \in S$, $\underline{x}' \in S'$ and $\underline{y} \in \mathsf{Im}\,(\underline{Y})$,

$$\mathrm{P}\big[\,\underline{Y} = \underline{y} \mid \underline{X} = \underline{x}\,\big] = \frac{\mathrm{P}\big[\,\underline{Y} = \underline{y}\,;\, \underline{X} = \underline{x}\,\big]}{\mathrm{P}\big[\,\underline{X} = \underline{x}\,\big]}\,.$$

Also, the right-hand side is equal to

$$\frac{\mathrm{P}\big[\,\underline{Y} = \underline{y}\,;\, \underline{X} = \underline{x}\,\big] \cdot \mathrm{P}\big[\,\underline{X}' = \underline{x}'\,\big]}{\mathrm{P}\big[\,\underline{X} = \underline{x}\,\big] \cdot \mathrm{P}\big[\,\underline{X}' = \underline{x}'\,\big]}\,.$$

By hypothesis, $(\,\underline{X}, \underline{Y}\,)$ and $(\,\underline{X}', \underline{Y}'\,)$ are stochastically independent, the latter formula can therefore be written

$$\frac{\mathrm{P}\big[\,\underline{Y} = \underline{y}\,;\, \underline{X} = \underline{x}\,;\, \underline{X}' = \underline{x}'\,\big]}{\mathrm{P}\big[\,\underline{X} = \underline{x}\,;\, \underline{X}' = \underline{x}'\,\big]}$$

which is $\mathrm{P}\big[\,\underline{Y} = \underline{y} \;\mid\; (\underline{X} = \underline{x}\,;\, \underline{X}' = \underline{x}')\,\big]$ by definition. The result follows. Also, the second assertion of (2.5) is proved similarly. Let us now prove the final statement. We have

$$\mathrm{P}\big[\,\underline{Y} = \underline{y}\,;\, \underline{Y}' = \underline{y}' \;\mid\; (\underline{X} = \underline{x}\,;\, \underline{X}' = \underline{x}')\,\big] = \frac{\mathrm{P}\big[\,\underline{Y} = \underline{y}\,;\, \underline{Y}' = \underline{y}'\,;\, \underline{X} = \underline{x}\,;\, \underline{X}' = \underline{x}'\,\big]}{\mathrm{P}\big[\,\underline{X} = \underline{x}\,;\, \underline{X}' = \underline{x}'\,\big]}\,.$$

Since by hypothesis we have that $(\underline{X}, \underline{Y})$ and $(\underline{X}', \underline{Y}')$ are stochastically independent, then the right-hand side is equal to

$$\frac{\mathrm{P}\big[\,\underline{Y} = \underline{y}\,;\,\underline{X} = \underline{x}\,\big] \cdot \mathrm{P}\big[\,\underline{Y}' = \underline{y}'\,;\,\underline{X}' = \underline{x}'\,\big]}{\mathrm{P}\big[\,\underline{X} = \underline{x}\,\big] \cdot \mathrm{P}\big[\,\underline{X}' = \underline{x}'\,\big]}$$

which, by definition, is also

$$\mathrm{P}\big[\,\underline{Y} = \underline{y} \mid \underline{X} = \underline{x}\,\big] \cdot \mathrm{P}\big[\,\underline{Y}' = \underline{y}' \mid \underline{X}' = \underline{x}'\,\big]$$

or equivalently

$$\mathrm{P}\big[\,\underline{Y} = \underline{y} \mid (\underline{X} = \underline{x}\,;\,\underline{X}' = \underline{x}')\,\big] \cdot \mathrm{P}\big[\,\underline{Y}' = \underline{y}' \mid (\underline{X} = \underline{x}\,;\,\underline{X}' = \underline{x}')\,\big]\,.$$

The result follows. □

Hereafter, we study the impact of conditioning on vectors of random variables uniformly defined on the sum of sets defined in definition 5.

Proposition 6 (Conditioning)**.** *Consider a finite Abelian group G and a partition $V_1, \ldots, V_k$ of $\{1, \ldots, n\}$. Let $a_1, \ldots, a_k$ be elements of G and I be a subset of $\{1, \ldots, n\}$. If $\underline{X} \sim \mathcal{U}\left(\sum_{i=1}^{k} S_{V_i}^n(a_i)\right)$, then for any $\underline{x} \in \sum_{i=1}^{k} S_{V_i}^n(a_i)$,*

$$\underline{X} \mid \underline{X}_I = \underline{x}_I \sim \mathcal{U}\left(\sum_{i=1}^{k} S_{V_i \setminus I}^n\left(a_i - \sum_{j \in V_i \cap I} \underline{x}_j\right) + \pi_I(\underline{x})\right).$$

Proof. Let i be an element of $\{1, \ldots, k\}$. Consider $\underline{x}, \underline{y} \in S_{V_i}^n(a_i)$ and $I \subseteq V_i$. Let us compute

$$\mathrm{P}\big[\pi_{V_i}(\underline{X}) = \underline{y} \mid \underline{X}_I = \underline{x}_I\big] = \frac{\mathrm{P}\big[\,\pi_{V_i}(\underline{X}) = \underline{y}\,;\,\underline{X}_I = \underline{x}_I\,\big]}{\mathrm{P}\big[\,\underline{X}_I = \underline{x}_I\,\big]} = \begin{cases} \dfrac{\mathrm{P}\big[\,\pi_{V_i}(\underline{X}) = \underline{y}\,\big]}{\mathrm{P}\big[\,\underline{X}_I = \underline{x}_I\,\big]} & \text{if } \underline{x}_I = \underline{y}_I\,, \\ 0 & \text{otherwise.} \end{cases}$$

According to the third assertion of proposition 4, $\pi_{V_i}(\underline{X}) \sim \mathcal{U}\left(S_{V_i}^n(a_i)\right)$. Also, if $I \neq V_i$ then $\underline{Y}_I \sim \mathcal{U}\left(G^{\#I}\right)$. Therefore,

$$\mathrm{P}\big[\pi_{V_i}(\underline{X}) = \underline{y} \mid \underline{X}_I = \underline{x}_I\big] = \begin{cases} \dfrac{1/(\#G^{\#V_i - 1})}{1/(\#G^{\#I})} = \dfrac{1}{\#G^{\#V_i - \#I - 1}} & \text{if } \underline{x}_I = \underline{y}_I\,, \\ 0 & \text{otherwise.} \end{cases}$$

If $I = V_i$ then $\underline{Y}_I \sim \mathcal{U}\left(S_{\{1, \ldots, \#V_i\}}^{\#V_i}(a_i)\right)$. Hence,

$$\mathrm{P}\big[\pi_{V_i}(\underline{X}) = \underline{y} \mid \underline{X}_I = \underline{x}_I\big] = \begin{cases} \dfrac{1/(\#G^{\#V_i - 1})}{1/(\#G^{\#V_i - 1})} = 1 & \text{if } \underline{x}_I = \underline{y}_I\,, \\ 0 & \text{otherwise.} \end{cases}$$

It follows that

$$\pi_{V_i}(\underline{X}) \mid \underline{X}_I = \underline{x}_I \ \sim \ \mathcal{U}\Big(S^n_{V_i \setminus I}\Big(a_i - \sum_{j \in I} \underline{x}_j\Big) + \pi_I(\underline{x})\Big) .$$

According to the first and the third statement of proposition 4, $\underline{X} = \sum_{i=1}^k \pi_{V_i}(\underline{X})$ and the random variables $\pi_{V_i}(\underline{X})$ are stochastically independent. It follows from proposition 5 that for any $I \subseteq \{1, \ldots, n\}$,

$$\pi_{V_i}(\underline{X}) \mid \underline{X}_I = \underline{x}_I \ \sim \ \pi_{V_i}(\underline{X}) \ \mid \ \underline{X}_{I \cap V_i} = \underline{x}_{I \cap V_i}$$

and that $\pi_{V_i}(\underline{X}) \mid \underline{X}_I = \underline{x}_I$ are stochastically independent. Given the above discussion

$$\pi_{V_i}(\underline{X}) \mid \underline{X}_{I \cap V_i} = \underline{x}_{I \cap V_i} \ \sim \ \mathcal{U}\Big(S^n_{V_i \setminus I}\Big(a_i - \sum_{j \in I \cap V_i} \underline{x}_j\Big) + \pi_{I \cap V_i}(\underline{x})\Big).$$

Therefore,

$$\pi_{V_i}(\underline{X}) \mid \underline{X}_I = \underline{x}_I \ \sim \ \mathcal{U}\Big(S^n_{V_i \setminus I}\Big(a_i - \sum_{j \in I \cap V_i} \underline{x}_j\Big) + \pi_{I \cap V_i}(\underline{x})\Big) .$$

Since $\pi_{V_i}(\underline{X}) \mid \underline{X}_I = \underline{x}_I$ are stochastically independent, therefore, according to the last statement of proposition 4

$$\sum_{i=1}^k \pi_{V_i}(\underline{X}) \mid \underline{X}_I = \underline{x}_I \ \sim \ \mathcal{U}\left(\sum_{i=1}^k S^n_{V_i \setminus I}\Big(a_i - \sum_{j \in I \cap V_i} \underline{x}_j\Big) + \pi_I(\underline{x})\right) .$$

The result follows. □

Let us now state and prove a proposition that will be useful in the proof of lemma 1. An example is given below the statement for the sake of clarity.

Proposition 7. *Let $b \in \{0, 1\}$. Consider the subset $J \subseteq \{1, \ldots, n\}$, the partition $V_1, \ldots, V_k$ of $\{1, \ldots, n\}$ and the partition $V_{k+1}, \ldots, V_{k+l}$ of $\{1, \ldots, n+b\}+n$. Define for any $x, y \in \{1, \ldots, 2n + b\}$*

$$\begin{aligned} x \sim y \Longleftrightarrow\ & \exists i \in \{1, \ldots, k+l\} \text{ such that } x, y \in V_i \\ & \text{or } \ \exists i, j \in \{1, \ldots, k+l\} \text{ such that } V_i \cap (V_j - n) \cap J \neq \emptyset \\ & \qquad \text{and } (x \in V_i, y \in V_j \text{ or } y \in V_i, x \in V_j) . \end{aligned}$$

Denote by $\approx$ the transitive closure of $\sim$. Then,

1. *$\eqsim$ is an equivalence relation on $\{1,\dots,2n+b\}$.*

2. *There exists a partition $S_1,\dots,S_m$ of $\{1,\dots,l+k\}$ such that the equivalence classes of $\{1,\dots,2n+b\}$ for $\eqsim$ are :*

$$X_i = \bigcup_{j\in S_i} V_j, \quad i = 1,\dots,m\,.$$

Moreover, for any $i = 1\dots m$, if $k_i \triangleq \#S_i$, then $\exists i_1,\dots,i_{k_i} \in S_i$ and $r_2,\dots,r_{k_i} \in J$ such that for any t with $1 < t \leq k_i$ we have

$$r_t \in V_{i_t} \text{ and } r_t + n \in \bigcup_{j=1}^{t-1} V_{i_j} \text{ if } i_t \in \{1,\dots,l\}$$

and (2.6)

$$r_t \in \bigcup_{j=1}^{t-1} V_{i_j} \text{ and } r_t + n \in V_{i_t} \text{ if } i_t \in \{1,\dots,l\} + k\,.$$

Also, for any $i \in \{1,\dots,m\}$, for any $x \in J \cap X_i$ we have $x + n \in X_i$ and for any $x \in (J+n) \cap X_i$ we have $x - n \in X_i$.

Proof. The reflexivity of $\sim$ is a direct consequence of its definition and the fact that $(V_i)_{i=1\dots l+k}$ is a partition of $\{1,\dots,2n+b\}$. The symetry of $\sim$ follows immediately from its definition. The relation $\eqsim$ being the transitive closure of $\sim$, it follows that $\eqsim$ is an equivalence relation.

Let us compute the equivalence classes $(X_i)_{i=1,\dots,m}$ of $\{1,\dots,2n+b\}$ for $\eqsim$. Consider an equivalence class X_i with $i \in \{1,\dots,m\}$.

Define

$$S_i = \left\{ j \mid X_i \cap V_j \neq \emptyset \text{ and } j \in \{1,\dots,l+k\} \right\}.$$

We have $X_i \subseteq \bigcup_{j\in S_i} V_j$ by definition of S_i. Also, $X_i \neq \emptyset$ because X_i is an equivalence class. Noting that if $x \in V_j$ for a fixed $j \in \{1,\dots,l+k\}$, then $V_j \subseteq X_i$ according to the definition of $\sim$ (*resp.* $\eqsim$). It follows that $\bigcup_{j\in S_i} V_j \subseteq X_i$. We conclude that $X_i = \bigcup_{j\in S_i} V_j$.

Remembering that $(X_i)_{i=1,\dots,m}$ and $(V_i)_{i=1,\dots,l+k}$ are a partition of $\{1,\dots,2n+b\}$ it follows that $(S_i)_{i=1,\dots,m}$ is a partition of $\{1,\dots,l+k\}$.

If $k_i = 1$, condition (2.6) is empty and nothing has to be proved. Suppose that $k_i \geq 2$.

Define

$$T_i = S_i \cap \{1, \ldots, k\} \text{ and } Q_i = S_i \cap (\{1, \ldots, l\} + k)\,.$$

Given the definition of the relation $\sim, \#T_i \geq 1$ and $\#Q_i \geq 1$. Therefore, $T_i \neq \emptyset$ and $\exists i_1 \in T_i$. The set X_i being an equivalence class and since $k_i \geq 2$, it implies that there exists $t \in S_i \setminus \{i_1\}$ such that there exist $x \in V_{i_1}$ and $y \in V_t$ satisfying $x \eqsim y$. If $x \sim y$ then $V_t \subseteq Q_i$ because $V_{i_1} \subseteq T_i$. Define $i_2 = t$. We have $V_{i_1} \cap (V_{i_2} - n) \cap J \neq \emptyset$. There exists $r_2 \in J$ such that $r_2 \in V_{i_1}$ and $r_2 + n \in V_{i_2}$. If $x \eqsim y$ and $x \not\sim y$, the transitive closure of the relation $\eqsim$ implies that there exist element (we may suppose distinct without loss of generality) $j_1, \ldots, j_r \in S_i \setminus \{i_1\}$ such that

$$\begin{aligned} x &\sim x_{j_1} \text{ with } x_{j_1} \in V_{j_1} \\ x_{j_1} &\sim x_{j_2} \text{ with } x_{j_2} \in V_{j_2} \\ &\vdots \\ x_{j_{r-1}} &\sim x_{j_r} \text{ with } x_{j_r} \in V_{j_r} \\ x_{j_r} &\sim y\,. \end{aligned}$$

It follows that $V_{i_1} \cap (V_{j_1} - n) \cap J \neq \emptyset$. Define $i_2 = j_1$. We have that $V_{i_1} \cap (V_{i_2} - n) \cap J \neq \emptyset$. There exists $r_2 \in I$ such that $r_2 \in V_{i_1}$ and $r_2 + n \in V_{i_2}$. The other elements i_s and r_s are built successively. Suppose distinct elements $i_1, i_2, \ldots, i_r \in S_i$ with $r < k_i$ and elements $r_2, \ldots, r_r$ such that for any $s \leq r$

$$r_s \in V_{i_s} \text{ and } r_s + n \in \bigcup_{j=1}^{s-1} V_{i_j} \text{ if } i_s \in T_i$$

and

$$r_s \in \bigcup_{j=1}^{s-1} V_{i_j} \text{ and } r_s + n \in V_{i_s} \text{ if } i_s \in Q_i\,.$$

Define $V = \bigcup_{j=1}^{r} V_{i_j}$ and consider $x \in V_{i_1}$. By hypothesis, $r < k_i$. Therefore there exists $y \in X_i \setminus V$. The set X_i being an equivalence class, we have $x \eqsim y$. If $x \sim y$, there exists an index we will denote i_{r+1} such that $y \in V_{i_{r+1}}$. This index $i_{r+1} \notin \{i_1, \ldots, i_r\}$ because $y \in X_i \setminus V$. According to the definition of $\sim$, $\exists r_{r+1} \in J$ such that

$$r_{r+1} \in V_{i_{r+1}} \text{ and } r_{r+1} + n \in V \text{ if } i_{r+1} \in T_i$$

and

$$r_{r+1} \in V \text{ and } r_{r+1} + n \in V_{i_{r+1}} \text{ if } i_{r+1} \in Q_i\,.$$

If $x \not\sim y$, according to the transitive closure of $\overline{\sim}$, there exist elements (that we may suppose distincts without loss of generality) $j_1, \ldots, j_r \in S_i \setminus \{i_1\}$ such that

$$\begin{aligned} x &\sim x_{j_1} \text{ with } x_{j_1} \in V_{j_1} \\ x_{j_1} &\sim x_{j_2} \text{ with } x_{j_2} \in V_{j_2} \\ &\vdots \\ x_{j_{r-1}} &\sim y \text{ with } y \in V_{j_r} \end{aligned}$$

If $j_1 \notin \{i_1, \ldots, i_r\}$, define $i_{r+1} = j_1$ and by definition of $\sim$ there exists $r_{r+1} \in V_{i_{r+1}}$ and $r_{r+1} + n \in V$ if $i_{r+1} \in T_i$ and $r_{r+1} \in V$ and $r_{r+1} + n \in V_{i_{r+1}}$ if $i_{r+1} \in Q_i$. Otherwise, let t be the greatest element such that $j_1, j_2, \ldots, j_t \in \{i_1, \ldots, i_r\}$. Necessarily, $t < v$ because $V_{j_v} \cap V = \emptyset$. Define $i_{r+1} = j_{t+1}$. By construction $x_{j_t} \sim x_{j_{t+1}}$ and therefore $\exists r_{r+1} \in J$ such that

$$r_{r+1} \in V_{i_{r+1}} \text{ and } r_{r+1} + n \in V \text{ if } i_{r+1} \in T_i$$

and

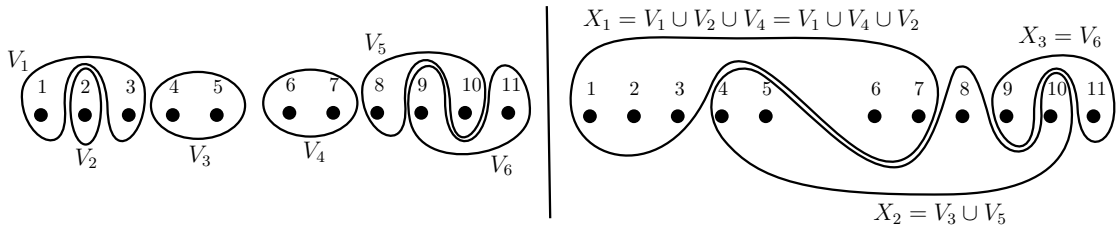
$$r_{r+1} \in V \text{ and } r_{r+1} + n \in V_{i_{r+1}} \text{ if } i_{r+1} \in Q_i\,.$$

Consider $x \in J \cap X_i$. By definition, there exists $j \in S_i \cap \{1, \ldots k\}$ such that $x \in V_j$. Also, there exists $t \in \{1, \ldots, l\} + k$ such that $x + n \in V_t$. Therefore, $x \in J \cap V_j \cap (V_t - n)$. It follows that $J \cap V_j \cap (V_t - n) \neq \emptyset$ which implies that $x \sim x + n$. By definition X_i is a class of equivalence, therefore $x + n \in X_i$.

Finally, consider $x \in (J+n) \cap X_i$. By definition, there exists $j \in S_i \cap (\{1, \ldots, l\} + k)$ such that $x \in V_j$. Also, there exists $t \in \{1, \ldots, l\}$ such that $x - n \in V_t$. Therefore, $x - n \in J \cap V_t \cap (V_j - n)$. This implies that $J \cap V_t \cap (V_j - n) \neq \emptyset$ which implies that $x \sim x - n$. By definition X_i is a class of equivalence, therefore $x - n \in X_i$. □

For the sake of clarity, let us now illustrate the application of proposition 7 on a simple example. Consider $n = 5$ and $b = 1$. Consider $J = \{1, 2, 5\}$ and the partitions V_1, V_2, V_3 of $\{1, \ldots 5\}$ and V_4, V_5, V_6 of $\{6, \ldots, 11\}$. Also, consider the V_i's to be defined as in the left side of the following figure. The purpose of proposition 7 is to determine the effect of an equivalence relation defined from a set J over the partition $V_1, V_2, V_3, V_4, V_5, V_6$ of $\{1, \ldots, 11\}$. The right side of the figure illustrates such an effect. Note that in this example we have $S_1 = \{1, 2, 4\}, S_2 = \{3, 5\}$ and $S_3 = \{6\}$.

The last part of the above proposition states that we can order the V_j's of a set



X_i in such a way that there exist r_t and r_{t+n} such that one of the two belongs to $\cup_{j=1}^{t-1} V_{i_j}$ and the other one belongs to V_{i_t} and this, for any t. This result will be very important to apply recursively proposition 3 (merging) in the proof of lemma 2.2.1. In what follows, we analyze the recursive step of the $\mathcal{R}_n$ algorithm.

2.2.2 Analysis of $\mathcal{R}_n$.

In the following, we aim to prove by induction the general results related to $\mathcal{R}_n$ given in theorem 2.2.3. In order to do this, we start by analyzing the recursive step of $\mathcal{R}_n$. Remember that this step correspond to the merging of two smaller schemes via a linear layer of randomness, as explained at the beginning of this section. After this, we analyze the complete execution of $\mathcal{R}_n$.

Analysis of the recursive step of the $\mathcal{R}_n$ algorithm.

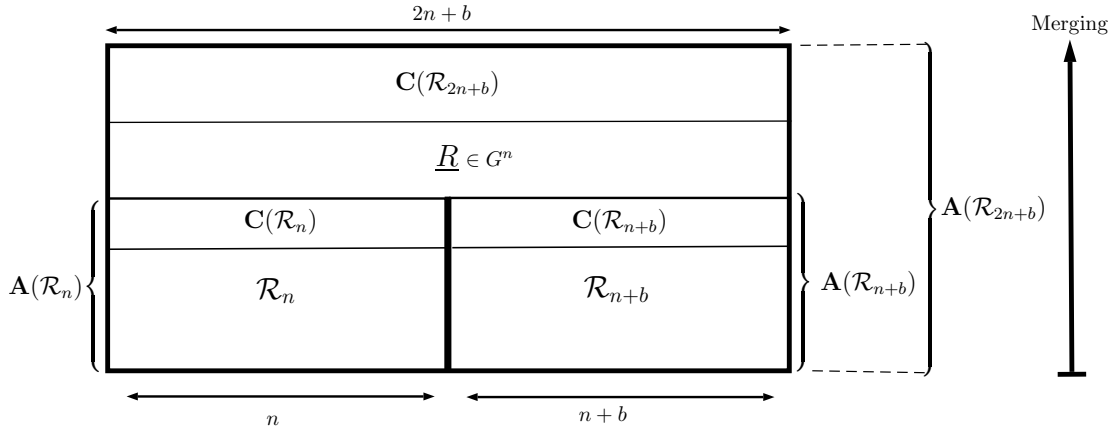
Consider the merging of the recursive calls $\mathcal{R}_n$ and $\mathcal{R}_{n+b}$ via the linear layer of randomness of size $2n + b$ and with respect to the insertion of random elements given at step 18 of the algorithm. The effect of such a merging is analyzed in lemma 2.2.1. The second type of merging, which corresponds to the insertion of random elements as described by step 20 will be treated in lemma 2.2.2. In any case, the merging results in a scheme of size $2n + b$ which follows the recursive definition of the $\mathcal{R}_n$ algorithm.

The context. In order to conduct our theoretical analysis, the resulting merged scheme $\mathcal{R}_{2n+b}$ is viewed as a block composed of the two schemes $\mathcal{R}_n$, $\mathcal{R}_{n+b}$ and the linear layer of randomness considered as a random vector $\underline{R}$. The computation of the linear layer of randomness is performed after the execution of $\mathcal{R}_n$ and $\mathcal{R}_{n+b}$, on the outputs of the two schemes ($\mathcal{R}_n$ and $\mathcal{R}_{n+b}$), and produces a random vector as result with additionnal randomness.

The overall computation of the merging is decomposed into several random vectors which hold the different random variables that are considered during the merging. We distinguish the different random vectors as follows. Consider the different blocks involved in the merging, *i.e.* $\mathcal{R}_{2n+b}$, $\mathcal{R}_n$, $\mathcal{R}_{n+b}$. We denote by $\mathbf{A}(\mathcal{R}_n)$ (resp.

$\mathbf{A}(\mathcal{R}_{n+b})$) the vector that is composed of all input random variables, intermediate random variables and output random variables involved in the computation of $\mathcal{R}_n$ (resp. $\mathcal{R}_{n+b}$). The vector of output variables of $\mathcal{R}_n$ (resp. $\mathcal{R}_{n+b}$) is denoted by $\mathbf{C}(\mathcal{R}_n)$ (resp. $\mathbf{C}(\mathcal{R}_{n+b})$). This can be done similarly for $\mathcal{R}_{2n+b}$ that performs the merging with the random vector $\underline{R}$. Namely, $\mathbf{A}(\mathcal{R}_{2n+b})$ is composed of $\mathbf{A}(\mathcal{R}_n)$, $\mathbf{A}(\mathcal{R}_{n+b})$, $\underline{R}$, and the output variables of the merging. This last vector will be referred to as the *adversary view* in the following. Finally, the vector of output variables of the merging will be denoted by $\mathbf{C}(\mathcal{R}_{2n+b})$.

For the sake of clarity, the next figure illusrates the merging performed in $\mathcal{R}_{2n+b}$ and displays all vectors previously defined that are involved in the computations. The inputs are at the bottom of the figure, while the merging is illustated at the top of the figure where is displayed the resulting output vector.



The result of the merging, *i.e.* the output $\mathbf{C}(\mathcal{R}_{2n+b})$ of $\mathcal{R}_{2n+b}$ is given by

$$\mathbf{C}(\mathcal{R}_{2n+b}) = (\mathbf{C}(\mathcal{R}_n) + \underline{R}\,,\, \mathbf{C}(\mathcal{R}_{n+b}) - \phi_b(\underline{R}))$$

where $b \in \{0,1\}$, $\phi_b(\underline{R}) = \underline{R}$, if $b = 0$ and $\phi_b(\underline{R}) = (\underline{R}, 0)$, if $b = 1$. Also, $\underline{R} \sim \mathcal{U}(G^n)$ and $\underline{R}$ is stochastically independent from any other random variable.

In what follows we consider possible adversary observations on the vectors displayed in the above figure during the merging, *i.e.* only some coordinates of these vectors are considered. The shrinked vectors of adversary observations we consider in the following are $\mathbf{A}(\mathcal{R}_n)_{I_\mathsf{L}}$, $\mathbf{A}(\mathcal{R}_{n+b})_{I_\mathsf{R}}$, $\underline{R}_{J^\mathsf{c}}$ and $\mathbf{C}(\mathcal{R}_{2n+b})_K$ with $J \subseteq \{1,\ldots,n\}$ and $K \subseteq \{1,\ldots,2n+b\}$. From this, we can build the adversary global view during the merging, *i.e.* ,

$$\mathbf{A}(\mathcal{R}_{2n+b})_I = \left(\mathbf{A}(\mathcal{R}_n)_{I_\mathsf{L}}, \mathbf{A}(\mathcal{R}_{n+b})_{I_\mathsf{R}}, \underline{R}_{J^\mathsf{c}}, \mathbf{C}(\mathcal{R}_{2n+b})_K\right)$$

where $I = I_{\mathsf{L}} \cup I_{\mathsf{R}} \cup J^{\mathsf{c}} \cup K$.

One of the main purpose of the following lemma is to analyze how the shapes of conditional distributions of vectors of adversary observations on $\mathcal{R}_n$ and $\mathcal{R}_{n+b}$ propagate to $\mathcal{R}_{2n+b}$ during the merging. Namely, considering the distributions of $\mathbf{C}(\mathcal{R}_n) \mid \mathbf{A}(\mathcal{R}_n)_{I_{\mathsf{L}}}$ and $\mathbf{C}(\mathcal{R}_n) \mid \mathbf{A}(\mathcal{R}_n)_{I_{\mathsf{R}}}$ before the merging, we analyze what is the shape of the distribution $\mathbf{C}(\mathcal{R}_{2n+b}) \mid \mathbf{A}(\mathcal{R}_{2n+b})_I$ after the merging. The last part of the lemma gives some conditions between the parameters of the distributions.

Lemma 2.2.1. *Consider the merging in $\mathcal{R}_{2n+b}$ of the left and right stochastically independent schemes respectively given by $\mathcal{R}_n$ and $\mathcal{R}_{n+b}$ as described above. Also, consider that the merging is performed following the insertion of random elements as given by step 18 of algorithm 2. Suppose that there exist applications $\rho_{1,i} : \mathsf{Im}\big(\mathbf{A}(\mathcal{R}_n)_{I_{\mathsf{L}}}\big) \mapsto G$ and $\rho_{2,i} : \mathsf{Im}\big(\mathbf{A}(\mathcal{R}_{n+b})_{I_{\mathsf{R}}}\big) \mapsto G$, respectively for $i = 1, \ldots, s$ and $i = s+1, \ldots, s+l$ such that for any $\underline{a_1} \in \mathsf{Im}\big(\mathbf{A}(\mathcal{R}_n)_{I_{\mathsf{L}}}\big)$ and any $\underline{a_2} \in \mathsf{Im}\big(\mathbf{A}(\mathcal{R}_{n+b})_{I_{\mathsf{R}}}\big)$,*

$$\mathbf{C}(\mathcal{R}_n) \mid \mathbf{A}(\mathcal{R}_n)_{I_{\mathsf{L}}} = \underline{a_1} \sim \mathcal{U}\left(\textstyle\sum_{i=1}^{s} S_{V_i}^{n}\big(\rho_{1,i}(\underline{a_1})\big)\right)$$

and

$$\mathbf{C}(\mathcal{R}_{n+b}) \mid \mathbf{A}(\mathcal{R}_{n+b})_{I_{\mathsf{R}}} = \underline{a_2} \sim \mathcal{U}\left(\textstyle\sum_{i=s+1}^{s+l} S_{V_i}^{n+b}\big(\rho_{2,i}(\underline{a_2})\big)\right),$$

where $V_1, \ldots, V_s$ is a partition of $\{1, \ldots, n\}$ and $V_{s+1}, \ldots, V_{s+l}$ is a partition of $\{1, \ldots, n+b\}$. Then,

- *There exist applications $\rho_i : \mathsf{Im}\big(\mathbf{A}(\mathcal{R}_{2n+b})_I\big) \mapsto G$ and a partition $X'_1, \ldots, X'_{m'}$ of $\{1, \ldots, 2n+b\}$ such that for any $\underline{a} \in \mathsf{Im}\big(\mathbf{A}(\mathcal{R}_{2n+b})_I\big)$*

$$\mathbf{C}(\mathcal{R}_{2n+b}) \mid \mathbf{A}(\mathcal{R}_{2n+b})_I = \underline{a} \sim \mathcal{U}\left(\textstyle\sum_{i=1}^{m'} S_{X'_i}^{2n+b}\big(\rho_i(\underline{a})\big)\right).$$

- *Assuming that $\#I < (2n+b)/2$. Also, assuming that if $\#I_{\mathsf{L}} < n/2$ (resp. $\#I_{\mathsf{R}} < (n+b)/2$), there exists $i \in \{1, \ldots, s\}$ (resp. $j \in \{1, \ldots, l\} + s$) such that $\#V_i \geq n - \#I_{\mathsf{L}}$ (resp. $\#V_j \geq n + b - \#I_{\mathsf{R}}$). Then, there exists $i \in \{1, \ldots, m'\}$ such that $\#X'_i \geq 2n + b - \#I$.*

Proof. The first part of the proof consists in proving that the shape of conditional distributions of $\mathcal{R}_n$ and $\mathcal{R}_{n+b}$ propagate to $\mathcal{R}_{2n+b}$.

Consider $V_1, \ldots, V_s$ a partition of $\{1, \ldots, n\}$ and $V_{s+1}, \ldots, V_{s+l}$ a partition of $\{1, \ldots, n+b\}$. According to the hypothesis and to the third statement of proposition 4

$$\pi_{V_i}(\mathbf{C}(\mathcal{R}_n)) \quad | \, \mathbf{A}(\mathcal{R}_n)_{I_{\mathsf{L}}} = \underline{a_1} \quad \sim \mathcal{U}\Big(S_{V_i}^{n}\big(\rho_{1,i}(\,\underline{a_1}\,)\big)\Big),$$

for all $i \in \{1, \ldots, s\}$. We also have

$$\pi_{V_i}(\mathbf{C}(\mathcal{R}_{n+b})) \, | \, \mathbf{A}(\mathcal{R}_{n+b})_{I_{\mathsf{R}}} = \underline{a_2} \sim \mathcal{U}\Big(S_{V_i}^{n+b}(\rho_{2,i}(\,\underline{a_2}\,))\Big),$$

for any $i \in \{s+1, \ldots, s+l\}$. Also, the random variables $\pi_{V_i}(\mathbf{C}(\mathcal{R}_n)) \mid \mathbf{A}(\mathcal{R}_n)_{I_{\mathsf{L}}} = \underline{a_1}$ are stochastically independent as well as the random variables $\pi_{V_i}(\mathbf{C}(\mathcal{R}_{n+b})) \mid \mathbf{A}(\mathcal{R}_{n+b})_{I_{\mathsf{R}}} = \underline{a_2}$.

Also, the vectors

$$\Big(\pi_{V_1}(\mathbf{C}(\mathcal{R}_n)), \ldots, \pi_{V_s}(\mathbf{C}(\mathcal{R}_n)),\, \mathbf{A}(\mathcal{R}_n)_{I_{\mathsf{L}}}\Big)$$

and

$$\Big(\pi_{V_{s+1}}(\mathbf{C}(\mathcal{R}_{n+b})), \ldots, \pi_{V_{s+l}}(\mathbf{C}(\mathcal{R}_{n+b})),\, \mathbf{A}(\mathcal{R}_{n+b})_{I_{\mathsf{R}}}\Big)$$

are stochastically independent because $\big(\mathbf{C}(\mathcal{R}_n), \mathbf{A}(\mathcal{R}_n)_{I_{\mathsf{L}}}\big)$ is stochastically independent of $(\mathbf{C}(\mathcal{R}_{n+b}), \mathbf{A}(\mathcal{R}_{n+b})_{I_{\mathsf{R}}})$. In order to cast the left and right schemes into a single scheme, let us now define

$$\underline{C_i} \triangleq \begin{cases} \Big(\pi_{V_i}(\,\mathbf{C}(\mathcal{R}_n)\,)\,,\, \underbrace{0, \ldots, 0}_{n+b}\Big) & \text{if } i \in \{1, \ldots, s\}\,, \\ \Big(\underbrace{0, \ldots, 0}_{n}\,,\, \pi_{V_i}(\,\mathbf{C}(\mathcal{R}_{n+b})\,)\Big) & \text{if } i \in \{s+1, \ldots, s+l\}\,. \end{cases}$$

We have

$$\underline{C_i} \;\mid\; \mathbf{A}(\mathcal{R}_n)_{I_{\mathsf{L}}} = \underline{a_1} \quad \sim\; \mathcal{U}\Big(S_{V_i}^{2n+b}\big(\rho_{1,i}(\,\underline{a_1}\,)\big)\Big), \quad \text{for any } i \in \{1, \ldots, s\}$$

and

$$\underline{C_i} \;\mid\; \mathbf{A}(\mathcal{R}_{n+b})_{I_{\mathsf{R}}} = \underline{a_2} \;\sim\; \mathcal{U}\Big(S_{V_{i+n}}^{2n+b}\big(\rho_{2,i}(\,\underline{a_2}\,)\big)\Big), \quad \text{for any } i \in \{s+1, \ldots, s+l\}\,.$$

According to proposition 5, for any $i = \{1, \ldots, s\}$

$$\underline{C_i} \;\mid\; \mathbf{A}(\mathcal{R}_n)_{I_{\mathsf{L}}} = \underline{a_1} \;\sim\; \underline{C_i} \;\mid\; (\mathbf{A}(\mathcal{R}_n)_{I_{\mathsf{L}}} = \underline{a_1}\,;\, \mathbf{A}(\mathcal{R}_{n+b})_{I_{\mathsf{R}}} = \underline{a_2})$$

and for any $\{s+1, \ldots, s+l\}$,

$$\underline{C_i} \;\mid\; \mathbf{A}(\mathcal{R}_{n+b})_{I_{\mathsf{R}}} = \underline{a_2} \;\sim\; \underline{C_i} \;\mid\; (\mathbf{A}(\mathcal{R}_n)_{I_{\mathsf{L}}} = \underline{a_1}\,;\, \mathbf{A}(\mathcal{R}_{n+b})_{I_{\mathsf{R}}} = \underline{a_2})\,.$$

Also, the random variables $\underline{C_i} \mid (\mathbf{A}(\mathcal{R}_n)_{I_\mathsf{L}}; \mathbf{A}(\mathcal{R}_{n+b})_{I_\mathsf{R}})$ are stochastically independent. Define

$$g_i(\underline{a_1}, \underline{a_2}) = \begin{cases} \rho_{1,i}(\underline{a_1}) & \text{if } i \in \{1, \ldots, s\}, \\ \rho_{2,i}(\underline{a_2}) & \text{if } i \in \{s+1, \ldots, s+l\} \end{cases}$$

and

$$V'_i = \begin{cases} V_i & \text{if } i \in \{1, \ldots, s\}, \\ n + V_i & \text{if } i \in \{s+1, \ldots, s+l\}. \end{cases} \tag{2.7}$$

We have

$$\underline{C_i} \mid (\mathbf{A}(\mathcal{R}_n)_{I_\mathsf{L}} = \underline{a_1}; \mathbf{A}(\mathcal{R}_{n+b})_{I_\mathsf{R}} = \underline{a_2}) \sim \mathcal{U}\left(S_{V'_i}^{2n+b}\big(g_i(\underline{a_1}, \underline{a_2})\big)\right)$$

and $V'_1, \ldots, V'_s$ is a partition of $\{1, \ldots, n\}$ and $V'_{s+1}, \ldots, V'_{s+l}$ is a partition of $\{1, \ldots, n+b\} + n$.

Denote by J the set of indices belonging to $\{1, \ldots, n\}$ such that $\underline{R}_J$ is not in $\mathbf{A}(\mathcal{R}_{2n+b})_I$. Since $\underline{R}$ and $\big(\mathbf{A}(\mathcal{R}_n)_{I_\mathsf{L}}, \mathbf{C}(\mathcal{R}_n), \mathbf{A}(\mathcal{R}_{n+b})_{I_\mathsf{R}}, \mathbf{C}(\mathcal{R}_{n+b})\big)$ are stochastically independent, then according to proposition 5

$$\underline{C_i} \mid (\mathbf{A}(\mathcal{R}_n)_{I_\mathsf{L}} = \underline{a_1}; \mathbf{A}(\mathcal{R}_{n+b})_{I_\mathsf{R}} = \underline{a_2})$$

is distributed as

$$\underline{C_i} \mid \big(\mathbf{A}(\mathcal{R}_n)_{I_\mathsf{L}} = \underline{a_1}; \mathbf{A}(\mathcal{R}_{n+b})_{I_\mathsf{R}} = \underline{a_2}; \underline{R}_{J^\mathsf{c}} = \underline{r}_{J^\mathsf{c}}\big) \tag{2.8}$$

and the random variables $\underline{C_i} \mid \big(\mathbf{A}(\mathcal{R}_n)_{I_\mathsf{L}} = \underline{a_1}; \mathbf{A}(\mathcal{R}_{n+b})_{I_\mathsf{R}} = \underline{a_2}; \underline{R}_{J^\mathsf{c}} = \underline{r}_{J^\mathsf{c}}\big)$ are stochastically independent. Remember that we aim at evaluating the distribution of

$$\mathbf{C}(\mathcal{R}_{2n+b}) \mid \mathbf{A}(\mathcal{R}_{2n+b})_I = \underline{a} \tag{2.9}$$

where $\underline{a} \in \mathsf{Im}\,(\mathbf{A}(\mathcal{R}_{2n+b})_I)$ and

$$\mathbf{A}(\mathcal{R}_{2n+b})_I = \Big(\mathbf{A}(\mathcal{R}_n)_{I_\mathsf{L}}, \mathbf{A}(\mathcal{R}_{n+b})_{I_\mathsf{R}}, \underline{R}_{J^\mathsf{c}}, \mathbf{C}(\mathcal{R}_{2n+b})_K\Big).$$

Let us now evaluate the distribution of

$$\mathbf{C}(\mathcal{R}_{2n+b}) \;\Big|\; \Big(\mathbf{A}(\mathcal{R}_n)_{I_\mathsf{L}} = \underline{a_1}; \mathbf{A}(\mathcal{R}_{n+b})_{I_\mathsf{R}} = \underline{a_2}; \underline{R}_{J^\mathsf{c}} = \underline{r}_{J^\mathsf{c}}\Big) \tag{2.10}$$

or equivalently the distribution of

$$\sum_{i=1}^{s+l} \underline{C_i} + \sum_{i=1}^{n} \left(\tau_i^{2n+b}(\underline{R}_i) - \tau_{i+n}^{2n+b}(\underline{R}_i)\right) \;\Big|\; \underbrace{\Big(\mathbf{A}(\mathcal{R}_n)_{I_\mathsf{L}} = \underline{a_1}; \mathbf{A}(\mathcal{R}_{n+b})_{I_\mathsf{R}} = \underline{a_2}; \underline{R}_{J^\mathsf{c}} = \underline{r}_{J^\mathsf{c}}\Big)}_{E(\underline{a_1}, \underline{a_2}, \underline{r}_{J^\mathsf{c}})}$$

and where the application $\tau^{2n+b}_{i\in\{1,\dots,n\}}$ is as in definition 3. In what follows, the parameters of E will be omitted for the sake of clarity. This equivalently gives

$$\sum_{i=1}^{s+l}\underline{C_i}+\sum_{i\in J}\left(\tau_i^{2n+b}(\underline{R_i})-\tau_{i+n}^{2n+b}(\underline{R_i})\right)+\sum_{i\in J^c}\left(\tau_i^{2n+b}(\underline{R_i})-\tau_{i+n}^{2n+b}(\underline{R_i})\right)\ \Big|\ E\,.$$

By replacing some random variables by their values, we get

$$\sum_{i=1}^{s+l}\underline{C_i}+\sum_{i\in J}\left(\tau_i^{2n+b}(\underline{R_i})-\tau_{i+n}^{2n+b}(\underline{R_i})\right)+\sum_{i\in J^c}\left(\tau_i^{2n+b}(\underline{r_i})-\tau_{i+n}^{2n+b}(\underline{r_i})\right)\ \Big|\ E\,.$$

Applying proposition 7, there exists a partition $X_1,\dots,X_m$ of $\{1,\dots,2n+b\}$ which is

$$X_i=\bigcup_{j\in S_i}V'_j \text{ for any } i\in\{1,\dots,m\}$$

and $S_1,\dots,S_m$ is a partition of $\{1,\dots,s+l\}$. Hence (2.10) follows the same distribution as

$$\sum_{j=1}^{m}\Big(\sum_{i\in S_j}\underline{C_i}\Big)+\sum_{j=1}^{m}\left(\sum_{i\in J\cap X_j}\left(\tau_i^{2n+b}(\underline{R_i})-\tau_{i+n}^{2n+b}(\underline{R_i})\right)+\sum_{i\in J^c}\left(\tau_i^{2n+b}(\underline{r_i})-\tau_{i+n}^{2n+b}(\underline{r_i})\right)\right)\ \Big|\ E\,.$$

Also,

$$\sum_{i\in J^c}\tau_i^{2n+b}(\underline{r_i})-\tau_{i+n}^{2n+b}(\underline{r_i})=\sum_{i\in J^c}\tau_i^{2n+b}(\underline{r_i})-\sum_{i\in J^c+n}\tau_i^{2n+b}\big(\underline{r}_{i-n}\big)$$

which is also

$$\sum_{j=1}^{m}\left(\sum_{i\in J^c\cap X_j}\tau_i^{2n+b}(\underline{r_i})-\sum_{i\in(J^c+n)\cap X_j}\tau_i^{2n+b}(\underline{r}_{i-n})\right).$$

Therefore, (2.10) follows the same distribution as

$$\sum_{j=1}^{m}\left(\sum_{i\in S_j}\underline{C_i}+\sum_{i\in J\cap X_j}\tau_i^{2n+b}(\underline{R_i})-\tau_{i+n}^{2n+b}(\underline{R_i})+\sum_{i\in J^c\cap X_j}\tau_i^{2n+b}(\underline{r_i})-\sum_{i\in(J^c+n)\cap X_j}\tau_i^{2n+b}(\underline{r}_{i-n})\right)\ \Big|\ E\,.$$

Applying proposition 7, we know that there exist $\{r_2,\dots,r_{k_j}\}\subseteq J\cap X_j$ satisfying some properties and we may say that (2.10) follows the same distribution as

$$\sum_{j=1}^{m}\left(\sum_{i\in S_j}\underline{C_i}+\sum_{i\in\{r_2,\dots,r_{k_j}\}}\left(\tau_i^{2n+b}(\underline{R_i})-\tau_{i+n}^{2n+b}(\underline{R_i})\right)+\sum_{i\in(J\cap X_j)\setminus\{r_2,\dots,r_{k_j}\}}\left(\tau_i^{2n+b}(\underline{R_i})-\tau_{i+n}^{2n+b}(\underline{R_i})\right)\right.$$
$$\left.+\sum_{i\in J^c\cap X_j}\tau_i^{2n+b}(\underline{r_i})-\sum_{i\in(J^c+n)\cap X_j}\tau_i^{2n+b}(\underline{r}_{i-n})\right)\ \Big|\ E\,.$$

Remembering that (2.8) is uniformly distributed over $S_{V_i'}^{2n+b}\big(g_i(\underline{a_1},\underline{a_2})\big)$ and applying recursively proposition 3 (merging), we have

$$\sum_{i\in S_j}\underline{C_i}+\sum_{i\in\{r_2,\ldots,r_{k_j}\}}\big(\tau_i^{2n+b}(\underline{R_i})-\tau_{i+n}^{2n+b}(\underline{R_i})\big)\;\Big|\;E\;\sim\;\mathcal{U}\left(S^{2n+b}_{\bigcup\limits_{i\in S_j}V_i'}\Big(\sum_{i\in S_j}g_i(\underline{a_1},\underline{a_2})\Big)\right)$$

or equivalently

$$\sum_{i\in S_j}\underline{C_i}+\sum_{i\in\{r_2,\ldots,r_{k_j}\}}\big(\tau_i^{2n+b}(\underline{R_i})-\tau_{i+n}^{2n+b}(\underline{R_i})\big)\;\Big|\;E\;\sim\;\mathcal{U}\left(S^{2n+b}_{X_j}\Big(\sum_{i\in S_j}g_i(\underline{a_1},\underline{a_2})\Big)\right).$$

According to proposition 7 if $i\in J\cap X_j$ then $i+n\in X_j$. Therefore, according to proposition 1

$$\sum_{i\in S_j}\underline{C_i}+\sum_{i\in\{r_2,\ldots,r_{k_j}\}}\big(\tau_i^{2n+b}(\underline{R_i})-\tau_{i+n}^{2n+b}(\underline{R_i})\big)+\sum_{i\in(J\cap X_j)\setminus\{r_2,\ldots,r_{k_j}\}}\big(\tau_i^{2n+b}(\underline{R_i})-\tau_{i+n}^{2n+b}(\underline{R_i})\big)\;\Big|\;E$$

is uniformly distributed over $S_{X_j}^{2n+b}\Big(\sum_{i\in S_j}g_i(\underline{a_1},\underline{a_2})\Big)$. Also, according to proposition 2

$$\sum_{i\in S_j}\underline{C_i}+\sum_{i\in\{r_2,\ldots,r_{k_j}\}}\big(\tau_i^{2n+b}(\underline{R_i})-\tau_{i+n}^{2n+b}(\underline{R_i})\big)$$
$$+\sum_{i\in(J\cap X_j)\setminus\{r_2,\ldots,r_{k_j}\}}\big(\tau_i^{2n+b}(\underline{R_i})-\tau_{i+n}^{2n+b}(\underline{R_i})\big)+\sum_{i\in J^{\mathsf{c}}\cap X_j}\tau_i^{2n+b}(\underline{r_i})-\sum_{i\in(J^{\mathsf{c}}+n)\cap X_j}\tau_i^{2n+b}(\underline{r_i-n})\;\Big|\;E$$

is uniformly distributed over

$$S_{X_j}^{2n+b}\Big(\sum_{i\in S_j}g_i(\underline{a_1},\underline{a_2})+\sum_{i\in J^{\mathsf{c}}\cap X_j}\underline{r_i}-\sum_{i\in(J^{\mathsf{c}}+n)\cap X_j}\underline{r_i-n}\Big).$$

Finally, summing the different (independent) random vectors and remembering that the condition E is $(\mathbf{A}(\mathcal{R}_n)_{I_{\mathsf{L}}}=\underline{a_1};\mathbf{A}(\mathcal{R}_{n+b})_{I_{\mathsf{R}}}=\underline{a_2};\underline{R}_{J^{\mathsf{c}}}=\underline{r}_{J^{\mathsf{c}}})$, we have that (2.10) is uniformly distributed over

$$\sum_{j=1}^{m}S_{X_j}^{2n+b}\Big(q_j(\underline{a_1},\underline{a_2},\underline{r}_{J^{\mathsf{c}}})\Big)$$

where

$$q_j(\underline{a_1},\underline{a_2},\underline{r}_{J^{\mathsf{c}}})=\sum_{i\in S_j}g_i(\underline{a_1},\underline{a_2})+\sum_{i\in J^{\mathsf{c}}\cap X_j}\underline{r_i}-\sum_{i\in(J^{\mathsf{c}}+n)\cap X_j}\underline{r_i-n}.$$

Applying proposition 6, we have that

$$\mathbf{C}(\mathcal{R}_{2n+b}) \mid \Big(\mathbf{A}(\mathcal{R}_n)_{I_{\mathsf{L}}} = \underline{a_1}\,;\, \mathbf{A}(\mathcal{R}_{n+b})_{I_{\mathsf{R}}} = \underline{a_2}\,;\, \underline{R}_{J^{\mathsf{c}}} = \underline{r}_{J^{\mathsf{c}}}\,;\, \mathbf{C}(\mathcal{R}_{2n+b})_K = \underline{a}_K\Big)$$

is uniform over

$$\sum_{j=1}^{m} S^{2n+b}_{X_j \setminus K}\left(q_j(\underline{a_1}, \underline{a_2}, \underline{r}_{J^{\mathsf{c}}}) - \sum_{i \in X_j \cap K} \underline{a}_i\right) + \pi_K(\,\underline{a}\,)$$

where $S^{2n+b}_{\emptyset} = \emptyset$. Considering (2.9) and remembering that $\underline{a} \in \mathsf{Im}\big(\mathbf{A}(\mathcal{R}_{2n+b})_I\big)$, we therefore have

$$\mathbf{C}(\mathcal{R}_{2n+b}) \mid \mathbf{A}(\mathcal{R}_{2n+b})_I = \underline{a} \;\sim\; \mathcal{U}\left(\sum_{j=1}^{m} S^{2n+b}_{X_j \setminus K}\big(\rho_j(\underline{a})\big) + \sum_{i \in K} S^{2n+b}_i(\underline{a}_i)\right)$$

where the ρ_j's are defined in the statement. This concludes the first part of the result.

