Secure and Verified Cryptographic Implementations in the Random Probing Model

Abdul Rahman Taleb

SORBONNE UNIVERSITÉ

CRYPTOEXPERTS
WE INNOVATE TO SECURE YOUR BUSINESS

PH.D. THESIS

Secure and Verified Cryptographic Implementations in the Random Probing Model

Thèse de doctorat présentée et soutenue publiquement le 7 novembre 2023 par

Abdul Rahman Taleb

pour obtenir le grade de

Docteur de Sorbonne Université

École Doctorale Informatique, Télécommunications et Électronique de Paris - ED 130

Spécialité: Informatique

Jury

Président du Jury :	François-Xavier STANDAERT	Université catholique de Louvain-la-Neuve, Belgique
Directeur de thèse :	Damien VERGNAUD	Sorbonne Université, France
Encadrants Industriels :	Matthieu RIVAIN	CryptoExperts, France
	Sonia BELAÏD	CryptoExperts, France
Rapporteurs :	Sebastian FAUST	Université de technologie de Darmstadt, Allemagne
	Svetla NIKOVA	Université catholique de Leuven, Belgique
Examinateurs :	Benjamin GRÉGOIRE	Inria Sophia Antipolis, France
	François-Xavier STANDAERT	Université catholique de Louvain-la-Neuve, Belgique
	Rina ZEITOUN	IDEMIA, France

Abstract

The masking countermeasure is among the most potent countermeasures to counteract side-channel attacks. Leakage models have been exhibited to theoretically reason on the security of such masked implementations. So far, the most widely used leakage model is the probing model, but it has been recently challenged as it does not fully capture the capabilities of a side-channel adversary. To capture a broader class of attacks, another model was introduced, referred to as the random probing model. From a leakage parameter p, each wire of the circuit leaks its value with probability p. The random probing model enjoys practical relevance thanks to a reduction to the noisy leakage model, which is admitted as the suitable formalization for power and electromagnetic side-channel attacks. In addition, the random probing model is much more convenient than the noisy leakage model to prove the security of masking schemes.

In this thesis, we study more closely the random probing model and define the first framework dedicated to it. We formalize a composition property for secure random probing gadgets and exhibit its relation to the strong non-interference (SNI) notion used in probing security. We then revisit the expansion idea proposed by Ananth, Ishai, and Sahai (CRYPTO 2018) and introduce a compiler that builds a random probing secure circuit from small base gadgets, achieving a random probing expandability (RPE) property. We then provide an in-depth analysis of the RPE security notion, allowing us to obtain much more efficient instantiations of the expansion technique, with constructions tolerating a leakage probability of up to 2^{-7}, against 2^{-26} for the previous construction and an improved complexity of $\mathcal{O}(\kappa^{3.2})$ against $\mathcal{O}(\kappa^{7.87})$ for the previous constructions, where κ is the security parameter. We also show that our constructions achieve a quadratic complexity in κ asymptotically as the number of shares grows. Further attempts to optimize constructions include generalizing the RPE approach by considering a dynamic choice of the base gadgets at each step in the expansion. We show that such techniques can further reduce the complexity from quadratic to quasi-linear while tolerating good leakage rates.

Finally, we introduce IronMask, a new versatile verification tool for masking security. IronMask is the first to verify standard simulation-based security notions in the probing model and recent notions in the random probing model. It supports any masking gadgets with linear randomness (e.g., addition, copy, and refresh gadgets) as well as quadratic gadgets (e.g., multiplication gadgets) that might include non-linear randomness (e.g., by refreshing their inputs) while providing complete verification results for both types of gadgets.

We conclude this thesis by discussing other research projects in the random probing model and suggestions for future works.

Résumé

La contre-mesure de masquage fait partie des contre-mesures les plus puissantes contre les attaques par canaux auxiliaires. Des modèles de fuite ont été introduits pour raisonner théoriquement sur la sécurité de telles implémentations masquées. Jusqu'à présent, le modèle de fuite le plus largement utilisé est le modèle nommé "probing", mais il a été récemment remis en question car il ne rend pas pleinement compte des capacités d'un adversaire par canaux auxiliaires. Ainsi, un autre modèle a été introduit, appelé "random probing". Dans ce modèle, à partir d'un paramètre de fuite p, chaque fil du circuit fuit sa valeur avec probabilité p. Le modèle random probing est pratique grâce à sa réduction au modèle "noisy leakage", qui est admis comme la formalisation appropriée pour la fuite pendant les attaques par canaux auxiliaires. De plus, le modèle random probing est bien plus pratique que le modèle noisy leakage pour prouver la sécurité des implémentations masquées.

Dans cette thèse, nous étudions de plus près le modèle random probing et définissons le premier cadre qui lui est dédié. Nous formalisons une propriété de composition pour les gadgets sécurisés dans ce modèle et montrons sa relation avec la notion de non-interférence forte (SNI) utilisée dans le modèle de sécurité "probing". Nous revisitons ensuite l'idée d'expansion proposée par Ananth, Ishai et Sahai (CRYPTO 2018) et introduisons un compilateur qui construit un circuit sécurisé dans le modèle random probing à partir de petits gadgets de base. Nous proposons ensuite une analyse approfondie, permettant d'obtenir des instanciations beaucoup plus efficaces de la technique d'expansion, avec des constructions tolérant une probabilité de fuite allant jusqu'à 2^{-7}, contre 2^{-26} pour les constructions précédentes et une complexité améliorée de $\mathcal{O}(\kappa^{3.2})$ contre $\mathcal{O}(\kappa^{7.87})$ pour les constructions précédentes, où *kappa* est le paramètre de sécurité. Nous montrons également que nos constructions atteignent une complexité quadratique en κ asymptotiquement en augmentant le nombre de partages pendant le masquage. D'autres tentatives d'optimisation des constructions incluent la généralisation de l'approche en considérant un choix dynamique des gadgets de base à chaque étape de l'expansion. Nous montrons que de telles techniques peuvent réduire davantage la complexité du quadratique au quasi-linéaire tout en tolérant de bons taux de fuite.

Enfin, nous présentons IronMask, un nouvel outil de vérification polyvalent pour la sécurité des circuits masqués. IronMask est le premier à vérifier les notions de sécurité standard basées sur la simulation dans le modèle probing et les notions récentes du modèle random probing. Il prend en charge tous les gadgets masqués qui utilisent des variables aléatoires de façon linéaire (par exemple, les gadgets d'addition et de copie) ainsi que les gadgets quadratiques (par exemple, les gadgets de multiplication) qui peuvent inclure des variables aléatoires non linéaires, tout en fournissant des résultats de vérification complets. pour les deux types de gadgets.

Nous concluons cette thèse en discutant d'autres projets de recherche dans le modèle random probing et en suggérant des travaux futurs.

Acknowledgements

It is a privilege in this world to be able to dream, to laugh, to love. It is a privilege to sleep in peace without worrying about when all of it could vanish. It is a privilege to have an ambition, a goal, and people who love and support you along the road. I am privileged to have had the opportunity to pursue and finish a PhD, and I couldn't have done it alone.

First, I would like to thank Sebastian Faust and Svetla Nikova for reviewing my manuscript. I would also like to thank Benjamin Grégoire, François-Xavier Standaert, and Rina Zeitoun for accepting to be the examiners of my jury.

Then, I would like to thank my PhD director, Damien Vergnaud, who started as my teacher and then my internship advisor before allowing me to do a PhD and guiding me to CryptoExperts. I owe him this success, and I sincerely appreciate his humane and fatherly side, in addition to the professionalism that I got to see.

I would also like to thank my advisors at CryptoExperts, Matthieu Rivain, and Sonia Belaïd, who have taught me so much and trusted me with many projects during the past three and a half years. It always felt like collaborative work with them, not an advisor-student relationship, which made everything more efficient and productive. I would also like to thank them for their emotional support. I thank Sonia for her encouraging messages after every accomplishment or supportive messages whenever things are slightly derailed. I thank Matthieu for never doubting me for a second. I can't help but think of the amount of trust he gave me during the CRY.ME project, which I valued very much. But I also thank both of them for the exciting scientific and social discussions we always had and the fun we shared during many team buildings (It's not every day that you get to do a karaoké with your supervisors!).

Next, I would like to thank my past and present colleagues at CryptoExperts, with whom I shared lunches, bars, or fun moments at team buildings. We taught each other about our different cultures, which always made a place for interesting discussions. I think of Ryad Benadjila, Nicolas Bon, Thibauld Feneuil, Bruno Gonzalez, Louis Goubin, Darius Mercadier, Viet-Sang Nguyen, Pascal Paillier, Ronan Thoraval, Matthias Trannoy, Aleksei Udovenko, Junwei Wang (and of course, Matthieu and Sonia).