The second part of the proof consists in dealing with the parameters of the above distributions.

Assume that $\#I < (2n+b)/2$. Also, assume that if $\#I_{\mathsf{L}} < n/2$ (resp. $\#I_{\mathsf{R}} < (n+b)/2$), there exists $i \in \{1, \ldots, s\}$ (resp. $j \in \{1, \ldots, l\} + s$) such that $\#V_i \geq n - \#I_{\mathsf{L}}$ (resp. $\#V_j \geq n + b - \#I_{\mathsf{R}}$). Let us prove that there exists $i \in \{1, \ldots, m'\}$ such that $\#(X_i \setminus K) \geq 2n + b - \#I$.

First, consider the case $b = 0$. Suppose that $\#I < n$, *i.e.* $\#I_{\mathsf{L}} + \#I_{\mathsf{R}} + \#J^{\mathsf{c}} + \#K < n$. Also, consider the case $\#I_{\mathsf{L}} < n/2$ and $\#I_{\mathsf{R}} < n/2$.

By hypothesis, it exists $i_0 \in \{1, \ldots, s\}$ and it exists $j_0 \in \{1, \ldots, l\} + s$ such that $\#V'_{i_0} \geq n - \#I_{\mathsf{L}}$ and $\#V'_{j_0} \geq n - \#I_{\mathsf{R}}$ where V'_{i_0} and V'_{j_0} are defined as in (2.7).

If $V'_{i_0}, V'_{j_0} \subseteq X_i$ for a fixed $i \in \{1, \ldots, m\}$, then $\#X_i \geq \#V'_{i_0} + \#V'_{j_0}$. Furthermore, $\#V'_{i_0} + \#V'_{j_0} \geq 2n - (\#I_{\mathsf{L}} + \#I_{\mathsf{R}})$ and therefore

$$\#X_i \geq 2n - (\#I_{\mathsf{L}} + \#I_{\mathsf{R}} + \#J^{\mathsf{c}})\,.$$

Remembering that $\#(X_i \setminus K) \geq \#X_i - \#K$, we have

$$\begin{aligned} \#(X_i \setminus K) &\geq 2n - (\#I_{\mathsf{L}} + \#I_{\mathsf{R}} + \#J^{\mathsf{c}} + \#K) \\ &\geq 2n - \#I. \end{aligned}$$

If V'_{i_0} and V'_{j_0} belong to different X_i's, then it exist $k, r \in \{1, \ldots, m\}$ such that $k \neq r$ and $V'_{i_0} \subseteq X_k$ and $V'_{j_0} \subseteq X_r$. Since $X_1, \ldots, X_m$ is a partition of $\{1, \ldots, 2n+b\}$

and since $k \neq r$, we have $X_r \cap X_k \neq \emptyset$. According to the definition of the relation of equivalence, $V'_{i_0} \cap (V'_{j_0} - n) \subseteq J^{\mathsf{c}}$. Also,

$$n = \#\left(V'_{i_0} \cup (V'_{j_0} - n)\right)^c + \#V'_{i_0} + \#\left(V'_{j_0} - n\right) - \#\left(V'_{i_0} \cap (V'_{j_0} - n)\right)$$

which implies that

$$\#\left(V'_{i_0} \cap (V'_{j_0} - n)\right) \geq \#V'_{i_0} + \#\left(V'_{j_0} - n\right) - n.$$

Therefore

$$\#J^{\mathsf{c}} \geq \#\left(V'_{i_0} \cap (V'_{j_0} - n)\right) \geq \#V'_{i_0} + \#\left(V'_{j_0} - n\right) - n.$$

Also, $\#V'_{i_0} \geq n - \#I_{\mathsf{L}}$ and $\#V'_{j_0} \geq n - \#I_{\mathsf{R}}$. It follows that $\#J^{\mathsf{c}} \geq n - (\#I_{\mathsf{L}} + \#I_{\mathsf{R}})$. Finally, we have $\#J^{\mathsf{c}} < n - (\#I_{\mathsf{L}} + \#I_{\mathsf{R}}) - \#K$ by hypothesis. This leads to a contradiction and therefore this case is not possible.

Let us now address the case where $\#I_{\mathsf{L}} \geq n/2$ and $\#I_{\mathsf{R}} < n/2$. By hypothesis, there exists $j_0 \in \{1, \ldots, l\} + s$ such that $\#V'_{j_0} \geq n - \#I_{\mathsf{R}}$. Let $k \in \{1, \ldots, k\}$ such that $V'_{j_0} \subseteq X_k$. Let us begin by deriving a lower bound on $\#X_k$.

Observe that for any $x \in V'_{j_0} \cap J$, $x \in X_k$ because $V'_{j_0} \subseteq X_k$. By proposition 7 we have that $x+n \in X_k$ and that $x+n \notin V'_{j_0}$. Therefore, noting that $V'_{j_0} \cap J = V'_{j_0} \setminus J^{\mathsf{c}}$

$$\#X_k \geq \#V'_{j_0} + \#(V'_{j_0} \setminus J^{\mathsf{c}})$$

with $\#(V'_{j_0} \setminus J^{\mathsf{c}}) \geq \#V'_{j_0} - \#J^{\mathsf{c}}$ and therefore

$$\#X_k \geq 2\#V'_{j_0} - \#J^{\mathsf{c}}\,.$$

Also, by hypothesis, $\#V'_{j_0} \geq n - \#I_{\mathsf{R}}$ and $\#I_{\mathsf{R}} < \#I_{\mathsf{L}}$ which gives us

$$\begin{aligned}\#X_k &\geq 2(n - \#I_{\mathsf{R}}) - \#J^{\mathsf{c}} \\ &\geq 2n - (\#I_{\mathsf{R}} + \#I_{\mathsf{L}} + \#J^{\mathsf{c}})\,.\end{aligned}$$

Therefore,

$$\#(X_k \setminus K) \geq 2n - (\#I_{\mathsf{R}} + \#I_{\mathsf{L}} + \#J^{\mathsf{c}} + \#K)\,.$$

Finally, it follows that $\#(X_k \setminus K) \geq 2n - \#I$ which concludes this case. Note that the case $\#I_{\mathsf{L}} < n/2$ and $\#I_{\mathsf{R}} \geq n/2$ is similar because of the symetry.

Let us now consider the case $b = 1$. Suppose that $\#I < (2n+1)/2$, *i.e.* $\#I_{\mathsf{L}} + \#I_{\mathsf{R}} + \#J^{\mathsf{c}} + \#K < (2n+1)/2$ or equivalently $\#I \leq n$, *i.e.* $\#I_{\mathsf{L}} + \#I_{\mathsf{R}} + \#J^{\mathsf{c}} + \#K \leq n$. Also, consider the case $\#I_{\mathsf{L}} < n/2$ and $\#I_{\mathsf{R}} < (n+1)/2$.

By hypothesis, there exists $i_0 \in \{1,\dots,s\}$ and there exists $j_0 \in \{1,\dots,l\}+s$ such that $\#V'_{i_0} \geq n - \#I_{\mathsf{L}}$ and $\#V'_{j_0} \geq (n+1) - \#I_{\mathsf{R}}$. Also, if $V'_{i_0}, V'_{j_0} \subseteq X_i$ for a fixed $i \in \{1,\dots,m\}$, then

$$\#X_i \geq \#V'_{i_0} + \#V'_{j_0}$$

which gives us that

$$\#X_i \geq (2n+1) - (\#I_{\mathsf{L}} + \#I_{\mathsf{R}} + \#J^{\mathsf{c}})\,.$$

Remembering that $\#(X_i \setminus K) \geq \#X_i - \#K$ we have

$$\#(X_i \setminus K) \geq (2n+1) - (\underbrace{\#I_{\mathsf{L}} + \#I_{\mathsf{R}} + \#J^{\mathsf{c}} + \#K}_{\#I})\,.$$

If V'_{i_0} and V'_{j_0} belong to different X_i's, then there is $k, r \in \{1,\dots,m\}$ such that $k \neq r$ and $V'_{i_0} \subseteq X_k$ and $V'_{j_0} \subseteq X_r$. According to the definition of the relation of equivalence, $V'_{i_0} \cap (V'_{j_0} - n) \subseteq J^{\mathsf{c}}$. Also,

$$n+1 = \#\left(V'_{j_0} \cup (V'_{j_0} - n)\right)^c + \#V'_{i_0} + \#\left(V'_{j_0} - n\right) - \#\left(V'_{i_0} \cap (V'_{j_0} - n)\right)\,.$$

This implies that

$$\begin{aligned}\#\left(V'_{i_0} \cap (V'_{j_0} - n)\right) &\geq \#V'_{i_0} + \#\left(V'_{j_0} - n\right) - (n+1)\\ &\geq \#V'_{i_0} + \#V'_{j_0} - (n+1)\,.\end{aligned}$$

Therefore,

$$\#J^{\mathsf{c}} \geq \#V'_{i_0} + \#V'_{j_0} - (n+1)\,.$$

By hypothesis, this means that $\#J^{\mathsf{c}} \geq n - (\#I_{\mathsf{L}} + \#I_{\mathsf{R}})$. However, we also have by hypothesis that

$$\#J^{\mathsf{c}} \leq n - (\#I_{\mathsf{L}} + \#I_{\mathsf{R}}) - \#K\,.$$

The only possible case is when $\#K = 0$ and $\#J^{\mathsf{c}} = n - (\#I_{\mathsf{L}} + \#I_{\mathsf{R}})$ or equivalently when $\#J = \#I_{\mathsf{L}} + \#I_{\mathsf{R}}$ because $\#J^{\mathsf{c}} = n - \#J$. Also, according to the last statement of proposition 7,

$$\#X_k \geq \#V'_{i_0} + \#\left(V'_{i_0} \cap J\right)$$

and

$$\#X_r \geq \#V'_{j_0} + \#\left(V'_{j_0} \cap (J+n)\right)\,.$$

It follows on the one hand that

$$\#X_k + \#J^{\mathsf{c}} \geq \#V'_{i_0} + \#\left(V'_{i_0} \cap J\right) + \#\left(V'_{i_0} \cap J^{\mathsf{c}}\right) \geq 2\#V'_{i_0}$$

since $J \cup J^{\mathsf{c}} = \{1, \dots n\}$.

On the other hand, we have

$$\begin{aligned}
\#X_r + \#J^{\mathsf{c}} &\geq \#V'_{j_0} + \#\left(V'_{j_0} \cap (J+n)\right) + \#\left(V'_{j_0} \cap (J^{\mathsf{c}}+n)\right) \\
&\geq \#V'_{j_0} + \#\left((V'_{j_0} - n) \cap (J \cup J^{\mathsf{c}})\right) \\
&\geq \#V'_{j_0} + \#\left((V'_{j_0} - n) \cap \{1, \dots, n\}\right) \\
&\geq \#V'_{j_0} + \#V'_{j_0} - 1 \\
&\geq 2\#V'_{j_0} - 1\,. \qquad (2.11)
\end{aligned}$$

It turns out that $\max(\#X_k, \#X_r) \geq (\#X_k + \#X_r)/2$ and therefore

$$\max(\#X_k, \#X_r) \geq \#V'_{i_0} + \#V'_{j_0} - \#J^{\mathsf{c}} - 1/2\,.$$

The hypothesis states that $\#V'_{i_0} \geq n - \#I_{\mathsf{L}}$ and $\#V'_{j_0} \geq n + 1 - \#I_{\mathsf{R}}$, giving

$$\max(\#X_k, \#X_r) \geq 2n + 1 - (\#I_{\mathsf{L}} + \#I_{\mathsf{R}} + \#J^{\mathsf{c}}) - 1/2\,.$$

Finally, because $\#K = 0$ we have $K = \emptyset$ and therefore $X_k \setminus K = X_k$ and $X_r \setminus K = X_r$. Therefore,

$$\begin{aligned}
\max(\#\left(X_k \setminus K\right), \#\left(X_r \setminus K\right)) &\geq 2n + 1/2 - (\#I_{\mathsf{L}} + \#I_{\mathsf{R}} + \#J^{\mathsf{c}} + \#K) \\
&\geq 2n + 1 - (\#I_{\mathsf{L}} + \#I_{\mathsf{R}} + \#J^{\mathsf{c}} + \#K)
\end{aligned}$$

because $\#\left(X_k \setminus K\right)$ and $\#\left(X_r \setminus K\right)$ are integers.

Let us address the case where $\#I_{\mathsf{L}} < n/2$ and $\#I_{\mathsf{R}} \geq (n+1)/2$. By hypothesis, $\#V'_{i_0} \geq n - \#I_{\mathsf{L}}$. Also,

$$\#X_k \geq \#V'_{i_0} + \#\left(V'_{i_0} \cap J\right)\,.$$

Therefore,

$$\#X_k + \#J^{\mathsf{c}} \geq \#V'_{i_0} + \#\left(V'_{i_0} \cap J\right) + \#\left(V'_{i_0} \cap J^{\mathsf{c}}\right) \geq 2\#V'_{i_0}\,.$$

It follows that,

$$\#X_k \geq 2(n - \#I_{\mathsf{L}}) - \#J^{\mathsf{c}}\,. \qquad (2.12)$$

Also, $\#I_{\mathsf{L}} < n/2$. Therefore, $\#I_{\mathsf{L}} \leq (n-1)/2$. Moreover, $\#I_{\mathsf{R}} \geq (n+1)/2$ implies that $\#I_{\mathsf{R}} - 1 \geq (n-1)/2$. We conclude that $\#I_{\mathsf{L}} \leq \#I_{\mathsf{R}} - 1$. Relation (2.12) may therefore be written as

$$\#X_k \geq 2n - (\#I_{\mathsf{L}} + \#I_{\mathsf{L}} + \#J^{\mathsf{c}})$$

which gives

$$\#X_k \geq 2n+1-(\#I_{\mathsf{R}}+\#I_{\mathsf{L}}+\#J^{\mathsf{c}})\,.$$

Remembering that $\#\left(X_k \setminus K\right) \geq \#X_k - \#K$, we have

$$\#(X_k \setminus K) \geq 2n+1-(\#I_{\mathsf{R}}+\#I_{\mathsf{L}}+\#J^{\mathsf{c}}+\#K)\,.$$

Let us finish the proof by addressing the last case, *i.e.* $\#I_{\mathsf{L}} \geq n/2$ and $\#I_{\mathsf{R}} < (n+1)/2$. We have $\#I_{\mathsf{R}} \leq \#I_{\mathsf{L}}$. By hypothesis, $\#V'_{j_0} \geq (n+1) - \#I_{\mathsf{R}}$. Again, $\#X_k \geq \#V'_{j_0} + \#\left(V'_{j_0} \cap J\right)$ and therefore as in (2.11), we have

$$\#X_k + \#J^{\mathsf{c}} \geq 2\#V'_{j_0} - 1 \geq 2\cdot(n+1-\#I_{\mathsf{R}}) - 1$$

This gives

$$\begin{aligned}\#X_k &\geq 2n+1-(\#I_{\mathsf{R}}+\#I_{\mathsf{R}}+\#J^{\mathsf{c}})\\ &\geq 2n+1-(\#I_{\mathsf{R}}+\#I_{\mathsf{L}}+\#J^{\mathsf{c}})\,.\end{aligned}$$

Finally, remembering that $\#\left(X_k \setminus K\right) \geq \#X_k - \#K$, we have

$$\#\left(X_k \setminus K\right) \geq 2n+1-(\#I_{\mathsf{R}}+\#I_{\mathsf{L}}+\#J^{\mathsf{c}}+\#K)$$

which concludes the last case and hence the proof as well. It follows that (2.11) may be rewritten as

$$\mathbf{C}(\mathcal{R}_{2n+b}) \;\mid \mathbf{A}(\mathcal{R}_{2n+b})_I = \underline{a} \sim \mathcal{U}\left(\sum_{i=1}^{m'} S_{X'_i}^{2n+b}\big(\rho_i(\underline{a})\big)\right)$$

where the (X'_i)'s are the non-empty sets $(X_i \setminus K)$'s and the element of K. Note that they form a partition of $\{1,\ldots,2n+b\}$ because the X_i's form a partition of the same set. □

Remember that the previous Lemma was associated to the insertion of the random elements in algorithm 2 following the first alternative. Let us now prove a lemma that is associated to the second alternative for the insertion of the random elements (see step 20 of the $\mathcal{R}_n$ algorithm).

Lemma 2.2.2. *The statement of Lemma* 2.2.1 *applies also when the design of* $\mathcal{R}_{2n+b}$ *from* $\mathcal{R}_n$ *and* $\mathcal{R}_{n+b}$ *is obtained by the construction :*

$$\mathbf{C}(\mathcal{R}_{2n+b}) = (\mathbf{C}(\mathcal{R}_n) + \underline{R}, \mathbf{C}(\mathcal{R}_{n+b}) - \phi_b^\star(\underline{R})) \leftarrow \mathcal{R}_{2n+b}^\star$$

where

$$\phi_b^\star = \begin{cases}\underline{R} & \text{if } b=0\\ (0,\underline{R}) & \text{if } b=1\,.\end{cases}$$

Proof. **(Sketch).** Define the two permutations π_b for $b \in \{0,1\}$ such that $\pi_b \circ \phi_b = \phi_b^\star$, where ϕ_b is defined in Lemma 2.2.1. Note that π_0 is the identity on G^n. Observe that

$$\begin{aligned}(\mathbf{C}(\mathcal{R}_n) + \underline{R}, \mathbf{C}(\mathcal{R}_{n+b}) - \phi_b^\star(\underline{R})) &= \big(\mathbf{C}(\mathcal{R}_n) + \underline{R}, \mathbf{C}(\mathcal{R}_{n+b}) - \pi_b(\phi_b(\underline{R}))\big) \\ &= \big(\mathbf{C}(\mathcal{R}_n) + \underline{R}, \pi_b(\pi_b^{-1}(\mathbf{C}(\mathcal{R}_{n+b})) - \phi_b(\underline{R}))\big) \\ &= (\mathsf{Id}, \pi_b)\big(\mathbf{C}(\mathcal{R}_n) + \underline{R}, \pi_b^{-1}(\mathbf{C}(\mathcal{R}_{n+b})) - \phi_b(\underline{R})\big) \\ &= (\mathsf{Id}, \pi_b)\big(\mathbf{C}(\mathcal{R}_n) + \underline{R}, \mathbf{C}(\pi_b^{-1}(\mathcal{R}_{n+b})) - \phi_b(\underline{R})\big)\end{aligned}$$

where $\pi_b^{-1}(\mathcal{R}_{n+b})$ denotes the composition of the permutation π_b^{-1} after the scheme $\mathcal{R}_{n+b}$. Note that if $(V_i)_{i\in\{1,\dots,l\}+s}$ is a partition of $\{1,\dots,n+b\}$ then $(\pi_b^{-1}(V_i))_{i\in\{1,\dots,l\}+s}$ is also a partition of the same set. Furthermore, the cardinality of any subset of $\{1,\dots,n+b\}$ is preserved under the permutation π_b. It turns out that up to a redefinition of some spaces, $\mathcal{R}_{n+b}$ satisfies the hypothesis of Lemma 2.2.1 if and only if $\pi_b^{-1}(\mathcal{R}_{n+b})$ does so. Therefore, Lemma 2.2.1 applies to the scheme defined by

$$\big(\mathbf{C}(\mathcal{R}_n) + \underline{R}, \mathbf{C}(\pi_b^{-1}(\mathcal{R}_{n+b})) - \phi_b(\underline{R})\big)$$

and provides the shape and constraints on the parameters of the above vector conditional to the view. The above arguments on the invariance properties related to a permutation extends the statement of Lemma 2.2.1 to

$$(\mathsf{Id}, \pi_b)\big(\mathbf{C}(\mathcal{R}_n) + \underline{R}, \mathbf{C}(\pi_b^{-1}(\mathcal{R}_{n+b})) - \phi_b(\underline{R})\big)\,.$$

The result follows. □

Analysis of the complete execution of $\mathcal{R}_n$.

The following proposition will be used to prove the main result regarding the recursive $\mathcal{R}_n$.

Proposition 8. *For any $n \in]2^k, 2^{k+1}]$, there exist $n_0, m_0 \in [2^{k-1}, 2^k]$ such that $n = m_0 + n_0$ with $m_0 \le n_0 \le m_0 + 1$.*

Proof. Consider n even. By hypothesis, $n \in]2^k, 2^{k+1}]$. Equivalently, $n \in [2^k + 2, 2^{k+1}]$. Therefore $n/2 \in [2^{k-1} + 1, 2^k]$ or equivalently $n/2 \in]2^{k-1}, 2^k]$. In that case, $n = n/2 + n/2$ and $n/2 \in [2^{k-1}, 2^k]$.

Consider n odd. By hypothesis, $n \in]2^k, 2^{k+1}]$. Therefore, $n \in]2^k, 2^{k+1} - 1]$ and $n - 1 \in [2^k, 2^{k+1} - 2]$. It follows that $(n-1)/2 \in [2^{k-1}, 2^k - 1]$ and $(n+1)/2 \in [2^{k-1}, 2^k]$. Consequently, $n = (n-1)/2 + (n+1)/2$ with $(n-1)/2 \le (n+1)/2 \le (n-1)/2 + 1$. The result follows. □

Let us state our main theorem concerning the execution of $\mathcal{R}_n$. Remember that $\mathcal{R}_n$ (alg. 2) describes a class of algorithms. This is due to the degree of freedom given by steps 18 and 20 of the algorithm, which allow two different ways to insert random elements during the processing of any linear layer.

Theorem 2.2.3. *Let $n \geq 2$. For any set I such that $\#I < n/2$ and any $\underline{a} \in \mathrm{Im}\big(\mathbf{A}(\mathcal{R}_n)_I\big)$, there exists a partition $V_1, \ldots, V_k$ of $\{1, \ldots, n\}$ and applications $\rho_i : \mathrm{Im}\big(\mathbf{A}(\mathcal{R}_n)_I\big) \mapsto G$ for any $\underline{a} \in \mathrm{Im}(\mathbf{A}(\mathcal{R}_n)_I)$ and*

$$\mathbf{C}(\mathcal{R}_n) \mid \mathbf{A}(\mathcal{R}_n)_I = \underline{a} \ \sim \ \mathcal{U}\left(\sum_{i=1}^{k} S^n_{V_i}\big(\rho_i(\underline{a})\big)\right),$$

with

$$max_i \#V_i \geq n - \#I\,.$$

Proof. Let consider the case $n = 2$. By definition, $\mathbf{C}(\mathcal{R}_2) = (R, -R)$ with $R \sim \mathcal{U}(G)$. The only set I such that $\#I < 2/2$ is the empty set and we have

$$\mathbf{C}(\mathcal{R}_2) \mid \emptyset \sim \mathcal{U}\left(S^2_{\{1,2\}}(0)\right)$$

with $\#\{1,2\} \geq 2 - 0$ and $\{1,2\}$ is a partition of $\{1,2\}$. Note that for any I with $\#I > 0$, $\mathbf{A}(\mathcal{R}_2)_I$ is a function of R and we have for any $r \in G$

$$\mathbf{C}(\mathcal{R}_2) \mid R = r \ \sim \ \mathcal{U}\left(S^2_{\{1\}}(r) + S^2_{\{2\}}(-r)\right)$$

and $\{1\}, \{2\}$ is a partition of $\{1,2\}$.

Let consider the case $n = 3$. By definition, $\mathbf{C}(\mathcal{R}_3) = (R_1, R_2 - R_1, -R_2)$ with $(R_1, R_2) \sim \mathcal{U}(G \times G)$. For any set I such that $\#I < 3/2$, the components of the vector $\mathbf{A}(\mathcal{R}_3)_I$ are elements of $\{\emptyset, \{R_1\}, \{R_2\}, \{R_2 - R_1\}\}$ or bijective functions of those variables. Now consider the case $\#I = 0$. This means that $\mathbf{A}(\mathcal{R}_3)_I = \emptyset$. According to proposition 6,

$$\mathbf{C}(\mathcal{R}_3) \mid \emptyset \ \sim \ \mathcal{U}\left(S^3_{\{1,2,3\}}(0)\right)$$

and $3 = \max_i \#V_i \geq 3 - 0$. Next, consider the case $\#I = 1$. If $\mathbf{A}(\mathcal{R}_3)_I = (R_1)$ and if $\mathbf{A}(R_3)_I = (R_2)$ or a bijective function of this variable we have

$$\begin{cases} \mathbf{C}(\mathcal{R}_3) \mid R_1 = r_1 \ \sim \ \mathcal{U}\left(S^3_{\{1\}}(r_1) + S^3_{\{2,3\}}(-r_1)\right) \\ \mathbf{C}(\mathcal{R}_3) \mid R_2 = r_2 \ \sim \ \mathcal{U}\left(S^3_{\{1,2\}}(r_2) + S^3_{\{3\}}(-r_2)\right) \end{cases}$$

and $2 = \max_i \#V_i \geq 3 - 1$. Also, if $\mathbf{A}(\mathcal{R}_3)_I = (R_2 - R_1)$,

$$\mathbf{C}(\mathcal{R}_3) \mid (R_2 - R_1) = r \ \sim \ \mathcal{U}\left(S^3_{\{1,3\}}(-r) + S^3_{\{2\}}(r)\right)$$

and $2 = \max_i \#V_i \geq 3 - 1$. Consider now the case $\#I > 1$, then there exists a bijection between $\mathbf{A}(\mathcal{R}_3)_I$ and (R_1, R_2). We have

$$\mathbf{C}(\mathcal{R}_3) \mid (R_1 = r_1 \,;\, R_2 = r_2) \ \sim \ \mathcal{U}\left(S^3_{\{1\}}(r_1) + S^3_{\{2\}}(r_2 - r_1) + S^3_{\{3\}}(-r_2)\right)$$

and $\mathbf{C}(\mathcal{R}_3) \mid \mathbf{A}(\mathcal{R}_3)_I = \underline{a}$ follows a similar distribution up to a change of variable of r_1 and r_2. In summary,

$$\mathbf{C}(\mathcal{R}_3) \ \mid \ \mathbf{A}(\mathcal{R}_3)_I = \underline{a} \ \sim \ \mathcal{U}\left(\sum_{i=1}^{k} S^3_{V_i}\big(\rho_i(\underline{a})\big)\right)$$

for any I. Moreover $\max_i \#V_i \geq 3 - \#I$ if $\#I < 3/2$.

Applying lemma 2.2.1 to $\mathcal{R}_2$, we deduce that the result is correct for $n = 4$. Let us now define $\mathcal{P}(k)$ as :

The result is correct for any n such that $2 \leq n \leq 2^k$.

As proved above, $\mathcal{P}(2)$ is correct. Suppose $\mathcal{P}(k)$ is correct ($k \geq 2$) and let us prove that $\mathcal{P}(k+1)$ is then also correct. We still have to prove that the statement is correct for $n \in]2^k, 2^{k+1}]$. According to proposition 8, for any $n \in]2^k, 2^{k+1}]$, there exist $n_0, m_0 \in [2^{k-1}, 2^k]$ such that $n = m_0 + n_0$ with $m_0 \leq n_0 \leq m_0 + 1$.

If n is an even integer of $]2^k, 2^{k+1}]$, then $n = 2m_0$ and $m_0 \in [2^{k-1}, 2^k]$. By the recurrence hypothesis, the result is correct for m_0. Applying lemma 2.2.1, we deduce that the statement extend to $2m_0$.

If n is an odd integer of $]2^k, 2^{k+1}]$, then $n = m_0 + (m_0 + 1)$ and $m_0, (m_0 + 1) \in [2^{k-1}, 2^k]$. By the recurrence hypothesis, the result is correct for m_0 and $(m_0 + 1)$. Applying indifferently lemma 2.2.1 or lemma 2.2.2, we deduce that the statement extends to $2m_0 + 1$. $\mathcal{P}(k+1)$ is therefore proved and the result follows. □

2.2.3 SNI Analysis of Our Refreshing Algorithm.

In the following, we address the second part of our analysis, *i.e.* the SNI property of our recursive refreshing schemes in the serial case. This is the purpose of the next theorem.

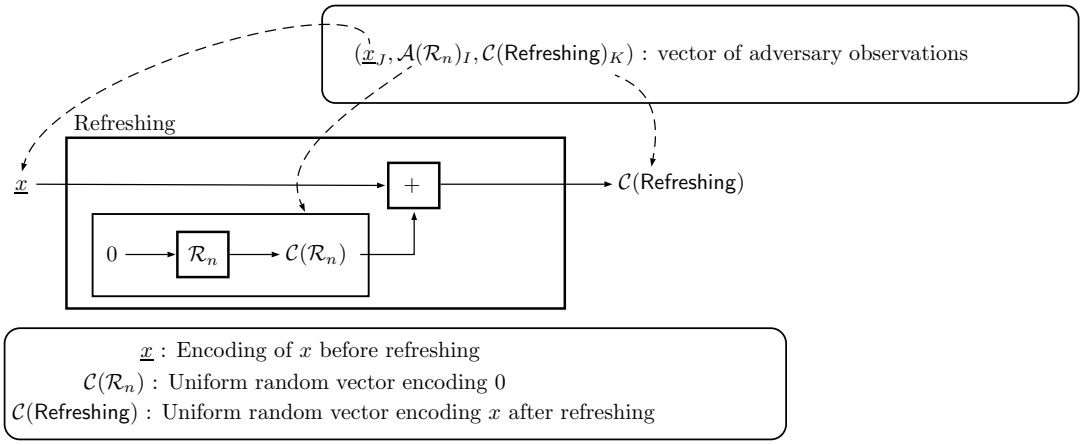


Figure 2.1: Illustration of the adversary vectors of observations during the execution of the refreshing via the algorithm $\mathcal{R}_n$.

The context. Consider the execution of the Refreshing algorithm (alg. 3) illustrated in the following figure.
Consider the different vectors displayed in the above figure. The input values, *i.e.* the components of $\underline{x}$, are considered to be fixed in what follows. Following the SNI definition for a refreshing gadget given in section 2, in order to prove that $\mathsf{Refreshing}(\underline{x})$ is SNI, we need to prove that the distribution of

$$\left(\mathbf{A}(\mathcal{R}_n)_I, \underline{x}_J, \mathbf{C}(\mathsf{Refreshing})_K\right)$$

depends on at most $\#I + \#J$ components of $\underline{x}$ for any subset I, J, K such that $\#I + \#J + \#K < n$. Also, $\underline{x}_J$ being fixed and of size $\#J$, the problem reduces to prove that $\left(\mathbf{A}(\mathcal{R}_n)_I, \mathbf{C}(\mathsf{Refreshing})_K\right)$ depends of at most $\#I$ components of $\underline{x}$ for any subset I and K such that $\#I + \#K < n - \#J$.

Theorem 2.2.4. *Let $\underline{x} \in G^n$ be a (fixed) sharing of s,* i.e. $\sum_{i=1}^n \underline{x}_i = s$. *Denote by $\mathcal{R}_n$ the probabilistic algorithm 2 and $\mathbf{C}(\mathcal{R}_n)$ its output vector. Then, the algorithm defined by*

$$\mathsf{Refreshing}(\underline{x}) = \underline{x} + \mathbf{C}(\mathcal{R}_n)$$

is a refreshing algorithm and is SNI.

Proof. According to theorem 2.2.3, $\mathbf{C}(\mathcal{R}_n) \sim \mathcal{U}\left(S^n_{\{1,\dots,n\}}(0)\right)$. For any fixed $\underline{x} \in G^n$,

$$\underline{x} + \mathbf{C}(\mathcal{R}_n) \sim \mathcal{U}\left(S^n_{\{1,\dots,n\}}\left(\sum_{i=1}^n \underline{x}_i\right)\right)$$

or equivalently $\mathsf{Refreshing}(\underline{x})$ is uniform over $S^n_{\{1,\dots,n\}}(s)$. This in particular means that $\mathsf{Refreshing}(\underline{x})$ is a refreshing algorithm. By definition $\mathbf{C}(\mathsf{Refreshing})_K = (\underline{x} + \mathbf{C}(\mathcal{R}_n))_K$. Therefore, $\mathbf{C}(\mathsf{Refreshing})_K$ depends algebraically on $\#K$ components of $\underline{x}$. It follows that the distribution of $\mathbf{C}(\mathsf{Refreshing})_K$ depends of at most $\#K$ components of $\underline{x}$. Remembering that $\mathbf{A}(\mathcal{R}_n)_I$ is independent of $\underline{x}$, we have that the distribution of $(\mathbf{A}(\mathcal{R}_n)_I, \mathbf{C}(\mathsf{Refreshing})_K)$ depends on at most $\#K$ components of $\underline{x}$.

If $\#I \geq n/2$, then according to the global constraint we have that

$$\#K < n - (\#I + \#J) < n/2 - \#J \leq n/2 \leq \#I\,.$$

Subsequently, $\mathsf{Refreshing}(\underline{x})$ is SNI in this case.

In what follows, we assume that $\#I < n/2$. According to theorem 2.2.3, for any I such that $\#I < n/2$ and for any view $\underline{a} \in \mathsf{Im}\left(\mathbf{A}(\mathcal{R}_n)_I\right)$, there exists a partition $V_1, \dots, V_k$ of $\{1, \dots, n\}$ and applications $\rho_i : \mathsf{Im}(\mathbf{A}(\mathcal{R}_n)_I) \mapsto G$ such that $\sum_{i=1}^k \rho_i(\underline{a}) = 0$ for any $\underline{a} \in \mathsf{Im}(\mathbf{A}(\mathcal{R}_n)_I)$ and

$$\mathbf{C}(\mathcal{R}_n) \mid \mathbf{A}(\mathcal{R}_n)_I = \underline{a} \;\sim\; \mathcal{U}\left(\sum_{i=1}^{k} S^n_{V_i}(\rho_i(\underline{a}))\right)$$

and there exists $t_0 \in \{1, \dots, k\}$ such that

$$\#V_{t_0} \geq n - \#I. \tag{2.13}$$

According to the global constraint $\#K < n - \#I - \#J$. Therefore, $\#K < n - \#I$. Also $\#V_{t_0} \geq n - \#I$ according to (2.13). It follows that $\#K < \#V_{t_0}$.

Note that $K = (V_{t_0} \cap K) \cup (V^{\mathsf{c}}_{t_0} \cap K)$. On one hand, we have that $V_{t_0} \cap K \underset{\neq}{\subseteq} V_{t_0}$ because $\#K < \#V_{t_0}$. It follows that

$$\mathbf{C}(\mathcal{R}_n)_{V_{t_0} \cap K} \;\mid\; \mathbf{A}(\mathcal{R}_n)_I = \underline{a} \sim \mathcal{U}\left(G^{\#(V_{t_0} \cap K)}\right).$$

Therefore,

$$\underline{x}_{V_{t_0} \cap K} + \mathbf{C}(\mathcal{R}_n)_{V_{t_0} \cap K} \mid \mathbf{A}(\mathcal{R}_n)_J = \underline{a} \;\sim\; \mathcal{U}\left(G^{\#(V_{t_0} \cap K)}\right)$$

and is therefore independent of $\underline{x}_K$. On the other hand, $(V^{\mathsf{c}}_{t_0} \cap K) \subseteq V^{\mathsf{c}}_{t_0}$ and $\#V^{\mathsf{c}}_{t_0} = n - \#V_{t_0}$. Therefore according to (2.13),

$$\#(V^{\mathsf{c}}_{t_0} \cap K) \leq n - \#V_{t_0} \leq \#I\,.$$

It follows that

$$\underline{x}_{V_{t_0}^{\mathsf{c}} \cap K} + \mathbf{C}(\mathcal{R}_n)_{V_{t_0}^{\mathsf{c}} \cap K} \mid \mathbf{A}(\mathcal{R}_n)_I = \underline{a}$$

depends on at most $\#I$ components of $\underline{x}$. It turns out that $\mathbf{C}(\mathsf{Refreshing})_K \mid \mathbf{A}(\mathcal{R}_n)_I = \underline{a}$ depends on at most $\#I$ components of $\underline{x}$. We deduce that

$$\left(\mathbf{A}(\mathcal{R}_n)_I, \mathbf{C}(\mathsf{Refreshing})_K\right) \mid \mathbf{A}(\mathcal{R}_n)_I = \underline{a}$$

follows a distribution depending on at most $\#I$ components of $\underline{x}$.

Finally, noting that $\mathbf{A}(\mathcal{R}_n)_I$ follows a distribution which does not depend of any component of $\underline{x}$, we deduce that $(\mathbf{A}(\mathcal{R}_n)_I, \mathbf{C}(\mathsf{Refreshing})_K)$ follows a distribution depending of at most $\#I$ components of $\underline{x}$. The results follows. □

2.3 Conversion into an SNI Iterative Refreshing.

In this section we address the iterative counterpart of our SNI recursive refreshing algorithm. In particular, we are interested in deriving a method for converting our recursive algorithm $\mathcal{R}_n$ into a security-wise equivalent iterative one. It is worth mentioning that while such conversions are quite straighforward in the case the size of input parameters are powers of two, it is however more challenging in the general case without any additional memory. Because the security analysis of our recursive approach depends on the sequence of internal computations during its execution, if one manages to replicate the same sensitive computation history in a different way, then the two approaches are security-wise equivalent. Consequently, converting our recursive approach into an iterative one can be done by making sure every sensitive computation is equivalent in both methods. In what follows, we refer to the recursive $\mathcal{R}_n$ algorithm as $\mathsf{rec\_}\mathcal{R}_n$ and the iterative version of $\mathcal{R}_n$ as $\mathsf{ite\_}\mathcal{R}_n$. Before deriving the method to perform the actual conversion, we start by a few important remarks that have to be taken into account for this conversion.

2.3.1 Preliminary Remarks.

For converting our SNI recursive approach into an iterative processing with same security, note that we only have to convert the subroutine $\mathsf{rec\_}\mathcal{R}_n$. The rest of the computation of the refreshing algorithm only consists in summing up the output of $\mathsf{rec\_}\mathcal{R}_n$ onto the input sensitive sharing that has to be refreshed (see steps 1 to 2 of algorithm 3. Therefore, the main problem we tackle in the following is to derive a way to retrieve all the call parameters of the $\mathsf{rec\_}\mathcal{R}_n$ algorithm "on-the-fly", *i.e.* without storing them.

A generic method for converting recursive into iterative algorithms. The method we propose in this section achieves the conversion of $\mathsf{rec\_}\mathcal{R}_n$ into an iterative algorithm $\mathsf{ite\_}\mathcal{R}_n$. However, our method can be applied to convert any similar recursive algorithm into an iterative processing. Namely, any recursive algorithm following a double recursion such as $\mathsf{rec\_}\mathcal{R}_n$, which successively divides the size of the input parameters by two, can be converted into an equivalent iterative algorithm. In order to do so, it is important to determine the computation history of the recursive algorithm. This can be done by analyzing the type of traversal is actually performed by the algorithm on the associated *recursion trees*.

Computation history - Recursion trees of $\mathsf{rec\_}\mathcal{R}_n$. As already mentioned, $\mathsf{rec\_}\mathcal{R}_n$ follows a double recursion, *i.e.* , $\mathcal{R}_n$ successively calls $\mathcal{R}_{\lfloor n/2 \rfloor}$ and $\mathcal{R}_{\lceil n/2 \rceil}$ until the final conditions $n = 2$ and $n = 3$ are reached. It also updates its input data of size n by applying single linear layers of randomness (see steps 14 to 22) only after the subroutines $\mathcal{R}_{\lfloor n/2 \rfloor}$ and $\mathcal{R}_{\lceil n/2 \rceil}$ have returned. Such a processing corresponds to a *depth-first post-order traversal* of the recursion trees. In this section, we will only consider linear layers for which the random elements are inserted following step 18 of the algorithm.

It can be observed that walking the tree in the *reverse level-order*, *i.e.* , level by level from the leaves to the root produces the same global computation history than for a depth-first post-order traversal, even though some of the computations may be performed in a different order. This is depicted in the following figure which displays the recursion tree associated to $\mathsf{rec\_}\mathcal{R}_9$. The tree on the left side corresponds to the post-order traversal of $\mathsf{rec\_}\mathcal{R}_n$. We displayed the recursive calls by dotted arrows. The numbers associated to the arrows indicate the order of the recursive calls. The circled numbers refer to the order in which the different nodes are processed. The tree on the right illustrates the reverse level-order traversal we want to emulate during the processing of $\mathsf{ite\_}\mathcal{R}_n$.

Computing the indexes of the labels. Note that the indexes of the labels of such trees correspond to the size of the different call parameters we wish to retrieve. As already explained, determining thee indexes dives the size of the linear layers of randomness (see steps 14 to 22 of the $\mathcal{R}_n$ algorithm) that have to be computed at execution.

Perfect binary recursion trees. Moreover, for any $n \in [2^k; 2^{k-1}[$, two cases can be distinguished whether $n \in [2^k; 3 \cdot 2^{k-1}]$ or $n \in]3 \cdot 2^{k-1}; 2^{k+1}[$. In particular, $\mathsf{rec\_}\mathcal{R}_n$ breaks the recursion further down in the case $n \in]3 \cdot 2^{k-1}; 2^{k+1}[$ and the

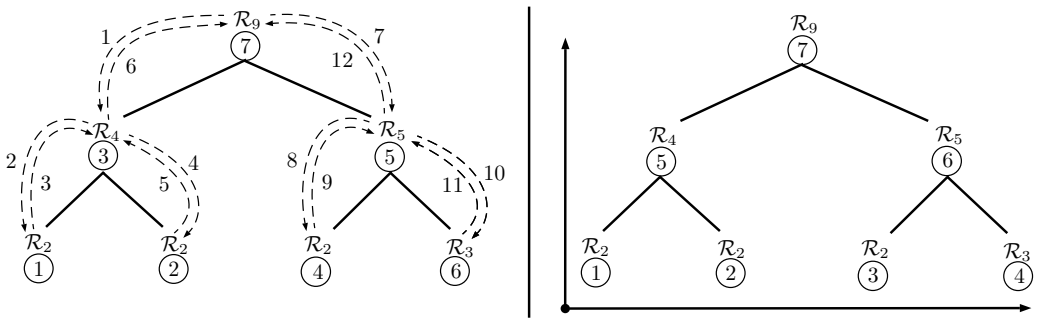


Figure 2.2: Traversals of the recursion tree associated to $\mathcal{R}_9$.

corresponding recursion trees are not perfect binary trees. As an example, consider the recursion trees produced for the executions of $\mathcal{R}_9$ and $\mathcal{R}_{13}$. Note that $2^3 \leq 9, 13 < 2^4$ and $9 \in]2^3; 3 \cdot 2^2]$ while $13 \in]3 \cdot 2^2; 2^4[$.

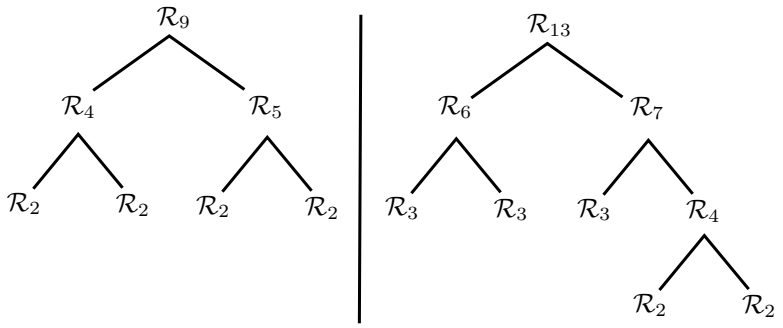


Figure 2.3: Recursion trees associated to $\mathcal{R}_9$ and $\mathcal{R}_{13}$.

In the following, it is more practical to consider perfect binary trees only, for any $2^k \leq n < 2^{k+1}$. Therefore, for any $n \in]3 \cdot 2^{k-1}; 2^{k+1}[$ we will consider the case $\mathcal{R}_4$ as a final condition, that integrates the two subsequent smaller cases $\mathcal{R}_2$.

On the architecture and associated features. The efficiency of our method in practice is even better if one can reverse the bits of registers efficiently and also determine the number of leading zeros within a register quickly. This can be done on ARM-based 32-bit microcontrollers with the RBIT and CLZ instructions, in one clock cycle. We therefore write our resulting iterative version of $\mathcal{R}_n$ for 32-bit architectures and so that it can be easily implemented in assembly language. Also, since we manipulate registers in what follows, we choose to represent them following the *little-endianness*. For example, for $n = 2^k + \sum_{i=0}^{k-1} \gamma_i \cdot 2^i$, it is represented

by

$$(\underbrace{0,\ldots,0}_{32-k},1,\gamma_{k-1},\ldots,\gamma_1,\gamma_0)\,.$$

For the sake of clarity, we will often omit the leading zeros when they are not necessary. The rest of the section is written according to this choice.

Based on the previous remarks, we now aim at determining the size of the recursion call parameters for any execution of $\mathsf{rec\_}\mathcal{R}_n$, for any n. Remember that they are computed by computing successive Euclidean divisions by two from the size n of the initial input parameter. In order to achieve this, we construct perfect binary trees related to such sequences of Euclidean divisions by two. In particular, we associate to the labels of our perfect binary trees the appropriate remainders computed from a given sequence of Euclidean divisions by 2 of n. The goal is to derive a method to understand how these remainders propagate from the root to the leafs. From this, we are able to retrieve any call parameter our original SNI algorithm $\mathsf{rec\_}\mathcal{R}_n$, for any given initial parameter size n.

2.3.2 Euclidean-division-based Perfect Binary Trees.

Let us start by defining useful subspaces of $\mathbb{F}_2^k$ over which we index the nodes of the trees at a given level. We consider that $\mathbb{F}_2^k$ is ordered according to the lexicographic order in the rest of this section.

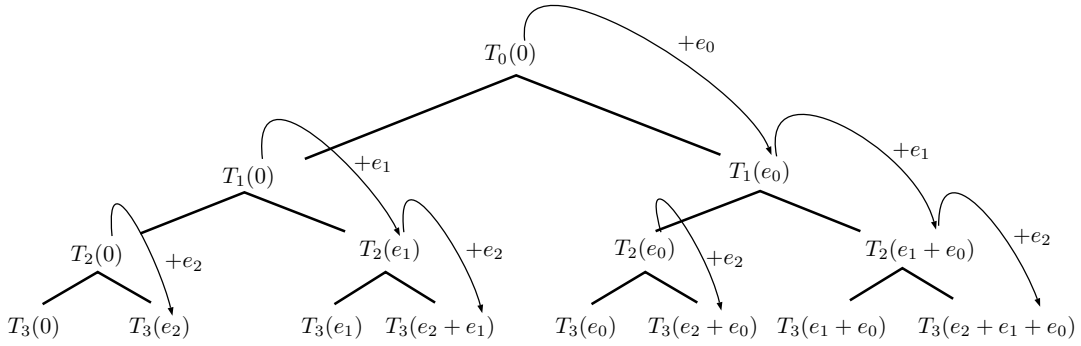
Ordering the indexes. Let $e_0,\ldots,e_{k-1}$ be a canonical basis of the vector space $\mathbb{F}_2^k$, *i.e.* , for any $i=\{0,\ldots,k-1\}$

$$e_i=(\underbrace{0,\ldots,0}_{i},1,0,\ldots,0)\,.$$

For any $i\in\{0,\ldots,k-1\}$, let V_i be subspaces of $\mathbb{F}_2^k$ and define $V_0=\{0\}$ and for any $i\in\{1,\ldots k-1\}$

$$V_i=\langle e_0,\ldots,e_{i-1}\rangle\,.$$

Let $k=\lfloor\log_2 n\rfloor$ and consider a perfect binary tree T of height k. In what follows, a node is referred to as $T_i(\omega)$, for any $i\in\{0,\ldots,k-1\}$ and any $\omega\in V_i$, where i is the level of the tree and ω is the index of the node at level i. Also, for any node $T_i(\omega)$, its left child is $T_{i+1}(\omega)$ and its right child is $T_{i+1}(\omega+e_i)$. The next figure illustrate this on a small example, for a tree T of height $k=3$.
Note that the indexes of the nodes at any level are ordered from left to right according to the lexicographic order on $\mathbb{F}_2^k$. Namely, for $i=1$ we have $0<e_0$, for



$i = 2$ we have $0 < e_1 < e_0 < e_0 + e_1$ and for $i = 3$ we have $0 < e_2 < e_1 < e_2 + e_1 < e_0 < e_2 + e_0 < e_1 + e_0 < e_2 + e_1 + e_0$. Let us now derive relations associated to successive Euclidean divisions of n by two.

Sequences of Euclidean divisions. Let $n = 2^k + \sum_{i=0}^{k-1} \gamma_i \cdot 2^i$ with $k \geq 1$. Observe that $2^k \leq n_0 < 2^{k+1}$. We now define a list of n_i's such that $n_k = 1$ and for any $i = \{0, \ldots, k-1\}$

$$n_k = 1 \text{ and } n_i = 2 \cdot n_{i+1} + \gamma_i \,, \text{ for any } i = \{0, \ldots, k-1\} \,. \tag{2.14}$$

This relation can also be written

$$n_0 = 2^i \cdot n_i + \sum_{j=0}^{i-1} \gamma_j \cdot 2^j \quad \text{for } i \in \{0, \ldots, k\} \,. \tag{2.15}$$

In what follows, we aim at relating the two above relations to our perfect binary trees associated n. More precisely, we aim at constructing a perfect binary tree of height k as above such that the sum of the value of all nodes at each level is $\sum_{j=0}^{i-1} \gamma_j \cdot 2^j$ and also each right child is greater or equal to the left child and this by at most one. This is made accurate by the following theorem.

Theorem 2.3.1. *Let $n_0 = 2^k + \sum_{i=0}^{k-1} \gamma_i \cdot 2^i$ with $k \geq 1$ and $\gamma_i \in \{0, 1\}$ for any $i \in \{0, \ldots, k-1\}$. Let T be a perfect binary tree of height k and associated with n_0. Consider that $T_0(\omega) = 0$ for $\omega \in V_0$. Also, if for any $i = \{0, \ldots, k-1\}$ and for any $\omega \in V_i$ we have*

$$T_{i+1}(\omega + \gamma_i \cdot e_i) = \gamma_i \text{ and } T_{i+1}(\omega + \overline{\gamma_i} \cdot e_i) = T_i(\omega) \,,$$

then, for any $i \in \{0, \ldots, k\}$ we have

$$\sum_{\omega \in V_i} T_i(\omega) = \sum_{j=0}^{i-1} \gamma_j \cdot 2^j \tag{2.16}$$

and for any $\omega \in V_i$*, for any* $i \in \{0, \dots, k-1\}$*, we also have*

$$T_{i+1}(\omega + e_i) \geq T_{i+1}(\omega)\,. \tag{2.17}$$

Also, for any $l \in \{1, \dots, k\}$,

$$T_l\Big(\omega + \textstyle\sum_{i=s}^{l-1} \overline{\gamma_i} \cdot e_i\Big) = T_s(\omega)\,,\ \textit{for any } \omega \in V_s\textit{, for any } s \leq l\,, \tag{2.18}$$

and

$$T_l\Big(\omega + \gamma_s \cdot e_s + \textstyle\sum_{i=s+1}^{l-1} \overline{\gamma_i} \cdot e_i\Big) = \gamma_s\,,\ \textit{for any } \omega \in V_s \textit{ and for any } s < l\,. \tag{2.19}$$

Finally,

$$T_l\Big(\textstyle\sum_{i=0}^{l-1} \overline{\gamma_i} \cdot e_i\Big) = 0. \tag{2.20}$$

Proof. The affirmation (2.16) is true for $i = 0$. Indeed, by definition we have $\sum_{\omega \in V_0} T_0(\omega) = 0$. Now assume that $\sum_{\omega \in V_i} T_i(\omega) = \sum_{j=0}^{i-1} \gamma_j \cdot 2^j$ for some $i \in \{0, \dots, k-1\}$ and let us evaluate $\sum_{\omega \in V_{i+1}} T_{i+1}(\omega)$. By definition,

$$V_{i+1} = V_i \cup (V_i + e_i)\,.$$

Therefore,

$$\sum_{\omega \in V_{i+1}} T_{i+1}(\omega) = \sum_{\omega \in V_i} T_{i+1}(\omega) + \sum_{\omega \in V_i} T_{i+1}(\omega + e_i)\,.$$

If $\gamma_i = 1$, by definition $T_{i+1}(\omega) = T_i(\omega)$ and $T_{i+1}(\omega + e_i) = 1$ for any $\omega \in V_i$. Hence,

$$\sum_{\omega \in V_{i+1}} T_{i+1}(\omega) = \sum_{\omega \in V_i} T_i(\omega) + (\#V_i) = \sum_{\omega \in V_i} T_i(\omega) + 2^i\,. \tag{2.21}$$

Also, by assumption we have that $\sum\limits_{\omega \in V_i} T_i(\omega) = \sum_{j=0}^{i-1} \gamma_j \cdot 2^j$. Hence,

$$\sum_{\omega \in V_{i+1}} T_{i+1}(\omega) = \sum_{j=0}^{i-1} \gamma_j \cdot 2^j + 2^i = \sum_{j=0}^{i} \gamma_j \cdot 2^j\,.$$

Now if $\gamma_i = 0$, similarly, $T_{i+1}(\omega) = 0$ and $T_{i+1}(\omega + e_i) = T_i(\omega)$ for any $\omega \in V_i$. By assumption,

$$\sum_{\omega \in V_{i+1}} T_{i+1}(\omega) = 0 + \sum_{\omega \in V_i} T_i(\omega) = 0 \cdot 2^i + \sum_{j=0}^{i-1} \gamma_j \cdot 2^j\,. \tag{2.22}$$

Therefore,

$$\sum_{\omega \in V_{i+1}} T_{i+1}(\omega) = \sum_{j=0}^{i} \gamma_j \cdot 2^j$$

and the result of affirmation (2.16) follows. Moreover, if $\gamma_i = 1$ (resp. $\gamma_i = 0$), we can deduce from relation (2.13) (resp. (2.22)) that $T_{i+1}(\omega + e_i) \geq T_{i+1}(\omega)$ for any $\omega \in V_i$. The result of affirmation (2.17) follows. Now, in order to prove affirmation (2.18), let us define the following proposition

$$\mathcal{P}(l) : T_l(\omega + \textstyle\sum_{i=s}^{l-1} \overline{\gamma_i} \cdot e_i) = T_s(\omega) \quad \text{for any } \omega \in V_s \text{ and for any } s \leq l\,.$$

Let us prove by induction that $\mathcal{P}(l)$ is true for any s and l such that $0 \leq s \leq l$. The base case $\mathcal{P}(1)$ results from the definition. Indeed, if $s = l = 1$ we trivially have $T_1(\omega) = T_1(\omega)$ and if $s = 0$, $T_1(\omega + \overline{\gamma_0} \cdot e_0) = T_0(\omega)$ for $\omega \in V_0 = \{0\}$. Assume that $\mathcal{P}(l)$ holds. Let us prove that $\mathcal{P}(l+1)$ is true. By definition,

$$T_{l+1}(\omega + \overline{\gamma_l} \cdot e_l) = T_l(\omega) \quad \text{for any } \omega \in V_l\,. \tag{2.23}$$

However $\omega + \sum_{i=s}^{l-1} \overline{\gamma_i} \cdot e_i \in V_l$ for any $\omega \in V_s$. Therefore, for any $\omega \in V_s$

$$T_{l+1}\Big(\omega + \textstyle\sum_{i=s}^{l-1} \overline{\gamma_i} \cdot e_i + \overline{\gamma_l} \cdot e_l\Big) = T_l\Big(\omega + \textstyle\sum_{i=s}^{l-1} \overline{\gamma_i} \cdot e_i\Big)\,.$$

By induction hypothesis, for any $s \leq l$ and any $\omega \in V_s$ we have

$$T_{l+1}\Big(\omega + \textstyle\sum_{i=s}^{l-1} \overline{\gamma_i} \cdot e_i + \overline{\gamma_l} \cdot e_l\Big) = T_s(\omega)$$

or equivalently

$$T_{l+1}\Big(\omega + \textstyle\sum_{i=s}^{l} \overline{\gamma_i} \cdot e_i\Big) = T_s(\omega)\,.$$

This proves $\mathcal{P}(l+1)$ and therefore concludes the induction from which the result of affirmation (2.18) follows. Let us now prove affirmation (2.19). By the previous result, for any $l \in \{1, \ldots, k\}$ and for any $s \leq l - 1$,

$$T_l\Big(\omega + \textstyle\sum_{i=s+1}^{l-1} \overline{\gamma_i} \cdot e_i\Big) = T_{s+1}(\omega)\,, \text{ for any } \omega \in V_{s+1}\,.$$

Also, $\omega + \gamma_s \cdot e_s \in V_{s+1}$ for any $\omega \in V_s$. Therefore,

$$T_l\Big(\omega + \gamma_s \cdot e_s + \textstyle\sum_{i=s+1}^{l-1} \overline{\gamma_i} \cdot e_i\Big) = T_{s+1}(\omega + \gamma_s \cdot e_s)\,, \text{for any } \omega \in V_s\,.$$

Observe that $T_{s+1}(\omega + \gamma_s \cdot e_s) = \gamma_s$ by definition. Hence, for any $s < l$,

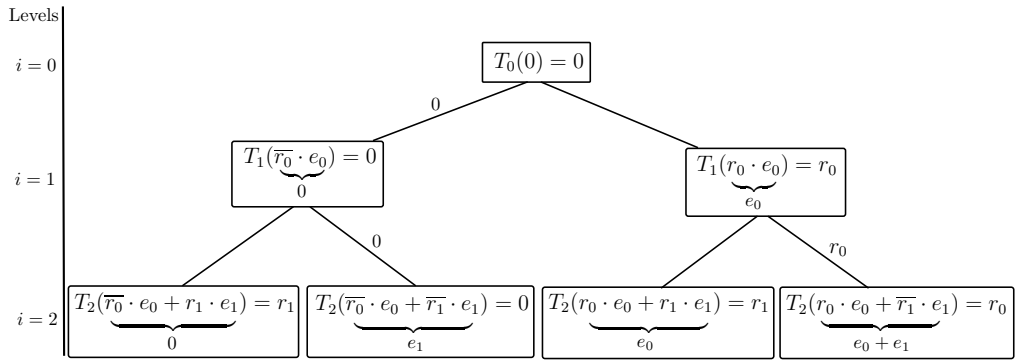
$$T_l\Big(\omega + \gamma_s \cdot e_s + \textstyle\sum_{i=s+1}^{l-1} \overline{\gamma_i} \cdot e_i\Big) = \gamma_s\,, \text{ for any } \omega \in V_s\,.$$

This proves the result of affirmation (2.19). Finally, by applying to $s = 0$ the result of affirmation (2.18) we thus have

$$T_l\Big(\omega + \textstyle\sum_{i=0}^{l-1} \overline{\gamma_i} \cdot e_i\Big) = T_0(\omega) = 0\,, \text{ for any } \omega \in V_0 = \{0\}\,,$$

which implies the result of the last affirmation. □

For the sake of clarity, let us now illustrate this theorem on a small example. Let $n_0 = 5$ and $k = \lfloor \log_2 n_0 \rfloor = 2$. Consider the binary representation of n_0, *i.e.* $(\gamma_1, \gamma_0) = (0, 1)$. Let e_0, e_1 be a canonical basis of $\mathbb{F}_2^2$. Remember that $\mathbb{F}_2^2$ is ordered according to the lexicographic order. Construct the list of n_i's according to relation (2.14), *i.e.* $(n_0, n_1, n_2) = (5, 2, 1)$. According to relation (2.15), we can write $n_0 = 1 \cdot 5$, $n_0 = 2 \cdot 2 + \gamma_0$ and $n_0 = 4 \cdot 1 + \gamma_0 + 2 \cdot \gamma_1$. Finally we have $V_0 = \{0\}, V_1 = \{0, e_0\}, V_2 = \{0, e_1, e_0, e_0 + e_1\}$. Following the previous theorem, the perfect binary tree of height k associated to $n_0 = 5$ is therefore



It can be seen that for each level of the tree depicted in the previous picture the relations (2.16) to (2.20) are satisfied. The following addresses a way to evaluate of the nodes of our euclidean division based perfect binary trees efficiently. Namely, we derive a method to determine the remainder that is associated to some given node.

2.3.3 Efficient Evaluation of the Trees.

Remember that we chose to derive an efficient method for $32 - bit$ architectures with respect to little-endianness. Let T be a perfect binary tree, associated with n, that is built as in theorem 2.3.1. Remember that T is of height k. We aim at evaluating all nodes of such trees, *i.e.* $T_l(\omega)$ for any $\omega \in V_l$ and any $l \in \{1, \ldots, k\}$. Note that the root $T_0(0)$ is always equal to 0. Our method enables to compute any node from the binary representations of its index and n.

Binary representations of the indexes. For any node $T_l(\omega)$, let us start by discussing the binary representations of ω. Note that by definition, for any $\omega \in V_l$ we have $\omega = \sum_{i=0}^{l-1} \omega_i \cdot e_i$. From this, following the definition of the canonical basis elements of $\mathbb{F}_2^k$ and with respect to a 32-bit architecture, any $\omega \in V_l$ over $\mathbb{F}_2^k$ is represented by a vector of the form

$$\underline{\omega} = \big(\underbrace{0,\ldots,0}_{32-k},\underbrace{\omega_0,\ldots,\omega_{k-1}}_{k}\big) = (0,\ldots,0,\omega_0,\ldots,\omega_{l-1},0,\ldots,0)$$

where $\omega_i \in \{0,1\}$. However, we can give more detailed representations of $\omega \in V_l$ by first seeing V_l as the following union set

$$V_l = \left(\bigcup_{s<l}\Big(V_s + \gamma_s \cdot e_s + \sum_{i=s+1}^{l-1} \overline{\gamma_i} \cdot e_i\Big)\right) \cup \left\{\sum_{i=0}^{l-1} \overline{\gamma_i} \cdot e_i\right\}. \tag{2.24}$$

Let us prove this above formula by induction. The formula is valid for the base case $l = 1$, *i.e.*

$$\begin{aligned} V_1 &= \bigcup_{s<1}\left(V_s + \gamma_s \cdot e_s + \sum_{i=1}^{0} \overline{\gamma_i} \cdot e_i\right) \cup \sum_{i=0}^{0} \overline{\gamma_i} \cdot e_i \\ &= V_0 + \gamma_0 \cdot e_0 \cup \overline{\gamma_0} \cdot e_0 \\ &= \{\gamma_0 \cdot e_0, \overline{\gamma_0} \cdot e_0\} \\ &= \{0, e_0\}. \end{aligned}$$

Assume now that relation (2.24) is valid parameter l. Let us prove that it is valid for parameter $l+1$, *i.e.*

$$V_{l+1} = \left(\bigcup_{s<l+1}\Big(V_s + \gamma_s \cdot e_s + \sum_{i=s+1}^{l} \overline{\gamma_i} \cdot e_i\Big)\right) \cup \left\{\sum_{i=0}^{l} \overline{\gamma_i} \cdot e_i\right\}.$$

We have that

$$\begin{aligned} V_{l+1} &= (V_l + \gamma_l \cdot e_l) \cup (V_l + \overline{\gamma_l} \cdot e_l) \\ &= (V_l + \gamma_l \cdot e_l) \cup \left(\left(\left(\bigcup_{s<l}\Big(V_s + \gamma_s \cdot e_s + \sum_{i=s+1}^{l-1} \overline{\gamma_i} \cdot e_i\Big)\right) \cup \left\{\sum_{i=0}^{l-1} \overline{\gamma_i} \cdot e_i\right\}\right) + \overline{\gamma_l} \cdot e_l\right) \\ &= (V_l + \gamma_l \cdot e_l) \cup \left(\left(\bigcup_{s<l}\Big(V_s + \gamma_s \cdot e_s + \sum_{i=s+1}^{l} \overline{\gamma_i} \cdot e_i\Big)\right) \cup \left\{\sum_{i=0}^{l} \overline{\gamma_i} \cdot e_i\right\}\right) \\ &= \left(\bigcup_{s<l+1}\Big(V_s + \gamma_s \cdot e_s + \sum_{i=s+1}^{l} \overline{\gamma_i} \cdot e_i\Big)\right) \cup \left\{\sum_{i=0}^{l} \overline{\gamma_i} \cdot e_i\right\}. \end{aligned}$$

We know from relation (2.19) of theorem 2.3.1 that for any $\omega \in V_l \setminus \sum_{i=0}^{l-1} \overline{\gamma_i} \cdot e_i$ we have

$$T_l(\omega + \gamma_s \cdot e_s + \sum_{i=s+1}^{l-1} \overline{\gamma_i} \cdot e_i) = \gamma_s\,, \text{ for any } \omega \in V_s \text{ and any } s < l\,.$$

Also, for $\omega = \sum_{i=0}^{l-1} \overline{\gamma_i} \cdot e_i$, we have $T_l(\omega) = 0$ according to relation (2.20).

Therefore, for any $\omega \in V_l \setminus \sum_{i=0}^{l-1} \overline{\gamma_i} \cdot e_i$, we have vectors of the form

$$\underline{\omega} = (0, \ldots, 0, \omega_0, \ldots, \omega_{s-1}, \gamma_s, \overline{\gamma_{s+1}}, \ldots, \overline{\gamma_{l-1}}, 0, \ldots, 0)\,. \tag{2.25}$$

In the case $\omega = \sum_{i=0}^{l-1} \overline{\gamma_i} \cdot e_i$, $\underline{\omega} = (0, \ldots, 0, \overline{\gamma_0}, \ldots, \overline{\gamma_{l-1}})$. The method we develop hereafter allows to retrieve γ_s, if existing, from $\underline{\omega}$ as in (2.25), the binary representation of n and some elementary operations that can be performed efficiently on 32-bit architectures.

Number of leading zeros of binary representations. The main idea is to use the CLZ instruction to count the number of leading zeros of some well-chosen vectors. From this, we can determine the value of any node $T_l(\omega)$, for any $\omega \in V_l$ and any $l \in \{1, \ldots, k\}$. The following is written according to the little-endianness. Consider now the reversed representation of (2.25), *i.e.*

$$\underline{\omega} = (\underbrace{\underbrace{0, \ldots, 0}_{k-l}, \underbrace{\overline{\gamma_{l-1}}, \ldots, \overline{\gamma_{s+1}}, \gamma_s, \omega_{s-1}, \ldots, \omega_0}_{l}}_{k}, \underbrace{0, \ldots, 0}_{32-k})\,. \tag{2.26}$$

From the binary representation of n, *i.e.* $(0, \ldots, 0, \gamma_k, \gamma_{k-1}, \ldots, \gamma_0)$, define an additionnal vector $\underline{a}$ such that

$$\underline{a} = (\underbrace{0, \ldots, 0}_{k-l}, \underbrace{\overline{\gamma_{l-1}}, \ldots, \overline{\gamma_{s+1}}, \overline{\gamma_s}, \overline{\gamma_{s-1}}, \ldots \overline{\gamma_0}}_{l}, \underbrace{0, \ldots, 0}_{32-k})\,. \tag{2.27}$$

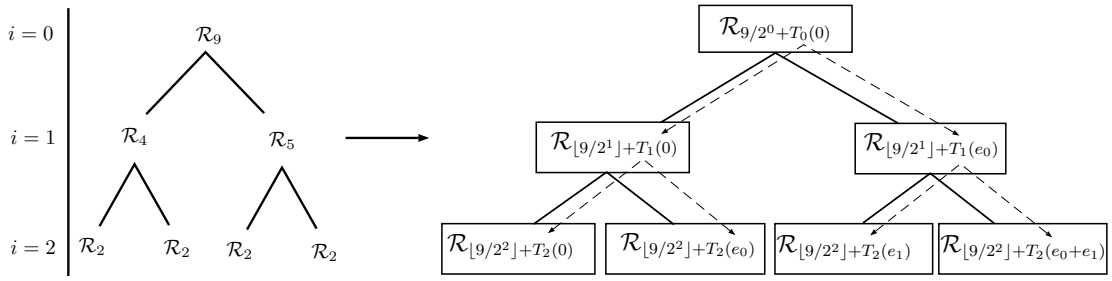
Computing the bit-wise XOR between $\underline{\omega}$ as in (2.26) and $\underline{a}$ as above produces

$$(\underbrace{\underbrace{0, \ldots\ldots\ldots\ldots, 0}_{\mathsf{CLZ}(\underline{\omega} \oplus \underline{a})}, \underbrace{1, \omega_{s-1} \oplus \overline{\gamma_{s-1}}, \ldots, \omega_0 \oplus \overline{\gamma_0}}_{s+1}}_{k}, 0, \ldots, 0) \tag{2.28}$$

where $1 = \gamma_s \oplus \overline{\gamma_s}$. From this last vector, since $\mathsf{CLZ}(\underline{\omega} \oplus \underline{a}) = k - (s+1)$, we deduce that $s = k - 1 - \mathsf{CLZ}(\underline{\omega} \oplus \underline{a})$. Note that if $s < 0$, this means that γ_s does not exist and therefore we are in the case $\omega = \sum_{i=0}^{l-1} \overline{\gamma_i} \cdot e_i$ for which it holds $T_l(\omega) = 0$. If $s \geq 0$, it means we are in the case $\omega \in V_l \setminus \sum_{i=0}^{l-1} \overline{\gamma_i} \cdot e_i$ for which it holds $T_l(\omega) = \gamma_s$ and γ_s can be easily obtained from the binary representation of n, *i.e.* $(0, \ldots, 0, \gamma_k, \ldots, \gamma_s, \ldots, \gamma_0)$.

2.3.4 Iterative Algorithm.

From theorem 2.3.1, we can determine any recursive call parameter of rec_$\mathcal{R}_n$ and hence obtain the size of the linear layers that have to be computed at execution. As already mentioned in the preliminary remarks of this section, remember that the algorithm rec_$\mathcal{R}_n$ breaks down the recursion until the final conditions $n = 2$ and $n = 3$ are reached. This means in particular that the corresponding sequence of euclidean divisions as defined in (2.14) does not have to be computed until $n_k = 1$. The final divisions can be omitted. This result in deriving our perfect binary trees only for a height equals to $k - 1$. The following figure illustrates an example for rec_$\mathcal{R}_9$. The perfect binary tree of height $k - 1$ and that is associated to $n = 9$ is displayed in dotted lines.



The sizes of the linear layers. It can be seen on the previous figure that we obtain relation (2.15) by summing up all indexes of the nodes at any level of the tree on the right side of the figure. This gives us the actual sizes of the different linear layers of randomness. They are obtained by adding the quotient of the division of 9 by a power of two. It is also worth mentioning that we do not need to pre-compute the tree T for deriving the ite_$\mathcal{R}_n$. We only emulate it by deriving the indexes of the nodes only, for all different levels. Considering all that have been previously discussed, we are now ready to derive our iterative version of $\mathcal{R}_n$.

The linear layers. We start by describing a simple algorithm that is used in ite_$\mathcal{R}_n$ for computing the linear layers of size n. This algorithm basically corresponds to steps 14 to 22 of the rec_$\mathcal{R}_n$ algorithm. It is however useful for the sake of clarity to separate this processing from the rest of the computation.

The iterative algorithm. Let us now derive the actual iterative algorithm. Remember that the perfect tree of height $k - 1$ that is associated to n is actually evaluated on-the-fly from its indexes only. No pre-computation is required. The processing basically consists in evaluating every node of the perfect binary tree T, computing the size of the linear linear of randomness that is associated to a given

Algorithm 4 Linear_layer

Require: A sharing $\underline{y} = (y_1, \dots y_n)$, the size n of the linear layer, an index j.
Ensure: The sharing $\underline{y}$ combined with random elements.

1: **for** i **from** j **to** $j + \lfloor n/2 \rfloor$ **do**
2: $\quad r_i \xleftarrow{\$} G$
3: $\quad y_i \leftarrow y_i + r_i$
4: $\quad y_{i+\lfloor n/2 \rfloor} \leftarrow y_{i+\lfloor n/2 \rfloor} - r_i$
5: **end for**
6: **return** $\underline{y}$

node of T, and applying successively the different layers by calling algorithm 2.3.4 just described. The evaluation of the nodes is performed such that the processing emulates a reverse level-order traversal of the recursion tree associated to rec_$\mathcal{R}_n$. It therefore iterates on the levels i from the leaves $i = k - 1$ to the root $i = 0$ and processes all the nodes indexed by $\omega \in V_i$, with respect to the lexicographic order on $\mathbb{F}_2^{k-1}$. Also, the algorithm is written such that for any $n \in]3 \cdot 2^{k-1}; 2^{k+1}[$ the execution of $\mathcal{R}_4$ integrates the two subsequent smaller executions of $\mathcal{R}_2$ (see preliminary remarks).
As it was the case for rec_$\mathcal{R}_n$, the actual refreshing is done by summing up the output of ite_$\mathcal{R}_n$ onto the sensitive data that have to be refreshed (see algorithm 3). By doing so, the iterative processing also satisfies the SNI requirements.

2.4 Conversion Into a Parallel Bounded Moment Algorithm.

In this section we address the conversion of our generic SNI refreshing scheme into its parallel counterpart. Similarly than for the previous conversion, the main problem we tackle now is to convert the execution of the $\mathcal{R}_n$ subroutine of our refreshing scheme into a parallel processing, denoted by parallel_$\mathcal{R}_n$ in the rest of the section. Thanks to theorem 1 of [8], converting an SNI algorithm that is proven secure at any masking order into its parallel counterpart renders the latter secure at any masking order under the bounded leakage model, which is relevant for parallel implementations. Note that the authors of [24] pointed out that the bounded moment model for parallel implementations may not offer the expected security. However, the study of the security of parallel implementations is new and need to be more studied. In this section, we start by giving some preliminary remarks that are useful beforehand tackling the actual conversion which leads to parallel_$\mathcal{R}_n$ and subsequently to our bounded moment secure parallel refreshing algorithm. Then,

Algorithm 5 ite_$\mathcal{R}_n$ (for 32-bit architectures)

Require: A sharing $\underline{y} = (y_1, \dots y_n)$ of 0, the binary representation of n, *i.e.* $\underline{n} = (0, \dots, 0, \gamma_k, \gamma_{k-1}, \dots, \gamma_0)$, $k - 1 = \lfloor \log_2 n \rfloor - 1$.
Ensure: A random sharing $\underline{y}$ of 0.

for i from $k-1$ **down to** 0 **do** ▷ From the leaves to the root.
  Compute $\underline{a}$ as in (2.27) for $l = i$.
  $j \leftarrow 1$ ▷ Points to a share of $\underline{y}$
  for every $\underline{\omega}$ **in** V_i **do** ▷ All nodes of a level
*** Computation of the size of the linear layers ***
    $\underline{\omega} \leftarrow \mathsf{reverse}(\underline{\omega})$ ▷ See (2.26)
    $s \leftarrow k - \mathsf{CLZ}(\underline{a} \oplus \underline{\omega}) - 1$ ▷ See (2.28)
    $ll\_size \leftarrow \lfloor n/2^i \rfloor$
    if $s \geq 0$ **then**
      $ll\_size \leftarrow ll\_size + ((n \gg s)\ \&\ 1)$ ▷ Updates size with γ_s
    end if
*** Computation of the linear layers ***
    if $i = k - 1$ **then** ▷ For the leaves only
      if $ll\_size = 3$ **then**
        $\underline{y} \leftarrow \mathsf{Linear\_layer}(\underline{y}, ll\_size, j + 1)$
      else if $ll\_size = 4$ **then**
        $\underline{y} \leftarrow \mathsf{Linear\_layer}\big(\underline{y}, ll\_size/2, j\big)$
        $\underline{y} \leftarrow \mathsf{Linear\_layer}\big(\underline{y}, ll\_size/2, j + 2\big)$
      end if
    end if
    $\underline{y} \leftarrow \mathsf{Linear\_layer}\big(\underline{y}, ll\_size, j\big)$
    $j \leftarrow j + ll\_size$
  end for
end for
return $\underline{y}$

we define some masks and vectors that are involved in parallel_$\mathcal{R}_n$. Finally, we give the algorithms.

2.4.1 Preliminary Remarks.

We follow an approach that is similar to the recursive to iterative conversion described in the previous section. Namely, remembering that reverse level-order traversals of the recursion trees are equivalent to the depth-first post-order traversals corresponding to rec_$\mathcal{R}_n$, parallel_$\mathcal{R}_n$ will emulate reverse level-order traver-

sals. However, on the contrary to $\mathsf{ite\_}\mathcal{R}_n$, *i.e.* the iterative version of $\mathcal{R}_n$ in the serial setting for which each share is manipulated in different clock cycles at execution, $\mathsf{parallel\_}\mathcal{R}_n$ manipulates all shares of a sharing at once for each clock cycle. It means that this time we aim at processing complete levels of the recursion trees at once, from the leaves to the root. Also, by observing that the evaluation of a complete level of a recursion tree updates all shares at once, this approach therefore requires to define a "packing" strategy for the shares. It simply consists in taking into account the word size of the architecture to pack several shares into a given register represented as a vector. By doing so, all packed shares can be operated at once time by applying a single instruction on such a word. In this section, we consider the word length to be big enough so that an entire sharing fits inside one word. Also, and as it was the case for the recursive to iterative conversion, a register is represented following the little-endianness. However, it would have been possible to consider big-endianness. The algorithm we propose in the parallel setting transposes to either choice, on condition (minor) modifications are made. The following notations formalize this context.

Notations. Consider the group $(\mathbb{F}_2^\alpha, \oplus)$, where α is the size of the architecture for which we derive our parallel refreshing algorithm. The operator $\oplus$ is the bitwise XOR on two words of size α. Also, the bitwise AND is denoted by & and $A \ll i$ (resp. $A \gg i$) stands for the word A that has to be shifted from i positions to the left (resp. right). Moreover, $\sim$A denotes the bitwise complement of A. We denote by $\underline{\mathsf{R}[v]}$ the vector $(r_1, \ldots, r_v)$ of the values corresponding to v independent random elements of $\mathbb{F}_2^\beta$. Finally, $\underline{u}[v] \in \mathbb{F}_2^{\beta \cdot v}$ with $\beta \cdot v \leq \alpha$ a vector of length v whose coordinates are $u \in \mathbb{F}_2^\beta$, *i.e.* , $\underline{u}[v] = (\underbrace{u, \ldots, u}_{v})$. In the following u will either 0 or 1, therefore denoting all-zero or all-one vectors respectively.

Packing strategy, little-endianness and consequences. Consider a sharing $\underline{x}$ of x as defined until now, *i.e.* $\underline{x} = (x_1, \ldots, x_n)$. From now on, we consider $\underline{x}$ over $\mathbb{F}_2^{\beta n}$ with $\beta n \leq \alpha$ and therefore each of the shares $x_1, \ldots, x_n$ is now considered as an element of $\mathbb{F}_2^\beta$. Also, registers are of size α and are represented following little-endianness. In the following we also choose to define the packing strategy accordingly. Namely, a sharing $\underline{x}$ of n shares is now represented as

$$\underline{x} = (\underbrace{0, \ldots, 0}_{\alpha - \beta n}, x_n, \ldots, x_1)$$

where the $\underline{x_i}$'s are represented by vectors of the form $(a_{\beta-1}, \ldots, a_0)$ with $a_j \in \{0, 1\}$ for any $j \in \{0, \ldots, \beta - 1\}$. By doing so, the smaller indexes correspond to the coordinates on the right side of the vectors. For the sake of clarity, we will omit

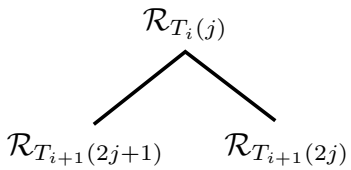
the $\alpha - \beta n$ leading zeros of sharings. We will therefore write vectors over $\mathbb{F}_2^{\beta n}$, taking only into consideration the relevant shares.

Our parallel algorithm still relies on perfect binary trees that are associated to n by considering the same sequences of Euclidean divisions by two as defined in the previous section (see (2.14)). However, this time we do not only consider the remainders of such sequences in the definition of the trees. We also change the indexing of the nodes in a way that is more appropriate for the current conversion. Additionally, we take into account the packing strategy just described, which results in a slight change of the definition of the trees. This is made accurate hereafter.

Euclidean division based trees associated to n. Consider $n = 2^k + \sum_{i=0}^{k-1} \gamma_i \cdot 2^i$ with $k \geq 1$. Let T be a perfect binary tree of height $k-1$. A node of T will be noted as $T_i(j)$ where i is the level and j is the index of the node within level i. The left child and the right child of $T_i(j)$ are $T_{i+1}(2j+1)$ and $T_{i+1}(2j)$ respectively. The tree T of height $k-1$ is recursively defined as follows. The root $T_0(0)$ is equal to n and for any $i \in \{1, \dots k-2\}$ we define

$$T_{i+1}(2j+1) = \left\lceil \frac{T_i(j)}{2} \right\rceil \quad \text{and} \quad T_{i+1}(2j) = \left\lfloor \frac{T_i(j)}{2} \right\rfloor ,$$

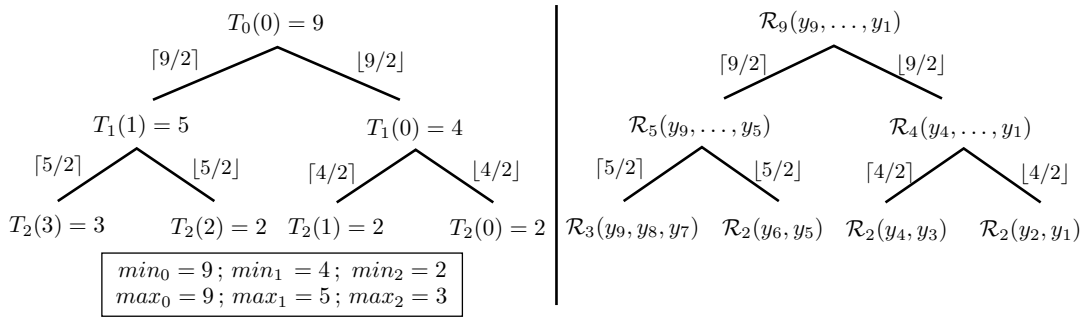
for any $j \in \mathbb{Z}_{2^i}$ where $\mathbb{Z}_{2^i}$ is identified to its smallest representatives, *i.e.* $\mathbb{Z}_{2^i} = \{0, \dots, 2^i - 1\}$. Note that this time the elements at each level sums up to n, *i.e.* $\sum_{j \in \mathbb{Z}_{2^i}} T_i(j) = n$. Note that the T associated to n gives the actual size of all call parameters derived from an execution of $\mathcal{R}_n$. We can therefore describe the recursion tree associated to the execution of $\mathcal{R}_n$ from the perfect binary tree T of height $k-1$ that is associated to n. Namely, we can write $\mathcal{R}_n = \mathcal{R}_{T_0(0)}$ and for any $i \in \{1, \dots, k-1\}$,



Also, we define

$$min_i \triangleq min_{j \in \mathbb{Z}_{2^i}} T_i(j) \quad \text{and} \quad max_i \triangleq max_{j \in \mathbb{Z}_{2^i}} T_i(j) \,. \tag{2.29}$$

According to the design of the trees T, we have $min_i \leq max_i \leq min_i + 1$. As an example consider the following figure. On the left side is displayed the perfect binary tree T of height 2 that is associated to 9 and on the right side is displayed the corresponding recursion tree with input parameters.

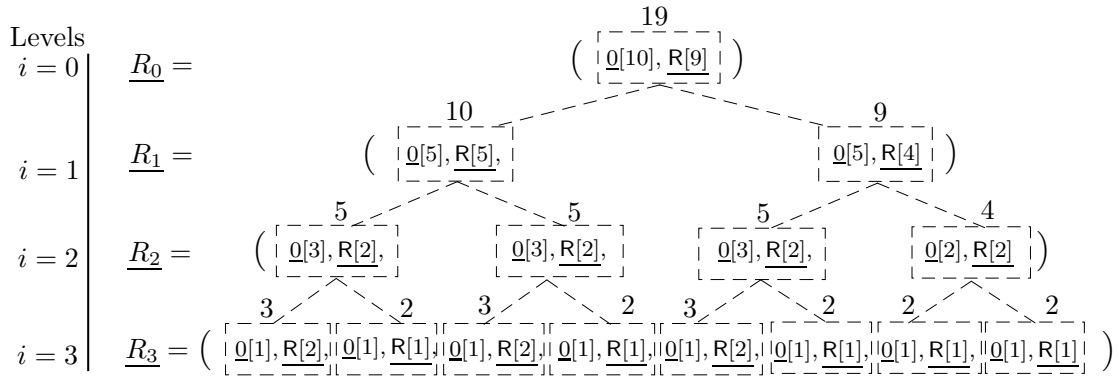


The basic idea is to emulate a reverse level-order traversal of the recursion tree displayed on the right side of the above figure from the perfect binary tree T displayed on the left side. Also, in the parallel setting, we aim at processing complete levels at once. Therefore, the parallel processing corresponding to the above example would first update the input vector $\underline{y} = (y_9, \ldots, y_1)$ by computing $\big(\mathcal{R}_3(y_9, y_8, y_7), \mathcal{R}_2(y_6, y_5), \mathcal{R}_2(y_4, y_3), \mathcal{R}_2(y_2, y_1)\big)$. Then, it would update $\underline{y}$ by computing $\big(\mathcal{R}_5(y_9, \ldots, y_5), \mathcal{R}_4(y_4, \ldots, y_1)\big)$ and finally it would output $\underline{y} = \big(\mathcal{R}_9(y_9, \ldots, y_1)\big)$. This sequence of computations basically consists in applying at once the different linear layers of randomness corresponding to the distinct $\mathcal{R}_n(\cdot)$ of a level. In what follows, we give more details about the parallel processing of all the linear layers of randomness as described by an entire level of the recursion trees.

Parallel processing of distinct linear layers. Each linear layer of algorithm 2 ($\mathcal{R}_n$) pairs together coordinates of $\underline{y}$ by XOR*ing* the same (freshly generated) single random element r on two distinct coordinates of the input vector $\underline{y}$ and the offset depends of the level of recursion. The parallel processing of all the linear layers at a level has to correspond to the serial processing. The difference in the parallel setting is that we would manipulate (freshly generated) random vectors of size $\alpha > \beta n$. The parallel processing of all linear layers of a level is performed as follows : we first extract some random elements from the freshly generated random vector and then we position them into a register such that all corresponding coordinates of the input vector $\underline{y}$ are paired at once time. We aim at processing this step efficiently and also at optimizing the number of random vectors that have to be globally used to perform all linear layers.

Constructing vectors of random elements. Consider the following example. Assume that we want to execute $\mathcal{R}_{19}$ in a parallel way. The next figure represents the values of each node of the Euclidean division based tree of height 3 that is associated to 19 and displays the vectors of random elements that we

need to construct for computing the different linear layers in a parallel way. Remember that $\underline{0}[v] = (0, \ldots, 0)$ of length v where 0 is represented over $\mathbb{F}_2^\beta$ and that $\underline{\mathsf{R}[v]} = (r_1, \ldots, r_v)$ is a vector of the values corresponding to v independent random elements of $\mathbb{F}_2^\beta$.