I am lucky to have had two teams of colleagues, the ones at CryptoExperts and the ones at ALMASTY and LIP6. I had exceptional moments with my ALMASTY team, enjoying board games, conferences, retreats, and much more. Of course, I thank Damien Vergnaud as my director. But I also thank Ahmed Khulaif Alharbi, whom I met at the end of my Ph.D. and who has already shown me so much kindness. I thank Florette Martinez for her help whenever I needed it, Ambroise Fleury for his adventurous spirit, and Charles Bouillaget for the insightful discussions. I thank Mickaël Hamdad and Julia Sauvage, whom I first met as students before becoming ALMASTY colleagues, which is a transition that I much appreciated. I thank Andersson Calle Viera, with whom I shared terrific moments at conferences and a common interest in the side-channel field. I thank Orel Cosseron for the shared culinary passion we have. Finally, I thank Samuel Bouaziz–Ermann and Jules Maire for the second-degree jokes,

lots of second-degree jokes. Jules, although not always fruitful, your attempt at making jokes is much appreciated. I finally thank Lucas Ottow for the fun moments during Journées C2.

Moreover, I am thankful to Camille Mutschler, a fellow PhD student I met at the beginning of my journey, and we instantly clicked. Having started working in the side-channel field simultaneously, we have had the opportunity to share many conferences, collaborate on exciting projects, and share memorable moments as friends, not just colleagues. It is rare to find lasting friendships, and I am grateful for this one.

My research is the fruit of collaboration with researchers I was grateful to meet throughout my PhD. I sincerely thank François-Xavier Standaert again for hosting me in his lab and teaching me many things about the practical side of side-channel attacks. Working with him was a great pleasure. I am also thankful to Gaëtan Cassiers for our collaborations and for helping me understand how to handle side-channel acquisition tools. I then express my gratitude to the ninjas from NinjaLab for our collaborations and their insightful help. I thank Thomas Roche for never hesitating to share his expertise in the field and for accepting to work with me on some projects. I also thank Victor Lomné for our discussions during projects and conferences. Indeed, I thank Camille again for our collaboration.

In addition, I thank Karine Heydemann and Quentin Meunier for helping me with my research projects. I thank Quentin Meunier for allowing me to work with him and for his expertise in the field. I thank him for his constant interest in my work and for his exciting questions that always allowed me to understand better what I do. I am delighted to have had successful collaborations with him.

I also thank all the researchers and PhD students I got to discuss or spend memorable moments with at conferences and seminars. I thank Agathe Beaugrand, Jean-Sébastien Coron, Marc Gourjon, Loïc Masure, Charles Momin, Maximilian Orlt, Emmanuel Prouff, Jürgen Pulkus, Rafael Carrera Rodriguez, Lorenzo Casalino, and all the researchers I got to speak to even for a brief moment.

The CRY.ME project was a considerable part of my journey at CryptoExperts. Hence, I would like to thank the team from ANSSI for this opportunity. I thank Chrysanthi Mavromati for managing the project and offering me future opportunities. I additionally thank Jérémy Jean, Louiza Khati, Ange Martinelli, and my teammates from CryptoExperts Matthieu, Ryad, Sonia, and Thibauld.

I already had experience in the research field before my Ph.D. I thank Alice Othmani for my first-ever scientific publication, which has already taught me so much about how to write good papers. I also thank Professor Noboru Kunihiro for being my supervisor during my exchange program in Tokyo, which was my first research experience in the side-channel field. I thank him for his interest in my PhD work and inviting me to talk in his seminars and workshops.

I couldn't have achieved anything without some close people constantly being by my side. My mother, sister, and two brothers, I love you, and thank you for being by my side. Through the ups and downs, we have always made the best out of it. Knowing someone is always there to catch me if I fall is comforting. I cherish our moments together and hope that we stay united no matter what. My father, I dedicate all this success and hard work to your soul. You have looked out for us and helped us get to where we are now. I hoped you would see me now, but life had other plans. You are always with us and in our hearts.

I am thankful for my long-lasting friendship with Talal Afyouni. I am blessed to have someone in my life whom I don't have to fake any feeling around, who sees my angriness, happiness, sadness, and craziness and still takes the whole package. It was relieving that I

could be myself around him during my moments of defeat or despair, where I had to force a smile around others. We have grown together, learned together, and were reunited again in another country, only to prove that we are meant to last. I will always be short of words to describe our relationship, so I can only say thank you.

I am also profoundly thankful to Arij Riabi. We met during our bachelor's but quickly bonded and developed a solid, hard-to-break friendship. Our shared struggle of studying abroad only made us stronger. Our bad jokes that only we find funny are one of our pillars. We are different yet have much in common, and our mutual trust and feelings are indestructible. It is one of those friendships you know will never fade away, the ones you live and thrive for. For that, I am always indebted.

I thank Quentin Bernier for always being there and looking out for me. I thank him for putting up with me whenever I struggled. I also thank him for our pleasant moments and shared interests. I am happy to have another good one in my circle and wish that our bond only grows stronger. Quentin, as I unintentionally stole your thunder by having my defense on your birthday, I wish you a very happy birthday and a year full of success.

I also thank Ivana and François Garlot for being good friends and sharing great evenings with me. It was my means of escape from work and anxiety on several occasions.

Finally, I thank Snookie for being an extremely annoying and adorable cat.

Contents

Chapter 1

Introduction

1.1 Cryptography Then and Now

We all have secrets, and we all look for privacy. This constant desire for protection gave birth to cryptography, the art of message encryption. Cryptography protects information and communications through codes, hiding a message from anyone who is not supposed to read it. It existed way before the emergence of computers. The ancient Greeks, for instance, are believed to have used the scytale as a method of transposition cipher during military campaigns. The scytale is a cylinder with a strip of parchment wound around it. To encrypt a message into a ciphered one, one can write it on the leather piece wrapped around the rod and then unwrap it. To decrypt the ciphered message (*i.e.* retrieve the original message), one needs the rod with the correct diameter to wind the strip and read the message. If an incompatible cylinder is used, the message appears as a set of randomly chosen letters. As simple as the technique may seem, it proves that humans have always looked for ways to secure communications.

Like everything else, cryptography has evolved into a much broader field, especially with the rise of technology and modern computers. In the digital world, modern cryptography is a branch of applied mathematics and computer science. It implements encryption algorithms as programs relying on mathematical tools. These algorithms should prevent attackers and eavesdroppers from breaching the privacy of communications. Modern cryptographic algorithms and protocols follow Kerckhoff's law by being public, with a secret parameter called the key being the only secret element. Essentially, anyone with access to this secret key is authorized to participate and read the communication, while it is computationally impossible for someone without the key to do the same. The computational impossibility is a widely admitted principle in the conception of modern cryptography. This impossibility is because cryptosystems rely on mathematical problems that are difficult to solve in a reasonable time without access to the secret parameter. In this case, one can only hope to break such a cryptosystem using an exhaustive key search (or brute force attack), which comes with exponential complexity in the security parameters and becomes quickly impossible to use.

Nowadays, cryptography aims at providing confidentiality, integrity, and authentication. Confidentiality means no one can read private messages except for authorized parties, while integrity ensures that unauthorized parties can not modify the data. Finally, authentication allows the receiver to check the identity of the sender of a message. In addition to all of these properties, digital signatures of electronic documents can ensure non-repudiation like a handwritten signature: the author of a signature cannot deny it afterward. With the development of modern cryptography, many other security features were rendered possible such as secure multiparty computations, group signatures, etc.

1.2 From Black-Box to Gray-Box Security

Cryptographic primitives are considered secure based on the hardness of the mathematical problems they use. This security is considered in what we call the *black-box* setting, where an adversary only has access to the inputs and outputs of the algorithm. However, these algorithms are implemented on physical devices vulnerable to *side-channel attacks* as revealed in the late nineties [63]. These attacks allow an adversary to measure physical quantities emanating from the device, revealing information about sensitive data in many cases. Such observations are made in the *gray-box* setting and can be the execution time, device temperature, power consumption, or electromagnetic radiations during the algorithm execution. In the gray-box model, one must analyze the implementation of a cryptosystem on the considered device, including data representation, function calls, and data manipulation.

Attacking the physical implementation of cryptosystems has gained much popularity since the discovery of side-channel attacks. More generally, we identify a family of physical attacks ranging from passive attacks, such as side-channel attacks as mentioned earlier, and more active attacks, such as fault attacks, where an attacker can tamper with the physical device and inject faults to alter the execution of the algorithm, potentially resulting in the reveal of secret information.