The vectors $\underline{R_i}$ described above are composed of all independent random elements that are required for the computation of all linear layers of a given level i in parallel. However, in order to actually perform the pairing of corresponding coordinates of the input vector $\underline{y}$ as described above, such vectors $\underline{R_i}$ have to be shifted from a certain offset which depends on the level i. This will position the random elements correctly for the pairing. We do so by defining shifted vectors $\underline{SR_i}$ as

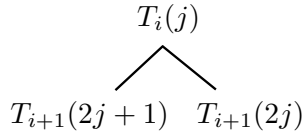
$$\begin{aligned}
\underline{SR_0} &= \underline{R_0} \oplus (\underline{R_0} \ll 10 \cdot \beta)\,, \\
\underline{SR_1} &= \underline{R_1} \oplus (\underline{R_1} \ll 5 \cdot \beta)\,, \\
\underline{SR_2} &= \underline{R_2} \oplus (\underline{R_2} \ll 2 \cdot \beta)\,, \\
\underline{SR_3} &= \underline{R_3} \oplus (\underline{R_3} \ll \beta)\,.
\end{aligned}$$

Parallel processing of $\mathcal{R}_n$. We consider that we have access to a random generator of vectors of $(\mathbb{F}_2^\beta)^n$. Our algorithm $\mathsf{parallel\_}\mathcal{R}_n$ works as follows : we generate as many random vectors of $(\mathbb{F}_2^\beta)^n$ as there are levels of linear layers to compute, *i.e.* , k for a tree T of height $k-1$. Then, the linear layers are computed successively from the leaves $i = k-1$ to the root $i = 0$. This is done by considering successively one of the k generated random vectors and by first constructing the vector $\underline{R_i}$ that is associated to level i from the selected random vector. Then $\mathsf{parallel\_}\mathcal{R}_n$ simply consists in updating $\underline{y}$ successively by computing $\underline{y} = \underline{y} \oplus \underline{SR_i}$ for $i = k-1, \ldots, 0$.

The processing of $\mathsf{parallel\_}\mathcal{R}_n$ just described and more precisely of its associated sequence of parallel linear layers requires to construct the vectors $\underline{R_i}$ from freshly generated random vectors of $(\mathbb{F}_2^\beta)^n$. It also requires to construct the shifted vectors $\underline{SR_i}$ which only consists in applying the right offsets for the shifts. This is made accurate in the following.

2.4.2 Linear Layers.

Let us now compute the vectors $\underline{R_i}$ and $\underline{SR_i}$ appearing during the execution of $\mathcal{R}_n$ for $n \geq 2$. Consider the binary expansion of $n = 2^k + \sum_{i=0}^{k-1} \gamma_i \cdot 2^i$ and its associated Euclidean division tree T, *i.e.* $T_0(0) = n$ and for any $i = 0, \ldots, k-2$ and any $j \in \mathbb{Z}_{2^i}$:



Let us first define some masks $\underline{M_i}$'s in order to generate vector $\underline{R_i}$'s from a random vector of $(\mathbb{F}_2^\beta)^n$, denoted by $\underline{R_{\mathsf{Gen}}}$ in the following.

Vectors of extracted random elements. As it was the case for the recursive to iterative conversion, we deal with the leaves separately from the other levels of the trees. Also as explained in the preliminary remarks of section 2.3, the height of recursion trees differs by 1 whether $n \in [2^k; 3 \cdot 2^{k+1}]$ or $n \in]3 \cdot 2^{k+1}; 2^{k+1}[$. Let us start by defining the vectors $\underline{R_i}$ composed of extracted random elements as follows. For any, $i \in \{0, \ldots, k-2\}$,

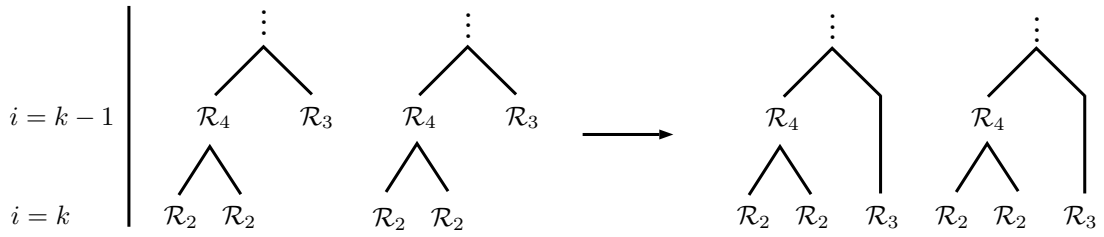
$$\underline{R_i} = \left(\underline{0}[\,\lceil T_i(j)/2 \rceil\,], \underline{\mathsf{R}[\,\lfloor T_i(j)/2 \rfloor\,]} \mid j \in \mathbb{Z}_{2^i} \right),$$

where $\underline{0}[\lceil T_i(j)/2 \rceil]$ follows the definition given in the notations of this section and $\underline{\mathsf{R}[\lfloor T_i(j)/2 \rfloor}$ denotes the concatenation of $\lfloor T_i(j)/2 \rfloor$ independent random elements of $\mathbb{F}_2^\beta$. The linear layers associated to the leaves of T are handled differently whether $n \in [2^k; 3 \cdot 2^{k+1}]$ or $n \in]3 \cdot 2^{k+1}; 2^{k+1}[$. The difference is that if $n \in [2^k; 3 \cdot 2^{k+1}]$, then we compute the linear layers associated to the leaves in one step while we do it in two steps if $n \in]3 \cdot 2^{k+1}; 2^{k+1}[$. Consider the first case, *i.e.* $n \in [2^k; 3 \cdot 2^{k-1}]$. In this case, the leaves of T are either 2 or 3 and we define

$$\underline{R_{k-1}} = \left(\underline{0}[\,\lfloor T_{k-1}(j)/2 \rfloor\,], \underline{\mathsf{R}[\,\lceil T_{k-1}(j)/2 \rceil\,]} \mid j \in \mathbb{Z}_{2^{k-1}} \right) \tag{2.30}$$

If $n \in]3 \cdot 2^{k-1}; 2^{k+1}[$, we compute two vectors R_{k-1} and R'_{k-1}. This is illustrated in the following example for $n = 14$ for which $k = 3$ and $14 \in]3 \cdot 2^2; 2^4[$. We only displayed the two last levels of the recursion tree associated to an execution of $\mathcal{R}_{14}$. The tree on the right side of the figure illustrates how we deal with the case $n \in]3 \cdot 2^{k-1}; 2^{k+1}[$. We first compute the linear layers associated to $\mathcal{R}_2$ and $\mathcal{R}_3$ and in a second step, we only compute the linear layers associated to $\mathcal{R}_4$.

Remember that the trees T have height $k-1$ and therefore we emulate the computation of $\mathcal{R}_4$ and $\mathcal{R}_3$ from the leaves of T corresponding to level $k-1$ of the



latter. Let us define

$$\underline{\mathsf{R}_{2,3}[u]} = \begin{cases} \underline{0}[1], \underline{\mathsf{R}[2]} & \text{if } u = 3 \\ \underline{0}[1], \underline{\mathsf{R}[1]}, \underline{0}[1], \underline{\mathsf{R}[1]} & \text{if } u = 4 \end{cases} \text{ and } \underline{\mathsf{R}_4[u]} = \begin{cases} \underline{0}[3] & \text{if } u = 3 \\ \underline{0}[2], \underline{\mathsf{R}[2]} & \text{if } u = 4\,. \end{cases}$$

From this, we define

$$\underline{R_{k-1}} = \left(\underline{\mathsf{R}_{2,3}[T_{k-1}(j)]} \mid j \in \mathbb{Z}_{2^{k-1}}\right) \text{ and } \underline{R'_{k-1}} = \left(\underline{\mathsf{R}_4[T_{k-1}(j)]} \mid j \in \mathbb{Z}_{2^{k-1}}\right). \quad (2.31)$$

Let us now derive the shifted vectors $\underline{SR_i}$ from the $\underline{R_i}$ just defined. From the definition of the tree T, it is not difficult to prove that for any $i = 0, \ldots, k-2$ and any $j \in \mathbb{Z}_{2^i}$,

$$T_{i+1}(2j) = min_{i+1} \text{ if } \gamma_i = 0 \text{ and } T_{i+1}(2j+1) = max_{i+1} \text{ otherwise.}$$

Therefore, if $n \in [2^k; 2^{k+1}[$, the vector $\underline{SR_i}$ corresponding to level i of the tree and for any $i = 0, \ldots, k-2$ is computed as

$$\underline{SR_i} = \underline{R_i} \oplus \left(\underline{R_i} \lll x \cdot \beta\right)$$

where $x = min_{i+1}$ if $\gamma_i = 0$ and $x = max_{i+1}$ otherwise, by considering $n = 2^k + \sum_{i=0}^{k-1} \gamma_i \cdot 2^i$. Also, if $n \in [2^k; 3 \cdot 2^{k-1}]$, the linear layers associated to the leaves of the recursion tree are computed in one step from

$$\underline{SR_{k-1}} = \underline{R_{k-1}} \oplus \left(\underline{R_{k-1}} \lll \beta\right),$$

where $\underline{R_{k-1}}$ is as in (2.30). Finally, if $n \in]3 \cdot 2^{k-1}; 2^{k+1}[$, the linear layers associated to the leaves of the recursion tree are computed in two steps from

$$\underline{SR_k} = \underline{R_k} \oplus \left(\underline{R_k} \lll \beta\right) \text{ and } \underline{SR_{k-1}} = \underline{R_{k-1}} \oplus \left(\underline{R_{k-1}} \lll 2 \cdot \beta\right),$$

where $\underline{R_k}$ and $\underline{R_{k-1}}$ are as in (2.31). In the following we explain how to extract the appropriated random elements from random vectors $\underline{R_{\mathsf{Gen}}}$ of $(\mathbb{F}_2^\beta)^n$ to actually derive the vectors $\underline{R_i}$.

Masks for extracting elements of random vectors. We define a list of masks,

denoted by $\underline{M_i}$ in what follows. Their definition follows the definition of the vectors $\underline{R_i}$. Namely, these masks only consists in replacing the chunks of random elements $\underline{\mathsf{R[v]}}$ in the definition of the vectors $\underline{R_i}$ by chunks of ones of same sizes, *i.e.* $\underline{1}[u]$. By doing so, we obtain the masks $\underline{M_0}, \ldots, \underline{M_{k-1}}$ (and $\underline{M'_{k-1}}$ if $n \in]3 \cdot 2^{k-1}; 2^{k+1}[$) and the vectors $\underline{R_i}$ can be constructed by simply computing

$$\underline{R_i} \triangleq \underline{R_{\mathsf{Gen}}} \,\&\, \underline{M_i}\,,$$

where $\underline{R_{\mathsf{Gen}}}$ is some random vector freshly generated.

Remark. The above procedure extracts only half random elements of every vector $\underline{R_{\mathsf{Gen}}}$ in order to construct the R_i's. With additional operations, it is also possible to construct two consecutive R_i's from the same $\underline{R_{\mathsf{Gen}}}$. This would allow to reduce the number of generated random vectors $\underline{R_{\mathsf{Gen}}}$ by a factor of about 2. However, this would also lead to a very technical algorithm which we omitted in this article.

2.4.3 Parallel Algorithm.

In the following, we derive the actual $\mathsf{parallel\_}\mathcal{R}_n$ algorithm from which the bounded moment secure refreshing algorithm can be derived straightforwardly. For the sake of clarity we provide beforehand $\mathsf{parallel\_}\mathcal{R}_n$ an algorithm for computing linear layers of randomness in parallel.

Algorithm 5 parallel_Linear_Layers

Require: A sharing $\underline{y} = (y_n, \ldots, y_1)$, a mask $\underline{M}$, an offset x.
Ensure: The sharing $\underline{y}$ updated with random elements.

1: $\underline{R_{\mathsf{Gen}}} \xleftarrow{\$} \mathbb{F}_2^{\beta \cdot n}$
2: $\underline{R} \leftarrow \underline{R_{\mathsf{Gen}}} \,\&\, \underline{M}$
3: $\underline{SR} \leftarrow \underline{R} \oplus (\underline{R} \ll x \cdot \beta)$
4: $\underline{y} \leftarrow \underline{y} \oplus \underline{SR}$
5: **return** $\underline{y}$

The following algorithm addresses the complete execution of $\mathsf{parallel\_}\mathcal{R}_n$. It makes use of the above parallel_Linear_Layers algorithm as a subroutine.

Algorithm 7 parallel_$\mathcal{R}_n$

Require: $\underline{y} = (y_n, \ldots, y_1)$, $n = (1, \gamma_{k-1}, \ldots, \gamma_0)$, the masks $(\underline{M_i})_{0 \le i \le k-1}$ and $\underline{M'_{k-1}}$ if $n \in]3 \cdot 2^{k-1}; 2^{k+1}[$, $(min_i)_{0 \le i \le k-2}, (max_i)_{0 \le i \le k-2}$.
Ensure: A random sharing $\underline{y}$ of 0.

$\underline{y} \leftarrow$ parallel_Linear_Layers$\big(\underline{y}, \underline{M_{k-1}}, 1\big)$
if $n \in]3 \cdot 2^{k-1}; 2^{k+1}[$ **then**
  $\underline{y} \leftarrow$ parallel_Linear_Layers$\big(\underline{y}, \underline{M'_{k-1}}, 2\big)$
end if
for i **from** $k-2$ **down to** 0 **do**
  if $\gamma_i = 0$ **then** $x \leftarrow min_{i+1}$ ▷ Corresponds to step 18 of alg. 2.
  else $x \leftarrow max_{i+1}$ ▷ Corresponds to step 20 of alg. 2.
  end if
  $\underline{y} \leftarrow$ parallel_Linear_Layers$\big(\underline{y}, \underline{M_i}, x\big)$
end for
return $\underline{y}$

Finally, the next algorithm is the Bounded Moment secure parallel refreshing scheme. It performs the actual refreshing of a sharing $\underline{x} = (x_n, \ldots, x_1)$ of some sensitive variable x.

Algorithm 5 Bounded Moment secure parallel refreshing scheme

Require: A sharing $\underline{x} = (x_n, \ldots, x_1)$, the masking order $n = (1, \gamma_{k-1}, \ldots, \gamma_0)$, the masks $(\underline{M_i})_{0 \le i \le k-1}$ and $\underline{M'_{k-1}}$ if $n \in]3 \cdot 2^{k-1}; 2^{k+1}[$, $(min_i)_{0 \le i \le k-2}, (max_i)_{0 \le i \le k-2}$.
Ensure: A refreshed sharing $\underline{z}$ of size n such that $\sum_{i=1}^{n} \underline{z}_i = \sum_{i=1}^{n} \underline{x}_i$

$\underline{y} \leftarrow 0$
$\underline{y} \leftarrow$ Parallel_$\mathcal{R}_n\big(\underline{y}, (\underline{M_i})_{0 \le i \le k-1}, (M'_{k-1}$if needed$), (min_i)_{1 \le i \le k-2}, (max_i)_{1 \le i \le k-2}\big)$
$\underline{z} \leftarrow \underline{x} \oplus \underline{y}$
return $\underline{z}$

Complexity. Our proposal requires $\lfloor \log_2 n \rfloor$ iterations and $\lceil \log_2 n \rceil$ random vectors while these values for the previous generic scheme [8] were both $\lceil n - 1 \rceil / 3$.

2.5 Conclusion

This chapter addressed the secure design of refreshing schemes which are one of the main primitives of masking countermeasures.

We provided a class of serial refreshing schemes over any finite Abelian group.

We proved that its security satisfies the strong SNI security requirements of the probing model for any masking order n. Our algorithm runs in $\mathcal{O}(n \cdot \log_2 n)$ and improves the time and randomness complexities of the best known method by a factor of about two in this context.

We provided a generic method for converting recursive implementations such as our recursive scheme into iterative ones and we applied this method on our recursive refreshing scheme. The resulting iterative scheme is equivalent to the former and therefore inherits its SNI property.

We proposed a parallel algorithm for a subset of the previous class, bounded moment secure for any masking order. Our proposal requires a number of iterations and random vectors which is logarithmic in the masking order while their number was linear in this parameter for the best known method so far.

Part III

A Generic Masking Scheme Based on the Polynomial Evaluation Method for Small Masking Orders

Chapter 1

Introduction

Masking is a sound countermeasure to protect implementations of block-cipher algorithms against Side Channel Analysis (SCA). Currently, the most efficient masking schemes use Lagrange's Interpolation Theorem in order to represent any S-box by a polynomial function over a binary finite field. Masking the processing of an S-box is then achieved by masking every operation involved in the evaluation of its polynomial representation. While the common approach requires to use the well-known Ishai-Sahai-Wagner (ISW) scheme in order to secure this processing, there exist alternatives. In the particular case of power functions, Genelle, Prouff and Quisquater proposed an efficient masking scheme (GPQ). However, no generalization has been suggested for polynomial functions so far. In next chapter, we solve the open problem of extending GPQ for polynomials, and we also solve the open problem of proving that both the original scheme and its variants for polynomials satisfy the t-SNI security definition. Our approach to extend GPQ is based on the cyclotomic method and results in an alternate cyclotomic method which is three times faster in practice than the original proposal in almost all scenarios we address. The best-known method for polynomial evaluation is currently CRV which requires to use the cyclotomic method for one of its step. We also show how to plug our alternate cyclotomic approach into CRV and again provide an alternate approach that outperforms the original in almost all scenarios. We consider the masking of n-bit S-boxes for $n \in [4; 8]$ and we get in practice 35% improvement of efficiency for S-boxes with dimension $n \in \{5, 7, 8\}$ and 25% for 6-bit S-boxes.

1.1 Related Works

In [56], Rivain and Prouff proposed the first efficient and provably secure masking scheme in the probing model for AES whose S-box consists in computing inversions in the finite field $\mathbb{F}_{2^8}$. Their idea was to express the corresponding inverse

function $x \mapsto x^{254}$ as a sequence of squares and nonlinear multiplications over $\mathbb{F}_{2^8}$. While squares are linear functions and are therefore easy to mask, they adapted the ISW multiplication gadget over $\mathbb{F}_2$ to the desired extension field $\mathbb{F}_{2^8}$ in order to mask the nonlinear multiplications. The proposed scheme was originally supposed to achieve d^{th} order security. However, the composition of their mask refreshing procedure with the ISW multiplication gadget induced a security flaw in the overall scheme [20]. A solution proposed in the same article was to avoid the use of the mask refreshing gadget by adapting the ISW scheme. The resulting secure multiplications are referred to as bilinear multiplications in the literature. It was only recently that the original scheme (without the bilinear multiplications) has been fixed in [6]. Namely, they proved that the composition of the multiplication gadget with a different mask refreshing procedure results in a safe construction with $d \geq t + 1$ shares by showing that both previous gadgets satisfy the t-SNI security definition.

The approach followed by Rivain and Prouff was extended to any n-bit S-box by Carlet, Goubin, Prouff, Quisquater and Rivain (CGPQR) in [14]. They showed that any n-bit S-box can be expressed as a sequence of linear transformations and nonlinear multiplications over $\mathbb{F}_{2^n}$, that is represented by a polynomial $S(x) = \sum_i a_i x^i$ over $\mathbb{F}_{2^n}$ using Lagrange's interpolation theorem. Thus, the CGPQR masking scheme consists in evaluating securely such polynomial over $\mathbb{F}_{2^n}$ by masking with ISW every nonlinear multiplication involved in the corresponding sequence. However, as the masking order grows, the secure processing of nonlinear multiplications quickly becomes expensive. Therefore, they also described two efficient heuristics called cyclotomic and parity-split methods that optimize the number of nonlinear multiplications required to evaluate the polynomial representation of generic S-boxes. Several methods have also improved CGPQR by further optimizing this number of nonlinear multiplications. Roy and Vivek [57] further reduced the complexity of several well known S-boxes and the currently best-known method for fast polynomial evaluation in $\mathbb{F}_{2^n}$ has been proposed by Coron, Roy and Vivek in [21] and is referred to as the CRV method in the rest of the paper. Recently, other constructions of multiplication circuits in finite fields than ISW have been proposed [12]. However, ISW remains the most efficient t-SNI scheme for orders of practical interest (i.e. orders 1, 2 and 3).

Different approaches can be used as alternatives to the higher-order CGPQR masking scheme [5, 15, 19, 29, 32, 40, 55]. Among them, the higher-order masking scheme introduced by Genelle, Prouff and Quisquater (GPQ) in [29] is a more efficient alternative for the AES than [56] (see [37]). The GPQ scheme is particularly efficient to mask S-boxes which are power functions but no generalization to mask

generic S-boxes has been proposed so far.

1.2 Contributions

In next chapter, we begin to prove the security of the GPQ masking scheme in the probing model under the stronger t-SNI security definition. Then, we show how to solve the open problem of extending GPQ to mask generic S-boxes (not only power functions). Specifically, our approach is based on the generic cyclotomic method proposed in [14], whose security so far relied on the ISW scheme. We show how to refine the use of GPQ when combined with the cyclotomic method so that it results in an alternate cyclotomic approach for polynomial evaluation over $\mathbb{F}_{2^n}$ that no longer requires ISW. We provide a description of our construction and prove that it satisfies the t-SNI requirements. We also provide an alternate approach for CRV. The latter requires the cyclotomic method in order to build a set of monomials in one of its steps. We show how to plug our alternate cyclotomic method into CRV, in order to efficiently compute those power functions with our previous construction. Moreover, our approach allows us to derive new parameters for CRV considered as irrelevant with the original proposal, but which are well-suited in our case. We then show that our alternate CRV construction is t-SNI. In practice, we consider the same scenarios for both our alternate approaches. We report the cost of polynomial evaluations with our approaches compared to the original ones where S-boxes are of dimension $n \in [4; 8]$. We improve by a factor 3 the efficiency of the original cyclotomic method in almost all scenarios (for $n \in \{5, 6, 7, 8\}$) and we improve by 35% the efficiency of the original CRV for S-boxes of dimension $n \in \{5, 7, 8\}$ and 25% for 6-bit S-boxes.

Next chapter presents the results of our article [50] we presented at the Conference on Cryptographic Hardware and Embedded Systems (CHES), in 2018.

Chapter 2

Mixing Additive and Multiplicative Masking for Secure Polynomial Evaluation Methods

The chapter is organized as follows. Section 1 provides background notions on masking and surrounding the probing model. We present GPQ in section 2 along with our t-SNI security proof. In Section 3 we recall aspects of the cyclotomic method, we address the extension of GPQ to the masking of generic S-boxes and we give security proofs regarding our alternate cyclotomic construction. In section 4, we describe the CRV method before showing how to derive an alternate approach that also enables to consider new parameters, and we also provide security proofs. Section 5 reports implementation results using our alternate approaches compared to the originals for S-boxes of dimension $n \in [4; 8]$. Eventually section 6 concludes the chapter.

Contents

2.1 Basics and Definitions

In this chapter, n denotes the bit-length of processed data. By default, variables in this paper are assumed to be defined in the field $\mathbb{F}_{2^n} \cong (\mathbb{F}_2[x]/(p(x)), \oplus, \otimes)$, where $p(x)$ is an irreducible polynomial of $\mathbb{F}_2[x]$ of degree n, $\oplus$ is the bitwise XOR operation and $\otimes$ denotes the polynomial multiplication modulo $p(x)$. These variables can also sometimes be viewed as elements of the vector space $\mathbb{F}_2^n$ defined over the field $(\mathbb{F}_2, \oplus, \odot)$, where $\odot$ is the AND operation. Some transformations may involve n-bit operations XOR, AND which shall be referred to by $\oplus^n, \odot^n$. The inverse of an element $x \in \mathbb{F}_{2^n}^*$ for the law $\otimes$ is x^{-1} where $\mathbb{F}_{2^n}^*$ denotes the set of invertible elements of $\mathbb{F}_{2^n}$.

2.1.1 Basics on Masking

As explained in the introduction, the masking countermeasure splits every sensitive variable x into $d = t+1$ shares $x_0, \ldots, x_d$ in such a way that the following relation is satisfied for a group operation $\perp$. Namely,

$$x_0 \perp x_1^{-1} \perp \ldots \perp x_d^{-1} = x\,. \tag{2.1}$$

where x_i^{-1} denotes the inverse of x_i w.r.t $\perp$. Usually, the d shares $x_1 \ldots, x_d$ are randomly generated and x_0 is processed so that (2.1) is satisfied. In this paper, $\perp$ either denotes the field addition $\oplus$ or the field multiplication $\otimes$. When $\perp = \oplus$ (resp. $\perp = \otimes$), the relation (2.1) induces an additive masking (resp. a multiplicative masking) of x. A $(d+1)$-tuple $(x_0, \ldots, x_d)$ satisfying (2.1) for $\perp = \oplus$ (resp. for $\perp = \otimes$) is called a d^{th} order additive (resp. d^{th} order multiplicative) sharing of x.

2.1.2 Useful t-SNI Gadgets

Several constructions in this article may involve gadgets whose security has already been analyzed in the literature. We hereafter recall the secure multiplication algo-

rithm as described in [56] and the mask refreshing procedure introduced by Duc, Dziembowski and Faust in [23]. Furthermore, it has been shown in [6] that both gadgets are t-SNI.

Algorithm 9 SecMult [56] (ISW)

Require: An order d, a $(d+1)$-sharing of x and a $(d+1)$-sharing of y.
Ensure: A $(d+1)$-sharing $(z_0, \ldots, z_d)$ of $(x \otimes y)$.

for $i = 0$ **to** d **do**
  $z_i \leftarrow x_i \otimes y_i$
end for
for $i = 0$ **to** d **do**
  for $j = i + 1$ **to** d **do**
    $r \xleftarrow{\$} \mathbb{F}_{2^n}$
    $z_i \leftarrow z_i \oplus r$
    $r \leftarrow x_i \otimes y_j \oplus r \oplus x_j \otimes y_i$
    $z_j \leftarrow z_j \oplus r$
  end for
end for
return $(z_0, \ldots, z_d)$

Alg. 10 presents the multiplication-based refreshing algorithm of [23].

Algorithm 10 Multiplication-Based Mask Refreshing Algorithm

Require: An order d and a $(d+1)$-sharing $(x_0, \ldots, x_d)$ of x.
Ensure: A $(d+1)$-sharing $(z_0, \ldots, z_d)$ of x.

for $i = 0$ **to** d **do**
  $z_i \leftarrow x_i$
end for
for $i = 0$ **to** d **do**
  for $j = i + 1$ **to** d **do**
    $r \xleftarrow{\$} \mathbb{F}_{2^n}$
    $z_i \leftarrow z_i \oplus r$
    $z_j \leftarrow z_j \oplus r$
  end for
end for
return $(z_0, \ldots, z_d)$

2.2 GPQ Scheme

Introduced by Genelle, Prouff and Quisquater in [27, 28, 29], the GPQ scheme securely evaluates power functions by mixing additive and multiplicative masking. Namely, (2.1) holds alternatively for $\perp = \oplus$ and $\perp = \otimes$. The additive masking is used to secure affine functions while multiplicative masking efficiently masks power functions as illustrated in Fig. 2.2. Thus, special transformations are necessary to convert an additive sharing into a multiplicative one and conversely. This strategy was initially addressed by Akkar and Giraud [2] but turned out to be not secure when a multiplicatively masked variable equals zero [31]. Genelle, Prouff and Quisquater solved this issue by proposing a secure implementation of the Dirac function that enables to multiplicatively mask the value zero [27, 28]. For the sake of self-completeness, we recall some algorithms of [27, 28, 29] that constitute GPQ and we also conduct a security analysis throughout this section to prove that the scheme actually satisfies the t-SNI property and not only the t-NI definition (as proven in the original paper).

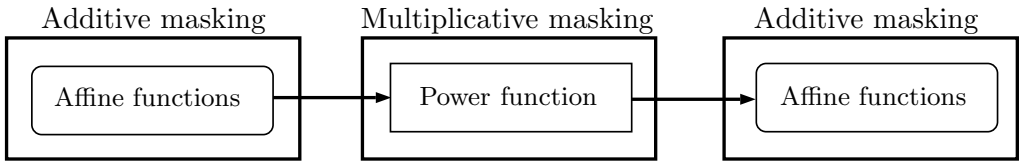


Figure 2.1: GPQ mixes additive and multiplicative masking.

From the above discussion, an additively masked element of $\mathbb{F}_{2^n}$ is mapped into $\mathbb{F}_{2^n}^*$ by adding it to its Dirac value so that the resulting non-zero element can be multiplicatively masked. Further details are given below.

2.2.1 Dirac

The Dirac function δ is defined over $\mathbb{F}_{2^n}$ by $\delta(x) = 1$ if $x = 0$ and $\delta(x) = 0$ otherwise. Hence for any $x \in \mathbb{F}_{2^n}$, it results $(x \oplus \delta(x)) \in \mathbb{F}_{2^n}^*$. The computation of the Dirac function of $x \in \mathbb{F}_{2^n}$ may be performed as follows.

Let $\overline{x} = (\overline{x_0}, \ldots, \overline{x_{n-1}})$ denote the bitwise complement of $x = (x_0, \ldots, x_{n-1})$, we have

$$\delta(x) = \overline{x_0} \odot \overline{x_1} \odot \cdots \odot \overline{x_{n-1}}\,,$$

where $\odot$ denotes the AND operation.

Computing one Dirac function at a time for several field elements may not be interesting in terms of efficiency (due to AND operations that have to be secured with ISW). However, bit-slicing enables to compute several Dirac functions simultaneously at a reasonable cost [28]. The latter approach is therefore preferred. In a nutshell, it computes the Dirac function of n elements of $\mathbb{F}_{2^n}$ viewed as a $(n \times n)$-matrix whose lines are actually treated as elements of $\mathbb{F}_2^n$. In the following, the field elements involved in the Secure-Dirac procedure are referred to as $x^{(k)}$ with $k \in \{0, 1, \ldots, n-1\}$. We hereafter recall the resulting algorithm that we also used in our implementations.

Algorithm 11 Secure-Dirac

Require: An order d, a length n and a $(d+1)$-sharing $(M_0, \cdots, M_d)$ of a binary $(n \times n)$-matrix M whose lines are the $x^{(k)}$'s.

Ensure: A $(d+1)$-sharing $(\Delta_0, \cdots, \Delta_d)$ of the n-bit vector $\Delta = (\delta(x^{(0)}), \cdots, \delta(x^{(n-1)}))$.

*** Compute the bitwise complement $\overline{M_0}$ of the $(n \times n)$-matrix M_0.*

1: $M_0 \longleftarrow \overline{M_0}$

*** Transpose the $(n \times n)$ matrices M_i for every $i \leq d$.*

2: **for** $i = 0$ **to** d **do**

3: $\quad t_i \longleftarrow (M_i)^{\mathsf{T}}$

4: **end for**

*** Refresh the shares.*

5: $(t_0^{(0)}, \ldots, t_d^{(0)}) \longleftarrow \text{Refresh}(t_0^{(0)}, \ldots, t_d^{(0)})$

6: $(\Delta_0, \ldots, \Delta_d) \longleftarrow (t_0^{(0)}, \ldots, t_d^{(0)})$

*** Process the Dirac computations.*

7: **for** $i = 1$ **to** $n-1$ **do**

8: $\quad (\Delta_0, \cdots, \Delta_d) \longleftarrow (\Delta_0, \cdots, \Delta_d) \odot^n (t_0^{(i)}, \ldots, t_d^{(i)})$

9: **end for**

10: **return** $(\Delta_0, \cdots, \Delta_d)$

The $\odot^n$ operation (Step 8 of Alg. 11) performs n secure multiplications over $\mathbb{F}_2$.

Remark 2. *In order to prove the following Lemma, we had to add a refreshing procedure (step 5) that was not originally required. In particular, this step requires the use of Alg. 10.*

Lemma 2.2.1. Secure-Dirac$(\cdot)$ *is* t-SNI. *Let $(M_i)_{0 \leq i \leq d}$ be the input and let $(\Delta_i)_{0 \leq i \leq d}$ be the output of Alg. 11. For any adversary set of at most t probed wires $\Omega = (\mathcal{I}, \mathcal{O})$, with $t \leq d$, there exists a set of input shares $\mathcal{S}$ such that $|\mathcal{S}| \leq |\mathcal{I}|$ and $\mathcal{S}$ is sufficient to simulate the adversary observation set Ω.*

Proof. For the sake of clarity, we divide the bit-sliced Secure-Dirac procedure into two stages as illustrated Fig. 2.2 and we give further details about the transformation before proving its security.

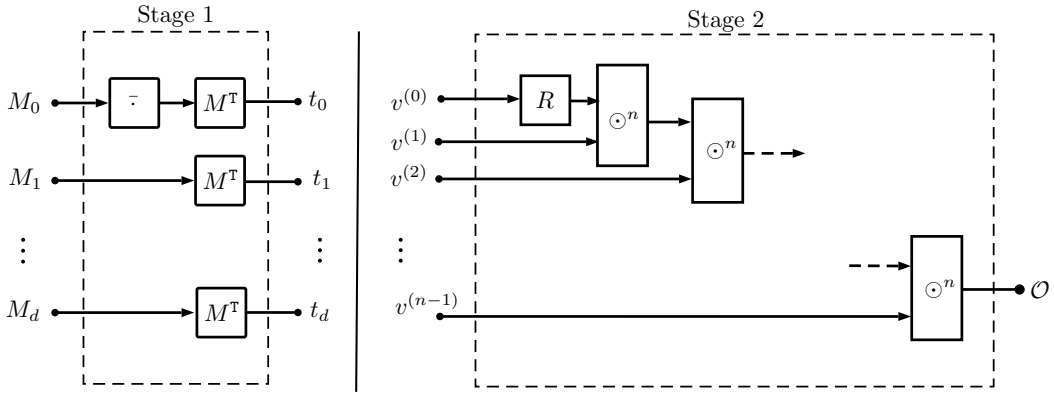


Figure 2.2: Gadget $\delta(\cdot)$.

Stage 1 is composed of the steps 1 to 4 of Alg. 11 and stage 2 illustrates the steps 5 to 9. As mentionned previously, the Secure-Dirac procedure uses bit-slicing and therefore processes simultaneously several elements of $\mathbb{F}_2^n$. We consider the case of n-bit architectures for which the transformation processes n elements of $\mathbb{F}_2^n$ at a time. These n elements are represented by a matrix $M \in \mathbb{F}_2^{n\times n}$ in such a way that each line of M is one element of $\mathbb{F}_2^n$. Since the procedure manipulates masked data, stage 1 therefore takes as input a $(d+1)$-sharing $(M_0, \ldots, M_d)$ of M, with $M_j \in \mathbb{F}_2^{n\times n}$ for every $j \in [0; d]$. We hereafter exhibit the matrices involve in the computation. Namely, we have

$$M_j = \begin{pmatrix} \left(\pi_0\left(x^{(0)}\right)\right)_j & \left(\pi_1\left(x^{(0)}\right)\right)_j & \cdots & \left(\pi_{n-1}\left(x^{(0)}\right)\right)_j \\ \left(\pi_0\left(x^{(1)}\right)\right)_j & \left(\pi_1\left(x^{(1)}\right)\right)_j & \cdots & \left(\pi_{n-1}\left(x^{(1)}\right)\right)_j \\ \vdots & \vdots & \ddots & \vdots \\ \left(\pi_0\left(x^{(n-1)}\right)\right)_j & \left(\pi_1\left(x^{(n-1)}\right)\right)_j & \cdots & \left(\pi_{n-1}\left(x^{(n-1)}\right)\right)_j \end{pmatrix}$$

where $\left(\pi_k\left(x^{(i)}\right)\right)_j$ is the projection of the k^{th} bit of the j^{th} share of the element $x^{(i)}$ with i in $[0; n-1]$, j in $[0; d]$ and k in $[0; n-1]$.

Stage 1 transposes the matrices M_j for every $j \in [0; d]$. Note that the bit-wise complement (step 1 of Alg. 11) is only performed over the elements of M_0. The transposed matrices are denoted by t_j and we therefore have $t_0 = \left(\overline{M_0}\right)^\intercal$ and

$t_j = (M_j)^\intercal$ for $j \in [1; d]$. Then, stage 1 outputs the $d+1$ binary $(n \times n)$-matrices $t_0, \ldots, t_d$ such that

$$t_j = \begin{pmatrix} \left(\pi_0\left(x^{(0)}\right)\right)_j & \left(\pi_0\left(x^{(1)}\right)\right)_j & \ldots & \left(\pi_0\left(x^{(n-1)}\right)\right)_j \\ \left(\pi_1\left(x^{(0)}\right)\right)_j & \left(\pi_1\left(x^{(1)}\right)\right)_j & \ldots & \left(\pi_1\left(x^{(n-1)}\right)\right)_j \\ \vdots & \vdots & \ddots & \vdots \\ \left(\pi_{n-1}\left(x^{(0)}\right)\right)_j & \left(\pi_{n-1}\left(x^{(1)}\right)\right)_j & \ldots & \left(\pi_{n-1}\left(x^{(n-1)}\right)\right)_j \end{pmatrix}.$$

Finally, stage 2 takes as inputs n distinct vectors $v^{(k)} = \left(t_0^{(k)}, \ldots, t_d^{(k)}\right)$ where $t_j^{(k)}$ is the k^{th} line of the matrix t_j. In other words, $t_j^{(k)}$ is a n-tuple composed of the k^{th} bits of the j^{th} shares of all input elements and $v^{(k)}$ is therefore composed of the k^{th} bit of all the shares of every input elements. In the following proof, we assume that if a single internal bit has to be simulated then the whole word corresponding to this single bit is required.

As in [6], the proof is constructed by composition. Namely, we construct the simulator for the whole circuit by simulating sequentially each inner gadget from right to left. We begin our security analysis by stage 2 which we also divide into two parts (see Fig. 2.3).

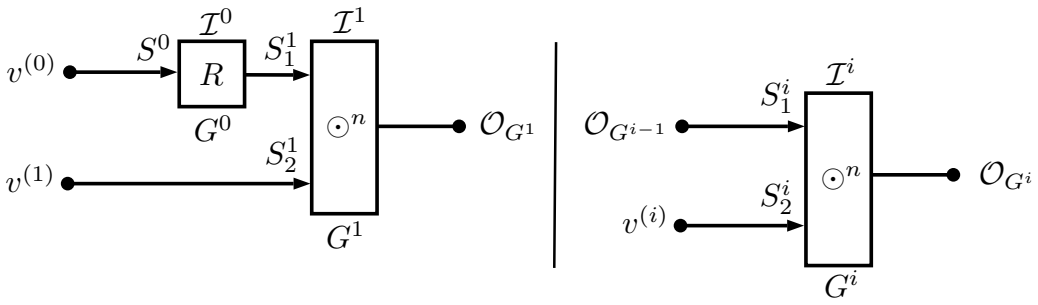


Figure 2.3: Stage 2 of Gadget $\delta(\cdot)$.

Let $\Omega = (\mathcal{I}, \mathcal{O})$ be an observation set made on stage 2 such that $\mathcal{I} = \bigcup_{0 \leq i \leq n-1} \mathcal{I}^i$ and such that the global constraint $\sum_{i=0}^{n-1} |\mathcal{I}^i| + |\mathcal{O}| \leq t$ is satisfied.

Let us first consider the right side of Fig. 2.3. For every $i \in [2, n-1]$, we want to simulate the observation set $\Omega^i = (\mathcal{I}^i, \mathcal{O}_{G^i})$ made on Gadget i. Since Gadget i is t-SNI and $|\mathcal{I}^{n-1} \cup \mathcal{O}| \leq t$ (by global constraint) and $|\mathcal{I}^i \cup \mathcal{O}_{G^i}| \leq t$ for $i \in [2, n-2]$ (by global constraint and simulation of Gadget $i+1$), we know that there exists

an observation set $\mathcal{S}^i = (\mathcal{S}_1^i, \mathcal{S}_2^i)$ such that $|\mathcal{S}_1^i| \leq |\mathcal{I}^i|$, $|\mathcal{S}_2^i| \leq |\mathcal{I}^i|$ and $\mathcal{S}_1^i \cup \mathcal{S}_2^i$ is sufficient to simulate Gadget i (*i.e.* simulate Ω^i) for every $i \in [2, n-1]$. As illustrated Fig. 2.3 and by the t-SNI property of Gadget i for every $i \in [2, n-1]$, the simulation of these Gadgets therefore requires at most $|\mathcal{I}^i|$ shares of $v^{(i)}$, and at most $|\mathcal{I}^i|$ shares of the output of Gadget $i-1$ for every $i \in [2, n-1]$.

Let us now also take into account the left side of Fig. 2.3. It has been shown in [6] that such a composition of t-SNI gadgets is t-SNI thanks to the additional t-SNI refreshing Gadget (Alg. 10). Thus, in order to simulate the observation set $\Omega^{0,1} = ((\mathcal{I}^0 \cup \mathcal{I}^1), \mathcal{O}_{G^1})$ and since $|\mathcal{I}^0 \cup \mathcal{I}^1 \cup \mathcal{O}_{G^1}| \leq t$ (by global constraint and simulation of Gadgets i for every $i \in [2, n-1]$), the corresponding simulator requires at most $|\mathcal{I}^0|$ shares of $v^{(0)}$ and at most $|\mathcal{I}^1|$ shares of $v^{(1)}$.

Altogether, the simulation of stage 2 requires at most $|\mathcal{I}^i|$ shares of $v^{(i)}$ for every $i \in [0, n-1]$.

Let us now take into account stage 1. As mentioned previously, $v^{(i)} = \left(t_0^{(i)}, \ldots, t_d^{(i)}\right)$ for every $i \in [0, n-1]$, which means that the $v^{(i)}$'s are composed of the i^{th} bit of all the shares of every input elements. Let us also remind that we assume that if a single internal bit has to be simulated then the whole word corresponding to this single bit is required. For the sake of clarity, we illustrate in Fig. 2.4 the propagation of a share $v^{(i)}$ throughout stage 1. As an example, we consider the case where the simulation requires the first share $t_0^{(i)}$ of $v^{(i)}$ and show how it is actually related to the first shares of each input elements.

The figure 2.4 below shows that if the simulation requires the j^{th} share of $v^{(i)}$, then the simulation actually requires the j^{th} shares of all input elements. Moreover and as discussed above, the simulation of the procedure requires $|\mathcal{I}^i|$ shares of $v^{(i)}$ for every $i \in [0, n-1]$. Oberserve that the simulation may involve $|\mathcal{I}^i|$ distinct shares of $v^{(i)}$ for every $i = [0, n-1]$. Therefore, at most $\sum_{i=0}^{n-1} |\mathcal{I}^i|$ distinct shares of each input element of the Secure-Dirac procedure are actually required. Also, by the global constraint $\sum_{i=0}^{n-1} |\mathcal{I}^i| \leq t$ and consequently the t-SNI property is satisfied for the whole circuit of the Secure-Dirac procedure. □

Remark 3. *In order to satisfy the t-SNI property, the shares of $v^{(0)}$ have to be refreshed in stage 2 thanks to Algorithm 10. This mask refreshing was not required in the original approach that only proves the security of* Secure-Dirac *under the less stronger t-NI security definition.*

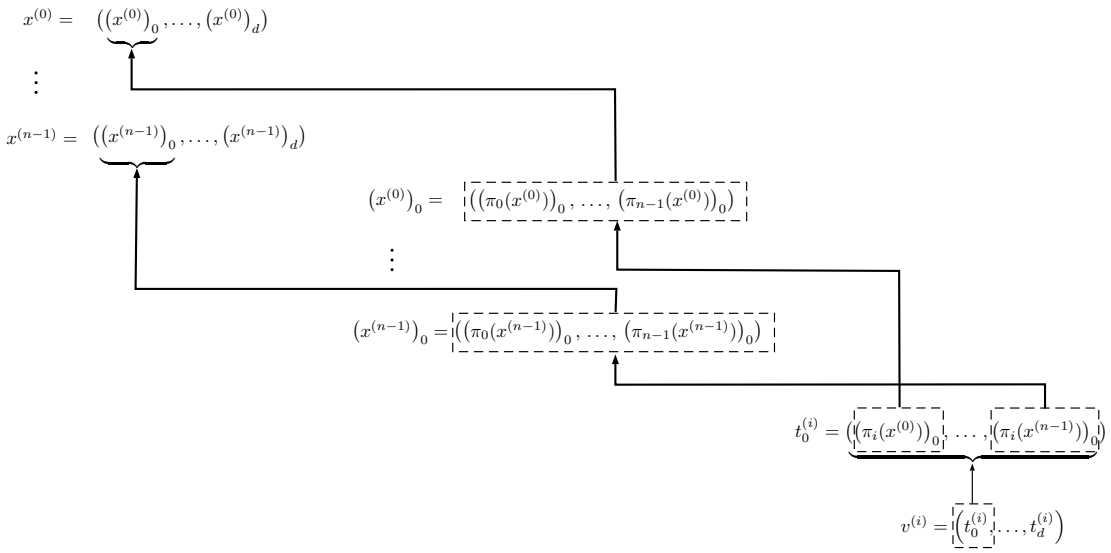


Figure 2.4: Linking the shares of the v^i's to the shares of the input elements.

For a given set of n additively masked field elements, their Dirac values can be computed with Alg. 11 and have to be added to their corresponding elements before converting them into multiplicative maskings. The complexity of the Secure-Dirac procedure is given at the end of this section. We now address the conversion transformations that enable to switch encodings for a non-zero masked element between its additive and multiplicative sharing.

2.2.2 Conversions

The general strategy consists in replacing sequentially each additive (resp. multiplicative) mask of the $(d+1)$-additive (resp. multiplicative) sharing of an element $x \in \mathbb{F}_{2^n}^*$ by a multiplicative (resp. additive) one. This strategy results in the following two algorithms. Alg. 12 describes the steps for an additive to multiplicative masking conversion and Alg. 13 describes the multiplicative to additive masking conversion. As in [29], these transformations are respectively called AMtoMM and MMtoAM.

The conversion AMtoMM described in algorithm 12 has been proven in [29] to satisfy the t-NI definition. We now prove the following theorem that states that AMtoMM$(\cdot)$ actually satisfies the t-SNI requirements.

Theorem 2.2.2. AMtoMM$(\cdot)$ *conversion is* t-SNI. *Let* $(x_i)_{0\leq i\leq d}$ *be the input and let* $(z_i)_{0\leq i\leq d}$ *be the output of Alg. 12. For any adversary set of at most* t *probed*

Algorithm 12 AMtoMM

Require: A $(d+1)$-additive sharing $(x_0, \ldots, x_d)$ of $x \in \mathbb{F}_{2^n}^*$
Ensure: A $(d+1)$-multiplicative sharing $(z_0, \ldots, z_d)$ of $x \in \mathbb{F}_{2^n}^*$

$z_0 \leftarrow x_0$
for $i = 1$ **to** d **do**
  $z_i \xleftarrow{\$} \mathbb{F}_{2^n}^*$
  $z_0 \leftarrow z_0 \dot{\otimes} z_i$
  for $j = 1$ **to** $d - i$ **do**
    $U \xleftarrow{\$} \mathbb{F}_{2^n}$
    $x_j \leftarrow z_i \dot{\otimes} x_j$
    $**$ *Refreshing of the additive share*
    $x_j \leftarrow x_j \oplus U$
    $z_0 \leftarrow z_0 \oplus x_j$
    $x_j \leftarrow U$
  end for
  $x_{d-i+1} \leftarrow z_i \dot{\otimes} x_{d-i+1}$
  $z_0 \leftarrow z_0 \oplus x_{d-i+1}$
end for
return $(z_0, z_1, \ldots, z_d)$

wires $\Omega = (\mathcal{I}, \mathcal{O})$, with $t \leq d$, there exists a set of input shares $\mathcal{S}$ such that $|\mathcal{S}| \leq |\mathcal{I}|$ and $\mathcal{S}$ is sufficient to simulate the adversary observation set Ω.