1.3 Masking Against Side-Channel Attacks

Since the discovery of side-channel attacks, several countermeasures have been studied to protect cryptographic algorithms. Among the different approaches, one of the most widely used is known as *masking*, simultaneously introduced by Chari, Jutla, Rao, and Rohatgi [35], and by Goubin and Patarin [52] in 1999. It consists in splitting a sensitive variable x into n random *shares*, among which any combination of $n-1$ shares does not reveal any secret information. This can be achieved by generating $n-1$ shares uniformly at random $x_1, \ldots, x_{n-1}$ and computing the last share x_n so that $x = x_1 * \ldots * x_{n-1} * x_n$ according to some group law $*$. We call n the number of shares and $n-1$ the *masking order*. When working on $\mathbb{F}_2$, we refer to the masking as Boolean masking and use the operation $+$. The masking order refers to the value $n-1$. The motivation of masking is to make it more difficult for an attacker to recover a secret by manipulating the shares instead of the sensitive value. Since each observation comes with noise, increasing the number of shares makes it harder to recover the secret, under some hypotheses on the leaking device. We optimally would like the security level to increase as we increase the number of shares n. Meanwhile, proving or validating such security levels in practice is not trivial.

When manipulating variables as n shares, one needs to adapt operations to preserve the security under the masking scheme while avoiding recombinations of shares. Indeed, such recombinations can sometimes lead to more revealed information about the secrets. Let us take an example of a simple addition $+$ operation over some group and n-shared variables $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$. Originally, to get the result of the addition, one performs $x + y$. An attacker observing side-channel measurements then tries to target the moment the value $x + y$ is computed during the leakage trace. When dealing with the shared variables, we can compute the addition as a shared variable $(z_1, \ldots, z_n)$ such that the recombination of the shares gives the desired result $x+y$, that is $z_1 + \ldots + z_n = x + y$. For instance, we can compute the addition sharewise as

$$z_1 = x_1 + y_1, \ldots, z_n = x_n + y_n \ .$$

In this case, an attacker must target the computation of each sharewise addition to retrieve

the result $x + y$. A less secure way to compute this addition is, for instance, by doing

$$z_1 = x_1 + \ldots + x_n, z_2 = y_1 + y_2, z_3 = y_3, \ldots, z_n = y_n \ .$$

Although the shared result still gives $z_1 + \ldots + z_n = x + y$, an attacker that can retrieve the value of the first share z_1 from the leakage traces already learns the value of the secret input x. The examples above showcase the importance of careful usage of the masking countermeasure as a misusage can sometimes reduce the security a masked implementation is intended to achieve. Constructing non-linear computations on shared variables becomes even more complex while trying to avoid share recombinations. For instance, to compute an n-shared multiplication between $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$, cross-products of the input shares can be performed, resulting in n^2 terms that then must be compressed back into an n-shared result.

In addition, this analysis of masked operations does not consider some physical defaults that can occur on a computing device. For instance, transitions occurring on memory buses or CPU registers between a previously processed value z_{i-1} and the current one z_i usually leak some information correlated to $z_{i-1} \oplus z_i$. Such defaults can also lower the initial security level of a masked implementation. The technical difficulties mentioned above give rise to the need for proper and rigorous ways to reason about the security of masked implementations.

1.4 Proven Security Against Side-Channel Attacks: Leakage Models

Providing security guarantees against side-channel attacks is generally tricky, and many works try to tackle this issue [46, 60, 55, 30]. When dealing with masked implementations, an empirical approach tries to assess the leakage given a certain number of leakage traces, mount well-known attacks, and check that the security is still guaranteed in the considered environment. This technique, however, does not provide a proven security level for a masked implementation. To reason about the security of such implementations in theory and provide security bounds for any circuit, the leakage is generally formalized in so-called leakage models. Ishai, Sahai, and Wagner (ISW) [59] first introduced the *t-probing model* for $t < n$, where a circuit is secure if the exact values of any set of t intermediate variables do not reveal any information on the secrets. This definition is supposed to reflect the security of masking by assuming that as t grows, an attacker would need more leakage traces to recover the secret, and since leakage traces come with noise, it becomes much more challenging to recover sensitive information. Variants of this model include the *robust probing model* [49] in hardware scenarios, which considers wider leakage to model physical effects such as glitches and transitions. In this scenario, t observations might reveal more than t intermediate computation variables.

Since its introduction, the probing model has been widely used by the community [74, 72, 40, 16, 41], thanks to it being very convenient to build security proofs. However, it raised several concerns regarding its relevance in practice [13, 56]. For instance, the authors of [13] show that the repeated manipulation of identical values can be exploited to retrieve the secrets in the context of *horizontal side-channel attacks*. These issues motivated the formalization of the *noisy leakage model* [70]. This model well captures the reality of embedded devices by assuming that each intermediate variable leaks a noisy function of its value. A circuit or implementation is secure in the noisy leakage model if the adversary cannot recover the secrets from a noisy function of each intermediate variable of the implementation. Unfortunately, proving the security of a masking scheme in this model is rather tedious.

Duc, Dziembowski, and Faust [45] proposed in 2014 a reduction from the noisy model to the t-probing model. The reduction relies on an intermediate leakage model, the *random*

probing model. The latter benefits from a tighter reduction with the noisy leakage model, which becomes independent of the size of the circuit. In a nutshell, it assumes that every wire in the circuit leaks with some constant leakage probability. This leakage probability is related to the amount of side-channel noise in practice. Then, a masked circuit is secure if there is a negligible probability that these leaking wires reveal information about the secrets.

1.5 Probing Model: Security and Composition

Proving security properties in the probing and random probing models amounts to exploring a combinatorial number of possible sets of probes and checking that they do not reveal any information about the secrets. For instance, a circuit using n-shared variables is t-probing secure for $t < n$ if the exact values of at most t leaking variables are independent of the secret. Such proofs come in two flavors: generic or exhaustive. Generic proofs are handwritten proofs that reason on the different forms of probes that can occur on the considered circuit. Generic proofs then check that we can not reveal information about the secrets for any considered form of probes. Such proofs do not exhaust all the possible sets of probes but "group" them into categories that can have a common effect on security. Meanwhile, exhaustive proofs, as their name indicates, rely on exhausting all possible sets of probes on the circuit and checking that no information on the secrets is revealed for each such set. When dealing with probing security and uniform input sharings, checking a set of probes is often done by computing the distribution of these probes and checking that it is independent of the distribution of the secrets.

Even in the simple probing model, proving the security of large masked circuits is a challenging problem using generic or exhaustive proofs. The most common solution is to build circuits from smaller sub-circuits named gadgets that implement a simple computation on masked data. Then, these gadgets can be composed to implement more complex computations, and the security proof only needs to care about the properties of the gadgets and their composition. In their seminal work [59], ISW introduced such gadgets for Boolean AND and NOT gates. These gadgets can be arbitrarily composed, but this scheme requires masking with $n = 2t+1$ shares. For performance reasons, the follow-up works primarily focused on using the optimal number of shares $n = t + 1$. For instance, the type-system of Barthe *et al.* [10] makes it possible to securely compose small gadgets that are proven to be (strong) non-interferent. These notions rely on the concept of *simulatability*. In a nutshell, the simulatability notion consists in constructing an algorithm called the *simulator*, which can simulate the distribution of a set of probes using some subset of the input shares of each input sharing. For instance, the t-non-interference (NI) notion imposes that the distribution of any set of at most t probes on a circuit or gadget can be simulated using at most t input shares of each n-shared input (with $t < n$). Observe that this notion does not necessarily suppose the uniformity of the input sharings and that if the latter are uniform sharings, then t-NI automatically implies t-probing security.

The t-strong non-interference (SNI) property is stronger than t-NI and imposes that any set of probes formed of t_1 probes on intermediate variables and t_2 probes on output variables of the gadget such that $t_1 + t_2 \leq t$ can be simulated using at most t_1 input shares of each input sharing. This property implies t-NI and allows for more efficient composition, especially for multiplication gadgets. However, SNI gadgets for affine operations could be more efficient. This issue was solved by the t-PINI definition (at the expense of slightly less efficient multiplication) [33].

Another direction to improve efficiency is to drop the requirement of arbitrary composition: while direct application of the ISW construction with $n = t + 1$ is not secure [72], it can be

fixed by adding refresh gadgets (which implement the identity function but re-randomize the sharings). This has been proposed in [10] with the `maskComp` tool that can compose weaker non-interferent (NI) gadgets by inserting SNI refresh gadgets (which can be derived from the ISW multiplication) in the circuit. Later, Belaïd, Goudarzi, and Rivain [20] introduced tight private circuits (TPC), another variant of the ISW scheme with $n = t + 1$ and additional refresh gadgets. The set of refresh gadgets inserted by their `tightProve` tool is tight in the sense that removing one of these gadgets is guaranteed to break t-probing security.