The AMtoMM$(\cdot)$ transformation converts an additively masked element $x \in \mathbb{F}_{2^n}^*$ into a multiplicative masking. Initially x is represented by a $(d+1)$-additive sharing $(X_0, \ldots, X_d)$ involving d additive masks $(X_i)_{1 \leq i \leq d}$ such that $\sum_{i=0}^{d} X_i = x$. A sequence of transformations is carried out over the successive intermediate maskings of x to finally produce a $(d+1)$-multiplicative sharing $(Z_0, \ldots, Z_d)$ of x such that $\prod_{i=0}^{d} Z_i = x$.

More precisely, Alg. 12 randomly generates multiplicative masks $(Z_i)_{0 \leq i \leq d-1}$ and computes the sequence $X^{(i+1)} = \pi_i(X^{(i)}, Z_i)$ for i in $[0; d-1]$ where $X^{(i)} = (X_0^{(i)}, X_1^{(i)}, \ldots, X_{d-i}^{(i)})$. Thus, $X^{(0)}$ is the input of Alg. 12, $X^{(d)}$ is the output and the other $X^{(i)}$'s are intermediate maskings of x. We illustrate the above discussion in Fig. 2.5.
The $\mathcal{I}^i$'s are random vectors whose components are some intermediate variables that appear during the corresponding transformation π_i and may also be composed of some of $X^{(i)}$'s shares. The $\mathcal{S}^i$'s specify which components of $X^{(i)}$ are considered. In the following we denote by $\widetilde{X}^{(i)}$ vectors only composed of the shares of $X^{(i)}$ and specified by $\mathcal{S}^i$. Also, $\mathcal{O} = \widetilde{X}^{(d)} \cup Z_{|_J}$ where $Z_{|_J} = (Z_i)_{i \in J}$.

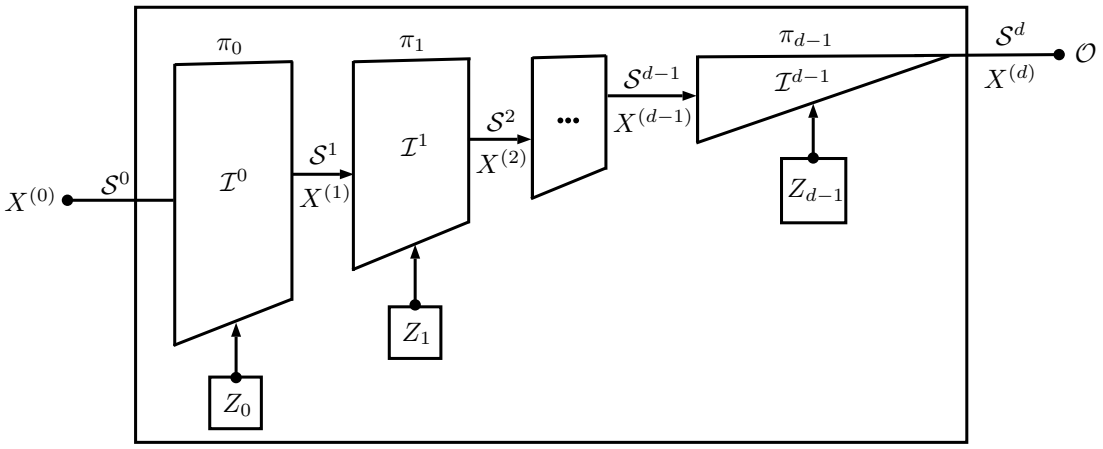


Figure 2.5: Internal functioning of Gadget AMtoMM$(\cdot)$.

Our goal is to prove that Gadget AMtoMM$(\cdot)$ is t-SNI.

Proof. Let $\Omega = (\mathcal{I}, \mathcal{O})$ be an adversary observation set constructed over Gadget AMtoMM$(\cdot)$ with $\mathcal{I} = \cup_{i=0}^{d-1} \mathcal{I}^i$ such that the global constraint $|\mathcal{I} \cup \mathcal{O}| \leq t$ is satisfied. In order to prove that Alg. 12 is t-SNI, we prove that $\Omega = (\mathcal{I}, \mathcal{O})$ may be simulated from a set of its input shares $\mathcal{S}^0$ with $|\mathcal{S}^0| \leq |\mathcal{I}|$. More precisely, we prove that any adversary view Ω satisfying the global constraint may be expressed as a function ρ of $\widetilde{X}^{(0)}$ and a uniform random vector $\mathcal{U}$ such that $\Omega = \rho(\widetilde{X}^{(0)}, \mathcal{U})$, where $\widetilde{X}^{(0)} \in \mathbb{F}_{2^n}^{|\mathcal{S}^0|}$ and $|\mathcal{S}^0| \leq |\mathcal{I}|$. This is achieved in two steps. We first prove that the adversary view Ω may be expressed in terms of a vector $(\widetilde{X}^{(0)}, \widetilde{X}^{(1)}, \ldots, \widetilde{X}^{(d)}, z_{|J \cup L})$ and a uniform random vector $(\mathcal{U}_{d-2}, \cdots, \mathcal{U}_0)$. We then prove that $(\widetilde{X}^{(0)}, \widetilde{X}^{(1)}, \ldots, \widetilde{X}^{(d)}, z_{|J \cup L})$ may be expressed as $h(\widetilde{X}^{(0)}, \mathcal{U}')$ with $|\mathcal{S}^0| \leq |\mathcal{I}|$ and $\mathcal{U}'$ is a uniform random vector.

Let us first build the sequence of $\mathcal{S}^i$'s.

If $X^{(d)}$ is a component of $\mathcal{O}$, then $\mathcal{S}^d = 0$ and thus $|\mathcal{S}^d| = 1$, otherwise $\mathcal{S}^d = \emptyset$ and $|\mathcal{S}^d| = 0$. Then the other $\mathcal{S}^i$'s are defined according to the following discussion.

We start from the end of the evaluation of AMtoMM$(\cdot)$ and we therefore first consider the transformation

$$\pi_{d-1}(X^{(d-1)}, Z_{d-1}) \longrightarrow (X^{(d)}),$$

where $X^{(d-1)} = (X_0^{(d-1)}, X_1^{(d-1)})$ and $X^{(d)} = X_0^{(d)}$.

We list hereafter the variables computed during this transformation :

- $W_0^{(d)} = X_0^{(d-1)} \cdot Z_{d-1}$
- $W_1^{(d)} = X_1^{(d-1)} \cdot Z_{d-1}$
- $X_0^{(d)} = W_0^{(d)} + W_1^{(d)}$

Let $\mathcal{I}^{d-1}$ be a subset of $\{X_0^{(d-1)}, X_1^{(d-1)}, W_0^{(d)}, W_1^{(d)}\}$ and $\overset{\circ}{\mathcal{I}}{}^{d-1} = \mathcal{I}^{d-1} \backslash \{X_0^{(d-1)}, X_1^{(d-1)}\}$. On the one hand, if $\overset{\circ}{\mathcal{I}}{}^{d-1} = \emptyset$ then $\mathcal{I}^{d-1}$ is only composed of input variables of π_{d-1}, thus all variables of $\mathcal{I}_{d-1}$ may be expressed from $\widetilde{X}^{(d-1)} \in \mathbb{F}_{2^n}^{|S^{d-1}|}$ with $|S^{d-1}| = |\mathcal{I}^{d-1}|$. On the other hand, if $\overset{\circ}{\mathcal{I}}{}^{d-1} \neq \emptyset$, noting that $W_0^{(d)}$ (resp. $W_1^{(d)}$) can be expressed from $X_0^{(d-1)}$ (resp. $X_1^{(d-1)}$) and Z_{d-1}, $\mathcal{I}^{d-1}$ may be expressed in terms of $\widetilde{X}^{(d-1)}$ with $|S^{d-1}| \leq |\mathcal{I}^{d-1}|$. Consequently in any cases, all variables of $\mathcal{I}^{d-1}$ may be expressed as a function of Z_{d-1} and $\widetilde{X}^{(d-1)} \in \mathbb{F}_{2^n}^{|S^{d-1}|}$, i.e.

$$\mathcal{I}^{d-1} = \rho_{d-1}(\widetilde{X}^{(d-1)}, Z_{d-1}) \text{ with } |S^{d-1}| \leq |\mathcal{I}^{d-1}|\,. \tag{2.2}$$

Let us define the sets $S_i = \{0, \ldots, d-i\}$ and $T_i = \{1, \ldots, d-i-1\}$ for $i = (d-2)\cdots 0$.

Our goal is now to prove that for any $i = (d-2)\cdots 0$, for any $\mathcal{I}^i$ with $\overset{\circ}{\mathcal{I}}{}^i \neq \emptyset$ and for any subset $\mathcal{S}^{i+1} \subseteq S_{i+1}$ there exists a subset $\mathcal{S}^i \subseteq S_i$, a uniform vector $\mathcal{U}_i$ stochastically independent of all other random vectors and an application ρ_i such that

$$\mathcal{I}^i = \rho_i(\widetilde{X}^{(i)}, \widetilde{X}^{(i+1)}, Z_i, \mathcal{U}_i) \text{ with } |\mathcal{S}^i| \leq |\mathcal{I}^i| + |\mathcal{S}^{i+1}|\,. \tag{2.3}$$

We list hereafter the variables involved in the π_i transformation, with $i \in \{0, \ldots, d-2\}$:

- $X_j^{(i+1)} \sim \mathcal{U}(\mathbb{F}_{2^n})$ with $j \in T_i$
- $W_j^{(i+1)} = X_j^{(i)} \cdot Z_i$ with $j \in S_i$
- $Y_j^{(i+1)} = W_j^{(i+1)} + X_j^{(i+1)}$ with $j \in T_i$
- $H_j^{(i+1)} = W_0^{(i+1)} + \sum_{t=1}^{j} Y_t^{(i+1)} = H_{j-1}^{(i+1)} + Y_j^{(i+1)}$ with $j \in T_i$. Note that $H_0^{(i+1)} = W_0^{(i+1)}$.
- $X_0^{(i+1)} = H_{d-i-1}^{(i+1)} + W_{d-i}^{(i+1)}$.

We have

$$\mathcal{I}^i = (X^{(i)}_{|_{I_i}}, W^{(i+1)}_{|_{K_i}}, Y^{(i+1)}_{|_{L_i}}, H^{(i+1)}_{|_{Q_i}}),$$

with $I_i \subseteq S_i$, $K_i \subseteq S_i$, $L_i \subseteq T_i$, $Q_i \subseteq T_i$.

First note that $X^{(i)}_{|_{I_i}}$ and $W^{(i)}_{|_{K_i}}$ may be expressed as a function of $X^{(i)}_{|_{I_i \cup K_i}}$ and Z_i. Therefore, $\mathcal{S}^i \supseteq I_i \cup K_i$. For any $j \in \mathcal{S}^{i+1} \cap L_i$, $\widetilde{X}^{(i+1)}$ and $X^{(i)}_j$ are necessary to simulate $Y^{(i+1)}_j$. Therefore, $\mathcal{S}^i \supseteq \mathcal{S}^{i+1} \cap L_i$. For any $j \in L_i \setminus \mathcal{S}^{i+1}$, $Y^{(i+1)}_j$ may be simulated from a uniform random variable stochastically independent from any other random variables.

If $Q_i = \emptyset$, $\mathcal{I}^i$ may be expressed as a function of $\widetilde{X}^{(i+1)}$, $\widetilde{X}^{(i)}$ (with $\mathcal{S}^i = I_i \cup K_i \cup (\mathcal{S}^{i+1} \cap L_i)$), Z_i and a uniform random vector $\mathcal{U}_i$ stochastically independent from any other random variables. It follows that

$$|\mathcal{S}^i| \leq |I_i| + |K_i| + |\mathcal{S}^{i+1}| \leq |\mathcal{I}^i| + |\mathcal{S}^{i+1}|.$$

The condition (2.3) is therefore satisfied in this case.

If $Q_i \neq \emptyset$, i.e. $Q_i = \{s_1, \cdots, s_t\}$. Applying a well determined invertible linear application to $H^{(i+1)}_{|_{Q_i}}$, we observe that the simulation of $H^{(i+1)}_{|_{Q_i}}$ is equivalent to the simulation of the vector

$$\left(X^{(i)}_0 \cdot Z_i + \sum_{j=1}^{s_1} Y^{(i+1)}_j, \sum_{j=s_1+1}^{s_2} Y^{(i+1)}_j, \ldots, \sum_{j=s_{t-1}+1}^{s_t} Y^{(i+1)}_j \right).$$

If the interval $[1; s_1] \subseteq \mathcal{S}^{i+1}$ then the first component of the above vector may be expressed as a function of $\widetilde{X}^{(i+1)}$, Z_i and the random variables $X^{(i)}_j$ for $j \in [1; s_1] \cup \{0\}$. Otherwise, the first component may be simulated from a uniform random variable stochastically independent from any other random variables. If the interval $[s_i; s_{i+1}] \subseteq \mathcal{S}^{i+1}$ then the corresponding component of the vector may be expressed as a function of $\widetilde{X}^{(i+1)}$, Z_i and the random variables $X^{(i)}_j$ for $j \in [s_i; s_{i+1}]$. Otherwise, this component may be simulated from a uniform random variable stochastically independent from any other random variables. It follows that $\mathcal{I}^i$ may be expressed as a function of $\widetilde{X}^{(i)}$ (with $\mathcal{S}^i = I_i \cup K_i \cup (\mathcal{S}^{i+1} \cap L_i) \cup \mathcal{S}^{i+1} \cup \{0\}$), Z_i and a uniform random vector $\mathcal{U}_i$ stochastically independent from any other random variables. Since $Q_i \neq \emptyset$ by assumption, it follows that

$$|\mathcal{S}^i| \leq |I_i| + |K_i| + |L_i| + |\mathcal{S}^{i+1}| + 1 \leq |\mathcal{I}^i| + |\mathcal{S}^{i+1}|.$$

The condition (2.3) is therefore satisfied in this case.

Observe that for any $\mathcal{I}^i$ with $\overset{\circ}{\mathcal{I}}{}^i = \emptyset$ we have $\mathcal{I}^i = \widetilde{X}^{(i)}$ with $\mathcal{S}^i = \mathcal{I}^i$. In this case, we have therefore $|\mathcal{S}^i| = |\mathcal{I}^i|$.
Define $L = \{i \mid \overset{\circ}{\mathcal{I}}{}^i \neq \emptyset\}$ with $\overset{\circ}{\mathcal{I}}{}^i = \mathcal{I}^i \backslash \widetilde{X}^{(i)}$. Let us prove that $(\widetilde{X}^{(0)}, \widetilde{X}^{(1)}, \ldots, \widetilde{X}^{(d)}, Z_{|_{J\cup L}})$ may be expressed as $h(\widetilde{X}^{(0)}, \mathcal{U}')$ with $|\mathcal{S}^0| \leq |\mathcal{I}|$, $\mathcal{U}'$ is a uniform random variable stochastically independent of $\widetilde{X}^{(0)}$ and $\widetilde{X}^{(i)}$ are vectors only composed of the shares of $X^{(i)}$ which are specified by $\mathcal{S}^i$.

Suppose that there does not exist an indice k such that $|\mathcal{S}^k| = d - k + 1$, then none of the sets $\mathcal{S}^i$ are $S_i = \{0, \ldots, d - i\}$ and thus all the $\widetilde{X}^{(i)}$'s and $Z_{|_{J\cup L}}$ are uniform stochastically independent random vectors. It follows that $(\widetilde{X}^{(0)}, \widetilde{X}^{(1)}, \ldots, \widetilde{X}^{(d)}, Z_{|_{J\cup L}})$ may clearly be expressed as $h(\widetilde{X}^{(0)}, \mathcal{U}')$.

Define now k as the smallest index such that $|S^k| = d - k + 1$. Note that $k \geq 1$ by the global constraint $|\mathcal{I} \cup \mathcal{O}| \leq t$.

For the case $k = d$, all the $\widetilde{X}^{(i)}$ with i in $[0; d - 1]$ are uniform stochastically independent random vectors of $\mathbb{F}_{2^n}^{|\mathcal{S}^i|}$ respectively. Also, $(\widetilde{X}^{(d)}, Z_{|_{J\cup L}})$ is a uniform random vector of $(\mathbb{F}_{2^n}^*)^{1+|J\cup L|}$ with $|J \cup L| \leq t - 1$ according to the global constraint $|\mathcal{I} \cup \mathcal{O}| \leq t$. It follows that if $k = d$ then $(\widetilde{X}^{(0)}, \widetilde{X}^{(1)}, \ldots, \widetilde{X}^{(d)}, Z_{|_{J\cup L}})$ may be expressed as $h(\widetilde{X}^{(0)}, \mathcal{U}')$.

Let us now assume that $1 \leq k \leq d - 1$. Gathering conditions (2.2) and (2.3), we have $|\mathcal{S}^k| \leq \sum_{i=k}^{d-1} |\mathcal{I}^i| \leq |\mathcal{I}|$. Also, $|\mathcal{I}| + |\mathcal{O}| \leq |\mathcal{I} \cup \mathcal{O}| \leq t$ by the global constraint. It follows that $|\mathcal{O}| \leq t - |\mathcal{S}^k|$. Note that

$$X^{(k)} = (\underbrace{x \cdot \prod_{i=0}^{k-1} Z_i + \sum_{j=1}^{d-i} X_j^{(k)}}_{X_0^{(k)}}, X_1^{(k)}, \ldots, X_{d-k}^{(k)}).$$

Remembering that $t \leq d$ and that $|S^k| = d - k + 1$, we have $|\mathcal{O}| \leq k - 1$. It follows that at most $k - 1$ Z_i's have to be simulated among the k Z_i's in the expression of $X_0^{(k)}$, *i.e.* $p^* = x \cdot \prod_{i=0}^{k-1} Z_i$ is therefore a uniform random variable of $\mathbb{F}_{2^n}^*$ stochastically independent of the random variables in the set $\mathcal{O}$. All other random variables $\widetilde{X}^{(i)}$ with $i > k$ and $Z_{|_{(J\cup L)\cap\{k,\ldots,d-1\}}}$ may be build from p^*, random vectors of $\mathbb{F}_{2^n}^{|\mathcal{S}^i|}$ and $(\mathbb{F}_{2^n}^*)^{|(J\cup L)\cap\{k,\ldots,d-1\}|}$. From the above discussion, it follows that $(\widetilde{X}^{(0)}, \widetilde{X}^{(1)}, \ldots, \widetilde{X}^{(d)}, Z_{|_{J\cup L}})$ may be expressed as $h(\widetilde{X}^{(0)}, \mathcal{U}')$ with $|\mathcal{S}^0| \leq |\mathcal{I}|$ and $\mathcal{U}'$ is a uniform random vector.

Finally, from conditions (2.2) and (2.3) it follows that $\Omega = (\mathcal{I}, \mathcal{O})$ may be expressed as $\rho(\widetilde{X}^{(0)}, \mathcal{U})$ which means that $\mathsf{AMtoMM}(\cdot)$ is t-SNI. □

The other conversion that deals with getting an additive masking from a multiplicative one is described by Alg. 13. This conversion has only been proven to satisfy the t-NI property. We prove similarly to the previous conversion that it satisfies the stronger security definition.

Algorithm 13 MMtoAM

Require: A $(d+1)$-multiplicative sharing $(z_0, \ldots, z_d)$ of $x \in \mathbb{F}_{2^n}^*$
Ensure: A $(d+1)$-additive sharing $(x_0, \ldots, x_d)$ of $x \in \mathbb{F}_{2^n}^*$

$x_0 \leftarrow z_0$
for $i = 1$ **to** d **do**
  $x_i \xleftarrow{\$} \mathbb{F}_{2^n}$
  $x_0 \leftarrow x_0 \oplus x_i$
  $x_0 \leftarrow x_0 \dot{\otimes} z_i^{-1}$
  for $j = 1$ **to** i **do**
    $x_j \leftarrow x_j \dot{\otimes} z_i^{-1}$
    $U \xleftarrow{\$} \mathbb{F}_{2^n}$
    ** *Refreshing of the additive share*
    $x_j \leftarrow x_j \oplus U$
    $z_0 \leftarrow z_0 \oplus x_j$
    $x_j \leftarrow U$
  end for
end for
return $(x_0, x_1, \ldots, x_d)$

Theorem 2.2.3. $\mathsf{MMtoAM}(\cdot)$ *conversion is* t-SNI. *Let* $(x_i)_{0 \le i \le d}$ *be the input and let* $(z_i)_{0 \le i \le d}$ *be the output of Alg. 13. For any adversary set of at most* t *probed wires* $\Omega = (\mathcal{I}, \mathcal{O})$, *with* $t \le d$, *there exists a set of input shares* $\mathcal{S}$ *such that* $|\mathcal{S}| \le |\mathcal{I}|$ *and* $\mathcal{S}$ *is sufficient to simulate the adversary observation set* Ω.

Proof. The proof concerning $\mathsf{MMtoAM}(\cdot)$ is very similar to the one of $\mathsf{AMtoMM}(\cdot)$. It consists essentially in interchanging the role of the additive masks with the multiplicative ones in the previous proof of $\mathsf{AMtoMM}(\cdot)$. □

The GPQ scheme involves Alg. 11, 12 and 13. We now give further details about the evaluation of power functions with GPQ.

2.2.3 Power Function Processing

We outline the processing of a power function as follows. Consider a power $\alpha \in [0, 2^n - 1]$ and an element $x \in \mathbb{F}_{2^n}$ that is initially additively masked. First, the GPQ processing requires to compute the Dirac function of x and add the result to it in order to map the field element into $\mathbb{F}^*_{2^n}$. Then, x is converted into a multiplicative sharing in order to process the power function $x \mapsto x^\alpha$. Afterwards, x^α is converted back into an additive sharing and the resulting element is mapped from $\mathbb{F}^*_{2^n}$ back into $\mathbb{F}_{2^n}$ to be further processed by linear operations only. This processing is illustrated Fig. 2.6.

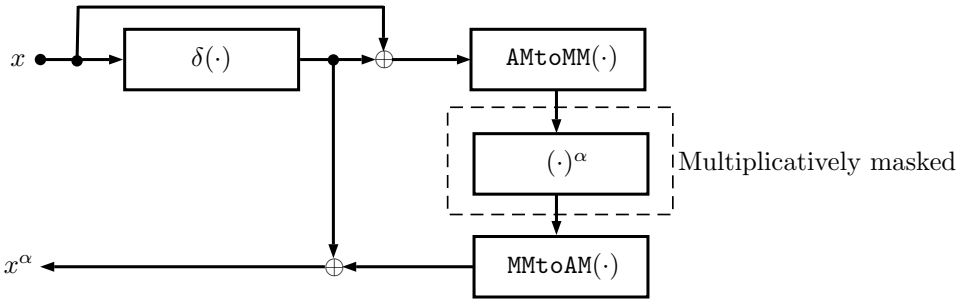


Figure 2.6: GPQ power function processing : $x \mapsto x^\alpha$.

The classical approach to securely process a power function $x \mapsto x^\alpha$ consists in expressing it in terms of squares and multiplications over $\mathbb{F}_{2^n}$, the latter being secured with the ISW multiplication gadget. This approach was first proposed by Rivain and Prouff in [56] for AES whose S-box can be represented as a single monomial over $\mathbb{F}_{2^n}$ (*i.e.* $x \mapsto x^{254}$). The study of masking power functions has been generalized by the work of Carlet, Goubin, Prouff, Quisquater and Rivain in [14]. They defined the notion of masking complexity for a n-bit S-box as the minimal number of nonlinear multiplications required to evaluate its polynomial representation over $\mathbb{F}_{2^n}$, and they computed the masking complexity of all power functions over $\mathbb{F}_{2^n}$ for $n \leq 11$. Their approach involves the notion of cyclotomic class and addition chain which are recalled hereafter.

Cyclotomic class. The cyclotomic class of α denoted by C_α, $\alpha \in [0; 2^n - 2]$ is defined by

$$C_\alpha = \{\alpha \cdot 2^i \bmod 2^n - 1 \ ; \ i \in [0; n-1]\}\,.$$

As the Frobenius map $x \mapsto x^2$ over $\mathbb{F}_{2^n}$ is linear, any $\alpha_i \in C_\alpha$ can be computed from any $\alpha_j \in C_\alpha$ with $\alpha_j \neq \alpha_i$ only using linear transformations. Hence, powers whose

exponents lie in the same cyclotomic class have the same masking complexity. The authors of [14] have related the problem of computing the masking complexity for an element α whose cyclotomic class is C_α to finding the shortest addition chain for α, $C_{\alpha_0} \to C_{\alpha_1} \to \ldots \to \mathsf{C}_{\alpha_k}$, such that $C_{\alpha_0} = C_1, C_{\alpha_k} = C_\alpha$, and for every $i \in [1; k]$, there exist $j, l \leq i$ such that $\alpha_i = \alpha_j + \alpha_l$ where $\alpha_j \in C_{\alpha_j}$ and $\alpha_l \in C_{\alpha_l}$. The resulting chain decomposes any power x^α in terms of linear operations (*i.e.* squares) and nonlinear multiplications between powers whose exponents belong to different cyclotomic classes. On the contrary to the classical approach, GPQ does not require ISW to secure the sequence that decomposes a power function. More precisely, multiplications which are nonlinear when an additive masking is involved may be performed by element-wise field multiplications between the shares of the multiplicatively masked values and hence ISW is no longer required. In fact, a power function $x \mapsto x^\alpha$ can even be tabulated with GPQ leading to great efficiency gains. Such an implementation choice costs 2^n bytes of memory to store the table, which is reasonable for a power function over $\mathbb{F}_{2^n}$ with $n < 10$.

In the following, when power functions cannot be tabulated, we use the procedure Eval-Chain$(\cdot)$ that takes as inputs a multiplicatively masked element x and an addition chain for α and that outputs the desired power x^α multiplicatively masked. Note that the cost of Eval-Chain$(\cdot)$ is negligible with GPQ. However, in order to minimize the complexity of an evaluation, it is always better to find the shortest possible addition chains.

Algorithm 14 Secure Power Function Evaluation

Require: An order d, an addition chain $\mathcal{A}$ for α, and a $(d+1)$-additive sharing of x
Ensure: A $(d+1)$-additive sharing $(y_0, \ldots, y_d)$ of x^α

*** Mapping from $\mathbb{F}_{2^n}$ to $\mathbb{F}_{2^n}^*$.*
1: $(\Delta_0, \ldots, \Delta_d) \leftarrow$ Secure-Dirac $(x_0, \ldots, x_d)$
2: $(x_0, \ldots, x_d) \leftarrow (x_0, \ldots, x_d) \oplus (\Delta_0, \ldots, \Delta_d)$
*** Convert into multiplicative masking*
3: $(z_0, \ldots, z_d) \leftarrow$ AMtoMM $(x_0, \ldots, x_d)$
*** Evaluate the chain*
4: $(z_0^\alpha, \ldots, z_d^\alpha) \leftarrow$ Eval-Chain $((z_0, \ldots, z_d), \mathcal{A})$
*** Convert back into additive masking*
5: $(y_0, \ldots, y_d) \leftarrow$ MMtoAM $(z_0^\alpha, \ldots, z_d^\alpha)$
*** Mapping from $\mathbb{F}_{2^n}^*$ to $\mathbb{F}_{2^n}$.*
6: $(y_0, \ldots, y_d) \leftarrow (y_0, \ldots, y_d) \oplus (\Delta_0, \ldots, \Delta_d)$
7: **return** $(y_0, \ldots, y_d)$

Complexity. Let us denote by $\mathsf{C}_\delta, \mathsf{C}_{\mathsf{AMtoMM}}$ and $\mathsf{C}_{\mathsf{MMtoAM}}$ respectively the costs of Alg. 11, 12 and 13 and by $\mathsf{C}_{\mathsf{GPQ}}$ the overall cost of a power function processing

with GPQ (Alg. 14). For each algorithm, we express their cost in terms of the costs of their elementary operations. To that end, let us also denote by $\mathsf{C}_{\mathsf{T}}, \mathsf{C}_{\oplus}, \mathsf{C}_{\odot}, \mathsf{C}_{\otimes}$ respectively the costs of $(n \times n)$-matrix transpositions, $\oplus$,$\odot$ and $\otimes$ operations. At last, $\mathsf{C}_{\oplus^n}$ and $\mathsf{C}_{\odot^n}$ denote the cost of n-bit operations $\oplus, \odot$. We have,

$$\mathsf{C}_{\delta} = \frac{(d+1)}{n} \times \mathsf{C}_{\mathsf{T}} + \frac{((2d\,(n-1)+n)(d+1)}{n} \times \mathsf{C}_{\oplus^n} + \frac{(n-1)(d+1)^2}{n} \times \mathsf{C}_{\odot^n}\,,$$

$$\mathsf{C}_{\mathsf{AMtoMM}} = d^2 \times \mathsf{C}_{\oplus} + \frac{d\,(3+d)}{2} \times \mathsf{C}_{\otimes}\,,$$

$$\mathsf{C}_{\mathsf{MMtoAM}} = d\,(2+d) \times \mathsf{C}_{\oplus} + \frac{d\,(3+d)}{2} \times \mathsf{C}_{\otimes}\,,$$

which gives,

$$\mathsf{C}_{\mathsf{GPQ}} = \mathsf{C}_{\delta} + \mathsf{C}_{\mathsf{AMtoMM}} + \mathsf{C}_{\mathsf{MMtoAM}}\,.$$

Security We now prove that GPQ is t-SNI. This is made accurate in the following theorem.

Theorem 2.2.4. GPQ *is* t-SNI. *Let* $(x_i)_{0\le i\le d}$ *be the input and let* $(y_i)_{0\le i\le d}$ *be the output of Alg. 14 (or equivalently of Fig. 2.7). For any adversary set of* t *probed wires* $\Omega = (\mathcal{I}, \mathcal{O})$*, with* $t \le d$*, there exists a set* $\mathcal{S}$ *of input shares such that* $|\mathcal{S}| \le |\mathcal{I}|$ *and* $\mathcal{S}$ *is sufficient to simulate the adversary observation set* Ω*.*

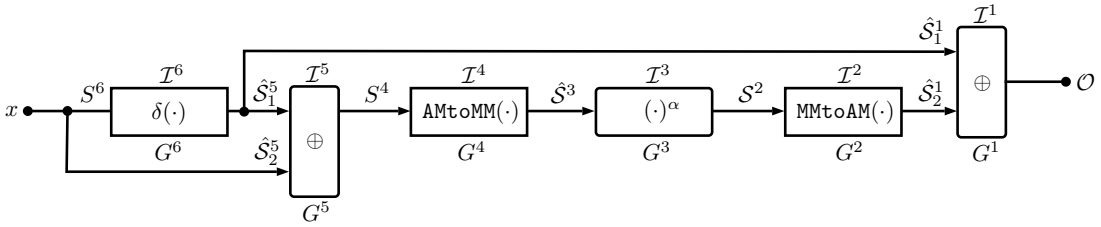


Figure 2.7: GPQ secure Gadget $(\cdot)^\alpha$.

Proof. As in [6], the proof is constructed by composition. Namely, we construct the simulator for the circuit of Fig. 2.7 by simulating sequentially each inner gadget from right to left.

Let $\Omega = (\mathcal{I}, \mathcal{O})$ be an observation set that has to be simulated, made on the whole circuit of Fig. 2.7 such that $\mathcal{I} = \bigcup_{1\le i\le 6} \mathcal{I}^i$ and such that the global constraint

$\sum_{i=1}^{6} |\mathcal{I}^i| + |\mathcal{O}| \leq t$ is satisfied.

Gadget 1 - Let $\Omega^1 = (\mathcal{I}^1, \mathcal{O})$ be an observation set made on Gadget 1. Since G^1 is affine-NI, we know that for every observation set Ω^1, there exists a set of input shares $\hat{\mathcal{S}}^1 = (\hat{\mathcal{S}}_1^1, \hat{\mathcal{S}}_2^1)$ such that $|\hat{\mathcal{S}}^1| \leq |\mathcal{I}^1 \cup \mathcal{O}|$ and the set $\hat{\mathcal{S}}^1$ is sufficient to simulate Ω^1.
Gadget 2 - Let $\Omega^2 = (\mathcal{I}^2, \hat{\mathcal{S}}_2^1)$ be an observation set made on Gadget 2. Since MMtoAM$(\cdot)$ is t-SNI and $|\mathcal{I}^2 \cup \hat{\mathcal{S}}_2^1| \leq |\mathcal{I}^2 \cup \mathcal{I}^1 \cup \mathcal{O}| \leq t$ (by simulation of Gadget 1 and the global constraint), we know that for every observation set Ω^2, there exists a set of input shares $\mathcal{S}^2$ such that $|\mathcal{S}^2| \leq |\mathcal{I}^2|$ and the set $\mathcal{S}^2$ is sufficient to simulate Ω^2.
Gadget 3 - Let $\Omega^3 = (\mathcal{I}^3, \mathcal{S}^2)$ be an observation set made on Gadget 3. Since G^3 is affine-NI, we know that for every observation set Ω^3, there exists an observation set $\hat{\mathcal{S}}^3$ such that $|\hat{\mathcal{S}}^3| \leq |\mathcal{I}^3 \cup \mathcal{S}^2| \leq |\mathcal{I}^3| + |\mathcal{I}^2|$ and the set $\hat{\mathcal{S}}^3$ is sufficient to simulate Ω^3.
Gadget 4 - Let $\Omega^4 = (\mathcal{I}^4, \hat{\mathcal{S}}^3)$ be an observation set made on Gadget 4. Since AMtoMM$(\cdot)$ is t-SNI and $|\mathcal{I}^4 \cup \hat{\mathcal{S}}^3| \leq t$ (by simulation of Gadget 3 and the global constraint) , we know that for every observation set Ω^4, there exists an observation set $\mathcal{S}^4$ such that $|\mathcal{S}^4| \leq |\mathcal{I}^4|$ the set $\mathcal{S}^4$ is sufficient to simulate Ω^4.
Gadget 5 - Let $\Omega^5 = (\mathcal{I}^5, \mathcal{S}^4)$ be an observation set made on Gadget 5. Since G^5 is affine-NI, we know that for every observation set Ω^5 there exists an observation set $\hat{\mathcal{S}}^5$ such that $|\hat{\mathcal{S}}^5| \leq |\mathcal{I}^5 \cup \mathcal{S}^4| \leq |\mathcal{I}^5| + |\mathcal{I}^4|$ and the set $\hat{\mathcal{S}}^5$ is sufficient to simulate Ω^5.
Gadget 6 - Let $\Omega^6 = (\mathcal{I}^6, (\hat{\mathcal{S}}^5 \cup \hat{\mathcal{S}}_1^1))$ be an observation set made on Gadget 6. Since $\delta(\cdot)$ is t-SNI and $|\mathcal{I}^6 \cup \hat{\mathcal{S}}^5 \cup \hat{\mathcal{S}}_1^1| \leq |\mathcal{I}^6 \cup \mathcal{I}^5 \cup \mathcal{I}^4 \cup \mathcal{I}^1 \cup \mathcal{O}| \leq t$ (by simulation of gadgets 5 and 1 and by the global constraint), we know that for every observation set Ω^6, there exists an observation set $\mathcal{S}^6$ such that $|\mathcal{S}^6| \leq |\mathcal{I}^6|$ and the set $\mathcal{S}^6$ is sufficient to simulate Ω^6.

To simulate the whole circuit, that is the observation set $\Omega = (\bigcup_{1 \leq i \leq 6} \mathcal{I}^i, \mathcal{O})$, the simulator requires $|\mathcal{S}^6 \cup \hat{\mathcal{S}}^5|$ shares of x. Since $|\mathcal{S}^6| \leq |\mathcal{I}^6|$, and $|\hat{\mathcal{S}}^5| \leq |\mathcal{I}^5| + |\mathcal{I}^4|$, we have that $|\mathcal{S}^6 \cup \hat{\mathcal{S}}^5| \leq \sum_{i=1}^{6} |\mathcal{I}^i| \leq t$ and therefore GPQ satisfies the t-SNI property. □

2.3 Polynomial GPQ : Alternate Cyclotomic Method

In this section, we describe how to extend the GPQ scheme to the masking of generic S-boxes. The main idea is outlined as follows. Since any n-bit S-box can be represented by a polynomial $S(x) = \sum a_i x^i$ over $\mathbb{F}_{2^n}$, a secure evaluation of $S(x)$ thus requires to securely process the corresponding sequence of linear

operations and power functions. As mentioned in the previous section, the common approach, referred to as the CGPQR method, would in turn decompose each power function in terms of squares and nonlinear multiplications over $\mathbb{F}_{2^n}$. Thereby, this approach involves ISW in order to secure these nonlinear multiplications. We propose to use GPQ to process the power functions in such manner that ISW is no longer required. However, a naive evaluation of the above writing of S that processes each power x^i with GPQ is not recommended in terms of efficiency. Indeed, such an evaluation would require the computation of a Dirac function along with conversions from an additive masking to a multiplicative masking and conversely for each monomial involved in the polynomial representation. Note that those transformations are costly to process (asymptotically they have the same complexity $\mathcal{O}(d^2)$ as ISW multiplications), thus we seek to minimize their number during a polynomial evaluation. A solution is provided by the cyclotomic method of [14] which we briefly present hereafter. Our approach is then detailed along with security proofs for the new proposed constructions.

2.3.1 Original Cyclotomic Method

Since the family of cyclotomic classes C_α is a partition of $[0, 2^n - 1]$, hence the polynomial representation of any S-box can be written

$$S(x) = a_0 + \left(\sum_{i=1}^{q} L_i(x^{\alpha_i}) \right) + a_{2^n-1} x^{2^n-1}, \tag{2.4}$$

where $L_i(x)$ denotes the linearized polynomial $\sum_j a_{i,j} x^{2^j}$ and q is the number of distinct cyclotomic classes of $[0; 2^n - 2]$. The cyclotomic method simply consists in deriving the powers x^{α_i} for each cyclotomic class as well as x^{2^n-1} if $a_{2^n-1} \neq 0$ and in evaluating $S(x)$. Following the CGPQR approach, it is required to find an addition chain for the x^{α_i}'s, $C_{\alpha_0} \to C_{\alpha_1} \to \ldots \to C_{\alpha_k}$ such that $C_{\alpha_0} = C_1$ and for every x^{α_i}, there exists $j \in [1, k]$ such that $C_{\alpha_i} = C_{\alpha_j}$. The addition chain decomposes the x^{α_i}'s as a sequence of squares and nonlinear multiplications over $\mathbb{F}_{2^n}$. The rest of the powers can be derived with Frobenius maps. Using CGPQR, ISW is involved to derive at least one power of each distinct cyclotomic classes of $[0; 2^n - 2]$. Therefore, the shorter the chain is, the better.

Complexity. Let us denote by $\mathsf{C}_{\mathsf{Cyclo}}$ the cost of evaluating polynomials with the CGPQR scheme and the cyclotomic method. Let us also denote by C_{SecMult} the cost of a finite field multiplication which is secured with ISW and let q be the number of distinct cyclotomic classes of $[0; 2^n - 2]$. Then the cost of masking generic S-boxes is

$$\mathsf{C}_{\mathsf{Cyclo}} = (q-1) \times \mathsf{C}_{\mathsf{SecMult}},$$

or $\mathsf{C}_{\mathsf{Cyclo}} = (q-2) \times \mathsf{C}_{\mathsf{SecMult}}$ if the S-box which is considered is balanced (see [14]). We now propose a different writing of (2.4) adapted for an evaluation with GPQ.

2.3.2 Our Alternate Proposal

We have that $x^{\alpha_i} = (x + \delta(x))^{\alpha_i} + \delta(x)$ which gives $L_i(x^{\alpha_i}) = L_i((x + \delta(x))^{\alpha_i}) + L_i(\delta(x))$ by linearity of L_i. Thus, (2.4) can be written as

$$S(x) = a_0 + L_1(x^{\alpha_1}) + \sum_{i=2}^{q} (L_i((x + \delta(x))^{\alpha_i}) + L_i(\delta(x))) + a_{2^n-1} x^{2^n-1},$$

where $L_i(\delta(x)) = \sum_j a_{i,j} \delta(x)^{2^j} = \sum_j a_{i,j}(1)^{2^j} \delta(x) = L_i(1) \cdot \delta(x)$, which gives

$$S(x) = a_0 + L_1(x^{\alpha_1}) + \sum_{i=2}^{q} (L_i((x + \delta(x))^{\alpha_i}) + L_i(1) \cdot \delta(x))) + a_{2^n-1} x^{2^n-1}.$$

According to the field equation, $x^{2^n-1} = 0$ if $x = 0$ and $x^{2^n-1} = 1$ otherwise. It follows that since $\delta(x) = 1$ if $x = 1$ and $\delta(x) = 0$ otherwise, we have $x^{2^n-1} = \delta(x) + 1$. Finally,

$$S(x) = a_0 + a_{2^n-1} + L_1(x^{\alpha_1}) + \sum_{i=2}^{q} L_i((x+\delta(x))^{\alpha_i}) + \left(\sum_{i=2}^{q} L_i(1) + a_{2^n-1} \right) \cdot \delta(x). \quad (2.5)$$

The above writing of $S(x)$ yields to a novel version of the cyclotomic method which shall be referred to as the alternate cyclotomic method in the following and which also extends GPQ to the evaluation of polynomials over $\mathbb{F}_{2^n}$.

We outline the steps of such an evaluation in Alg. 15. Similarly to the processing of a single power function (see Section 2.2.3), the procedure Eval-Chain$(\cdot)$ (Step 4 of Alg. 15) takes as inputs an element $(x + \delta(x)) \in \mathbb{F}_{2^n}^*$ along with an addition chain for all the x^{α_i}'s, evaluates the latter without ISW and outputs the desired powers $(x + \delta(x))^{\alpha_i}$ still multiplicatively masked. Note that the sequence of operations provided by the chain may lead to computing powers which are not one of the $(x + \delta(x))^{\alpha_i}$'s. However, only the $(x + \delta(x))^{\alpha_i}$'s are converted back into additive maskings at the end of the evaluation. Moreover, since Frobenius maps are less costly than conversions, the linearized polynomial $L_1(x)$ of (2.5), whose monomials are only powers of two, is always computed in additive masking.

Algorithm 15 Alternate Cyclotomic

Require: An order d, an addition chain $\mathcal{A}$, and a $(d+1)$-additive sharing of x
Ensure: A $(d+1)$-additive sharing $(S_0, \ldots, S_d)$ of $S(x)$

NOTE : *The (d+1)-additive sharing $(x_0, \ldots, x_d)$ of x is stored in memory*

*** Mapping from $\mathbb{F}_{2^n}$ to $\mathbb{F}_{2^n}^*$.*

$(\Delta_0, \ldots, \Delta_d) \leftarrow \mathsf{Secure\text{-}Dirac}(x_0, \ldots, x_d)$
$(x_0, \ldots, x_d) \leftarrow (\Delta_0, \ldots, \Delta_d) \oplus (x_0, \ldots, x_d)$

*** Convert into multiplicative masking and evaluate the addition chain.*

$(z_0, \ldots, z_d) \leftarrow \mathsf{AMtoMM}(x_0, \ldots, x_d)$
$(z^{\alpha_2}, \ldots, z^{\alpha_k}) \leftarrow \mathsf{Eval\text{-}Chain}((\mathsf{z_0}, \ldots, \mathsf{z_d}), \mathcal{A})$

*** Compute the linearized polynomials.*

$(L_0, \ldots, L_d) \leftarrow \mathsf{Linearize\text{-}Poly}(x_0, \ldots, x_d)$
for $i = 2$ **to** q **do**
    $(x_0^{(i)}, \ldots, x_d^{(i)}) \leftarrow \mathsf{MMtoAM}(z_0^{\alpha_i}, \ldots, z_d^{\alpha_i})$
    $(l_0^{(i)}, \ldots, l_d^{(i)}) \leftarrow \mathsf{Linearize\text{-}Poly}(x_0^{(i)}, \ldots, x_d^{(i)})$
    $(L_0, \ldots, L_d) \leftarrow (L_0, \ldots, L_d) \oplus (l_0^{(i)}, \ldots, l_d^{(i)})$
end for

*** Mapping from $\mathbb{F}_{2^n}^*$ to $\mathbb{F}_{2^n}$.*

$a \leftarrow a_0$
for $i = 1$ **to** $2^n - 1$ **do**
    $a \leftarrow a \oplus (a_i \cdot (\Delta_0, \ldots, \Delta_d))$
    $(S_0, \ldots, S_d) \leftarrow a_0 \oplus a_{2^n-1} \oplus (L_0, \ldots, L_d)$
end for
return $(S_0, \ldots, S_d)$

Complexity. Let us denote by $\mathsf{C}_{\mathsf{Alt-cyclo}}$ the cost of evaluating polynomials using the alternate cyclotomic method (Alg. 15). We do not take into account the costs of $\mathsf{Eval\text{-}Chain}(\cdot)$ and $\mathsf{Linearize\text{-}Poly}(\cdot)$ procedures as they can be computed with linear transformations. The cost of our alternate cyclotomic method for the evaluation of polynomials is therefore

$$\mathsf{C}_{\mathsf{Alt-cyclo}} = \mathsf{C}_{\delta} + \mathsf{C}_{\mathsf{AMtoMM}} + (q-2) \times \mathsf{C}_{\mathsf{MMtoAM}}\,,$$

where q is the number of distinct cylotomic classes of $[0; 2^n - 2]$.

For the sake of clarity, Table 2.1 lists the complexities of our proposal and the original method in terms of elementary operations as a function of the order d. Also, as operations $\oplus, \oplus^n, \odot, \odot^n$ have the same complexity in practice (see Section

2.5), we list them together. Operation $\mathsf{M}^\intercal$ denotes $(n \times n)$-matrix transpositions.

In order to proceed to a fair comparison it should be noted that field multiplications with our proposal do not have the same weight as the ones that are implemented following the original method. Our approach allows to implement field multiplications more efficiently (see Section 2.5).

Table 2.1: Complexities of our proposal and the original method in terms of elementary operations.