A more potent version of the probing model has been considered in the literature, known as the *region probing model* [4]. In this model, the adversary is not limited to t probes on the circuit but gets t probes per gadget (or *region*) of the circuit. This model is relevant in practice as being closer to actual side-channel leakages (providing information on all the gadgets of an implementation) and provides a tighter reduction to the noisy leakage model [45] than the probing model. In a recent work [53], Goudarzi, Prest, Rivain, and Vergnaud introduce the *input-output separation* (IOS) notion for simple composition in the region probing model. This notion acts as probing-composition scissors: the circuit is divided into regions separated by IOS gadgets whose probes can then be simulated separately.

Another extension of the t-probing model is the *t-robust* probing model. In the former, the exact values of t circuit variables are supposed to leak independently. However, in practice, physical defaults might generate a leakage of values carried by several wires simultaneously. In that case, a circuit secure in the t-probing model might only be secure in the t'-robust probing model for $t' < t$. Typical examples of physical defaults include transitions and glitches, and probes can be extended accordingly. For instance, in glitches, probes might reveal all the variables carried by coming wires up to the last synchronization point. Such probes thus include not only one but a set of wires. Similarly, pairs of values that are successively stored in the same memory cell may leak at once (when the second variable replaces the first one). Therefore, in transition-based leakage, probes are generally assumed to include specific pairs of wires. Hence, in the t-robust probing model, a set of t probes is replaced by a set of t extended probes, which can reveal the exact values of more than t variables. A circuit is then secure in this model if any set of t so-called extended probes is independent of the secret inputs.

A complementary line of research has been considering the optimization of the masked gadgets themselves. This culminated in the design of a $(n-1)$-NI multiplication gadget with randomness usage $n^2/4+\mathcal{O}(n)$ [16] (while the ISW multiplication is $(n-1)$-SNI and has $n^2/2+\mathcal{O}(n)$ randomness usage) and a $(n-1)$-SNI refresh gadget with complexity $\mathcal{O}(n \log n)$ [14]. Besides these arbitrary-order gadgets, there have also been many optimizations for low-order gadgets.

1.6 Random Probing Model: Security and Composition

Compared to the probing model, the random probing model is closer to the noisy leakage model and, in particular, captures *horizontal attacks*, which exploit the repeated manipulations of variables throughout the implementation. Classical probing secure schemes are also secure in the random probing model, but the tolerated leakage probability (a.k.a. leakage rate) might decrease with the masking order, which is not satisfactory from a practical viewpoint. Indeed, in practice, the leakage probability translates to some side-channel noise amount that the implementer might not be able to customize.

Before this thesis, only a few works have studied constructions in the random probing model. In [1, 4], the authors provide constructions in the random probing model based on expander graphs. The problem with these constructions is that they are conceptually involved, and their practical instantiation is complex. Also, the tolerated probability is not

made explicit. Then, in [3], the authors provide constructions based on multi-party computation (MPC) protocols and an expansion strategy. Their idea is to recursively apply an MPC protocol on a base circuit several times to achieve a desired security level in the random probing model. The authors provide an instantiation based on the Maurer MPC protocol [67], and explain that the complexity of the expansion for a circuit with $|C|$ gates is $\mathcal{O}(|C| \cdot \mathsf{poly}(\kappa))$ for some parameter κ but neither the polynomial nor the tolerated leakage probability are made explicit.

1.7 Automatic Verification Tools

As discussed in Section 1.5, exhaustive proofs for (random) probing security consist of enumerating a combinatorial number of probes and checking that they are independent of the secret inputs (with simulation-based notions, we must be able to perfectly simulate their distributions without the knowledge of the secret inputs). The manual verification of such properties on small gadgets is very error-prone [40]. Therefore, automatic tools are regularly built to apply formal verification on software and hardware masked implementations.

In 2012, Moss *et al.* [69] designed the first automatic type-based masking compiler to provide first-order security against DPA. Following this seminal work, Bayrak *et al.* [15] investigated the SMT-based method to evaluate the statistical independence between leakage and secrets. Eldib, Wang, and Schaumont [48] extended it to verifying higher-order targets using a similar notion to non-interference. Nevertheless, the complexity of their model counting approach restricts it to small masking orders.

Barthe *et al.* [9] then formalized the connection between the security of masked implementation and probabilistic non-interference. While their method does not entirely prevent the combinatorial explosion of observation sets for high orders, it helps gain several orders of magnitude and provides non-negligible complexity improvements. The resulting tool, maskVerif, thus verifies circuits at reasonable masking orders. After several improvements in the past few years [10, 6], maskVerif includes the verification of most probing-like security notions for different leakage models, including the robust probing model in the presence of glitches. Its extension into scVerif [32] captures even more advanced hardware side effects [12]. In the same line of work, checkMasks [38] offers the same functionalities with a more extensive scope (*e.g.*, verification of Boolean to arithmetic masking conversion) and a polynomial time verification on selected gadgets. In the same vein, Zhang *et al.* use abstraction-refinement techniques to improve scalability and precision with their tool `SCInfer` [75] whose complexity remains significant. Bordes and Karpman [28] also try to improve accuracy with their tool matverif.

In a parallel sequence of works, Rebecca [27] was designed to verify the probing security in the presence of glitches on Verilog implementations. It preceded the similar improvement on maskVerif to also handle hardware implementations. One step further, Coco [51] was designed to check masked software implementations given any possible architectural side effects. Inspired by Rebecca, it analyzes a CPU design as a hardware circuit and investigates several shares' potential leaks. Finally, SILVER [62] offers the verification of the classical probing-like security properties for hardware implementations with a method based on the analysis of probability distributions.

It is worth noting that before this thesis, no tools existed to verify security in the more realistic random probing model.

1.8 Contributions of the Thesis

This thesis focuses on secure and verified masked cryptographic implementations in the random probing model. Our contributions can be summarized as follows.

Security & Composition [18]. First, we introduce a composition security property to make gadgets composable in a global random probing secure circuit. We exhibit the relation between this new *random probing composability* (RPC) notion and the *strong non-interference* (SNI) notion, which is widely used in the context of probing security [10]. We also define a verification method to automatically exhibit the random probing security parameters of any small circuit whose variables leak with some probability p. In a nutshell, a circuit is (p, f)-random probing secure if it leaks information on the secret with probability $f(p)$, where $f(p)$ is the *failure probability function*. While it is limited to small circuits by complexity, the state-of-the-art shows that verifying those circuits can be particularly useful in practice (see e.g. the `maskVerif` tool [9]), for instance, to verify gadgets and then deduce global security through composition properties and low-order masked implementations. Following our work, the authors of a recent work [32] provide tighter composition notions to compose gadgets secure in the random probing model by computing probe distribution tables (PDTs). In a nutshell, a probe distribution table computes a different function $f(p)$ for each leaking set of input shares and output shares. While their representation allows for tighter composition than the RPC notion, the latter is used by our expansion strategy, which helps achieve arbitrarily large levels of security in the random probing model.

Expanding Compiler [18, 22]. Second, we revisit the modular approach of Ananth, Ishai, and Sahai [3], which uses an expansion strategy to get random probing security from a multi-party computation protocol. We introduce the powerful *expanding compiler* that builds random probing secure circuits from small base gadgets. We formalize the notion of *random probing expandability* (RPE) and show that a base gadget satisfying this notion can be securely used in the expanding compiler to achieve arbitrary/composable random probing security. The advantage of this approach is that it enables bootstrapping small gadgets (defined for a small number of shares) into a circuit, achieving arbitrary security in the random probing model while tolerating a constant and quantified leakage probability. We provide a detailed complexity analysis of our strategy and show that it depends on the size of the gadgets and a parameter that we define as the *amplification order*. In a nutshell, the amplification order refers to the size of the smallest set of probes in a gadget that reveals information about the secret inputs. We prove bounds on the optimal amplification orders achievable by any gadget. Then, we compare our approach with previous works, specifically the ISW construction and the modular approach from [3], and show that our expansion strategy is much more efficient and achieves better security levels in the random probing model. We also implement a proof-of-concept expanding compiler that can take any base gadgets as input and give the expanded gadgets as output (*i.e.*, gadgets on which we applied the expansion strategy). We additionally implement a protected AES implementation, which uses these output gadgets to benchmark the performance on a real-life secure implementation. The source code of these implementations is publicly available at:

`https://github.com/CryptoExperts/poc-expanding-compiler`

Optimal Expanding Compilers [22, 23]. Third, we provide an in-depth analysis of our expansion strategy and the expandability notion. We introduce generic constructions of gadgets achieving RPE for any number of shares and with nearly optimal parameters. We namely show how to construct addition, copy, and multiplication gadgets approaching these optimal

parameters and then instantiate these constructions for concrete gadgets on a small number of shares with a quadratic complexity with respect to the number of shares n. We also push the random probing expansion strategy one step further by analyzing a dynamic choice of the base gadgets. We consider a dynamic approach in which a new compiler is selected at each expansion step from a family of base compilers $\{\mathsf{CC}_i\}_i$. This approach is motivated by the previously mentioned generic gadget constructions, which achieve the RPE property for any number of shares n. While the asymptotic complexity of the expanding compiler decreases with n, the tolerated leakage probability p also gets smaller with n, which makes those constructions only practical for small values of n. We show that using our dynamic approach, we can get the best of both worlds: our dynamic expanding compiler enjoys the best tolerated probability and the best asymptotic complexity from the underlying family of RPE compilers $\{\mathsf{CC}_i\}_i$. This dynamic approach further motivates the design of asymptotic RPE gadgets to achieve better complexity. To improve on the previously constructed gadgets with quadratic complexity, we introduce new constructions achieving quasi-linear complexity. The addition and copy gadgets are based on the refresh gadget from Battistello, Coron, Prouff, and Zeitoun [14]. While for the multiplication gadget, we obtain a construction with a linear number of multiplications by tweaking the probing-secure multiplication gadget from Belaïd, Benhamouda, Passelègue, Prouff, Thillard, and Vergnaud [17]. As in the original construction, our RPE gadget imposes some constraints on the underlying finite field. We demonstrate that for any number of shares, there exists a (possibly large) finite field on which our construction can be instantiated, and we provide some concrete instantiations for some small number of shares. These new gadgets finally enable us to achieve nearly optimal complexity for the expanding compiler.

Automatic Verification Tool [21]. Finally, in a first attempt to verify security properties in the random probing model, we implement a verification tool VRAPS, which verifies random probing composition and expansion properties by computing the random probing security parameters as mentioned earlier with the failure function $f(p)$. VRAPS is the first tool to verify security properties in the random probing model. The source code of VRAPS is publicly available[1]. Later, we extend this tool into a more sophisticated one called IronMask, a versatile verification tool for masking security. IronMask is the first to verify standard simulation-based security notions in the probing model and composition and expandability notions in the random probing model introduced through our works. It supports any masking gadgets with linear randomness (*e.g.* addition, copy, and refresh gadgets) as well as quadratic gadgets (*e.g.* multiplication gadgets) that might include non-linear randomness (*e.g.* by refreshing their inputs) while providing complete verification results for both types of gadgets. We achieve this complete verifiability by introducing a new algebraic characterization for such quadratic gadgets and exhibiting a complete method to determine the sets of input shares necessary and sufficient to perform a perfect simulation of any set of probes. We report various benchmarks which show that IronMask is competitive with state-of-the-art verification tools in the probing model. IronMask is also several orders of magnitude faster than VRAPS –the only previous tool verifying random probing composability and expandability– as well as SILVER –the only previous tool providing complete verification for quadratic gadgets with non-linear randomness. Thanks to this completeness and increased performance, we obtain better bounds for the tolerated leakage probability of state-of-the-art random probing secure compilers. IronMask is open-source and publicly available at:

`https://github.com/CryptoExperts/IronMask`

[1] See `https://github.com/CryptoExperts/VRAPS`

1.8.1 Other Contributions

Other results were obtained during the thesis but are not discussed in this manuscript.

In [25], we unify and extend existing probing composition notions. Namely, we show that the composition approach of tight private circuits (TPC) introduced in [20] requires a stronger notion than the SNI initially considered by the authors. The required notion is the *free SNI* notion introduced by Coron and Spignoli in [42]. In a nutshell, the free SNI notion requires that the non-simulated output shares of a gadget be uniformly distributed and mutually independent of the simulated wires. We analyze and generalize the free SNI notion, show how to use it to correct and generalize the TCP proof from [20] and show strong connections between the free SNI notion and the IOS notion introduced in [53] for easy composition in the region probing model. We also investigate gadget constructions under the strong unified notions of free SNI and IOS and propose generic constructions of gadgets achieving these notions from simpler building blocks. To validate our proofs, we provide an efficient automatic verification method for these notions and integrate it into the IronMask tool. Finally, we extend the IOS composition framework proposed in [53] using our generalized framework to obtain a more efficient composition in the region probing model.

In [24], we present a complete methodology describing the steps to turn an abstract circuit into a physical implementation satisfying provable security against side-channel attacks in practice. We propose new tools to enforce or relax the physical assumptions the ideal noisy leakage model relies on and provide novel ways of including them in a physical implementation. We also highlight the design goals for an embedded device to reach high levels of proven security, discussing the limitations and open problems of the practical usability of the leakage models. Our goal is to show that it is possible to bridge theory and practice and to motivate further research to fully close the gap and get practical implementations proven secure against side-channel attacks on a physical device without any ideal assumption about the leakage.

Chapter 2

Preliminaries

In this chapter, we give the necessary preliminary notations and definitions used in the rest of this manuscript.

In the following, $\mathbb{K}$ shall denote a finite field. For any $n \in \mathbb{N}$, we shall denote $[n]$ the integer set $[n] = [1, n] \cap \mathbb{Z}$. For any tuple $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{K}^n$ (sometimes denoted $\widehat{x}$ depending on the context, we use $\boldsymbol{x}$ for a vector of variables and $\widehat{x}$ for a sharing) and any set $I \subseteq [n]$, we shall denote $\boldsymbol{x}|_I = (x_i)_{i \in I}$.

Also, for some $\widehat{\boldsymbol{x}} = (\widehat{x}_1, \ldots, \widehat{x}_\ell) \in (\mathbb{K}^n)^\ell$, we denote by $\boldsymbol{I}$ a collection of sets $\boldsymbol{I} = (I_1, \ldots, I_\ell)$ with $I_1 \subseteq [n]$, $\ldots$, $I_\ell \subseteq [n]$. We then denote $\widehat{\boldsymbol{x}}|_{\boldsymbol{I}} = (\widehat{x}_1|_{I_1}, \ldots, \widehat{x}_\ell|_{I_\ell})$ where $\widehat{x}_i|_{I_i} \in \mathbb{K}^{|I_i|}$ is the tuple as defined earlier, *i.e.* $\widehat{x}_i|_{I_i} = (x_{i,j})_{j \in I_i}$.

Finally, any two probability distributions D_1 and D_2 are said ε-*close*, denoted $D_1 \approx_\varepsilon D_2$, if their statistical distance is upper bounded by ε, that is

$$\mathsf{SD}(D_1; D_2) := \frac{1}{2} \sum_x |p_{D_1}(x) - p_{D_2}(x)| \leq \varepsilon \ ,$$

where $p_{D_1}(\cdot)$ and $p_{D_1}(\cdot)$ denote the probability mass functions of D_1 and D_2.

2.1 Circuit Compilers

We call an *arithmetic circuit* over a field $\mathbb{K}$ a labeled directed acyclic graph whose edges are *wires* and vertices are *arithmetic gates* processing operations over $\mathbb{K}$. We consider three types of arithmetic gates:

- an addition gate, of fan-in 2 and fan-out 1, computes an addition over $\mathbb{K}$,
- a multiplication gate, of fan-in 2 and fan-out 1, computes a multiplication over $\mathbb{K}$,
- a copy gate, of fan-in 1 and fan-out 2, outputs two copies of its input.

A *randomized arithmetic circuit* is equipped with an additional type of gate:

- a random gate, of fan-in 0 and fan-out 1, outputs a fresh uniform random value of $\mathbb{K}$.

A (randomized) arithmetic circuit is further formally composed of input gates of fan-in 0 and fan-out 1 and output gates of fan-in 1 and fan-out 0. Evaluating an ℓ-input m-output circuit C consists in writing an input $\boldsymbol{x} \in \mathbb{K}^\ell$ in the input gates, processing the gates from input gates to output gates, then reading the output $\boldsymbol{y} \in \mathbb{K}^m$ from the output gates. This is denoted by $\boldsymbol{y} = C(\boldsymbol{x})$. During the evaluation process, each wire in the circuit is assigned a value on $\mathbb{K}$. We call the tuple of all these wire values a *wire assignment* of C (on input $\boldsymbol{x}$).

Remark 1. *We sometimes consider additional types of gates, such as multiplication by a constant value. For simplicity, we introduce such gates only in the sections where needed.*

Definition 1 (Circuit Compiler)**.** *A* circuit compiler *is a triplet of algorithms* $(\mathsf{CC}, \mathsf{Enc}, \mathsf{Dec})$ *defined as follows:*

- CC *(circuit compilation) is a deterministic algorithm that takes as input an arithmetic circuit C and outputs a randomized arithmetic circuit $\widehat{C}$.*
- Enc *(input encoding) is a probabilistic algorithm that maps an input $\boldsymbol{x} \in \mathbb{K}^\ell$ to an encoded input $\widehat{\boldsymbol{x}} \in \mathbb{K}^{\ell'}$.*
- Dec *(output decoding) is a deterministic algorithm that maps an encoded output $\widehat{\boldsymbol{y}} \in \mathbb{K}^{m'}$ to a plain output $\boldsymbol{y} \in \mathbb{K}^m$.*

These three algorithms satisfy the following properties:

- ***Correctness:*** *For every arithmetic circuit C of input length ℓ, and for every $\boldsymbol{x} \in \mathbb{K}^\ell$, we have*

$$\Pr\left(\mathsf{Dec}\big(\widehat{C}(\widehat{\boldsymbol{x}})\big) = C(\boldsymbol{x}) \;\middle|\; \widehat{\boldsymbol{x}} \leftarrow \mathsf{Enc}(\boldsymbol{x})\right) = 1 \ , \ \textit{where } \widehat{C} = \mathsf{CC}(C).$$

- ***Efficiency:*** *For some security parameter $\lambda \in \mathbb{N}$, the running time of $\mathsf{CC}(C)$ is $\mathrm{poly}(\lambda, |C|)$, the running time of $\mathsf{Enc}(\boldsymbol{x})$ is $\mathrm{poly}(\lambda, |\boldsymbol{x}|)$ and the running time of $\mathsf{Dec}\big(\widehat{\boldsymbol{y}}\big)$ is $\mathrm{poly}(\lambda, |\widehat{\boldsymbol{y}}|)$, where $\mathrm{poly}(\lambda, q) = O(\lambda^{k_1} q^{k_2})$ for some constants k_1, k_2.*

2.2 Linear Sharing and Gadgets

In the following, the *n-linear decoding* mapping, denoted LinDec, refers to the function $\bigcup_n \mathbb{K}^n \to \mathbb{K}$ defined as

$$\mathsf{LinDec} : (x_1, \ldots, x_n) \mapsto x_1 + \cdots + x_n \ ,$$

for every $n \in \mathbb{N}$ and $(x_1, \ldots, x_n) \in \mathbb{K}^n$. We shall further consider that, for every $n, \ell \in \mathbb{N}$, on input $(\widehat{x}_1, \ldots, \widehat{x}_\ell) \in (\mathbb{K}^n)^\ell$ the n-linear decoding mapping acts as

$$\mathsf{LinDec} : (\widehat{x}_1, \ldots, \widehat{x}_\ell) \mapsto (\mathsf{LinDec}(\widehat{x}_1), \ldots, \mathsf{LinDec}(\widehat{x}_\ell)) \ .$$

Let us recall that for some tuple $\widehat{x} = (x_1, \ldots, x_n) \in \mathbb{K}^n$ and for some set $I \subseteq [n]$, the tuple $(x_i)_{i \in I}$ is denoted $\widehat{x}|_I$.

Definition 2 (Linear Sharing)**.** *Let $n, \ell \in \mathbb{N}$. For any $x \in \mathbb{K}$, an* n-linear sharing *of x is a random vector $\widehat{x} \in \mathbb{K}^n$ such that $\mathsf{LinDec}(\widehat{x}) = x$. It is said to be* uniform *if for any set $I \subseteq [n]$ with $|I| < n$ the tuple $\widehat{x}|_I$ is uniformly distributed over $\mathbb{K}^{|I|}$. A* n-linear encoding *is a probabilistic algorithm LinEnc which on input a tuple $\boldsymbol{x} = (x_1, \ldots, x_\ell) \in \mathbb{K}^\ell$ outputs a tuple $\widehat{\boldsymbol{x}} = (\widehat{x}_1, \ldots, \widehat{x}_\ell) \in (\mathbb{K}^n)^\ell$ such that $\widehat{x}_i$ is a uniform n-sharing of x_i for every $i \in [\ell]$.*

In the following, we shall call an *(n-share, ℓ-to-m) gadget*, a randomized arithmetic circuit that maps an input $\widehat{\boldsymbol{x}} \in (\mathbb{K}^n)^\ell$ to an output $\widehat{\boldsymbol{y}} \in (\mathbb{K}^n)^m$ such that $\boldsymbol{x} = \mathsf{LinDec}(\widehat{\boldsymbol{x}}) \in \mathbb{K}^\ell$ and $\boldsymbol{y} = \mathsf{LinDec}(\widehat{\boldsymbol{y}}) \in \mathbb{K}^m$ satisfy $\boldsymbol{y} = g(\boldsymbol{x})$ for some function g, which can be any arithmetic function over $\mathbb{K}$. In most cases (unless specified otherwise), we shall consider gadgets for three types of functions corresponding to the three types of gates: the addition $g : (x_1, x_2) \mapsto x_1 + x_2$, the multiplication $g : (x_1, x_2) \mapsto x_1 \cdot x_2$ and the copy $g : x \mapsto (x, x)$. We shall generally denote such gadgets G_{add}, G_{mult} and G_{copy} respectively. We also make use of the refresh gadget denoted G_{refresh}, which takes as input an n-sharing $\widehat{x}$ and outputs a new sharing $\widehat{y}$ such that $x_1 + \ldots + x_n = y_1 + \ldots + y_n$. In other words, G_{refresh} is similar to G_{copy}, except that it corresponds to the identity function $g : x \mapsto x$. As for the other gates (*e.g.* multiplication by a constant), we introduce the notation for the corresponding gadgets when needed.

Definition 3 (Standard Circuit Compiler)**.** *Let $\lambda \in \mathbb{N}$ be some security parameter and let $n = \text{poly}(\lambda)$. Let $\mathbb{B} = \{g : \mathbb{K}^\ell \to \mathbb{K}^m\}$ be an arithmetic circuit basis with sharing order n and base gadgets $\{G_g\}_{g\in\mathbb{B}}$. Typically when we consider $\mathbb{B}$ with the three types of gates introduced earlier (addition, multiplication and copy), we consider the gadgets G_{add}, G_{mult}, G_{copy} respectively. The* standard circuit compiler *with arithmetic basis $\mathbb{B}$,* sharing order *n, and* base gadgets *$\{G_g\}_{g\in\mathbb{B}}$ is the circuit compiler* $(\mathsf{CC}, \mathsf{Enc}, \mathsf{Dec})$ *satisfying the following:*

1. *The input encoding* Enc *is an n-linear encoding.*
2. *The output decoding* Dec *is the n-linear decoding mapping* LinDec*.*
3. *The circuit compilation* CC *consists in replacing each gate in the original circuit by an n-share gadget with corresponding functionality (G_g for a gate implementing the function g), and each wire by a set of n wires carrying an n-linear sharing of the original wire. If the input circuit is a randomized arithmetic circuit, each of its random gates is replaced by n random gates, which duly produce an n-linear sharing of a random value.*

For such a circuit compiler, the correctness and efficiency directly holds from the correctness and efficiency of the gadgets $\{G_g\}_{g\in\mathbb{B}}$.

2.3 (Strong) Non-Interference in the Probing Model

In the t-probing model, an adversary can choose a set of t probes on the wires in the target circuit. The adversary can then observe the exact values carried by the probes. An n-share circuit is then t-probing secure for $t \leq n-1$ if any such set of t probes is independent of the sensitive input shares, assuming all input sharings are uniform.

Proving the security of large masked circuits is a challenging problem. Hence, we usually build such circuits from smaller blocks called gadgets. However, the definition of t-probing security does not allow composition of gadgets. In other words, connecting t-probing secure gadgets does not necessarily give a t-probing secure circuit [40]. Hence, stronger notions are required to ensure this composition property. In [10], the authors introduce such notions. To do so, they rely on the *simulatability* property of sets of probes. Given an ℓ-to-m n-share gadget with input $\widehat{\boldsymbol{x}}$, a set of probes W is said to be t-simulatable if there exists a collection of sets $\boldsymbol{I} = (I_1, \ldots, I_\ell)$ such that $|I_i| \leq t$, $\forall i \in [\ell]$, and there exists an algorithm called simulator that can generate a simulation of the probes in W which have the same distribution as the probes in W, using only the knowledge of the input shares $\widehat{\boldsymbol{x}}|_{\boldsymbol{I}}$. In other words, given a fixed input sharing $\widehat{\boldsymbol{x}}$, the simulator only needs access to the values of $\widehat{\boldsymbol{x}}|_{\boldsymbol{I}}$ and can perfectly simulate the distribution of the probes in W. When $t \leq n-1$ and the input sharings are all uniform, and each set of t probes is t-simulatable, this implies t-probing security as the probes are independent of the input values not given to the simulator. Note that a trivial simulator always exists by setting $\boldsymbol{I} = ([n], \ldots, [n])$.