Operations	Our proposal	[14]
$\otimes$	$(q-1)\,d^2 + (3q-3)\,d$	$(q-2)\,d^2 + (2q-4)\,d + (q-2)$
$\oplus, \oplus^n, \odot, \odot^n$	$\left(q+2-\frac{3}{n}\right)d^2 + \left(2q+1-\frac{4}{n}\right)d + \left(2-\frac{1}{n}\right)$	$(2q-4)\,d^2 + (2q-4)\,d$
$\mathsf{M}^\intercal$	$d/n + 1/n$	–

From Table 2.1 it is obvious that the complexities of both our proposal and the original method mainly depend on the number of cyclotomic classes.

The end of the section is dedicated to prove the security of the resulting method under the t-SNI security definition.

Security. In order to analyze the security of our alternate cyclotomic method and for the sake of clarity, we divide the processing of the corresponding Gadget $\mathsf{Alt\text{-}Cy}(\cdot)$ into two parts as illustrated Fig. 2.8. The security of Gadgets $\mathsf{Alt\text{-}Cy}^1(\cdot)$ and $\mathsf{Alt\text{-}Cy}^2(\cdot)$ is analyzed separately. Then the security of $\mathsf{Alt\text{-}Cy}(\cdot)$ is induced by the secure composition of Gadgets $\mathsf{Alt\text{-}Cy}^1(\cdot)$ and $\mathsf{Alt\text{-}Cy}^2(\cdot)$. Note that Gadget R is a refreshing Gadget (Alg. 10).

Gadget $\mathsf{Alt\text{-}Cy}^1(\cdot)$ only involves affine gadgets. Indeed, Gadget $L_i(\cdot)$ corresponds to the procedure $\mathsf{Linearize\text{-}Poly}(\cdot)$ (step 9) of Alg. 15 which only involves linear operations (*i.e.* squares and additions) and Gadget $\mathcal{C}(\cdot)$ corresponds to step 8 which involves scalar multiplications and additions. Since the composition of affine gadgets is affine, hence $\mathsf{Alt\text{-}Cy}^1(\cdot)$ is $\mathsf{affine\text{-}NI}$. Regarding Gadget $\mathsf{Alt\text{-}Cy}^2(\cdot)$ we prove the following Lemma.

Lemma 2.3.1. $\mathsf{Alt\text{-}Cy}^2(\cdot)$ *is* t-SNI. *Let* $(x_i)_{0\leq i\leq d}$ *be the input and let* $(x'_i)_{0\leq i\leq d}, ((x+\delta(x))_i^{\alpha_j})_{0\leq i\leq d}$ *with* $j \in [2;q]$ *and* $(\delta(x)_i)_{0\leq i\leq d}$ *be the outputs of Gadget* $\mathsf{Alt\text{-}Cy}^2(\cdot)$. *For any adversary set of* t *probed wires* $\Omega = (\mathcal{I}, \mathcal{O})$, *with* $t \leq d$, *there exists a set*

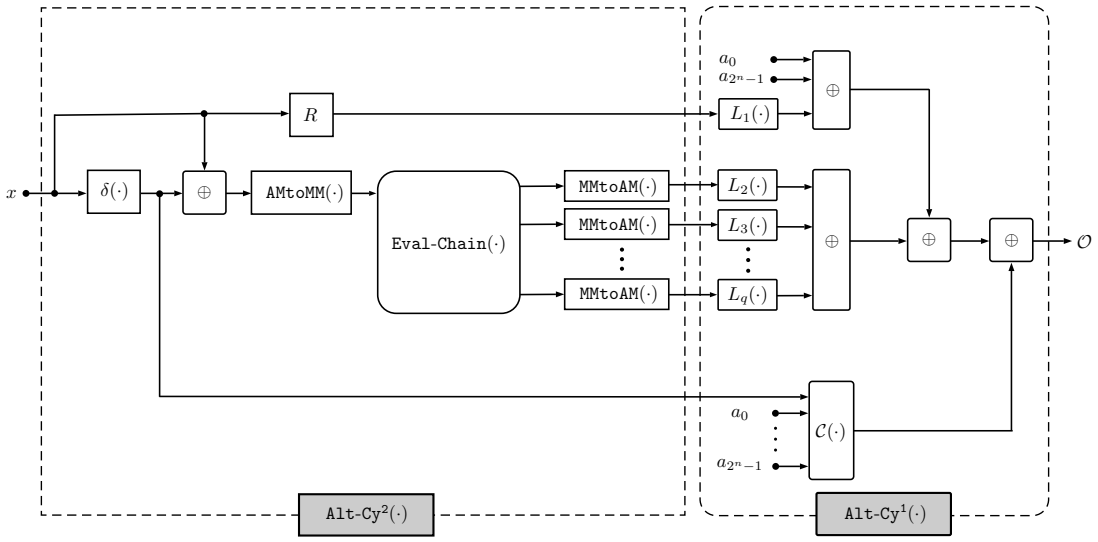


Figure 2.8: Gadget Alt-Cy$(\cdot)$: circuit of the alternate cyclotomic processing.

$\mathcal{S}$ of input shares such that $|\mathcal{S}| \leq |\mathcal{I}|$ and $\mathcal{S}$ is sufficient to simulate the adversary observation set Ω.

Proof. As previously, we build the simulator for the circuit Fig. 2.9 by simulating sequentially each inner gadget from right to left.

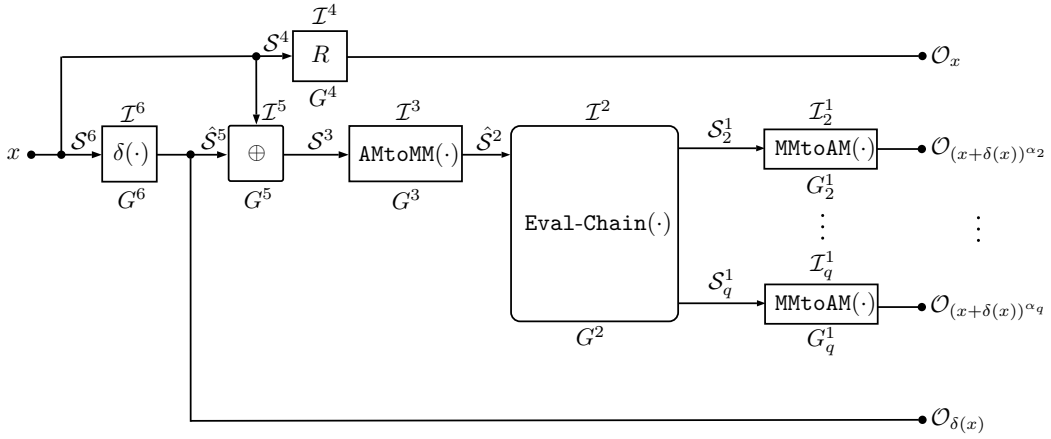


Figure 2.9: Gadget Alt-Cy$^2(\cdot)$.

Let $\Omega = (\mathcal{I}, \mathcal{O})$ be the adversary observation set that we want to simulate and which is made on the whole circuit of Fig. 2.9, with $\mathcal{O} = (\mathcal{O}_x, \mathcal{O}_{(x+\delta(x))^{\alpha_2}}, \ldots, \mathcal{O}_{(x+\delta(x))^{\alpha_q}}, \mathcal{O}_{\delta(x)})$

such that $\mathcal{I} = \left(\bigcup_{j=2}^{q} \mathcal{I}_j^1\right) \cup \left(\bigcup_{i=2}^{6} \mathcal{I}^i\right)$ and such that $\left(\sum_{j=2}^{q} |\mathcal{I}_j^1| + \sum_{i=2}^{6} |\mathcal{I}^i|\right) + |\mathcal{O}| \leq t$.

Gadgets 1. For every $j \in [2; q]$, let $\Omega_j^1 = (\mathcal{I}_j^1, \mathcal{O}_{(x+\delta(x))^{\alpha_j}})$ be an observation set made on G_j^1. Since G_j^1 is t-SNI and $|\mathcal{I}_j^1 \cup \mathcal{O}_{(x+\delta(x))^{\alpha_j}}| \leq t$ (by global constraint), we know that for every observation set Ω_j^1 there exists an observation set $\mathcal{S}_j^1$ for G_j^1 such that $|\mathcal{S}_j^1| \leq |\mathcal{I}_j^1|$ and $\mathcal{S}_j^1$ is sufficient to simulate Ω_j^1 for every $j \in [2, q]$.
Gadget 2. Let $\Omega^2 = (\mathcal{I}^2, (S_2^1, \ldots, S_q^1))$. Since G^2 is affine-NI, we know that for every observation set Ω^2 there exists an observation set $\hat{\mathcal{S}}^2$ such that $|\hat{\mathcal{S}}^2| \leq |\mathcal{I}^2 \cup \left(\bigcup_{2 \leq j \leq q} S_j^1\right)| \leq |\mathcal{I}^2| + \sum_{j=2}^{q} |\mathcal{I}_j^1|$ and the set of input shares $\hat{\mathcal{S}}^2$ is sufficient to simulate the adversary observation set Ω^2 made on Gadget 2.
Gadget 3. Let $\Omega^3 = (\mathcal{I}^3, \hat{\mathcal{S}}^2)$. Since G^3 is t-SNI and $|\mathcal{I}^3 \cup \hat{\mathcal{S}}^2| \leq t$ (by simulation of Gadget 2 and the global constraint), we know that for every observation set Ω^3 there exists an observation set $\mathcal{S}^3$ such that $|\mathcal{S}^3| \leq |\mathcal{I}^3|$ and $\mathcal{S}^3$ is sufficient to simulate Ω^3.
Gadget 4. Let $\Omega^4 = (\mathcal{I}^4, \mathcal{O}_x)$. Since G^4 is t-SNI and $|\mathcal{I}^4 \cup \mathcal{O}_x| \leq t$ (by the global constraint), we know that for every observation set Ω^4 there exists an observation set $\mathcal{S}^4$ such that $|\mathcal{S}^4| \leq |\mathcal{I}^4|$ and $\mathcal{S}^4$ is sufficient to simulate Ω^4.
Gadget 5. Let $\Omega^5 = (\mathcal{I}^5, \mathcal{S}^3)$. Since G^5 is affine-NI, we know that for every observation set Ω^5 there exists an observation set $\hat{\mathcal{S}}^5$ such that $|\hat{\mathcal{S}}^5| \leq |\mathcal{I}^5 \cup \mathcal{S}^3| \leq |\mathcal{I}^5| + |\mathcal{I}^3|$ and the set of input shares $\hat{\mathcal{S}}^5$ is sufficient to simulate the adversary observation set Ω^5 made on Gadget 5.
Gadget 6. Let $\Omega^6 = (\mathcal{I}^6, \hat{\mathcal{S}}^5 \cup \mathcal{O}_{\delta(x)})$. Since G^6 is t-SNI and $|\mathcal{I}^6 \cup \hat{\mathcal{S}}^5 \cup \mathcal{O}_{\delta(x)}| \leq |\mathcal{I}^6| + |\mathcal{I}^5| + |\mathcal{I}^3| + |\mathcal{O}_{\delta(x)}| \leq t$ (by simulation of Gadget 5 and the global constraint), we know that for every observation set Ω^6 there exists an observation set $\mathcal{S}^6$ such that $|\mathcal{S}^6| \leq |\mathcal{I}^6|$ and $\mathcal{S}^6$ is sufficient to simulate Ω^6.

In order to simulate Gadget Alt-$Cy^2(\cdot)$, the corresponding simulator requires the shares $S^6 \cup \hat{\mathcal{S}}^5 \cup \mathcal{S}^4$ and $|S^6 \cup \hat{\mathcal{S}}^5 \cup \mathcal{S}^4| \leq |\mathcal{I}^6| + |\mathcal{I}^5| + |\mathcal{I}^4| + |\mathcal{I}^3| \leq \sum_{j=2}^{q} |\mathcal{I}_j^1| + \sum_{i=2}^{6} |\mathcal{I}^i| \leq t$. Therefore, Gadget Alt-Cy$^2(\cdot)$ is t-SNI. □

Theorem 2.3.2. *Alternate cyclotomic is t-SNI. Let $(x_i)_{0 \leq i \leq d}$ be the input and let $(S_i)_{0 \leq i \leq d}$ be the output of Alg. 15 or equivalently of Gadget* Alt-Cy$(\cdot)$ *(see Fig. 2.8). For any adversary set of t probed wires $\Omega = (\mathcal{I}, \mathcal{O})$, with $t \leq d$, there exists a set $\mathcal{S}$ of input shares such that $|\mathcal{S}| \leq |\mathcal{I}|$ and $\mathcal{S}$ is sufficient to simulate the adversary observation set Ω.*

We illustrate Fig. 2.10 the circuit corresponding to our alternate cyclotomic method. We already analyzed Gadgets Alt-Cy$^1(\cdot)$ and Alt-Cy$^2(\cdot)$ and now we prove the security of the full construction by composition.

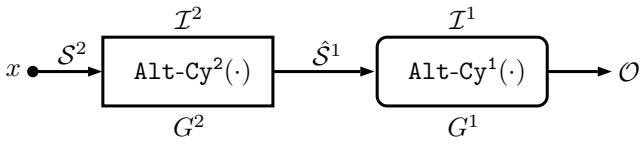


Figure 2.10: Gadget Alt-Cy$(\cdot)$.

Proof. Let $\Omega = ((\mathcal{I}^1 \cup \mathcal{I}^2), \mathcal{O})$ be an observation set to simulate for the circuit represented Fig. 2.10, such that the global constraint $|\mathcal{I}^1| + |\mathcal{I}^2| + |\mathcal{O}| \leq t$ is satisfied.

Gadget 1. Let $\Omega^1 = (\mathcal{I}^1, \mathcal{O})$. Since G^1 is affine-NI, we know that there exists an observation set $\hat{\mathcal{S}}^1$ such that $|\hat{\mathcal{S}}^1| \leq |\mathcal{I}^1| + |\mathcal{O}|$ and the set of input shares $\hat{\mathcal{S}}^1$ is sufficient to simulate the adversary observation set Ω^1 made on Gadget 1.
Gadget 2. Let $\Omega^2 = (\mathcal{I}^2, \hat{\mathcal{S}}^1)$. Since G^2 is t-SNI (by Lemma 2) and $|\mathcal{I}^2 \cup \hat{\mathcal{S}}^1| \leq t$ (by simulation of Gadget 1 and the global constraint), we know that there exists an observation set $\mathcal{S}^2$ such that $|\mathcal{S}^2| \leq |\mathcal{I}^2|$ and $\mathcal{S}^2$ is sufficient to simulate Ω^2.

In order to simulate Gadget Alt-Cy$(\cdot)$, the corresponding simulator requires the shares $\mathcal{S}^2$ of its input x and $|\mathcal{S}^2| \leq |\mathcal{I}^2| \leq \sum_{i=1}^{2} |\mathcal{I}^i| \leq t$. Therefore, Gadget Alt-Cy$(\cdot)$ is t-SNI. □

Gadgets Alt-Cy$^1(\cdot)$ and Alt-Cy$^2(\cdot)$ involved in our alternate cyclotomic method are also used in the next section in which we describe how to combine them to propose an alternate CRV method.

2.4 Alternate CRV Method

In this section we describe an alternate approach for the CRV method proposed by Coron, Roy and Vivek in [21] which is currently the best known method for polynomial evaluation over $\mathbb{F}_{2^n}$. The idea is to plug our polynomial evaluation method with GPQ (*i.e* our alternate cyclotomic method) into the CRV construction. First, we recall the original method, then we describe our alternate approach and we also show how it enables to derive new parameters more adapted to our case. Finally, we prove that the resulting construction is t-SNI.

2.4.1 Original CRV Method

The CRV method first consists in choosing a collection S of l cyclotomic classes among which C_0 and C_1 are always counted. Then it defines the union set L of

all integers in those cyclotomic classes. The original approach states that the set S has to be carefully chosen so that the monomials x^L can be computed with only $l-2$ nonlinear multiplications. It is moreover required that every monomial of $[0, 2^n-1]$ can be written as a product of some two monomials generated from L.

Denoting by $\mathcal{P}(x^L)$ the set of all polynomials in $\mathbb{F}_{2^n}$ whose monomials belong to the set x^L, CRV generates randomly $k-1$ polynomials $q_i(x) \in \mathcal{P}$ and tries to find k polynomials $p_i(x) \in \mathcal{P}$ such that

$$S(x) = \sum_{i=1}^{k-1} p_i(x) \cdot q_i(x) + p_k(x)\,. \tag{2.6}$$

From (2.6), CRV tries to solve a system of 2^n linear equations with $k \cdot |L|$ unknowns which are the coefficients of the p_i's. Such a system admits a solution for every choice of S if it has rank 2^n. To be of full rank, the necessary condition $k \cdot |L| \geq 2^n$ has to be satisfied.

Complexity. Let us denote by $\mathsf{C}_{\mathsf{CRV}}$ the overall cost of CRV. As mentioned in the above description, the set of monomials x^L requires $l-2$ nonlinear multiplications to be built, and $k-1$ additional nonlinear multiplications are necessary to compute (2.6). Following the CGPQR method, those nonlinear multiplications are secured with ISW, which cost is denoted by $\mathsf{C}_{\mathsf{SecMult}}$. Thus,

$$\mathsf{C}_{\mathsf{CRV}} = (l + k - 3) \times \mathsf{C}_{\mathsf{SecMult}}\,.$$

2.4.2 Our Alternate Proposal

The original approach imposes a constraint on the choice of cyclotomic classes that form the set S. Underlying this constraint is in fact the cyclotomic method. The latter enables to evaluate polynomials composed of l cyclotomic classes with $l-2$ nonlinear multiplications, as long as each nonlinear multiplication allows to reach a different cyclotomic class. Also, monomials that belong to C_0 or C_1 do not require nonlinear multiplications to be derived (see Section 4.1 of [14]).

On the other hand, our alternate cyclotomic approach does not imply to secure these $l-2$ nonlinear multiplications with ISW and thus makes the previous constraint obsolete. It evaluates polynomials with GPQ instead. Therefore, we propose to plug our alternate cyclotomic approach into the CRV construction only to build the precomputed set x^L. We emphasize that computing (2.6) still requires $k-1$ ISW multiplications.

New parameters. Our approach allows more freedom degree on the choice of cyclotomic classes to build x^L. Also, we can consider larger sets S. As an example, let us consider the secure evaluation of 8-bit S-boxes. It has been shown in [21] that choosing $l = 7$ and the set of cyclotomic classes $L = C_0 \cup C_1 \cup C_3 \cup C_7 \cup C_{29} \cup C_{87} \cup C_{251}$ gives a full rank system for some random choice of the polynomials $q_i(x)$. The precomputed set from which the monomials of the q_i's are picked up can thus be built with 5 nonlinear multiplications. Moreover, in order to satisfy the necessary condition $k \cdot |L| \geq 2^n$, such a choice for L ($|L| = 49$) implies that $k = 6$.

In our approach, we increase the size of S only to decrease the parameter k. To that end, we chose $l = 10$ and $L = C_0 \cup C_1 \cup C_{15} \cup C_{31} \cup C_{39} \cup C_{43} \cup C_{53} \cup C_{61} \cup C_{111} \cup C_{119}$ ($|L| = 69$) which implies that $k = 4$ and we have checked that the corresponding system is of full rank. Such settings would require a total of 11 nonlinear multiplications following the original approach. However, they are better suited for our alternate cyclotomic approach than those proposed in [21]. We also determined new sets of parameters for the cases $n \in \{5, 7\}$, which are given along with implementation results Section 2.5.

Complexity. Let us recall that C_{SecMult} denotes the cost of a finite field multiplication which is secured with ISW. The cost to build the precomputed set of monomials is denoted by C_{Set} and we denote by $C_{\mathsf{Alt-CRV}}$ the overall cost of our alternate CRV proposal. Note that C_{Set} represents the cost of our alternate cyclotomic proposal for polynomials that can be generated from l distinct cyclotomic classes. Thus,

$$\mathsf{C}_{\mathsf{Set}} = \mathsf{C}_{\delta} + \mathsf{C}_{\mathsf{AMtoMM}} + (l-2) \times \mathsf{C}_{\mathsf{MMtoAM}} \,,$$

and

$$\mathsf{C}_{\mathsf{Alt-CRV}} = \mathsf{C}_{\mathsf{Set}} + (k-1) \times \mathsf{C}_{\mathsf{SecMult}} \,.$$

For the sake of clarity, Table 2.2 below lists the complexities of our proposal and the original method in terms of elementary operations as a function of the order d. Also, as operations $\oplus, \oplus^n, \odot, \odot^n$ have the same complexity in practice (see Section 2.5), they are listed together. Operation $\mathsf{M}^{\intercal}$ denotes $(n \times n)$-matrix transpositions. As previously mentioned, in order to proceed to a fair comparison it should be noted that field multiplications with our proposal do not have the same weight as the ones that are implemented following the original method. Our approach allows to implement field multiplications more efficiently (see Section 2.5).

Security. We now describe the resulting alternate CRV construction that incorporates our alternate cyclotomic approach to build the precomputed set of monomials. Gadgets $\mathsf{Alt\text{-}Cy}^1(\cdot)$ and $\mathsf{Alt\text{-}Cy}^2(\cdot)$ of our alternate cyclotomic approach

Table 2.2: Complexities of our proposal and the original method in terms of elementary operations.

Operation	Our proposal
$\otimes$	$(l+k-2)\,d^2+(3l+2k-5)\,d+(k-1)$
$\oplus, \oplus^n, \odot, \odot^n$	$(l+2k-3/n)\,d^2+(2l+2k-4/n-1)\,d+(2-1/n)$
$\mathsf{M}^\intercal$	$d/n+1/n$

	[21]
$\otimes$	$(l+k-1)\,d^2+(2l+2k-2)\,d+(l+k-1)$
$\oplus$	$(2l+2k-2)\,d^2+(2l+2k-2)\,d+(l+k-1)$

which have been analyzed in the previous section are thus involved in the full construction as illustrated Fig. 2.11. Note that $\mathsf{Alt\text{-}Cy}^2(\cdot)$ enables to derive powers of the precomputed set and each gadget $\mathsf{Alt\text{-}Cy}^1(\cdot)$ enables to generate distinct linearized polynomials by affecting different coefficients to powers belonging to the precomputed set. Therefore, each composition of a Gadget $\mathsf{Alt\text{-}Cy}^2(\cdot)$ with a Gadget $\mathsf{Alt\text{-}Cy}^1(\cdot)$ generates a new polynomial. Note also that Gadgets $\mathsf{CRV}^1_{\mathsf{i}}(\cdot)$ correspond to the products of the $p_i(x)$'s with the $q_i(x)$'s of (2.6) for which we prove the following Lemma.

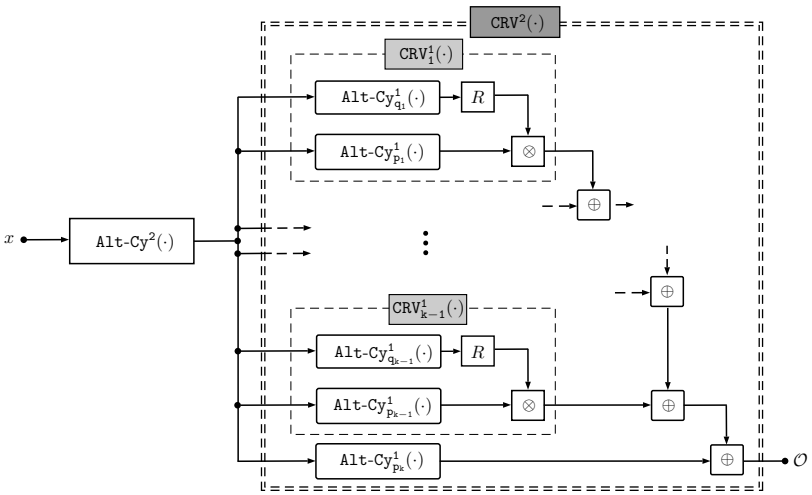


Figure 2.11: Gadget $\mathsf{Alt\text{-}CRV}(\cdot)$: circuit of the alternate CRV method.

Lemma 2.4.1. *Gadget* $\mathsf{CRV}_{\mathsf{i}}^{\mathbf{1}}(\cdot)$ *is* t-SNI. *Let* $(x_i)_{0\leq i\leq d}$, $(\delta(x)_i)_{0\leq i\leq d}$ *and also* $\left((x+\delta(x))_i^{\alpha_j}\right)_{0\leq i\leq d}$ *with* $j \in [2;q]$ *be the inputs and let* $(y_i)_{0\leq i\leq d}$ *be the output of Gadget* $\mathsf{CRV}_{\mathsf{i}}^{\mathbf{1}}(\cdot)$ *represented Fig. 2.11. For any adversary set of* t *probed wires* $\Omega = (\mathcal{I}, \mathcal{O})$, *with* $t \leq d$, *there exists a set* $\mathcal{S}$ *of input shares such that* $|\mathcal{S}| \leq |\mathcal{I}|$ *from which* Ω *can be perfectly simulated.*

Proof. Consider the following figure of Gadget $\mathsf{CRV}_{\mathsf{i}}^{\mathbf{1}}(\cdot)$.

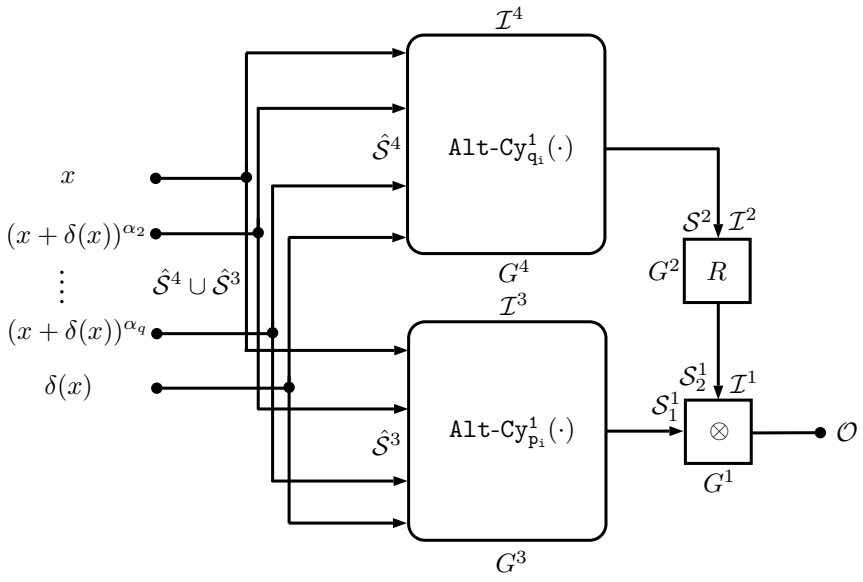


Figure 2.12: Gadget $\mathsf{CRV}_{\mathsf{i}}^{\mathbf{1}}(\cdot)$.

Let $\Omega = (\mathcal{I}, \mathcal{O})$ be an observation set that we want to simulate and which is made on the whole circuit of Fig. 2.12, such that $\mathcal{I} = \bigcup_{1\leq i\leq 4} \mathcal{I}^i$ and such that the global constraint $\sum_{i=1}^{4} |\mathcal{I}^i| + |\mathcal{O}| \leq t$ is satisfied.

Gadget 1. Let $\Omega^1 = (\mathcal{I}^1, \mathcal{O})$. Since G^1 is t-SNI and $|\mathcal{I}^1 \cup \mathcal{O}| \leq t$ (by global constraint), we know that there exists an observation set $\mathcal{S}^1 = (\mathcal{S}_1^1, \mathcal{S}_2^1)$ such that $|\mathcal{S}_1^1| \leq |\mathcal{I}^1|$, $|\mathcal{S}_2^1| \leq |\mathcal{I}^1|$ and $\mathcal{S}^1$ is sufficient to simulate Ω^1.
Gadget 2. Let $\Omega^2 = (\mathcal{I}^2, \mathcal{S}_2^1)$. Since G^2 is t-SNI and $|\mathcal{I}^2 \cup \mathcal{S}_2^1| \leq t$ (by simulation of Gadget 1 and the global constraint), we know that there exists an observation set $\mathcal{S}^2$ such that $|\mathcal{S}^2| \leq |\mathcal{I}^2|$ and $\mathcal{S}^2$ is sufficient to simulate Ω^2.
Gadget 3. Let $\Omega^3 = (\mathcal{I}^3, \mathcal{S}_1^1)$. Since G^3 is affine-NI, we know that there exists an observation set $\hat{\mathcal{S}}^3$ such that $|\hat{\mathcal{S}}^3| \leq |\mathcal{I}^3 \cup \mathcal{S}_1^1| \leq |\mathcal{I}^3| + |\mathcal{I}^1| \leq t$ and the set of input shares $\hat{\mathcal{S}}^3$ is sufficient to simulate the adversary observation set Ω^3 made on Gadget 3.

Gadget 4. Let $\Omega^4 = (\mathcal{I}^4, \mathcal{S}^2)$. Since G^4 is affine-NI, we know that there exists an observation set $\hat{\mathcal{S}}^4$ such that $|\hat{\mathcal{S}}^4| \leq |\mathcal{I}^4 \cup \mathcal{S}^2| \leq |\mathcal{I}^4| + |\mathcal{I}^2|$ and the set of input shares $\hat{\mathcal{S}}^4$ is sufficient to simulate the adversary observation set Ω^4 made on Gadget 4.

In order to simulate Gadget $\mathsf{CRV}_{\mathsf{i}}^{1}(\cdot)$, the corresponding simulator requires the shares $\hat{\mathcal{S}}^4 \cup \hat{\mathcal{S}}^3$ of each of its inputs and $|\hat{\mathcal{S}}^4 \cup \hat{\mathcal{S}}^3| \leq |\mathcal{I}^4| + |\mathcal{I}^3| + |\mathcal{I}^2| + |\mathcal{I}^1| \leq \sum_{i=1}^{4} |\mathcal{I}^i| \leq t$. Therefore, Gadget $\mathsf{CRV}_{\mathsf{i}}^{1}(\cdot)$ is t-SNI. □

Regarding Gadget $\mathsf{CRV}^{2}(\cdot)$ we prove the following Lemma.

Lemma 2.4.2. $\mathsf{CRV}^{2}(\cdot)$ *is* t-NI. *Let* $(x_i)_{0\leq i\leq d}$, $(\delta(x)_i)_{0\leq i\leq d}$ *and* $\left((x+\delta(x))_i^{\alpha_j}\right)_{0\leq i\leq d}, j \in [2, q]$ *be the inputs and let* $(y_i)_{0\leq i\leq d}$ *be the output of Gadget* $\mathsf{CRV}^{2}(\cdot)$ *represented Fig. 2.4.2. For any adversary set of* t *probed wires* $\Omega = (\mathcal{I}, \mathcal{O})$, *with* $t \leq d$, *there exists a set* $\mathcal{S}$ *of input shares such that* $|\mathcal{S}| \leq t$ *from which* Ω *can be perfectly simulated.*

Proof. Consider the following figure of Gadget $\mathsf{CRV}^{2}(\cdot)$

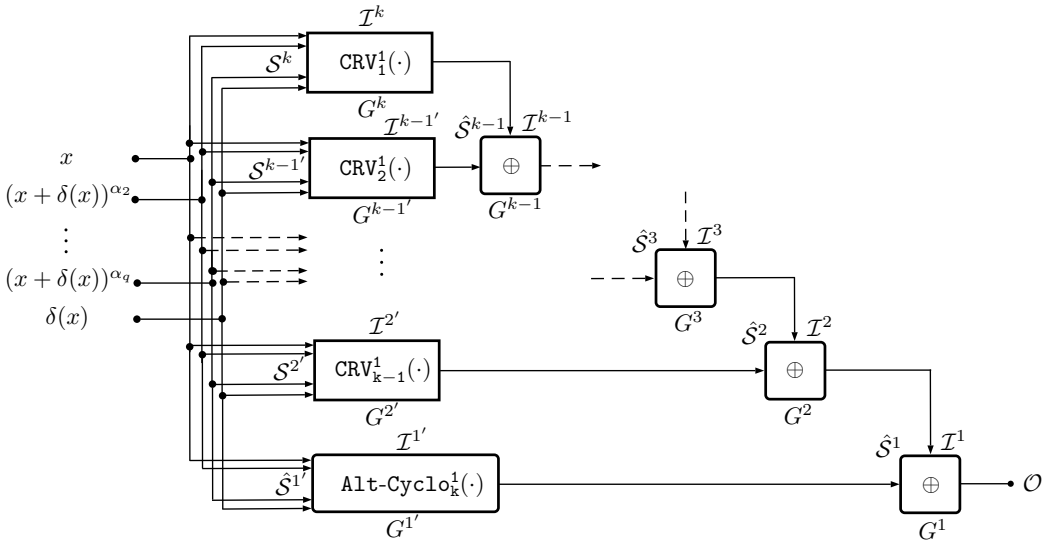


Figure 2.13: Gadget $\mathsf{CRV}^{2}(\cdot)$.

Let $\Omega = (\mathcal{I}, \mathcal{O})$ be an observation set to simulate, made on the whole circuit of Fig. 2.4.2, such that $\mathcal{I} = \left(\bigcup_{1\leq i\leq k} \mathcal{I}^i\right) \cup \left(\bigcup_{1\leq i\leq k-1} \mathcal{I}^{i'}\right)$ and such that the global constraint $\left(\sum_{i=1}^{k} |\mathcal{I}^i| + \sum_{i=1}^{k-1} |\mathcal{I}^{i'}|\right) + |\mathcal{O}| \leq t$ is satisfied.

Gadget 1. Let $\Omega^1 = (\mathcal{I}^1, \mathcal{O})$. Since G^1 is affine-NI, we know that there exists an observation set $\hat{\mathcal{S}}^1$ such that $|\hat{\mathcal{S}}^1| \leq |\mathcal{I}^1 \cup \mathcal{O}| \leq |\mathcal{I}^1| + |\mathcal{O}|$ and the set of input shares $\hat{\mathcal{S}}^1$ is sufficient to simulate Ω^1.
Gadget 1'. Let $\Omega^{1'} = (\mathcal{I}^{1'}, \hat{\mathcal{S}}^1)$. Since $\mathsf{Alt\text{-}Cy}^1_{\mathsf{t}}(\cdot)$ is affine-NI, we know that there exists an observation set $\hat{\mathcal{S}}^{1'}$ such that $|\hat{\mathcal{S}}^{1'}| \leq |\mathcal{I}^{1'} \cup \hat{\mathcal{S}}^1| \leq |\mathcal{I}^{1'}| + |\mathcal{I}^1| + |\mathcal{O}|$ and the set of input shares $\hat{\mathcal{S}}^{1'}$ is sufficient to simulate $\Omega^{1'}$.
Gadget i, for every $i \in [2; k-1]$. Let $\Omega^i = (\mathcal{I}^i, \hat{\mathcal{S}}^{i-1})$. Since $\oplus$ is affine-NI, we know that there exists an observation set $\hat{\mathcal{S}}^i$ for G^i such that $|\hat{\mathcal{S}}^i| \leq |\mathcal{I}^i \cup \hat{\mathcal{S}}^{i-1}| \leq |\bigcup_{1\leq j\leq i} \mathcal{I}^j \cup \mathcal{O}| \leq \sum_{j=1}^{i} |\mathcal{I}^j| + |\mathcal{O}|$. Moreover, the set of input shares $\hat{\mathcal{S}}^i$ is sufficient to simulate the adversary observation set Ω^i made on Gadget i.
Gadget i', for every $i \in [2; k-1]$. Let $\Omega^{i'} = (\mathcal{I}^{i'}, \hat{\mathcal{S}}^i)$. Since $\mathsf{CRV}^1_{\mathsf{i}}(\cdot)$ is t-SNI and $|\mathcal{I}^{i'} \cup \hat{\mathcal{S}}^i| \leq t$ (by simulation of Gadgets j for every $j \in [2, i]$ and the global constraint), we know that there exists an observation set $\mathcal{S}^{i'}$ for $G^{i'}$ such that $|\mathcal{S}^{i'}| \leq |\mathcal{I}^{i'}|$ and the set of input shares $\mathcal{S}^{i'}$ is sufficient to simulate the adversary observation set $\Omega^{i'}$ made on Gadget i'.
Gadget k. Let $\Omega^k = (\mathcal{I}^k, \hat{\mathcal{S}}^{k-1})$. Since $\mathsf{CRV}^1_1(\cdot)$ is t-SNI and $|\mathcal{I}^k \cup \hat{\mathcal{S}}^{k-1}| \leq \sum_{i=1}^{k} |\mathcal{I}^i| \cup |\mathcal{O}| \leq t$ (by simulation of Gadget i for $i \in [1, k-1]$ and the global constraint), we know that there exists an observation set $\mathcal{S}^k$ such that $|\mathcal{S}^k| \leq |\mathcal{I}^k|$ and $\mathcal{S}^k$ is sufficient to simulate Ω^k.

In order to simulate Gadget $\mathsf{CRV}^2(\cdot)$, the corresponding simulator requires the shares $\bigcup_{1\leq i\leq k-1} \mathcal{S}^{i'} \cup \mathcal{S}^k$ and $|\bigcup_{1\leq i\leq k-1} \mathcal{S}^{i'} \cup \mathcal{S}^k| \leq |\mathcal{I}^k| + \sum_{i=1}^{k-1} |\mathcal{I}^{i'}| + |\mathcal{I}^1| + |\mathcal{O}| \leq t$. Therefore Gadget $\mathsf{CRV}^2(\cdot)$ is affine-NI. □

We now prove the following theorem regarding the full construction of our alternate CRV approach.

Theorem 2.4.3. *Alternate* CRV *is* t-SNI. *Let* $(x_i)_{0\leq i\leq d}$ *be the input and let* $(y_i)_{0\leq i\leq d}$ *be the output of Gadget* $\mathsf{Alt\text{-}CRV}(\cdot)$ *represented Fig. 2.11 and 2.14. For any adversary set of* t *probed wires* $\Omega = (\mathcal{I}, \mathcal{O})$, *with* $t \leq d$, *there exists a set* $\mathcal{S}$ *of input shares such that* $|\mathcal{S}| \leq |\mathcal{I}|$ *from which* Ω *can be perfectly simulated.*

Proof. Let $\Omega = ((\mathcal{I}^1 \cup \mathcal{I}^2), \mathcal{O})$ be an observation set we want to simulate, made on the whole circuit represented Fig. 2.14 (or equivalently Fig. 2.11) such that $|\mathcal{I}^2| + |\mathcal{I}^1| + |\mathcal{O}| \leq t$.

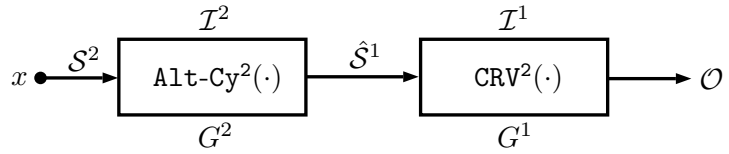


Figure 2.14: Gadget $\mathsf{Alt\text{-}CRV}(\cdot)$.

Gadget 1. Let $\Omega^1 = (\mathcal{I}^1, \mathcal{O})$. Since G^1 is affine-NI (by Lemma 4) and $|\mathcal{I}^1 \cup \mathcal{O}| \leq t$ (by the global constraint), we know that there exists an observation set $\mathcal{S}^1$ such that $|\mathcal{S}^1| \leq |\mathcal{I}^1| + |\mathcal{O}|$ and the set of input shares $\mathcal{S}^1$ is sufficient to simulate the adversary observation set Ω^1 made on Gadget 1.
Gadget 2. Let $\Omega^2 = (\mathcal{I}^2, \hat{\mathcal{S}}^1)$. Since G^2 is t-SNI (by Lemma 2) and $|\mathcal{I}^2 \cup \hat{\mathcal{S}}^1| \leq |\mathcal{I}^2| + |\mathcal{I}^1| + |\mathcal{O}| \leq t$ (by simulation of Gadget 1 and the global constraint), we know that there exists an observation set $\mathcal{S}^2$ such that $|\mathcal{S}^2| \leq |\mathcal{I}^2|$ and $\mathcal{S}^2$ is sufficient to simulate Ω^2.

To simulate Gadget Alt-CRV$(\cdot)$, the simulator needs the shares $\mathcal{S}^2$ of x and $|\mathcal{S}^2| \leq |\mathcal{I}^2| \leq |\mathcal{I}^1| + |\mathcal{I}^2| \leq t$. Therefore Alt-CRV$(\cdot)$ is t-SNI. □

2.5 Implementation Results

In this section we compare the efficiency of our alternate approach for the cyclotomic and the CRV methods with that of the original approach (CGPQR) for orders $d = 1, 2, 3$. We wrote the codes in assembly language for an 8051 based 8-bit architecture with bit-addressable memory and we provide implementation results for different settings related to S-boxes of size 4 to 8. For the sake of clarity, we begin to explicit our implementation choices and provide timings (in cycles) for several elementary operations. For elementary operations $\oplus$ and $\odot$, we experienced $\mathsf{C}_\oplus = \mathsf{C}_\odot = 1$ cycle.

Finite field multiplications. We tabulated them for S-boxes of dimension $n = 4$ at the cost of 2^8 bytes of memory and we experienced $\mathsf{C}_\otimes = 10$ cycles. For larger dimensions, the memory required to store such tables becomes prohibitive. In cases $n \in [5; 8]$, we implemented finite fields multiplications using exp/log tables. This approach still requires to store two tables with 2^n entries each, but offers a good trade-off between execution time and memory cost. The most tricky part of the exp/log multiplication is to manage the case where the inputs equal 0 while avoiding any conditional branch. In our GPQ based alternate approaches, there are always one non-zero input involved in field multiplications which yields to slightly more efficient field multiplications than in the classical CGPQR approach. A time constant field multiplication is executed in $\mathsf{C}_\otimes = 38$ cycles in the context of CGPQR while it only takes $\mathsf{C}_\otimes = 25$ cycles in our alternate proposals.

$(8 \times n)$-matrix transposition. We recalled in Section 2.2.1 a bit-sliced approach that computes n Dirac functions simultaneously over $\mathbb{F}_{2^n}$. The procedure (Alg. 11) involves $(n \times n)$-matrix transpositions. However, on 8-bit architectures we are able to simultaneously compute 8 Dirac functions at a time for any S-box dimension

lower or equal to 8 which consequently requires $(8 \times n)$-matrix transpositions. We experienced $\mathsf{C}_{\mathsf{T}} = 150$ cycles to transpose $(8 \times n)$-matrices for $5 \leq n \leq 8$. Note that 8-bit architectures allow us to fill each register with two elements of $\mathbb{F}_{2^4}$ in order to faster the transformation in that particular case leading to a cost $\mathsf{C}_{\mathsf{T}} = 75$ cycles.

We give costs of the transformations involve in GPQ along with the cost of a secure multiplication using ISW (Alg. 9) in Table 2.3.

Table 2.3: Costs of Secure-Dirac (Alg. 11), AMtoMM (Alg. 12), MMtoAM (Alg. 13) and SecMult (Alg. 9).

	Costs (in cycles)			
Order (d)	C_{δ}	$\mathsf{C}_{\mathsf{AMtoMM}}$	$\mathsf{C}_{\mathsf{MMtoAM}}$	$\mathsf{C}_{\mathsf{SecMult}}$
1	43	51	53	156
2	72	129	133	354
3	105	234	240	632

2.5.1 Cyclotomic Method

The cyclotomic method only consists in evaluating a polynomial whose monomials may belong to any of the q distinct cyclotomic classes of $[0, 2^n - 2]$. The classical CGPQR scheme requires to secure each of the $q - 1$ nonlinear multiplications with ISW ($q - 2$ if the S-box is balanced, see [14]). Considering our proposal, a secure polynomial evaluation implies to process 1 Secure-Dirac$(\cdot)$, 1 AMtoMM$(\cdot)$ and $q-1$ MMtoAM$(\cdot)$ ($q - 2$ if the S-box is balanced). Table 2.4 lists the costs (in cycles) to evaluate polynomials over $\mathbb{F}_{2^n}$ with $n \in [4; 8]$.

When finite field multiplications can be tabulated (when $n = 4$), our proposal does not lead to improvement of efficiency. In this case, the original approach is preferred. In all other scenarios, our proposal is approximatively 3 times faster at orders $d = 1, 2, 3$. Those results illustrates the efficiency of our extended version of GPQ for polynomials.

2.5.2 CRV Method

Regarding the CRV method, its processing can be divided into two main stages. First it requires to generate polynomials whose monomials are derived from a set of l distinct cyclotomic classes. This stage requires $l - 2$ nonlinear multiplications with the classical approach or 1 Secure-Dirac$(\cdot)$, 1 AMtoMM$(\cdot)$ and $l - 2$

Table 2.4: Costs of evaluating S-boxes of size $4 \leq n \leq 8$ with the cyclotomic method and our alternate proposal.

Method	Order (d)	n				
		4	5	6	7	8
Our proposal	1	**83**	**246**	**553**	**860**	**1677**
[14]		132	780	1716	2652	5148
Our proposal	2	276	**585**	**1362**	**2138**	**4205**
[14]		**174**	1770	3894	6018	11682
Our proposal	3	477	**1036**	**2445**	**3854**	**7603**
[14]		**293**	3160	6952	10744	20856

MMtoAM$(\cdot)$ with ours. Then the evaluation is completed with $k-1$ additional nonlinear multiplications secured with ISW for both approaches.

New parameters. As mentioned in Section 2.4.2, our proposal enables to consider new settings for parameters l,k and L. We list in Table 2.5 the settings that led to better performances in practice. We were able to derive more efficient parameters for S-boxes of dimension n with $n \in \{5, 7, 8\}$.

Table 2.5: New settings for parameters k and l of the CRV method.

n	l	k	$\lvert L\rvert$	L
5	5	2	21	$C_0 \cup C_1 \cup C_5 \cup C_7 \cup C_{15}$
7	8	3	50	$C_0 \cup C_1 \cup C_3 \cup C_9 \cup C_{11} \cup C_{15} \cup C_{21} \cup C_{43}$
8	10	4	69	$C_0 \cup C_1 \cup C_{15} \cup C_{31} \cup C_{39} \cup C_{43} \cup C_{53} \cup C_{61} \cup C_{111} \cup C_{119}$

We report in Table 2.6 the cost (in cycles) of the CRV method with the original approach compared to our proposal. Parameters l and k have been chosen accordingly to [21] for the original approach, while our alternate proposal uses our new settings for S-boxes of dimension $n \in \{5, 7, 8\}$.
Again in the particular case $n = 4$, the original approach is preferred since finite field multiplications can be tabulated. However, our alternate proposal outperforms the original in every other scenario.