Based on this simulatability notion, the authors of [10] introduce the non-intereference (NI) and strong non-interference (SNI) composition properties.

Definition 4 (Non-Interference (NI))**.** *Let n, ℓ and t be positive integers. An n-share gadget $G : (\mathbb{K}^n)^\ell \to \mathbb{K}^n$ is t-NI if there exists a deterministic algorithm Sim_1^G and a probabilistic algorithm Sim_2^G such that for every set $J \subseteq [n]$ and subset W of wire labels from G satisfying $|W| + |J| \leqslant t$, the following random experiment with any $\widehat{\boldsymbol{x}} \in (\mathbb{K}^n)^\ell$*

$$\boldsymbol{I} \leftarrow \mathsf{Sim}_1^G(W, J)$$
$$out \leftarrow \mathsf{Sim}_2^G\big(\widehat{\boldsymbol{x}}|_{\boldsymbol{I}}\big)$$

yields

$$|I_1| \leqslant t, \ldots, |I_\ell| \leqslant t \tag{2.1}$$

and

$$out \overset{id}{=} \left(\mathsf{AssignWires}(G, W, \widehat{\boldsymbol{x}}) \ , \ \widehat{\boldsymbol{y}}|_J\right) \tag{2.2}$$

where $\boldsymbol{I} = (I_1, \ldots, I_\ell)$ *and* $\widehat{\boldsymbol{y}} = G(\widehat{\boldsymbol{x}})$.

We can observe that t-NI is stronger than t-probing security and that t-NI does not suppose the uniformity of the input sharings. Indeed, t-NI with the uniformity of the input sharings imply t-probing security.

Next, t-SNI gadgets guarantee independence between the input and output shares in the presence of a t-probing adversary.

Definition 5 (Strong Non-Interference (SNI))**.** *Let n, ℓ and t be positive integers. An n-share gadget $G : (\mathbb{K}^n)^\ell \to \mathbb{K}^n$ is t-SNI if there exists a deterministic algorithm Sim_1^G and a probabilistic algorithm Sim_2^G such that for every set $J \subseteq [n]$ and subset W of wire labels from G satisfying $|W| + |J| \leqslant t$, the following random experiment with any $\widehat{\boldsymbol{x}} \in (\mathbb{K}^n)^\ell$*

$$\begin{aligned} \boldsymbol{I} &\leftarrow \mathsf{Sim}_1^G(W, J) \\ out &\leftarrow \mathsf{Sim}_2^G\left(\widehat{\boldsymbol{x}}|_{\boldsymbol{I}}\right) \end{aligned}$$

yields

$$|I_1| \leqslant |W|, \ldots, |I_\ell| \leqslant |W| \tag{2.3}$$

and

$$out \overset{id}{=} \left(\mathsf{AssignWires}(G, W, \widehat{\boldsymbol{x}}) \ , \ \widehat{\boldsymbol{y}}|_J\right) \tag{2.4}$$

where $\boldsymbol{I} = (I_1, \ldots, I_\ell)$ *and* $\widehat{\boldsymbol{y}} = G(\widehat{\boldsymbol{x}})$.

The authors of [10] show that building secure circuits from t-NI and t-SNI gadgets is possible by carefully inserting t-SNI refresh gadgets where needed.

2.4 Gadgets from the Literature

In this section, we introduce some widely used gadgets from the literature that we will use for our constructions in the rest of this manuscript.

2.4.1 ISW Construction

ISW Multiplication Gadget. The ISW multiplication gadget was introduced in [59] as a gadget that achieves security in the probing model. The gadget is proven to achieve $(n-1)$-probing security for any number of shares n. Later, the gadget was also proven to satisfy the stronger $(n-1)$-SNI notion in [10]. The ISW multiplication algorithm has quadratic complexity $\mathcal{O}(n^2)$ and uses $\dfrac{n(n+1)}{2}$ fresh random variables. It is described in Algorithm 1.

Algorithm 1: ISW Multiplication

Input : $(a_1, \ldots, a_n)$,$(b_1, \ldots, b_n)$ input sharings, $\{r_{ij}\}_{1 \leq i < j \leq n}$ random values
Output: $(c_1, \ldots, c_n)$ sharing of $a \cdot b$

for $i \leftarrow 1$ **to** n **do**
  $c_i \leftarrow a_i \cdot b_i$;
end
for $i \leftarrow 1$ **to** n **do**
  for $j \leftarrow i + 1$ **to** n **do**
    $c_i \leftarrow c_i + r_{ij}$;
    $r_{ji} \leftarrow (a_i \cdot b_j + r_{ij}) + a_j \cdot b_i$;
    $c_j \leftarrow c_j + r_{ji}$;
  end
end
return $(c_1, \ldots, c_n)$;

ISW Refresh Gadget. The ISW refresh gadget can be seen as an ISW multiplication between the input sharing and the n-tuple $(1, 0, \ldots, 0)$. This is formally depicted in Algorithm 2. Trivially, the gadget is also $(n-1)$-SNI in the probing model.

Algorithm 2: ISW Refresh

Input : $(a_1, \ldots, a_n)$ input sharing, $\{r_{ij}\}_{1 \leq i < j \leq n}$ random values
Output: $(c_1, \ldots, c_n)$ such that $c_1 + \cdots + c_n = a_1 + \cdots + a_n$

for $i \leftarrow 1$ **to** n **do**
  $c_i \leftarrow a_i$;
end
for $i \leftarrow 1$ **to** n **do**
  for $j \leftarrow 1$ **to** $i - 1$ **do**
    $c_i \leftarrow c_i + r_{ji}$;
  end
  for $j \leftarrow i + 1$ **to** n **do**
    $c_i \leftarrow c_i + r_{ij}$;
  end
end
return $(c_1, \ldots, c_n)$;

2.4.2 $\mathcal{O}(n \log n)$ Refresh Gadget

In [14], the authors introduce a new construction for a refresh gadget that benefits from quasi-linear complexity. Namely, the n-share gadget has complexity in $\mathcal{O}(n \log n)$. The authors prove that the gadget satisfies the strong notion $(n-1)$-SNI in the probing model, which makes it composable and secure in the probing model for any number of shares n. The description of the gadget is given in Algorithm 3.

Algorithm 3: QuasiLinearRefresh

Input : $(a_1, \ldots, a_n)$ input sharing
Output: $(d_1, \ldots, d_n)$ such that $d_1 + \cdots + d_n = a_1 + \cdots + a_n$

if $n = 1$ **then return** a_1;
if $n = 2$ **then**
  $r \leftarrow \$$;
  return $(a_1 + r, a_2 - r)$;
end
for $i \leftarrow 1$ **to** $\lfloor n/2 \rfloor$ **do**
  $r \leftarrow \$$;
  $b_i \leftarrow a_i + r$;
  $b_{\lfloor n/2 \rfloor + i} \leftarrow a_{\lfloor n/2 \rfloor + i} - r$;
end
if $n \mod 2 = 1$ **then** $b_n \leftarrow a_n$;
$(c_1, \ldots, c_{\lfloor n/2 \rfloor}) \leftarrow \mathsf{QuasiLinearRefresh}(b_1, \ldots, b_{\lfloor n/2 \rfloor})$;
$(c_{\lfloor n/2 \rfloor + 1}, \ldots, c_n) \leftarrow \mathsf{QuasiLinearRefresh}(b_{\lfloor n/2 \rfloor + 1}, \ldots, b_n)$;
for $i \leftarrow 1$ **to** $\lfloor n/2 \rfloor$ **do**
  $r \leftarrow \$$;
  $d_i \leftarrow c_i + r$;
  $d_{\lfloor n/2 \rfloor + i} \leftarrow c_{\lfloor n/2 \rfloor + i} - r$;
end
if $n \mod 2 = 1$ **then** $d_n \leftarrow c_n$;
return $(d_1, \ldots, d_n)$;

Chapter 3

Random Probing Security

In this chapter, we recall the definition of random probing security from the literature. Then, we discuss the contributions of this thesis in the following sections. Namely, we introduce the random probing composability notion, which allows us to securely compose small gadgets into a circuit with global random probing security. After that, we present our technique for computing the security parameters of a circuit in the random probing model. In a nutshell, a circuit is (p, ε)-random probing secure if any set of leaking wires, where each wire is added independently with probability p, can be perfectly simulated without knowing the secrets with probability $1 - \varepsilon$. In Section 3.3, we present our contributions on how to compute this failure probability as a function of the leakage probability p, which allows us to automatically verify the security of circuits in the random probing model using automatic verification tools like our tool IronMask discussed later in Chapter 7. The contributions in this chapter are published in [18].