Table 2.6: Costs of evaluating S-boxes of size $4 \leq n \leq 8$ with the CRV method and our alternate proposal.

		n				
Method	Order (d)	4	5	6	7	8
Our proposal	1	127	**402**	**559**	**713**	**972**
[21]		**88**	624	780	1092	1560
Our proposal	2	276	**939**	**1296**	**1685**	**2300**
[21]		**204**	1416	1770	2478	3540
Our proposal	3	477	**1668**	**2305**	**3012**	**4117**
[21]		**368**	2528	3160	4424	6320

2.6 Conclusion

In this chapter, we have proven the security of the power function masking scheme GPQ under the t-SNI definition. We have extended the GPQ scheme to the evaluation of polynomials over $\mathbb{F}_{2^n}$ and we have proven the security of the resulting construction under the t-SNI definition. Our extension results in an alternate cyclotomic method which we have plugged into the CRV construction in order to speed up polynomial evaluations. We have analyzed our alternate CRV construction and we have proven that it is t-SNI. Moreover, we have provided new sets of parameters that improve even more the efficiency of our alternate approach for CRV. We have given implementation results in several realistic scenarios where S-boxes are of dimension $n \in \{4, 5, 6, 7, 8\}$. Given those results, we argue that our t-SNI proposal for polynomial evaluation over $\mathbb{F}_{2^n}$ is a better alternative than the original approach in all scenarios where finite field multiplications are not tabulated.

Part IV

Fast Transform over Finite Fields of Characteristic p

Chapter 1

Introduction

Lin, Chung and Han introduced in 2014 the LCH-basis in order to derive FFT-based multipoint evaluation and interpolation algorithms with respect to this polynomial basis. Considering an affine space of $n = 2^j$ points, their algorithms require $O(n \cdot \log_2 n)$ operations in $\mathbb{F}_{2^r}$. The LCH-basis has then been extended over finite fields of characteristic p by Lin et al. in 2016 and an n-point evaluation algorithm has been derived for $n = p^j$ with complexity $O(n \cdot \log_p n \cdot p)$. However, the problem of interpolating polynomials represented in such a basis over $\mathbb{F}_{p^r}$ has not been addressed.

In next chapter, we fill this gap and also derive a faster algorithm for evaluating polynomials in the LCH-basis at multiple points over $\mathbb{F}_{p^r}$. We follow a different approach where we represent the multipoint evaluation and interpolation maps by well-defined matrices. We present factorizations of such matrices into the product of sparse matrices which can be evaluated efficiently. These factorizations lead to fast algorithms for both the multipoint evaluation and the interpolation of polynomials represented in the LCH-basis at $n = p^j$ points with optimized complexity $O(n \cdot \log_2 n \cdot \log_2 p \cdot \log_2 \log_2 p)$.

A particular attention is paid to provide in-place algorithms with high memory-locality. Our implementations written in C confirm that our approach improves the original transforms.

Remark 4. *The motivation of this chapter was the improvement of the paper [35] (and by extension [36]) which presents a design of a secure multiplication. This last project is not joined to this thesis due to lack of time but will be made available on ePrint very soon.*

Let $\mathbb{F}_{p^r}$ be the finite field with p^r elements and let us denote the set of polynomials

with respect to the monomial basis $\{1, x, x^2, \ldots\}$ by $\mathbb{F}_{p^r}[x]$. Let $\mathsf{M}(n)$ denote the number of field operations required to multiply two polynomials in $\mathbb{F}_{p^r}[x]$ of degree less than n, which may be taken to be in $O(n \log_2 n \log_2 \log_2 n)$. The standard multipoint evaluation and interpolation problems at n points of such polynomials over $\mathbb{F}_{p^r}$ can be solved with $O(\mathsf{M}(n) \log_2 n)$ operations in $\mathbb{F}_{p^r}$ [26].

The multipoint evaluation and interpolation can be performed with even lower algebraic complexities when applied to particular sets of evaluation points. The so-called Fast Fourier transform (FFT) [18] based algorithms offer complexities as low as $O(n \log_2 n)$ [45, 47, 48, 58].

1.1 Related Works

The LCH-basis presented in [45] is based on subspace polynomials over $\mathbb{F}_{2^r}$. It has been shown that the resulting FFT-based multipoint evaluation and interpolation on a affine space of $n = 2^j$ points of polynomials that are represented in that basis can be done in $O(n \cdot \log_2 n)$ operations over $\mathbb{F}_{2^r}$. The transforms have been mostly employed in coding theory and in particular applied to Reed-Solomon codes [44, 45, 46, 47, 48]. Basis conversion algorithms are also proposed in [48]. Furthermore, fast polynomial arithmetic in the LCH-basis are presented in [47]. In particular, a fast polynomial division and a fast half-GCD algorithm are derived. Besides, fast multiplications for long binary polynomials can be found in [17].

Regarding finite fields of characteristic p, [48] extends the LCH-basis over $\mathbb{F}_{p^r}$. However, only a solution for the forward transform is presented. It leads to a multipoint evaluation of polynomials in the LCH-basis of degree less than $n = p^j$ in $O(n \cdot \log_p n \cdot p)$ operations of $\mathbb{F}_{p^r}$. To the best of our knowledge, no solution has been provided to solving the interpolation problem of such polynomials over $\mathbb{F}_{p^r}$.

1.2 Contributions

Next chapter presents another approach than the original solution presented in [47] for solving the multipoint evaluation and interpolation problems. Namely, we express the map of the multipoint evaluation of polynomials in the LCH-basis by a matrix.

We then present a factorization of this matrix into the product of sparse matrices and show how to perform the global computation efficiently. Our approach relies on standard fast polynomial arithmetic over $\mathbb{F}_{p^r}$. It leads to a solution for

multipoint evaluation at $n = p^j$ points of polynomials that are represented in the LCH-basis over $\mathbb{F}_{p^r}$ optimized from $O(n \cdot \log_p n \cdot p)$ to $O(n \cdot \log_2 n \cdot \log_2 p \cdot \log_2 \log_2 p)$.

We deduce the invertibility of the matrix corresponding to the evaluation map from its factorization. As the inverse matrix corresponds to the interpolation map, we therefore also provide a solution for fast interpolation of polynomials in the LCH-basis over $\mathbb{F}_{p^r}$ in $O(n \cdot \log_2 n \cdot \log_2 p \cdot \log_2 \log_2 p)$. To the best of our knowledge, no solution was provided until now for the interpolation problem of such polynomials over $\mathbb{F}_{p^r}$. Moreover our approach leads to in-place algorithms with high memory-locality.

Next chapter presents the results of our article [51] we presented at the International Symposium on Symbolic and Algebraic Computation (ISSAC), in 2020.

Chapter 2

Fast Multipoint Evaluation and Interpolation of Polynomials in the LCH-basis over $\mathbb{F}_{p^r}$.

This chapter is organized as follows. Section 1 expresses the problem of evaluating and interpolating polynomials the LCH-basis over $\mathbb{F}_{p^r}$. Section 2 gives useful prerequisites. Then, section 3 expresses the evaluation map by a matrix and presents our factorization into the product of sparse matrices. The matrix of the inverse map is also exhibited. Section 4 shows how to derive fast algorithms for multipoint evaluation and interpolation of polynomials in the LCH-basis over $\mathbb{F}_{p^r}$. Experimental results are given in section 5 and finally section 6 concludes the chapter.

Contents

2.1 Description of the Problem

The first three paragraphs present material introduced in [45, 48]. Let $v_0, \ldots, v_{r-1}$ be a basis of $\mathbb{F}_{p^r}$ over $\mathbb{F}_p$, *i.e.* $\langle v_0, \ldots, v_{r-1}\rangle = \mathbb{F}_{p^r}$ and let $e_0, \ldots, e_{r-1}$ be the canonical basis of $\mathbb{F}_p^r$, *i.e.* , $\langle e_0, \ldots, e_{r-1}\rangle = \mathbb{F}_p^r$ where any basis element e_k is a vector $(e_{0,k}, \ldots, e_{r-1,k})$ with $e_{i,k} = 1$ if $k = i$ and 0 otherwise.

The LCH-basis over $\mathbb{F}_{p^r}$. For any $i, j \geq 0$ such that $i + j \leq r$, let U_i^j and V_i^j be respectively subspaces of $\mathbb{F}_{p^r}$ and $\mathbb{F}_p^r$ defined as

$$U_i^j = \langle v_i, \ldots, v_{i+j-1}\rangle, \quad V_i^j = \langle e_i, \ldots, e_{i+j-1}\rangle .$$

Observe that they form stricly ascending chains. Namely,

$$U_0^0 \subset U_0^1 \subset \ldots \subset U_0^r = \mathbb{F}_{p^r} \text{ and } V_0^0 \subset V_0^1 \subset \ldots \subset V_0^r = \mathbb{F}_p^r ,$$

with $U_0^0 = V_0^0 = \{0\}$. Now let $L_{U_0^k}$ be linearized polynomials defined over $\mathbb{F}_{p^r}$ and of degree p^k such that $\ker\big(L_{U_0^k}\big) = U_0^k$. They can be recursively defined from $L_{U_0^0}(x) = x$ by the following relation

$$L_{U_0^{k+1}}(x) = L_{U_0^k}(x)^p - L_{U_0^k}(v_k)^{p-1} \cdot L_{U_0^k}(x) . \tag{2.1}$$

and they are linear (see [13]). Note that $L_{U_0^k}$ is invariant over the cosets of U_0^k in $\mathbb{F}_{p^r}$. Namely, for any $x \in U_0^k = \langle v_0, \ldots, v_{k-1}\rangle$ and for any $y \in U_k^{r-k} = \langle v_k, \ldots, v_{r-1}\rangle$,

$$L_{U_0^k}(x + y) = L_{U_0^k}(y) .$$

For any $x \in \mathbb{F}_{p^r}$ and for any $\omega = (\omega_0, \ldots, \omega_{r-1}) \in \mathbb{F}_p^r$ define

$$\chi_\omega(x) = \prod_{k=0}^{r-1} L_{U_0^k}(x)^{\omega_k} .$$

Identifying $\omega_j \in \mathbb{F}_p$ to its minimal representative over $\mathbb{N}$, the degree of $\chi_\omega(x)$ is therefore $\sum_{j=0}^{r-1} \omega_j \cdot p^j$.

Invariance of χ_ω. From the invariance property of $L_{U_0^k}$ and the fact that $U_0^k \subset U_0^{k+1}$ for any $k \in \{0, \ldots, r-1\}$, it follows that $L_{U_0^k}$ is invariant over the cosets of U_0^s with $s \leq k$. Also, for any $\omega = (\omega_0, \ldots, \omega_{r-1}) \in V_i^j$, we have $\omega_k = 0$ for any $k < i$. Therefore, $\chi_\omega(x) = \prod_{k=0}^{r-1} L_{U_0^k}(x)^{\omega_k} = \prod_{k=i}^{r-1} L_{U_0^k}(x)^{\omega_k}$. It turns out that χ_ω is invariant over the intersection of the set of cosets of U_0^s with $s \leq k$ for $k \in \{i, \ldots, r-1\}$. This last intersection are the cosets of U_0^t with $t \leq i$.

Polynomials in the LCH-basis over $\mathbb{F}_{p^r}$. Any polynomial of $\mathbb{F}_{p^r}[x]/(x^{p^r} - x)$ can be written in the LCH-basis as

$$P(x) = \sum_{\omega \in \mathbb{F}_p^r} \alpha_\omega \cdot \chi_\omega(x)\,, \tag{2.2}$$

with $\alpha_\omega \in \mathbb{F}_{p^r}$ for any $\omega \in \mathbb{F}_p^r$. The classical multipoint evaluation problem consists in evaluating $P(x)$ for any $x \in \mathbb{F}_{p^r}$. The classical interpolation problem consists in recovering the coefficients α_ω with $\omega \in \mathbb{F}_p^r$ from the values $P(x)$ for any $x \in \mathbb{F}_{p^r}$.

In [48], the authors consider a variation of the problem where the polynomials are of the shape

$$\sum_{\omega \in V} \alpha_\omega \cdot \chi_\omega(x) \tag{2.3}$$

and the set of evaluation is $U + \mu$ with $\mu \in \mathbb{F}_{p^r}$, U and V being respectively subspaces of $\mathbb{F}_{p^r}$ and $\mathbb{F}_p^r$ with same cardinalities. A divide-and-conquer approach was first proposed in [45] for solving this multipoint evaluation problem over $\mathbb{F}_{2^r}$. By backtracking their method, the same authors derived an algorithm for the interpolation problem. The forward approach was then generalized to characteristic p in [48]. However, no method was provided for solving the interpolation problem over $\mathbb{F}_{p^r}$.

2.2 Prerequisites

In this section, we first define an ordering over the subspaces that are involved in the evaluation of some polynomial represented in the LCH-basis . Then, we remind some definitions about matrices and we state two results about them that are based on the chosen order.

Ordering. Consider a finite totally ordered set $(S, \leq)$ such that $\#S = n$, i.e. $S = \{s_0, s_1, \ldots, s_{n-1}\}$ such that $s_i \leq s_j \Leftrightarrow i \leq j$. We define the *index* application $ind : S \mapsto \mathbb{N}$ defined by $ind(s_i) = i$ for any $i = 0, \ldots, n-1$.

Also for any arbitrary total order on $\mathbb{F}_p$, in the rest of the paper we consider the lexicographic order denoted by $\leq$ on $\mathbb{F}_p^r$. We may also define a total order on $\mathbb{F}_{p^r}$ by requiring that

$$\phi(\underline{x}) \overset{\text{def}}{\leq} \phi(\underline{y}) \Leftrightarrow \underline{x} \leq \underline{y} \text{ for any } \underline{x}, \underline{y} \in \mathbb{F}_p^r\,.$$

The symbol $\leq$ is used indifferently to refer to the total order on (subsets) of $\mathbb{F}_p^r$ and $\mathbb{F}_{p^r}$.

Matrices. Consider two finite totally ordered subsets, i.e. $S_1 = \{x_0, \ldots, x_{n-1}\}$ and $S_2 = \{y_0, \ldots, y_{m-1}\}$ where the elements are respectively written from the smallest to the largest. In what follows we handle matrices over $\mathbb{F}_{p^r}$. The matrix A is defined by

$$\big((\mathsf{A}(x,y)\big)_{x\in S_1, y\in S_2} \triangleq \begin{pmatrix} a_{x_0,y_0} & a_{x_0,y_1} & \cdots & a_{x_0,y_{m-1}} \\ a_{x_1,y_0} & a_{x_1,y_1} & \cdots & a_{x_1,y_{m-1}} \\ \vdots & \vdots & \ddots & \vdots \\ a_{x_{n-1},y_0} & a_{x_{n-1},y_1} & \cdots & a_{x_{n-1},y_{m-1}} \end{pmatrix}.$$

The symbol $\otimes$ denotes the Kronecker product of matrices. The product of several matrices $\prod_{i=0}^{n} A_i$ stands for $A_0 \cdot A_1 \cdot \ldots \cdot A_n$. Finally, $\oplus_{i\in S} A_i$ denotes the block diagonal matrix $\mathrm{diag}(A_{s_0}, \ldots, A_{s_{n-1}})$. These notations can be found in [39]. The following well-known lemma can be found in [22].

Lemma 2.2.1. *For any two totally ordered sets $(S_1, \leq_1)$ and $(S_2, \leq_2)$, consider the sets $S_1 \times S_2$ and $S_2 \times S_1$ together with their respective lexicographic orders. Consider the matrix P defined by*

$$\mathsf{P} = (\mathsf{P}(x,y))_{x\in S_2\times S_1, y\in S_1\times S_2} \quad \text{with } \mathsf{P}(x,y) = \begin{cases} 1 & \text{if } \pi(y) = x \\ 0 & \text{otherwise} \end{cases}$$

where $\pi : S_1 \times S_2 \to S_2 \times S_1 : (s_1, s_2) \to (s_2, s_1)$. Also, for any matrix $\mathsf{A} = (\mathsf{A}(x,y))_{x,y\in S_2\times S_1}$, we have

$$\mathsf{P}^{\mathsf{T}} \cdot \mathsf{A} \cdot \mathsf{P} = (\mathsf{A}(\pi(x), \pi(y)))_{x,y\in S_1\times S_2}\,. \tag{2.4}$$

Also, note that for any $(s_1, s_2), (s_3, s_4) \in S_1 \times S_2$ and A such that

$$\mathsf{A} = \big(\mathsf{A}((s_1,s_2),(s_3,s_4))\big)_{(s_1,s_2),(s_3,s_4)\in S_1\times S_2}\,,$$

we then have

$$\mathsf{A} = \Big(\big(\mathsf{A}((s_1,s_2),(s_3,s_4))\big)_{s_2,s_4\in S_2}\Big)_{s_1,s_3\in S_1}\,. \tag{2.5}$$

Proof. Let us begin by proving relation (2.4). For any $z \in S_2 \times S_1$ and for any $y \in S_1 \times S_2$,

$$(\mathsf{A}\cdot\mathsf{P})(z,y) = \sum_{u\in S_2\times S_1} \mathsf{A}(z,u)\cdot\mathsf{P}(u,y) = \mathsf{A}(z,\pi(y))\,.$$

Therefore, for any $x, y \in S_1 \times S_2$,

$$(\mathsf{P}^{\mathsf{T}}\cdot\mathsf{A}\cdot\mathsf{P})(x,y) = \sum_{z\in S_2\times S_1} \mathsf{A}(z,\pi(y))\cdot\mathsf{P}(z,x) = \mathsf{A}(\pi(x),\pi(y))\,.$$

Let us now prove relation (2.5). For any two totally ordered sets $(S_1, \leq_1), (S_2, \leq_2)$ with $\#S_1 = n_1, \#S_2 = n_2$ and the lexicographic order on $S_1 \times S_2$, the index of $(s_1, s_2) \in S_1 \times S_2$ is given by

$$ind(s_1, s_2) = ind(s_1) \cdot \#S_2 + ind(s_2)\,. \tag{2.6}$$

Therefore, $A = \big(A((s_1, s_2), (s_3, s_4))\big)_{(s_1,s_2),(s_3,s_4) \in S_1 \times S_2}$ is composed of $n_1 \times n_1$ blocks each of size $\#S_2 \times \#S_2$. Therefore,

$$\mathsf{A} = \Big(\big(\mathsf{A}((s_1, s_2), (s_3, s_4))\big)_{s_2,s_4 \in S_2}\Big)_{s_1,s_3 \in S_1}\,.$$

□

Now, consider the map

$$\pi_{1,j} : \mathbb{F}_p \times \mathbb{F}_p^{j-1} \mapsto \mathbb{F}_p^{j-1} \times \mathbb{F}_p : (s, t) \mapsto \pi(s, t) = (t, s)\,.$$

Identify the spaces $\mathbb{F}_p \times \mathbb{F}_p^{j-1}, \mathbb{F}_p^{j-1} \times \mathbb{F}_p$ and $\mathbb{F}_p^j$. Define π_0 as the identity map and

$$\pi_{k,j} = \begin{cases} \underbrace{\pi_{1,j} \circ \pi_{1,j} \circ \ldots \circ \pi_{1,j}}_{k} & \text{if } k > 0\,, \\ (\pi_{-k,j})^{-1} & \text{if } k < 0\,. \end{cases}$$

For any $k \in \mathbb{Z}$, define $\mathsf{P}_{k,j} = (\mathsf{P}_{k,j}(x, y))_{x,y \in \mathbb{F}_p^j}$ where

$$\mathsf{P}_{k,j}(x, y) = \begin{cases} 1 & \text{if } \pi_{k,j}(y) = x \\ 0 & \text{otherwise.} \end{cases}$$

The second part of the following corollary can be found in [22].

Corollary 2.2.1.1. *We have $P_{j,j} = I_{p^j}$ and $P_{k,j} = P_{1,j}^{k \bmod j}$ for any $k \in \mathbb{Z}$. Moreover, if $A = (A(x, y))_{x,y \in \mathbb{F}_p^k}$ and $B = (B(x, y))_{x,y \in \mathbb{F}_p^{j-k}}$, then*

$$P_{k,j}^{\mathsf{T}} \cdot (B \otimes A) \cdot P_{k,j} = A \otimes B\,.$$

where $\mathbb{F}_p^k \times \mathbb{F}_p^{j-k}, \mathbb{F}_p^{j-k} \times \mathbb{F}_p^k$ and $\mathbb{F}_p^j$ are identified.

Proof. By definition of $\mathsf{P}_{k,j}$ and $\pi_{0,j}$, we have $\mathsf{P}_{0,j} = \mathsf{I}_{p^j}$. Also, for any $x, y \in \mathbb{F}_{p^j}$ and any $k \geq 1$

$$\begin{aligned} \left(\mathsf{P}_{k-1,j} \cdot \mathsf{P}_{1,j}\right)(x, y) &= \sum_{v \in \mathbb{F}_p^j} \mathsf{P}_{k-1,j}(x, v) \cdot \mathsf{P}_{1,j}(v, y) \\ &= \begin{cases} 1 & \text{if } \pi_{k,j}(y) = x\,, \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

Therefore, $\mathsf{P}_{k,j} = \mathsf{P}_{k-1,j} \cdot \mathsf{P}_{1,j}$. It follows that $\mathsf{P}_{k,j} = \mathsf{P}_{1,j}^k$ for any $k \geq 0$. Observing that $\pi_{j,j} = \underbrace{\pi_{1,j} \circ \ldots \circ \pi_{1,j}}_{j} = \text{id}$, we have $\mathsf{P}_{j,j} = \mathsf{I}_{p^j}$. Therefore, $\mathsf{P}_{k,j} = \mathsf{P}_{1,j}^{k \bmod j}$ for any $k \geq 0$. According to the definition of $\mathsf{P}_{k,j}$ and $\pi_{k,j}$, for any $k < 0, \mathsf{P}_{k,j} = \left(\mathsf{P}_{-k,j}\right)^{-1}$. The element $-k$ being positive, we have $\mathsf{P}_{-k,j} = \mathsf{P}_{1,j}^{-k}$. Therefore, $\mathsf{P}_{k,j} = \left(\mathsf{P}_{1,j}^{-k}\right)^{-1} = \left(\mathsf{P}_{1,j}^{-1}\right)^{-k}$. Also, $\mathsf{P}_{j,j} = \mathsf{P}_{1,j}^j = \mathsf{I}_{p^j}$. It follows that $\mathsf{P}_{1,j}^{-1} = \mathsf{P}_{1,j}^{j-1}$. It turns out that $\mathsf{P}_{k,j} = \left(\mathsf{P}_{1,j}^{j-1}\right)^{-k} = \mathsf{P}_{1,j}^{(j-1)\cdot(-k)} = \mathsf{P}_{1,j}^{(j-1)\cdot(-k) \bmod j} = \mathsf{P}_{1,j}^{k \bmod j}$.

The second part of the corollary follows immediately from Lemma 2.2.1. □

2.3 A Factorization of Two Matrices

In this section, we start by reducing the multipoint evaluation and interpolation problems expressed in Section 2.1 to a canonical form. Then we associate a matrix to the multipoint evaluation map. We give a factorization of this matrix into the product of sparse matrices. This factorization allows us to derive its inverse matrix which corresponds to the inverse map. Finally, we show that the inverse matrix also factorizes into the product of sparse matrices.

Canonical form. Consider fixed integers i and j such that $i, j \geq 0$ and $i + j \leq r$. Remember that $U_i^j = \langle v_i, \ldots, v_{i+j-1} \rangle$ and $V_i^j = \langle e_i, \ldots, e_{i+j-1} \rangle$. In what follows $U_0^i, U_i^j, U_{i+j}^{r-(i+j)}$ and V_i^j will be denoted respectively by U_L, U, U_R and V for the sake of clarity. Note that $\mathbb{F}_{p^r} = U_L \oplus U \oplus U_R$. Consider the multipoint evaluation and interpolation problems on a subset of polynomials of the shape

$$P_V(x) = \sum_{\omega \in V} \alpha_\omega \cdot \chi_\omega(x) \quad \text{for any } x \in U + \mu\,. \tag{2.7}$$

Note that because the vector basis $(v_i)_i$ of $\mathbb{F}_{p^r}$ may be freely chosen, the space U may be any vector space of $\mathbb{F}_{p^r}$ with dimension j.

Let us show that these problems may be reduced to a canonical form. Without loss of generality, we may assume that $\mu \in U_L \oplus U_R$. For any $x \in U$

$$P_V(x + \mu) = \sum_{\omega \in V} \alpha_\omega \cdot \chi_\omega(x + \mu)\,.$$

Writting μ as $\mu_L + \mu_R$ where $\mu_L \in U_L$ and $\mu_R \in U_R$, we have for any $\omega \in V$ and any $x \in U$ that

$$\chi_\omega(x + \mu_L + \mu_R) = \chi_\omega(x + \mu_R)$$

because χ_ω is invariant on the cosets of U_L. Hence for any $x \in U$

$$P_V(x+\mu) = \sum_{\omega \in V} \alpha_\omega \cdot \chi_\omega(x+\mu_R)\,.$$

We deduce that evaluating $P_V(x)$ over $U+\mu$ reduces to the evaluation of P_V over $U+\mu_R$.

Matrices of evaluation and interpolation maps. Given $n = p^j$ points $x \in U+\mu_R$, we define the linear evaluation map $E : \mathbb{F}_{p^r}^n \mapsto \mathbb{F}_{p^r}^n$ by $E\left((\alpha_\omega)_{\omega\in V}\right) = (P_V(x))_{x\in U+\mu_R}$. This linear map can be represented by the matrix

$$\mathsf{X} \triangleq \left(\chi_\omega(x)\right)_{x\in U+\mu_R, \omega\in V}\,. \tag{2.8}$$

A factorization of X for fast evaluation. Let us present a factorization of X into the product of sparse matrices. Let us first introduce descending chains of $U+\mu_R$ and V. More precisely, for any $k \in \{0,\ldots,j-1\}, U_k = \langle v_{i+k},\ldots,v_{i+j-1}\rangle + \mu_R$ and $V_k = \langle e_{i+k},\ldots,e_{i+j-1}\rangle$. Let $U_j = \{\mu_R\}$ and $V_j = (0,\ldots,0)$. We have $U_s \subset U_t$ and $V_s \subset V_t$ if $s > t$, also $U_0 = U+\mu_R$ and $V_0 = V$. Also, let us define the increasing chain of sets $U_{L,k} = U_0^{i+k}$ for $k \in \{0,\ldots,j-1\}$. These sets are related by the invariant $U_{L,k} \oplus U_k = U_L \oplus U_0 = U_0^{i+j} + \mu_R$ for any k which expresses different ways to decompose the affine space $U_0^{i+j} + \mu_R$. Finally, these sets are totally ordered according to Section 2.2.

Remark. Relation (2.9) in the following theorem gives essentially a matrix expression of the recurrence relation behind the divide-and-conquer approach developed in [48]. Its proof also relies on the invariance property of the LCH-basis .

Theorem 2.3.1. *Consider the linearized polynomials $L_k = L_{U_{L,k}}$ defined such that $ker(L_k) = U_{L,k}$ for $k = 0,\ldots,j-1$. The matrices X_k defined by $(\chi_\omega(x))_{x\in U_k,\omega\in V_k}$ satisfy the recursion*

$$\mathsf{X}_k = \left(\bigoplus_{x\in U_{k+1}} \mathsf{V}_k(x)\right) \cdot \mathsf{P}_{1,j-k} \cdot (\mathsf{I}_p \otimes \mathsf{X}_{k+1}) \cdot \mathsf{P}_{1,j-k}^{\mathsf{T}}\,. \tag{2.9}$$

for any $k = 0,\ldots,j-1$ where V_k is the Vandermonde matrix defined as

$$\mathsf{V}_k(x) = \left(L_k(x + c\cdot v_{i+k})^d\right)_{c,d\in\mathbb{F}_p}$$

for any $x \in U_{k+1}$ and $\mathsf{P}_{1,j-k}$ is defined in Section 2.2. Also, $\mathsf{X} = \mathsf{X}_0$ and we have that

$$\mathsf{X} = \prod_{k=0}^{j-1} \left(\left(\bigoplus_{x\in U_{k+1}} \mathsf{V}_k(x)\right) \otimes \mathsf{I}_{p^k}\right)\,, \tag{2.10}$$

or equivalently

$$X = \prod_{k=0}^{j-1} \left(\left(\bigoplus_{x \in U_{k+1}} \left(I_{p^k} \otimes V_k(x) \right) \right) \cdot B_k \right) \tag{2.11}$$

where

$$B_k = \begin{cases} I_{p^{j-(k+2)}} \otimes \left((I_p \otimes P_{1,k+1}) \cdot P_{1,k+2}^{\mathsf{T}} \right) & \text{if } 0 \leq k < j-1\,, \\ P_{1,j} & \text{if } k = j-1\,. \end{cases} \tag{2.12}$$

and $P_{1,j}$, $P_{1,k+1}$, $P_{1,k+2}$ are defined in Section 2.2.

Proof. Let us prove relation (2.9) by first showing that

$$X_k = \left(\bigoplus_{x \in U_{k+1}} V_k(x) \right) \cdot (X_{k+1} \otimes I_p)\ . \tag{2.13}$$

In what follows $\delta_b(d)$ denotes the *Kronecker delta function*, *i.e.* $\delta_b(d) = 1$ if $b = d$ and 0 otherwise. On the one hand we have

$$\left(\bigoplus_{x \in U_{k+1}} V_k(x) \right) = \left(\delta_x(z) \cdot \left(\left(L_k(x + c \cdot v_{i+k}) \right)^b \right)_{c,b \in \mathbb{F}_p} \right)_{x,z \in U_{k+1}}\ .$$

Also, applying Lemma 2.2.1 on the right-hand side gives

$$\left(\delta_x(z) \cdot \left(L_k(x + c \cdot v_{i+k}) \right)^b \right)_{(x,c),(z,b) \in U_{k+1} \times \mathbb{F}_p}\ .$$

Therefore for any $(x,c),(z,b) \in U_{k+1} \times \mathbb{F}_p$,

$$\left(\bigoplus_{x \in U_{k+1}} V_k(x) \right)_{(x,c),(z,b)} = \delta_x(z) \cdot \left(L_k(x + c \cdot v_{i+k}) \right)^b\ . \tag{2.14}$$

On the other hand we have

$$\begin{aligned} X_{k+1} \otimes I_p &= \left(\chi_\omega(z) \cdot I_p \right)_{z \in U_{k+1}, \omega \in V_{k+1}} \\ &= \left(\chi_{\phi_{k+1}(\omega)}(z) \cdot I_p \right)_{z, \omega \in U_{k+1}} \end{aligned}$$

where

$$\phi_k : V_k \mapsto U_k : \omega = \sum_{s=i+k}^{i+j-1} \omega_s \cdot e_s \mapsto x = \left(\sum_{s=i+k}^{i+j-1} \omega_s \cdot v_s \right) + \mu_R\,.$$

We stress that this bijection preserves the order of all elements. Also, for any $z, \omega \in U_{k+1}$ we have

$$\left(\chi_{\phi_{k+1}(\omega)}(z) \cdot I_p \right)_{z,\omega} = \left(\chi_{\phi_{k+1}(\omega)}(z) \cdot \left(\delta_b(d) \right)_{b,d \in \mathbb{F}_p} \right)_{z,\omega}\ ,$$

Also, by Lemma 2.2.1, the right-hand side gives

$$\left(\chi_{\phi_{k+1}(\omega)}(z)\cdot\delta_b(d)\right)_{(z,b),(\omega,d)\in U_{k+1}\times\mathbb{F}_p}$$

It turns out that for any $(z,b)\in U_{k+1}\times\mathbb{F}_p$, for any $(\omega,d)\in V_{k+1}\times\mathbb{F}_p$,

$$\left(\mathsf{X}_{k+1}\otimes I_p\right)_{(z,b),(\omega,d)} = \chi_\omega(z)\cdot\delta_b(d)\,. \tag{2.15}$$

For any $(x,c),(z,b)\in U_{k+1}\times\mathbb{F}_p$, for any $(\omega,d)\in V_{k+1}\times\mathbb{F}_p$ we have that

$$\left(\left(\bigoplus_{x\in U_{k+1}}\mathsf{V}_k(x)\right)\cdot\left(\mathsf{X}_{k+1}\otimes\mathsf{I}_p\right)\right)_{(x,c),(\omega,d)} = \sum_{z,b}\left(\bigoplus_{x\in U_{k+1}}\mathsf{V}_k(x)\right)_{(x,c),\,(z,b)}\cdot\left(\mathsf{X}_{k+1}\otimes\mathsf{I}_p\right)_{(z,b),\,(\omega,d)}$$

and from (2.15) and (2.14), the right-hand side reduces to

$$\sum_{z,b}\delta_x(z)\cdot\left(L_k(x+c\cdot v_{i+k})\right)^b\cdot\chi_\omega(z)\cdot\delta_b(d)\,.$$

Therefore,

$$\left(\left(\bigoplus_{x\in U_{k+1}}\mathsf{V}_k(x)\right)\cdot\left(\mathsf{X}_{k+1}\otimes\mathsf{I}_p\right)\right)_{(x,c),(\omega,d)} = L_k(x+c\cdot v_{i+k})^d\cdot\chi_\omega(x)\,. \tag{2.16}$$

Observe that for any $\omega\in V_{k+1}$, $\chi_\omega(x)$ is invariant over the cosets of $U_{L,k+1}$. Also, $v_{i+k}\in U_{L,k+1}=\langle v_0,\ldots,v_{i+k}\rangle$. Thus,

$$L_k(x+c\cdot v_{i+k})^d\cdot\chi_\omega(x) = L_k(x+c\cdot v_{i+k})^d\cdot\chi_\omega(x+c\cdot v_{i+k})\,.$$

Note that $L_k(x+c\cdot v_{i+k})^d=\chi_{d\cdot e_{i+k}}(x+c\cdot v_{i+k})$ and therefore we have that

$$\begin{aligned}L_k(x+c\cdot v_{i+k})^d\cdot\chi_\omega(x) &= \chi_{d\cdot e_{i+k}}(x+c\cdot v_{i+k})\cdot\chi_\omega(x+c\cdot v_{i+k})\\ &= \chi_{\omega+d\cdot e_{i+k}}(x+c\cdot v_{i+k})\end{aligned}$$

where $\chi_{\omega+d\cdot e_{i+k}}(x+c\cdot v_{i+k}) = (\mathsf{X}_k)_{(x,c),\,(\omega,d)}$. Hence,

$$L_k(x+c\cdot v_{i+k})^d\cdot\chi_\omega(x) = (\mathsf{X}_k)_{(x,c),\,(\omega,d)}\ .$$

This proves (2.13). Finally, by gathering relation (2.16) with the above relation and applying corollary 2.2.1.1

$$\mathsf{X}_{k+1}\otimes\mathsf{I}_p = \mathsf{P}_{1,j-k}\cdot\left(\mathsf{I}_p\otimes\mathsf{X}_{k+1}\right)\cdot\mathsf{P}_{1,j-k}^{\mathsf{T}}$$

which concludes the proof of relation (2.9).

Let us now prove relation (2.10) by induction. Consider the base case $k = j$. By definition, $\mathsf{X}_j = \left(\chi_\omega(x)\right)_{x\in U_j, \omega\in V_j}$ with $U_j = \{\mu_R\}$ and $V_j = \{(0,0,\ldots,0)\}$. Hence $\mathsf{X}_j = \prod_{k=0}^{r-1}\left(L_{U_{L,k}}(\mu_R)\right)^0 = 1$. Also, $\mathsf{X}_j = \prod_\emptyset = 1$. Now, assume it holds that

$$\mathsf{X}_n = \prod_{k=n}^{j-1}\left(\left(\bigoplus_{x\in U_{k+1}} \mathsf{V}_k(x)\right)\otimes \mathsf{I}_{p^{k-n}}\right).$$

Starting by applying relation (2.13), we have

$$\mathsf{X}_{n-1} = \left(\bigoplus_{x\in U_n} \mathsf{V}_{n-1}(x)\right)\cdot\left(\mathsf{X}_n \otimes \mathsf{I}_p\right).$$

Then, using the induction hypothesis we can write

$$\mathsf{X}_{n-1} = \left(\bigoplus_{x\in U_n} \mathsf{V}_{n-1}(x)\right)\cdot\left(\prod_{k=n}^{j-1}\left(\left(\bigoplus_{x\in U_{k+1}} \mathsf{V}_k(x)\right)\otimes \mathsf{I}_{p^{k-n}}\right)\otimes \mathsf{I}_p\right).$$

Noting that $\mathsf{I}_p = \prod_{k=n}^{j-1}\mathsf{I}_p$ and $\prod_i \mathsf{A}_i \otimes \prod_i \mathsf{B}_i = \prod_i(\mathsf{A}_i\otimes\mathsf{B}_i)$,

$$\begin{aligned}\prod_{k=n}^{j-1}\left(\left(\bigoplus_{x\in U_{k+1}} \mathsf{V}_k(x)\right)\otimes \mathsf{I}_{p^{k-n}}\right)\otimes \mathsf{I}_p \\ = \prod_{k=n}^{j-1}\left(\left(\bigoplus_{x\in U_{k+1}} \mathsf{V}_k(x)\right)\otimes \mathsf{I}_{p^{k-n}}\otimes \mathsf{I}_p\right).\end{aligned}$$

Also $\mathsf{I}_{p^{k-n}}\otimes\mathsf{I}_p = \mathsf{I}_{p^{k-n+1}}$, therefore

$$\mathsf{X}_{n-1} = \left(\bigoplus_{x\in U_n} \mathsf{V}_{n-1}(x)\right)\cdot\left(\prod_{k=n}^{j-1}\left(\left(\bigoplus_{x\in U_{k+1}} \mathsf{V}_k(x)\right)\otimes \mathsf{I}_{p^{k-n+1}}\right)\right).$$

Since for any matrix $\mathsf{A}, \mathsf{A}\otimes\mathsf{I}_1 = \mathsf{A}$ and $\mathsf{I}_1 = \mathsf{I}_{p^0}$, then

$$\mathsf{X}_{n-1} = \left(\left(\bigoplus_{x\in U_n} \mathsf{V}_{n-1}(x)\right)\otimes \mathsf{I}_{p^0}\right)\cdot\left(\prod_{k=n}^{j-1}\left(\left(\bigoplus_{x\in U_{k+1}} \mathsf{V}_k(x)\right)\otimes \mathsf{I}_{p^{k-n+1}}\right)\right),$$

or equivalently

$$\mathsf{X}_{n-1} = \prod_{k=n-1}^{j-1}\left(\left(\bigoplus_{x\in U_{k+1}} \mathsf{V}_k(x)\right)\otimes I_{p^{k-n+1}}\right).$$

Let us now prove relation (2.11). Observe that

$$\mathsf{X} = \prod_{k=0}^{j-1} \left(\bigoplus_{x \in U_{k+1}} \mathsf{V}_k(x) \right) \otimes \mathsf{I}_{p^k} = \prod_{k=0}^{j-1} \left(\bigoplus_{x \in U_{k+1}} \left(\mathsf{V}_k(x) \otimes \mathsf{I}_{p^k} \right) \right) .$$

Also, according to corollary 2.2.1.1,

$$\mathsf{V}_k(x) \otimes \mathsf{I}_{p^k} = \mathsf{P}_{1,k+1}^{\intercal} \cdot \left(\mathsf{I}_{p^k} \otimes \mathsf{V}_k(x) \right) \cdot \mathsf{P}_{1,k+1} .$$

Therefore,

$$\mathsf{X} = \prod_{k=0}^{j-1} \left(\bigoplus_{x \in U_{k+1}} \left(\mathsf{P}_{1,k+1}^{\intercal} \left(\mathsf{I}_{p^k} \otimes \mathsf{V}_k(x) \right) \cdot \mathsf{P}_{1,k+1} \right) \right) .$$

Noting that

$$\bigoplus_{x \in U_{k+1}} \left(\mathsf{P}_{1,k+1}^{\intercal} \left(\mathsf{I}_{p^k} \otimes \mathsf{V}_k(x) \right) \cdot \mathsf{P}_{1,k+1} \right) = \mathsf{C}_k^{\intercal} \cdot \left(\bigoplus_{x \in U_{k+1}} \left(\mathsf{I}_{p^k} \otimes \mathsf{V}_k(x) \right) \right) \cdot \mathsf{C}_k$$

where $\mathsf{C}_k = \left(\mathsf{I}_{p^{j-(k+1)}} \otimes \mathsf{P}_{1,k+1} \right)$ and $\mathsf{C}_k^{\intercal} = \left(\mathsf{I}_{p^{j-(k+1)}} \otimes \mathsf{P}_{1,k+1}^{\intercal} \right)$, we have

$$\mathsf{X} = \prod_{k=0}^{j-1} \left(\mathsf{C}_k^{\intercal} \cdot \left(\bigoplus_{x \in U_{k+1}} \left(\mathsf{I}_{p^k} \otimes \mathsf{V}_k(x) \right) \right) \cdot \mathsf{C}_k \right) .$$

Let us write this last formula in another way by combining successive factors. Let us evaluate C_0^T. We have

$$\mathsf{I}_{p^{j-(0+1)}} \otimes \mathsf{P}_{1,0+1}^{\intercal} = \mathsf{I}_{p^{j-1}} \otimes \mathsf{P}_{1,1}^{\intercal} = \mathsf{I}_{p^{j-1}} \otimes \mathsf{I}_p = \mathsf{I}_{p^j} .$$

Also C_{j-1} is

$$\mathsf{I}_{p^{j-j}} \otimes \mathsf{P}_{1,j} = \mathsf{I}_1 \otimes \mathsf{P}_{1,j} = \mathsf{P}_{1,j} .$$

Finally, for any k such that $0 \leq k < j-1$, we have

$$\begin{aligned} \mathsf{C}_k \cdot \mathsf{C}_{k+1}^{\intercal} &= \left(\mathsf{I}_{p^{j-(k+1)}} \otimes \mathsf{P}_{1,k+1} \right) \cdot \left(\mathsf{I}_{p^{j-(k+2)}} \otimes \mathsf{P}_{1,k+2}^{\intercal} \right) \\ &= \left(\mathsf{I}_{p^{j-(k+2)}} \otimes \mathsf{I}_p \otimes \mathsf{P}_{1,k+1} \right) \cdot \left(\mathsf{I}_{p^{j-(k+2)}} \otimes \mathsf{P}_{1,k+2}^{\intercal} \right) \\ &= \mathsf{I}_{p^{j-(k+2)}} \otimes \left(\left(\mathsf{I}_p \otimes \mathsf{P}_{1,k+1} \right) \cdot \mathsf{P}_{1,k+2}^{\intercal} \right) . \end{aligned}$$

The result follows. □

A factorization of X^{-1} for fast interpolation. As explained previously, X^{-1} is the matrix of the interpolation map for which the following corollary gives a factorization into the product of sparse matrices.

Corollary 2.3.1.1. *The matrix X as defined in theorem 2.3.1 is invertible and the inverse matrix X^{-1} is given by*

$$X^{-1} = \prod_{k=0}^{j-1} \left(\bigoplus_{x \in U_{j-k}} V_{j-k-1}^{-1}(x) \right) \otimes I_{p^{j-k-1}} \text{[1]} \tag{2.17}$$

or equivalently

$$X^{-1} = \prod_{k=0}^{j-1} \left(B_{j-k-1}^{-1} \cdot \left(\bigoplus_{x \in U_{j-k}} \left(I_{p^{j-k-1}} \otimes V_{j-k-1}^{-1}(x) \right) \right) \right) \tag{2.18}$$

where

$$B_k^{-1} = \begin{cases} I_{p^{j-(k+2)}} \otimes \left(P_{1,k+2} \cdot \left(I_p \otimes P_{1,k+1}^{\mathsf{T}} \right) \right) & \text{if } 0 \leq k < j-1\,, \\ P_{1,j}^{\mathsf{T}} & \text{if } k = j-1\,. \end{cases} \tag{2.19}$$

and $P_{1,j}$, $P_{1,k+1}$, $P_{1,k+2}$ are defined in Section 2.2.

Proof. By theorem 2.3.1, we have

$$\mathsf{X} = \prod_{k=0}^{j-1} \left(\bigoplus_{x \in U_{k+1}} \mathsf{V}_k(x) \right) \otimes \mathsf{I}_{p^k}\,.$$

Therefore, we have

$$\mathsf{X}^{-1} = \prod_{k=0}^{j-1} \left(\bigoplus_{x \in U_{j-k}} \mathsf{V}_{j-k-1}^{-1}(x) \right) \otimes \mathsf{I}_{p^{j-k-1}}\,.$$

Let us now prove relation (2.18). According to theorem 2.3.1,

$$\mathsf{X} = \prod_{k=0}^{j-1} \left(\left(\bigoplus_{x \in U_{k+1}} \left(\mathsf{I}_{p^k} \otimes \mathsf{V}_k(x) \right) \right) \cdot \mathsf{B}_k \right)$$

where

$$\mathsf{B}_k = \begin{cases} \mathsf{I}_{p^{j-(k+2)}} \otimes \left((\mathsf{I}_p \otimes \mathsf{P}_{1,k+1}) \cdot \mathsf{P}_{1,k+2}^{\mathsf{T}} \right) & \text{if } \ 0 \leq k < j-1\,, \\ \mathsf{P}_{1,j} & \text{if } \ k = j-1\,. \end{cases}$$

Therefore

$$\mathsf{X}^{-1} = \left(\prod_{k=0}^{j-1} \left(\left(\bigoplus_{x \in U_{k+1}} \left(\mathsf{I}_{p^k} \otimes \mathsf{V}_k(x) \right) \right) \cdot \mathsf{B}_k \right) \right)^{-1}\,.$$

[1] There was a typo in the published version [51]. The exponent should be $j-k-1$ as corrected here.

Remembering that $(\mathsf{A}\cdot\mathsf{B})^{-1}=\mathsf{B}^{-1}\cdot\mathsf{A}^{-1}$ and $(\mathsf{A}\otimes\mathsf{B})^{-1}=\mathsf{A}^{-1}\otimes\mathsf{B}^{-1}$ we have

$$\mathsf{X}^{-1}=\prod_{k=0}^{j-1}\left(\mathsf{B}_{j-k-1}^{-1}\cdot\left(\bigoplus_{x\in U_{j-k}}\left(\mathsf{I}_{p^{j-k-1}}\otimes\mathsf{V}_{j-k-1}^{-1}(x)\right)\right)\right).$$

Let us compute B_k^{-1} for $0\leq k\leq j-1$. For any k such that $0\leq k<j-1$, we have

$$(\mathsf{B}_k)^{-1}=\left(\mathsf{I}_{p^{j-(k+2)}}\otimes\left((\mathsf{I}_p\otimes\mathsf{P}_{1,k+1})\cdot\mathsf{P}_{1,k+2}^{\intercal}\right)\right)^{-1}.$$

Noting that $(\mathsf{A}\otimes\mathsf{B})^{-1}=\mathsf{A}^{-1}\otimes\mathsf{B}^{-1}$, and $(\mathsf{A}\cdot\mathsf{B})^{-1}=\mathsf{B}^{-1}\cdot\mathsf{A}^{-1}$, $\mathsf{P}_{1,k+2}^{-1}=\mathsf{P}_{1,k+2}^{\intercal}$ this gives

$$(\mathsf{B}_k)^{-1}=\mathsf{I}_{p^{j-(k+2)}}\otimes\left(\mathsf{P}_{1,k+2}\cdot\left(\mathsf{I}_p\otimes\mathsf{P}_{1,k+1}^{\intercal}\right)\right).$$

Finally,

$$\mathsf{B}_{j-1}^{-1}=(\mathsf{P}_{1,j})^{-1}.$$

The result follows □

2.4 Fast Multipoint Evaluation and Interpolation in the LCH-basis

In this section we present algorithms for fast multipoint evaluation and interpolation of polynomials represented in the LCH-basis over $\mathbb{F}_{p^r}$. These algorithms are respectively given in theorem 2.3.1 and corollary 2.3.1.1.
We use an *algebraic complexity model*, where the running time of an algorithm is measured in terms of the number of operations in $\mathbb{F}_{p^r}$. As customary, we use the O-notation to neglect constant factors. We denote by $\mathsf{M}:\mathbb{N}\to\mathbb{N}$ a function such that polynomials in $\mathbb{F}_{p^r}[X]$ of degree at most n can be multiplied in $\mathsf{M}(n)$ operations in $\mathbb{F}_{p^r}$. Using FFT multiplication, we can take $\mathsf{M}(n)\in O(n\log_2 n\log_2 log_2 n)$. Our algorithms rely on fast interpolation and multipoint evaluation of polynomials written in the monomial basis. Using algorithms of [26, Ch. 10], both p-point evaluation and interpolation can be done in $O(\mathsf{M}(p)\cdot\log_2 p)$.

2.4.1 Fast Multipoint Evaluation Algorithm in the LCH-basis over $\mathbb{F}_{p^r}$

In what follows, we show that any sparse matrix involved in the factorized expression of the matrix X can be processed efficiently by evaluations at p points of well-defined polynomials in the monomial basis with the standard fast algorithm described above.

Consider a polynomial represented in the LCH-basis , i.e. $P_V(x) = \sum_{\omega \in V} \alpha_\omega \cdot \chi_\omega(x)$ and recall that the evaluation map at the points $x \in U + \mu_R$ is represented by the matrix $\mathsf{X} = (\chi_\omega(x))_{x \in U+\mu_R, \omega \in V}$ where U and V are totally ordered. The multipoint evaluation of $P_V(x)$ at the p^j points $x \in U + \mu_R$ therefore amounts to compute

$$(P_V(x))_{x \in U+\mu_R} = \mathsf{X} \cdot (\alpha_\omega)_{\omega \in V} \,. \tag{2.20}$$

From theorem 2.3.1, X factorizes into the product of sparse matrices which are bloc diagonal matrices for which each bloc is a Vandermonde matrix of order p. This gives

$$(P_V(x))_{x \in U+\mu_R} = \left(\prod_{k=0}^{j-1} \left(\left(\bigoplus_{x \in U_{k+1}} \left(\mathsf{I}_{p^k} \otimes \mathsf{V}_k(x) \right) \right) \cdot \mathsf{B}_k \right) \right) \cdot (\alpha_\omega)_{\omega \in V}$$

where B_k is defined as in (2.12).

Now by letting $\nu(j) = (\alpha_\omega)_{\omega \in V}$, for any $k = j-1 \ldots 0$ we have

$$\nu(k) = \left(\left(\bigoplus_{x \in U_{k+1}} \left(\mathsf{I}_{p^k} \otimes \mathsf{V}_k(x) \right) \right) \cdot \mathsf{B}_k \right) \cdot \nu(k+1) \,, \tag{2.21}$$

which gives $\nu(0) = (P_V(x))_{x \in U+\mu_R}$. Note that the bloc diagonal matrix in (2.21) is composed of $\#U_{k+1}$ groups of p^k identical Vandermonde matrices of order p.

Fast processing of the sparse matrices. Let us denote by $\nu(k+1, x)$ the part of the vector $B_k \cdot \nu(k+1)$ that will be multiplied by $\mathsf{I}_{p^k} \otimes V_k(x)$. This vector may be further divided into p^k vectors of length p that will be denoted by $\nu(k+1, x, l)$ for $0 \leq l < p^k$. Thus

$$\nu(k, x, l) = \mathsf{V}_k(x) \cdot \nu(k+1, x, l) \tag{2.22}$$

for any iteration $j > k \geq 0$, for any $x \in U_{k+1}$ and for any $0 \leq l < p^k$. Observe now that (2.22) corresponds to p-point evaluations of well-defined polynomials of degree less than p over $\mathbb{F}_{p^r}[X]$. Namely, let $f \in \mathbb{F}_{p^r}[X]$ be polynomials of degree less than p defined as

$$f(X) = \sum_{d \in \mathbb{F}_p} \nu(k+1, x, l)_d \cdot X^d. \tag{2.23}$$

Therefore, evaluating f at the p points of the set $S_{x,k} = \{L_k(x + c \cdot v_{i+k}) \mid c \in \mathbb{F}_p\}$ corresponds indeed to the matrix-vector product of (2.22). Also, for any $x \in U_{k+1}$ there is by definition an $(c_j)_j$ such that $x = \sum_{j=i+k}^{r-1} c_j \cdot v_j$ and therefore for any $k = 0, \ldots, j-1$ we have according to the linearity of the polynomial L_k that

$$S_{x,k} = \left\{ \sum_{j=i+k}^{r-1} c_j \cdot L_k(v_j) + c \cdot L_k(v_{i+k}) \mid c \in \mathbb{F}_p \right\}. \tag{2.24}$$

We have the following algorithm for the multipoint evaluation of P_V at the $n = p^j$ points $x \in U + \mu_R$.

Algorithm 16 Fast multipoint evaluation of a polynomial represented in the LCH-basis over $\mathbb{F}_{p^r}$

Require: $P_V(x) = \sum_{\omega \in V} \alpha_\omega \cdot \chi_\omega(x)$ where $V = V_i^j$ and the set $U + \mu_R$ where $U = U_i^j$. The vector basis $(v_i)_i$ of $\mathbb{F}_{p^r}$.
Ensure: $\nu(0) = (P_V(x))_{x \in U + \mu_R}$.
1: Let $\nu(j) = (\alpha_\omega)_{\omega \in V}$.
2: **for** k **from** $j-1$ **down to** 0 **do**
3: Compute $\nu(k+1) = \mathsf{B}_k \cdot \nu(k+1)$.
4: **for** any $x \in U_{k+1}$ **do**
5: Compute $S_{x,k}$ as in (2.24).
6: **for** l **from** 0 **to** $p^k - 1$ **do**
7: Let $f(X) = \sum_{d \in \mathbb{F}_p} \nu(k+1, x, l)_d \cdot X^d$.
8: Call "Standard fast multipoint evaluation" of [26, Ch. 10].
9: Input : f, the set of evaluation points $S_{x,k}$.
10: Output : A vector $\big(f(s)\big)_{s \in S_{x,k}}$.
11: Set $\nu_k = \nu_k \,||\, \big(f(s)\big)_{s \in S_{x,k}}$.
12: **end for**
13: **end for**
14: **end for**
15: **return** $\nu(0) = (P_V(x))_{x \in U + \mu_R}$.

Complexity. In what follows, $n = p^j$ represents the size of the data in terms of elements of $\mathbb{F}_{p^r}$. The sets $S_{x,k}$ of step 5 are computed recursively thanks the recursive definition (2.1) of the linearized polynomials. Computing all the sets $S_{k,x}$ costs at most $O(j \cdot p^j) = O(\log_2 n \cdot n)$ operations in $\mathbb{F}_{p^r}$. Also, noting that $\#U_{k+1} = p^{j-k-1}$, it can be seen that algorithm 16 calls the subroutine Standard fast multipoint evaluation (step 8) $j \cdot p^{j-1}$ times in total. Remembering that this subroutine is in $O(\mathsf{M}(p) \log_2 p)$, therefore step 8 requires

$$c \cdot j \cdot p^{j-1}(\mathsf{M}(p) \cdot \log_2 p) \in O(j \cdot p^{j-1} \cdot (\mathsf{M}(p) \log_2 p))$$

operations in $\mathbb{F}_{p^r}$ where c is a constant. Also, by taking $\mathsf{M}(p) \in O(p \log_2 p \log_2 \log_2 p)$ and considering the cost of step 5, we have that Algorithm 16 is in

$$O(j \cdot p^j \cdot \log_2^2 p \cdot \log_2 \log_2 p) = O(n \cdot \log_p n \cdot \log_2^2 p \cdot \log_2 \log_2 p)\,,$$

or equivalently in

$$O(n \cdot \log_2 n \cdot \log_2 p \cdot \log_2 \log_2 p)\,.$$

Remark 5. *The algorithm we just described evaluates a polynomial $P_V(x)$ of degree strictly less than p, where $V = V_i^j$ at the points of the affine space $U + \mu_R$ with $\mu_R \in \mathbb{F}_{p^r}$. Each point of this affine space can be written as $\alpha \cdot x + \beta$, $\beta \in \mathbb{F}_{p^r}$ and $x \in \mathbb{F}_p$ and which aim at evaluating $P_V(\alpha \cdot x + \beta)$. The multipoint evaluation of $P_V(x)$ we presented can be efficiently performed by computing several standard multi-evaluations of some well-defined univariate polynomials of degree strictly less than p. It is worth mentioning that this can be done differently. In the following, we describe another method for solving the multipoint evaluation/interpolation. We wish to thank the anonymous reviewer who gave us some references at the rebuttal of the submission process of ISSAC 2020 in order to improve the efficiency of the evaluation of the Vandermonde matrix/vector product.*

The first thing to do is to compute the polynomial $Q(x) = P_V(x + \beta)$. This can be done using the algorithm that can be found in [1]. It directly reduces such evaluations to a simple convolution which can be evaluated by computing a product of polynomials. Then, the polynomial $T(x) = Q(\alpha \cdot x)$ is computed as follows. If $Q(x) = \sum_i q_i \cdot x^i$, then

$$Q(\alpha \cdot x) = \sum_i q_i \cdot (\alpha \cdot x)^i = \sum_i q_i \cdot \alpha^i \cdot x^i \,.$$

Also, since

$$\sum_i q_i \cdot \alpha^i \cdot x^i = \sum_i (q_i \cdot \alpha^i) \cdot x^i \,,$$

therefore $t_i = \alpha^i \cdot q_i$. It can also be observed that $T(x) = Q(\alpha \cdot x) = P(\alpha \cdot x + \beta)$.

From this, we now wish to evaluate $T(x)$ for any $x \in \mathbb{F}_p$. This can be done by first converting $T(x)$ into a Newton basis, i.e. *, $T(x) = \sum_i t_{n_i} \cdot \mathsf{Nb}_i(x)$ where $\mathsf{Nb}_i(x)$ is the i^{th} element of the Newton basis. We only need to compute the coefficients t_{n_i}, which can be done with a single (transposed) polynomial multiplication. The Newton basis does not have to be computed.*

The evaluation of $T(x)$ considered to be represented in a Newton basis can be performed as follows. First, we compute $T(0) = t_0 = t_{n_0}$. The evaluation of $T(x)$ over $\mathbb{F}_p^*$ can then be easily done with an additional polynomial multiplication. This is due to the fact that $\mathbb{F}_p^*$ is cyclic and therefore $\mathbb{F}_p^* = \{\gamma^i \mid i = 0, \ldots, p-2\}$. This concludes the method.

This approach reduces to performing efficiently a few polynomial multiplications for which we have fast algorithms. The overall complexity of the method is expected to reduce the complexity of our original approach by a $\log_2(p)$ factor, from $O(n \cdot \log_2 n \cdot \log_2 p \cdot \log_2 \log_2 p)$ to $O(n \cdot \log_2 n \cdot \log_2 \log_2 p)$

2.4.2 Fast Interpolation Algorithm in the LCH-basis over $\mathbb{F}_{p^r}$

In what follows we show that the sparse matrices involved in the factorized expressions of the matrix representing the interpolation map can be processed efficiently by applying the standard algorithm for fast interpolation of [26, Ch. 10].

Consider the coefficients $(\alpha_\omega)_{\omega\in V}$ of $P_V(x) = \sum_{\omega\in V} \alpha_\omega \cdot \chi_\omega(x)$ from the set of evaluations $(P_V(x))_{x\in U+\mu_R}$ with $\mu_R \in U_R$. By (2.20), we have

$$(P_V(x))_{x\in U+\mu_R} = \mathsf{X} \cdot (\alpha_\omega)_{\omega\in V}\,.$$

Since X is invertible by corollary 2.3.1.1, and

$$\mathsf{X}^{-1} = \prod_{k=0}^{j-1} \left(\mathsf{B}_{j-k-1}^{-1} \cdot \left(\bigoplus_{x\in U_{j-k}} \left(\mathsf{I}_{p^{j-k-1}} \otimes \mathsf{V}_{j-k-1}^{-1}(x) \right) \right) \right)$$

where B_{j-k-1}^{-1} is defined as in (2.19), clearly

$$(\alpha_\omega)_{\omega\in V} = \mathsf{X}^{-1} \cdot (P_V(x))_{x\in U+\mu_R}\,.$$

Remembering that $\nu(j) = (\alpha_\omega)_{\omega\in V}$ and that $\nu(0) = (P_V(x))_{x\in U+\mu_R}$, therefore for any $k = 0 \dots j-1$ we have

$$\nu(k+1) = \left(\mathsf{B}_{j-k-1}^{-1} \cdot \left(\bigoplus_{x\in U_{j-k}} \left(\mathsf{I}_{p^{j-k-1}} \otimes \mathsf{V}_{j-k-1}^{-1}(x) \right) \right) \right) \cdot \nu(k) \tag{2.25}$$

which inverses (2.21). In this case, the elementary operation is

$$\mathsf{V}_k^{-1}(x) \cdot \nu(k, x, l)$$

where $\nu(k, x, l)$ denotes the p elements of $\nu(k)$ that have to be multiplied with $\mathsf{V}_{j-k-1}^{-1}(x)$ with respect to (2.25).

Fast processing of the sparse matrices. The above elementary operation corresponds to a standard fast polynomial interpolation. More precisely, by interpolating from the evaluations $\nu(k, x, l)$ and the p points of the set $\{L_k(x + c\cdot v_{i+k}) \mid c \in \mathbb{F}_p\}$, we retrieve the polynomial f as in (2.23) from which we extract the coefficients $\nu(k+1, x, l)$. By doing so for any $0 \le k < j$, for any $x \in U_{k+1}$ and for any $0 \le l < p^k$, we successively retrieve all the $\nu(k)$ and in particular $\nu(j) = (\alpha_\omega)_{\omega\in V}$. The algorithm is given below.

Remark. Step 5 would normally require to evaluate the formal derivative $M(X)'$

Algorithm 17 Fast interpolation of polynomials represented in the LCH-basis over $\mathbb{F}_{p^r}$

Require: $\nu(0) = (P_V(x))_{x \in U + \mu_R}$, the set $U + \mu_R$ (where $U = U_i^j$) and $V = V_i^j$. The vector basis $(v_i)_i$ of $\mathbb{F}_{p^r}$.
Ensure: $\nu(j) = (\alpha_\omega)_{\omega \in V}$.
1: **for** k **from** 0 **to** $j-1$ **do**
2: **for** any $x \in U_{k+1}$ **do**
3: Compute $S_{x,k}$ as in (2.24).
4: **for** l from 0 **to** $p^k - 1$ **do**
5: Call "Standard fast interpolation" of [26, Ch. 10].
6: Input : $\nu(k, x, l)$, the set of evaluation points $S_{x,k}$.
7: Output: $f(X) = \sum_{d \in \mathbb{F}_p} \nu(k+1, x, l)_d \cdot X^d$.
8: Set $\nu(k+1) = \nu(k+1) \,||\, \nu(k+1, x, l)$.
9: **end for**
10: **end for**
11: Compute $\nu(k+1) = \mathsf{B}_{j-k-1}^{-1} \cdot \nu(k+1)$.
12: **end for**
13: **return** $\nu(j) = (\alpha_\omega)_{\omega \in V}$.

of $M(X) = \prod_{k \in \mathbb{F}_p}(X - S_{x,k})$. Noting that $S_{x,k}$ is an affine set, $M(X)$ is $T(X) - T(L_k(x))$ where $T(X) = X^p - L_k(v_{i+k})^{p-1} \cdot X$. Therefore $M'(X) = T'(X) = -L_k(v_{i+k})^{p-1}$ which is a constant. Consequently $M'(X)$ does not need to be evaluated at the different points as it would be required normally.

Complexity. In what follows, $n = p^j$ represents the size of the data in terms of elements of $\mathbb{F}_{p^r}$. The cost of algorithm 17 is mainly given by its steps 3 and 5. As previously explained, the cost of step 3 is at most $O(j \cdot p^j) = O(\log_p n \cdot n)$. Step 5 consists of multiple calls to the subroutine Standard fast interpolation. This subroutine is in $O(\mathsf{M}(p) \log_2 p)$ and is called as many times as the subroutine of algorithm 16. Therefore step 5 and more globally algorithm 17 are in

$$O(n \cdot \log_2 n \cdot \log_2 p \cdot \log_2 \log_2 p)\,.$$

In-place algorithms. The multipoint evaluation (resp. interpolation) of a polynomial expressed in the LCH-basis is a combination of permutations and calls to the subroutine Standard fast multipoint evaluation (resp. Standard fast interpolation) on p points. Permutations are based on perfect shuffle permutations. As explained in [59], a perfect shuffle can be expressed as the composition of two involutions and can therefore be implemented by simply swapping elements, which are in-place operations. The subroutines of fast multipoint evaluation and interpolation on p points can also be performed in-place following [30].

2.5 Experimental Results

We implemented and ran our algorithms on an Intel Core i5 CPU at 3,2 Ghz. Our implementations are written in C using the FLINT library [38]. We compared the timings of our algorithm for fast multipoint evaluation of polynomials over $\mathbb{F}_{p^r}$ with the original method proposed in [48].

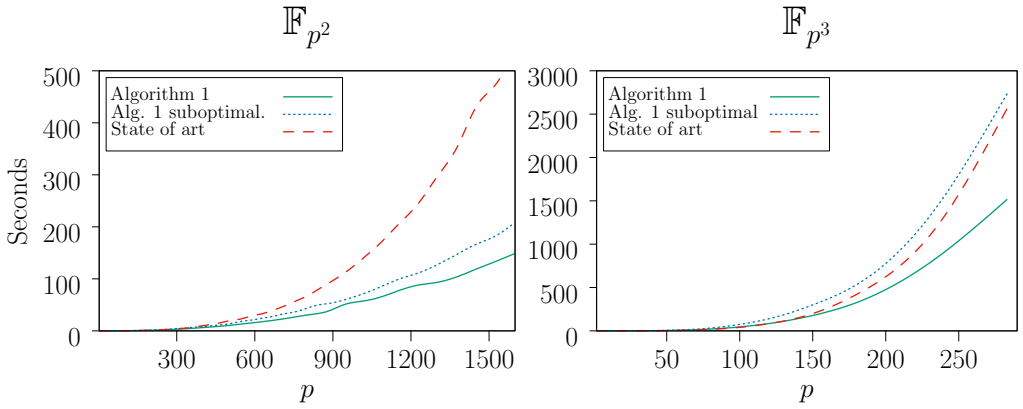


Figure 2.1: Timings for p^r-point evaluation over $\mathbb{F}_p^2$ (left) and $\mathbb{F}_p^3$ (right). Abscissa is characteristic p.

Algorithm 16 suboptimal consists in implementing formula (2.10) where the operands of the Kronecker product are commuted using a well-defined permutation. In this case, the sequence of Vandermonde matrices $V_k(x), x \in U_{k+1}$ is repeated p^k times. This lead to unnecessary memory transfers and/or recomputation of polynomials. Algorithm 16 based on formula (2.12) remedies this problem. In this case, each successive Vandermonde matrix is used p^k times in a row. Both algorithm 16 suboptimal and algorithm 16 were implemented following the classical (not in-place) version of the Standard fast multipoint evaluation and interpolation. In the state of the art [48], each single evaluation requires $O(p)$ operations while our approach performs p-point evaluations in $O\left(\mathsf{M}(p) \cdot \log_2(p)\right) = O\left(p \cdot \log_2^2 p \cdot \log_2 \log_2 p\right)$. This explains the improvement of total complexity for n-point evaluation of polynomials in the LCH-basis from $O(n \cdot \log_p n \cdot p)$ to $O(n \cdot \log_2 n \cdot \log_2 p \cdot \log_2 \log_2 p)$ as confirmed in Figure 2.1.

2.6 Conclusion

In this chapter, we tackled the problems of fast multipoint evaluation and interpolation of polynomials represented in the LCH-basis over $\mathbb{F}_{p^r}$.

We provided a fast algorithm for the multipoint evaluation problem. We reduced

such an evaluation to the problem of computing multiple multipoint evaluation of (standard) polynomials with respect to the monomial basis at p points over $\mathbb{F}_{p^r}$. By doing so, we optimized the complexity of the original method from $O(n \cdot \log_p n \cdot p)$ to $O(n \cdot \log_2 n \cdot \log_2 p \cdot \log_2 \log_2 p)$.

We also provided an algorithm for the fast interpolation problem which was left unsolved in [48] for finite fields of characteristic p. We reduced this problem to the one of computing multiple fast interpolation of (standard) polynomials with respect to the monomial basis at p points over $\mathbb{F}_{p^r}$. Our method is in $O(n \cdot \log_2 n \cdot \log_2 p \cdot \log_2 \log_2 p)$. We implemented both methods using the FLINT library and we showed that the improvement is confirmed in practice. Using permutations of the data, which are explicitely given, our algorithms satisfy high memory-locality. They can also be performed in-place.

Part V

General Conclusion and Perspectives

In this thesis, we provided new building blocks that can be used in the masking countermeasure, at any masking order. First, we provided refreshing schemes, for both the serial and the parallel cases in the revelant probing model and in the bounded moment leakage model respectively. Our serial schemes were proved to be SNI secure, the strongest security property in the probing model, for any finite Abelian group, at any masking order. In the two different contexts, our schemes improve the performances of the existing related schemes of the state of the art. Our constructions can be used for the masking at small masking orders but also for the masking at high masking orders.

Secondly, we proved the SNI property of the original GPQ masking scheme, dedicated to the masking of the AES block cipher. We extended the GPQ masking scheme to the masking of the implementations of any block ciphers. Our approach is based on existing polynomial evaluation methods over binary finite fields. The resulting extended version of GPQ we provide describes the first polynomial evaluation mixing Boolean and multiplicative masking. We also proved that it satisfies the SNI requirements. We also provided an alternate approach for the CRV polynomial evaluation method mixing our extended GPQ scheme and the commonly used ISW multiplications. The security of our alternate approach based on CRV was proved to be SNI at any masking order. We also derived new parameters for our alternate CRV polynomial evaluations, improving even more the performances in practice. This was verified for implementations in assembly language for an 8051 based 8-bit architecture at small masking orders $1, 2$ and 3. Interestingly, even if the complexity of our constructions are dominated by the quadratic complexities of the conversions, they were the fastest constructions in practice at the time for the masking orders that were implemented.

Thirdly, we improved a transform for evaluating/interpolating polynomials in the LCH-basis over finite fields of characteristic p and we also solved the interpolation problem for such a characteristic. The latter problem was left unsolved in the original works. We implemented our methods in C using the FLINT library and published the source codes on GitHub.

The following addresses different possible axes for future research.

Perspectives

Let us start with two short terms perspectives which are almost finalized. Firstly, we are writing a journal version of the work initiated in [51]. This journal version will take into account Remark 5 of Part IV, chapter 2. According to this remark,

we expect to improve the complexity of [51] by a logarithmic factor.

Secondly, we are building a secure multiplication based on [51]. This project is also being finalized.

The following presents longer terms perspectives of the work initiated by this thesis. Parallel software implementations are of interest. They offer great possibilities of efficiency improvements. However, the masking countermeasure for the parallel setting is still young. Interestingly, coupling effects due to micro-architectural features were reported in practice in a recent work [25]. These effects can lead to security issues. It would therefore be interesting to investigate these effects for specific processors and search for new ways to enhance the security of masking schemes in the parallel setting.

Also, hardware implementations are of great interest in the context of masking. Such implementations are typically masked differently than the standard masking countermeasure for software implementations. The masking schemes are usually called *threshold implementations* are were initiated in [53]. These constructions deal with additionnal physical effects called *glitches* [49] that occur in practice. Glitches greatly contribute to the overall power consumption of a running device and can also lead to a reduction of the masking order by leaving a sensitive variable not protected by its masks. It would also be interesting to investigate such constructions and conduct actual analysis in practice of the resulting masking schemes with the adequate equipments, *i.e.* an oscilloscope and FPGA or ASIC based implementations.

In any case, software and hardware implementations at small specific orders can sometimes also be improved, due to specific features of some well-chosen masking orders such as powers of two. Also, it is sometimes convenient to conduct analyses on a small range of well-chosen masking orders, for which one can derive specific and efficient methods. Namely, for small masking orders, it might be more interesting to derive quadratic countermeasures with small constants rather than quali-linear methods with large constants.

Finally, some of the implementations we did during this thesis can also be extended from 8-bit architectures to 32-bit architectures. The latter provide a large set of instructions that can lead to even better performances in practice.

Bibliography

[1] A. V. Aho, K. Steiglitz, and J. D. Ullman. Evaluating polynomials at fixed sets of points. *SIAM J. Comput.*, 4(4):533–539, 1975. doi: 10.1137/0204045. URL `https://doi.org/10.1137/0204045`.

[2] M. Akkar and C. Giraud. An implementation of DES and aes, secure against some attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, number Generators, pages 309–318, 2001. doi: 10.1007/3-540-44709-1_26. URL `https://doi.org/10.1007/3-540-44709-1_26`.

[3] R. Anderson and M. Kuhn. Tamper resistance: a cautionary note. 1996.

[4] M. Andrychowicz, S. Dziembowski, and S. Faust. Circuit compilers with o(1/\log (n)) leakage rate. In M. Fischlin and J. Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 586–615. Springer, 2016. doi: 10.1007/978-3-662-49896-5_21.

[5] J. Balasch, S. Faust, and B. Gierlichs. Inner product masking revisited. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 486–510, 2015. doi: 10.1007/978-3-662-46800-5_19. URL `http://dx.doi.org/10.1007/978-3-662-46800-5_19`.

[6] G. Barthe, S. Belaïd, F. Dupressoir, P. Fouque, and B. Grégoire. Compositional verification of higher-order masking: Application to a verifying masking compiler. *IACR Cryptology ePrint Archive*, 2015:506, 2015. URL `http://eprint.iacr.org/2015/506`.

[7] G. Barthe, S. Belaïd, F. Dupressoir, P. Fouque, B. Grégoire, P. Strub, and R. Zucchini. Strong non-interference and type-directed higher-order masking. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi,

editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 116–129. ACM, 2016. doi: 10.1145/2976749.2978427.

[8] G. Barthe, F. Dupressoir, S. Faust, B. Grégoire, F. Standaert, and P. Strub. Parallel implementations of masking schemes and the bounded moment leakage model. In J. Coron and J. B. Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - Paris, France, April 30 - May 4, 2017, Proceedings, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 535–566, 2017. doi: 10.1007/978-3-319-56620-7_19.

[9] G. Barthe, S. Belaïd, F. Dupressoir, P. Fouque, B. Grégoire, F. Standaert, and P. Strub. Improved parallel mask refreshing algorithms: generic solutions with parametrized non-interference and automated optimizations. *J. Cryptogr. Eng.*, 10(1):17–26, 2020. doi: 10.1007/s13389-018-00202-2.

[10] A. Battistello, J. Coron, E. Prouff, and R. Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In *CHES 2016 - Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, pages 23–39, 2016. doi: 10.1007/978-3-662-53140-2_2.

[11] A. Battistello, J.-S. Coron, E. Prouff, and R. Zeitoun. Horizontal side-channel attacks and countermeasures on the isw masking scheme. Cryptology ePrint Archive, Report 2016/540, 2016. `https://eprint.iacr.org/2016/540`.

[12] S. Belaïd, F. Benhamouda, A. Passelègue, E. Prouff, A. Thillard, and D. Vergnaud. Private multiplication over finite fields. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III*, pages 397–426, 2017. doi: 10.1007/978-3-319-63697-9_14. URL `https://doi.org/10.1007/978-3-319-63697-9_14`.

[13] E. R. Berlekamp. *Algebraic coding theory.* McGraw-Hill series in systems science. McGraw-Hill, 1968. ISBN 0070049033. URL `http://www.worldcat.org/oclc/00256659`.

[14] C. Carlet, L. Goubin, E. Prouff, M. Quisquater, and M. Rivain. Higher-order masking schemes for s-boxes. In *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, pages 366–384, 2012. doi: 10.1007/978-3-642-34047-5_21. URL `http://dx.doi.org/10.1007/978-3-642-34047-5_21`.

[15] C. Carlet, E. Prouff, M. Rivain, and T. Roche. Algebraic decomposition for probing security. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual*

Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I, pages 742–763, 2015. doi: 10.1007/978-3-662-47989-6_36. URL http://dx.doi.org/10.1007/978-3-662-47989-6_36.

[16] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards sound approaches to counteract power-analysis attacks. In M. J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999. doi: 10.1007/3-540-48405-1_26.

[17] M. Chen, C. Cheng, P. Kuo, W. Li, and B. Yang. Multiplying boolean polynomials with frobenius partitions in additive fast fourier transform. *CoRR*, abs/1803.11301, 2018. URL http://arxiv.org/abs/1803.11301.

[18] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Math. Comput.*, 19:297–301, 1965. ISSN 0025-5718; 1088-6842/e.

[19] J. Coron. Higher order masking of look-up tables. In *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, pages 441–458, 2014. doi: 10.1007/978-3-642-55220-5_25. URL http://dx.doi.org/10.1007/978-3-642-55220-5_25.

[20] J. Coron, E. Prouff, M. Rivain, and T. Roche. Higher-order side channel security and mask refreshing. In *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, pages 410–424, 2013. doi: 10.1007/978-3-662-43933-3_21. URL https://doi.org/10.1007/978-3-662-43933-3_21.

[21] J. Coron, A. Roy, and S. Vivek. Fast evaluation of polynomials over binary finite fields and application to side-channel countermeasures. In *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, pages 170–187, 2014. doi: 10.1007/978-3-662-44709-3_10. URL http://dx.doi.org/10.1007/978-3-662-44709-3_10.

[22] M. Davio. Kronecker products and shuffle algebra. *"IEEE Transactions on Computers"*, 30(2):116–125, 1981. doi: 10.1109/TC.1981.1675863. URL https://doi.org/10.1109/TC.1981.1675863.

[23] A. Duc, S. Dziembowski, and S. Faust. Unifying leakage models: From probing attacks to noisy leakage. In P. Q. Nguyen and E. Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 423–440. Springer, 2014. doi: 10.1007/978-3-642-55220-5_24.

[24] S. Gao, B. Marshall, D. Page, and E. Oswald. Share-slicing: Friend or foe? *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):152–174, 2020. doi: 10.13154/tches.v2020.i1.152-174. URL `https://doi.org/10.13154/tches.v2020.i1.152-174`.

[25] S. Gao, B. Marshall, D. Page, and E. Oswald. Share-slicing: Friend or foe? *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):152–174, 2020. doi: 10.13154/tches.v2020.i1.152-174. URL `https://doi.org/10.13154/tches.v2020.i1.152-174`.

[26] J. V. Z. Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 2 edition, 2003. ISBN 0521826462.

[27] L. Genelle, E. Prouff, and M. Quisquater. Secure multiplicative masking of power functions. In *Applied Cryptography and Network Security, 8th International Conference, ACNS 2010, Beijing, China, June 22-25, 2010. Proceedings*, pages 200–217, 2010. doi: 10.1007/978-3-642-13708-2_13. URL `http://dx.doi.org/10.1007/978-3-642-13708-2_13`.

[28] L. Genelle, E. Prouff, and M. Quisquater. Montgomery's trick and fast implementation of masked AES. In *Progress in Cryptology - AFRICACRYPT 2011 - 4th International Conference on Cryptology in Africa, Dakar, Senegal, July 5-7, 2011. Proceedings*, pages 153–169, 2011. doi: 10.1007/978-3-642-21969-6_10. URL `http://dx.doi.org/10.1007/978-3-642-21969-6_10`.

[29] L. Genelle, E. Prouff, and M. Quisquater. Thwarting higher-order side channel analysis with additive and multiplicative maskings. In *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, pages 240–255, 2011. doi: 10.1007/978-3-642-23951-9_16. URL `http://dx.doi.org/10.1007/978-3-642-23951-9_16`.

[30] P. Giorgi, B. Grenet, and D. S. Roche. Fast in-place algorithms for polynomial operations: division, evaluation, interpolation. In I. Z. Emiris and L. Zhi, ed-

itors, *ISSAC '20: International Symposium on Symbolic and Algebraic Computation, Kalamata, Greece, July 20-23, 2020*, pages 210–217. ACM, 2020. doi: 10.1145/3373207.3404061. URL https://doi.org/10.1145/3373207.3404061.

[31] J. D. Golic and C. Tymen. Multiplicative masking and power analysis of AES. In B. S. K. Jr., Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 198–212. Springer, 2002. doi: 10.1007/3-540-36400-5_16. URL https://doi.org/10.1007/3-540-36400-5_16.

[32] L. Goubin and A. Martinelli. Protecting AES with shamir's secret sharing scheme. In *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, pages 79–94, 2011. doi: 10.1007/978-3-642-23951-9_6. URL http://dx.doi.org/10.1007/978-3-642-23951-9_6.

[33] L. Goubin and J. Patarin. DES and differential power analysis (the "duplication" method). In Ç. K. Koç and C. Paar, editors, *CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999. doi: 10.1007/3-540-48059-5_15.

[34] D. Goudarzi and M. Rivain. How fast can higher-order masking be in software? In *Advances in Cryptology - EUROCRYPT 2017 - Paris, France, April 30 - May 4, 2017, Proceedings, Part I*, pages 567–597, 2017. doi: 10.1007/978-3-319-56620-7_20.

[35] D. Goudarzi, A. Joux, and M. Rivain. How to securely compute with noisy leakage in quasilinear complexity. In T. Peyrin and S. D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 547–574. Springer, 2018. doi: 10.1007/978-3-030-03329-3_19. URL https://doi.org/10.1007/978-3-030-03329-3_19.

[36] D. Goudarzi, T. Prest, M. Rivain, and D. Vergnaud. Probing security through input-output separation and revisited quasilinear masking. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):599–640, 2021. doi: 10.46586/tches.v2021.i3.599-640. URL https://doi.org/10.46586/tches.v2021.i3.599-640.

[37] V. Grosso, F. Standaert, and S. Faust. Masking vs. multiparty computation: how large is the gap for aes? *J. Cryptographic Engineering*, 4(1):47–57, 2014. doi: 10.1007/s13389-014-0073-y. URL `http://dx.doi.org/10.1007/s13389-014-0073-y`.

[38] W. Hart, F. Johansson, and S. Pancratz. FLINT: Fast Library for Number Theory, 2015. Version 2.5.2, `http://flintlib.org`.

[39] L. Hogben. *Handbook of linear algebra; 2nd ed.* Discrete Mathematics and Its Applications. Taylor and Francis, Hoboken, NJ, 2013. URL `https://cds.cern.ch/record/1641653`.

[40] Y. Ishai, A. Sahai, and D. A. Wagner. Private circuits: Securing hardware against probing attacks. In D. Boneh, editor, *Advances in Cryptology - CRYPTO 2003, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003. doi: 10.1007/978-3-540-45146-4_27.

[41] A. Journault and F. Standaert. Very high order masking: Efficient implementation and security evaluation. In *Cryptographic Hardware and Embedded Systems - CHES 2017 - Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 623–643, 2017. doi: 10.1007/978-3-319-66787-4_30.

[42] P. C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In N. Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996. doi: 10.1007/3-540-68697-5_9. URL `https://doi.org/10.1007/3-540-68697-5\_9`.

[43] P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999. doi: 10.1007/3-540-48405-1_25.

[44] R. Li, Q. Huang, and Z. Wang. Encoding of non-binary quasi-cyclic codes by lin-chung-han transform. In *IEEE Information Theory Workshop, ITW 2018, Guangzhou, China, November 25-29, 2018*, pages 1–5, 2018. doi: 10.1109/ITW.2018.8613313. URL `https://doi.org/10.1109/ITW.2018.8613313`.

[45] S. Lin, W. Chung, and Y. S. Han. Novel polynomial basis and its application to reed-solomon erasure codes. In *55th IEEE Annual Symposium on*

Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014, pages 316–325, 2014. doi: 10.1109/FOCS.2014.41. URL https://doi.org/10.1109/FOCS.2014.41.

[46] S. Lin, T. Y. Al-Naffouri, and Y. S. Han. Efficient frequency-domain decoding algorithms for reed-solomon codes. *CoRR*, abs/1503.05761, 2015. URL http://arxiv.org/abs/1503.05761.

[47] S. Lin, T. Y. Al-Naffouri, and Y. S. Han. FFT algorithm for binary extension finite fields and its application to reed-solomon codes. *IEEE Trans. Information Theory*, 62(10):5343–5358, 2016. doi: 10.1109/TIT.2016.2600417. URL https://doi.org/10.1109/TIT.2016.2600417.

[48] S. Lin, T. Y. Al-Naffouri, Y. S. Han, and W. Chung. Novel polynomial basis with fast fourier transform and its application to reed-solomon erasure codes. *IEEE Trans. Information Theory*, 62(11):6284–6299, 2016. doi: 10.1109/TIT.2016.2608892. URL https://doi.org/10.1109/TIT.2016.2608892.

[49] S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*. Springer-Verlag, Berlin, Heidelberg, 2007. ISBN 0387308571.

[50] A. Mathieu-Mahias and M. Quisquater. Mixing additive and multiplicative masking for probing secure polynomial evaluation methods. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):175–208, 2018. doi: 10.13154/tches.v2018.i1.175-208. URL https://doi.org/10.13154/tches.v2018.i1.175-208.

[51] A. Mathieu-Mahias and M. Quisquater. Fast multipoint evaluation and interpolation of polynomials in the lch-basis over $f_p{}^r$. In I. Z. Emiris and L. Zhi, editors, *ISSAC '20: International Symposium on Symbolic and Algebraic Computation, Kalamata, Greece, July 20-23, 2020*, pages 344–351. ACM, 2020. doi: 10.1145/3373207.3404009. URL https://doi.org/10.1145/3373207.3404009.

[52] A. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996. ISBN 0-8493-8523-7. doi: 10.1201/9781439821916. URL http://cacr.uwaterloo.ca/hac/.

[53] S. Nikova, C. Rechberger, and V. Rijmen. Threshold implementations against side-channel attacks and glitches. In P. Ning, S. Qing, and N. Li, editors, *Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings*, volume

4307 of *Lecture Notes in Computer Science*, pages 529–545. Springer, 2006. doi: 10.1007/11935308_38. URL https://doi.org/10.1007/11935308_38.

[54] E. Prouff and M. Rivain. Masking against side-channel attacks: A formal security proof. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 142–159, 2013. doi: 10.1007/978-3-642-38348-9_9. URL http://dx.doi.org/10.1007/978-3-642-38348-9_9.

[55] E. Prouff and T. Roche. Higher-order glitches free implementation of the AES using secure multi-party computation protocols. In *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, pages 63–78, 2011. doi: 10.1007/978-3-642-23951-9_5. URL http://dx.doi.org/10.1007/978-3-642-23951-9_5.

[56] M. Rivain and E. Prouff. Provably secure higher-order masking of AES. In S. Mangard and F. Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 2010. doi: 10.1007/978-3-642-15031-9_28. URL https://doi.org/10.1007/978-3-642-15031-9_28.

[57] A. Roy and S. Vivek. Analysis and improvement of the generic higher-order masking scheme of FSE 2012. In *Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings*, pages 417–434, 2013. doi: 10.1007/978-3-642-40349-1_24. URL http://dx.doi.org/10.1007/978-3-642-40349-1_24.

[58] J. van der Hoeven and R. Larrieu. The frobenius FFT. In *Proceedings of the 2017 ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2017, Kaiserslautern, Germany, July 25-28, 2017*, pages 437–444, 2017. doi: 10.1145/3087604.3087633. URL https://doi.org/10.1145/3087604.3087633.

[59] Q. Yang, J. Ellis, K. Mamakani, and F. Ruskey. In-place permuting and perfect shuffling using involutions. *Information Processing Letters*, 113 (10):386 – 391, 2013. ISSN 0020-0190. doi: https://doi.org/10.1016/j.

ipl.2013.02.017. URL http://www.sciencedirect.com/science/article/pii/S0020019013000719.

Résumé : De nos jours, les systèmes embarqués sont omniprésents et leurs applications sont de plus en plus nombreuses. La plupart des domaines industriels d'aujourd'hui dépendent des systèmes embarqués pour l'accomplissement de tâches spécifiques, parfois hautement sensibles. Actuellement, l'usage de systèmes embarqués est encore davantage accentué par la mise en place de "l'Internet des Objets", véritable révolution digitale. Un système embarqué est un système électronique et informatique contrôlant une partie spécifique d'un système plus large. De nombreuses contraintes, notamment liées à sa taille, doivent être prises en compte lors de sa conception. Il en résulte qu'un système embarqué est généralement à bas coûts, consomme peu de courant et dispose la plupart du temps d'une puissance de calculs relativement restreinte. Pour l'accomplissement de ses tâches spécifiques, un système embarqué collecte, manipule et échange des données parfois sensibles. Par ailleurs, un tel système est souvent directement accessible physiquement. Cette particularité peut alors être exploitée par une personne malintentionnée pour contrôler, extraire ou encore altérer les données sensibles manipulées par de tels systèmes. Dans un tel contexte, la mise en place de mécanismes de sécurité adaptés est primordiale. En particulier, il est crucial de sécuriser l'accès physique à un système embarqué, mais également de protéger les données sensibles manipulées ou stockées par le matériel. La cryptographie ou science du secret offre de nombreuses possibilités pour parvenir à sécuriser les données manipulées par un système embarqué. Cependant, dans ce contexte précis, certaines caractéristiques physiques liées à l'électronique des systèmes embarqués varient à l'exécution des implémentations d'algorithmes cryptographiques, garants de la sécurité de l'information. En particulier, la consommation de courant de l'appareil ou encore ses émanations électromagnétiques dépendent des données manipulées ainsi que des choix faits à l'implémentation. Ces caractéristiques physiques peuvent en outre être mesurées si l'accès physique à l'appareil est possible. L'exploitation de ces mesures a mené à des attaques dévastatrices communément appelées « attaques par canaux auxiliaires ». La mise en œuvre de ce type d'attaque permet d'extraire les données secrètes stockées ou manipulées par un appareil électronique, souvent sans grands efforts. Des contre-mesures particulières doivent donc être mises en place pour garantir la sécurité des implémentations d'algorithmes cryptographiques sans trop dégrader leurs performances à l'exécution. Le masquage est une solution largement déployée de nos jours, mais sa mise en œuvre correcte et efficace nécessite une analyse fine des solutions

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre: Sécurisation des implémentations d'algorithmes cryptographiques pour les systèmes embarqués

Mots clés: Masquage, Implémentations logicielles, Chiffrements par blocs, Systèmes embarqués, Attaques par canaux auxiliaires.

Résumé: De nos jours, les systèmes embarqués sont omniprésents. Ils trouvent des applications dans tous les domaines industriels et leur déploiement est encore davantage accentué par la mise en place de "l'Internet des Objets", véritable révolution digitale. Ces systèmes collectent, manipulent et échangent des données, parfois sensibles, opèrent souvent dans des environnements à risques et sont généralement physiquement accessibles par une personne malintentionnée, ce qui nécessite la mise en place de mécanismes de sécurité adaptés garantissant la protection de l'information. La cryptographie, ou science du secret, offre de nombreuses possibilités pour parvenir à securiser des appareils éléctroniques contrôlés par des systèmes embarqués. Cependant, ce contexte précis à une particularité. L'exécution d'implémentations d'algorithmes cryptographiques est fortement liée à l'électronique embarquée, menant à des attaques dévastatrices, exploitant un accès physique direct au matériel, permettant de retrouver facilement les données secrètes manipulées par le système. Des contre-mesures particulières doivent être mises en place pour pallier à ces problèmes. Le masquage est une solution largement déployée de nos jours, mais sa mise en œuvre correcte nécessite une analyse fine des solutions algorithmiques qu'elle propose, particulièrement dans un contexte où les ressources matérielles sont limitées.

Title: Securisation of implementations of cryptographic algorithms in the context of embedded systems

Keywords: Making, Software implementations, Block ciphers, Embedded Systems, Side-Channel Analysis.

Abstract: Embedded systems are ubiquitous. They find applications in all industrial fields and their deployment is even more accelerated by the birth of the "Internet of Things", which is expected to revolutionize our digital world. These systems collect, manipulate and exchange information, sometimes sensitive, in highly critical environments and are usually physically accessible by an unauthorized entity. This requires to develop and implement well-suited security mecanisms, guaranteing the security of information. Cryptography is the science of secrets and offers numerous ways to mitigate the risks that face electronic devices controlled by embedded systems. However, in such a context, the execution of the implementations of cryptographic algorithms is tied to the embedded electronics, leading to devastating attacks exploiting a direct physical access to the device, allowing the disclosure of the secret information manipulated by the system. Special countermeasures have to be implemented to mitigate these issues. Masking is a well-known solution, but its correct implementation requires a thorough analysis of algorithmic solutions it provides, especially in a context where devices have limited resources.

Maison du doctorat de l'Université Paris-Saclay
2ème étage aile ouest, Ecole normale supérieure Paris-Saclay
4 avenue des Sciences,
91190 Gif sur Yvette, France