3.1 Definition of Random Probing Security

Let $p \in [0, 1]$ be some constant leakage probability parameter. This parameter is sometimes called *leakage rate* in the literature. Informally, the p-random probing model states that during the evaluation of a circuit C, each wire leaks its exact value with probability p and leaks nothing with probability $1 - p$. Such leakage event is defined on each wire in the circuit, independently of all the other wires. For instance, the probability that all s wires of a circuit C leak their exact values is p^s, while only $t \leq s$ wires leak their values with probability $p^t \cdot (1 - p)^{s-t}$.

Notice that when a wire leaks, it gives the adversary the exact value it carries without any noise. This is not the case in real-life acquisitions where some noise perturbs the signal corresponding to a leaked value. Recall that in the t-probing model, at most t wires can leak their exact values, based on the assumption that retrieving secrets from noisy traces becomes exponentially hard when the noise becomes more important. In the random probing model, we have more flexibility than the probing model. Namely, we can represent this noise level by the leakage probability p, which increases as the noise level decreases.

In order to formally define the random probing leakage of a circuit, we shall consider two probabilistic algorithms:

- The *leaking-wires sampler* takes as input a randomized arithmetic circuit C and a probability $p \in [0, 1]$, and outputs a set W, denoted as

$$W \leftarrow \mathsf{LeakingWires}(C, p) \ ,$$

  where W is constructed by including each wire label from the circuit C with probability p to W (where all the probabilities are mutually independent), except for the output

wire labels. In other words, the output wires of C (*i.e.* the wires incoming output gates) are excluded by the $\mathsf{LeakingWires}$ sampler, whereas the input wires of C (*i.e.* the wires connecting input gates to subsequent gates) are included. Namely, W does not include any output wire label of C. This is because when composing several circuits (or gadgets), the output wires of a circuit are the input wires in the following circuit. This also relates to the widely admitted *only computation leaks* assumption [68]: the processing of a gate leaks information on its input values (and information on the output can be captured through information on the input).

- The *assign-wires sampler* takes as input a randomized arithmetic circuit C, a set of wire labels W (subset of the wire labels of C), and an input $\boldsymbol{x} \in \mathbb{K}^\ell$, and it outputs a $|W|$-tuple $\boldsymbol{w} \in (\mathbb{K} \cup \{\perp\})^{|W|}$, denoted as

$$\boldsymbol{w} \leftarrow \mathsf{AssignWires}(C, W, \boldsymbol{x}) \ ,$$

  where $\boldsymbol{w}$ corresponds to the assignments of the wires of C with label in W for an evaluation on input $\boldsymbol{x}$.

We can now formally define the random probing leakage of a circuit.

Definition 6 (Random Probing Leakage)**.** *The* p-random probing leakage *of a randomized arithmetic circuit* C *on input* $\boldsymbol{x}$ *is the distribution* $\mathcal{L}_p(C, \boldsymbol{x})$ *obtained by composing the leaking-wires and assign-wires samplers as*

$$\mathcal{L}_p(C, \boldsymbol{x}) \stackrel{id}{=} \mathsf{AssignWires}(C, \mathsf{LeakingWires}(C, p), \boldsymbol{x}) \ .$$

Definition 7 (Random Probing Security)**.** *A randomized arithmetic circuit* C *with* $\ell \cdot n \in \mathbb{N}$ *input gates (for* ℓ *inputs of* n *shares) is* (p, ε)-random probing secure *with respect to encoding* Enc *if there exists a simulator* Sim *such that for every* $\boldsymbol{x} \in \mathbb{K}^\ell$*:*

$$\mathsf{Sim}(C) \approx_\varepsilon \mathcal{L}_p(C, \mathsf{Enc}(\boldsymbol{x})) \ . \tag{3.1}$$

A circuit compiler $(\mathsf{CC}, \mathsf{Enc}, \mathsf{Dec})$ *is* (p, ε)-random probing secure *if for every (randomized) arithmetic circuit* C *the compiled circuit* $\widehat{C} = \mathsf{CC}(C)$ *is* $(p, |C| \cdot \varepsilon)$*-random probing secure where* $|C|$ *is the size of the original circuit.*

We can slightly tweak the definition above to consider *simulation with abort* as in [3]. The difference is that instead of considering that the distribution of $\mathsf{Sim}(C)$ is ε-close to that of $\mathcal{L}_p(C, \mathsf{Enc}(\boldsymbol{x}))$, we consider that the simulator either outputs an identical distribution or aborts (or fails) with probability ε. Formally, for any leakage probability $p \in [0, 1]$, the simulator Sim is defined as

$$\mathsf{Sim}(C) = \mathsf{SimAW}(C, \mathsf{LeakingWires}(C, p)) \tag{3.2}$$

where SimAW, the *wire-assignment simulator*, either returns $\perp$ (simulation failure) or a perfect simulation of the requested wires. Formally, the experiment

$$\begin{aligned} &W \leftarrow \mathsf{LeakingWires}(C, p) \\ &\mathit{out} \leftarrow \mathsf{SimAW}(C, W) \end{aligned}$$

leads to

$$\Pr[\mathit{out} = \perp] = \varepsilon \quad \text{and} \quad \big(\mathit{out} \mid \mathit{out} \neq \perp\big) \stackrel{\text{id}}{=} \big(\mathsf{AssignWires}(C, W, \mathsf{Enc}(\boldsymbol{x})) \mid \mathit{out} \neq \perp\big) \ . \tag{3.3}$$

It is not hard to see that if we can construct such a simulator SimAW for a randomized arithmetic circuit C, then this circuit is (p, ε)-random probing secure. This tweaked definition will be useful when we express ε as a function of p later in Section 3.3.

3.2 Composition in the Random Probing Model

It is challenging to construct an arbitrarily large secure circuit, especially with no prior knowledge of the structure of the circuit. To circumvent this issue, we usually build secure circuits from small building blocks that we call *gadgets*. For each type of operation (or gate) involved in the circuit, we construct an n-share gadget with the same functionality as the latter operation. Then, we transform our base circuit starting from the input gates: we replace each input wire with n wires carrying a sharing of the original wire. Also, each input random wire is replaced by n input random wires. Then, we replace each internal gate of the circuit with the corresponding gadget. A standard circuit compiler performs these transformations on the original circuit. A 2-share example is given in Figure 3.1.

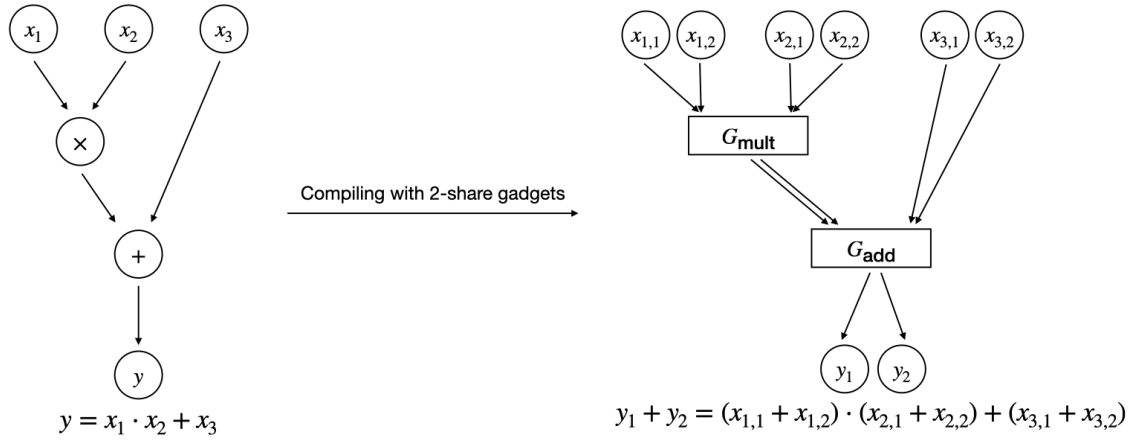

Figure 3.1: Illustration of compilation using a standard circuit compiler with 2-share gadgets. The inputs are x_1, x_2, x_3 and the output is y. Each arrow represents a wire carrying the value output by the gate represented as a circle. A gadget is represented as a rectangle since it consists of multiple gates.

Secure gadgets must satisfy a security notion that guarantees a global security level after composition. In the t-probing model, many security properties for composition have been introduced (t-SNI, t-PINI, *etc.*). In the random probing model, we introduce the *random probing composability* notion for a gadget.

Definition 8 (Random Probing Composability)**.** *Let $n, \ell, m \in \mathbb{N}$. An n-share gadget $G : (\mathbb{K}^n)^\ell \to (\mathbb{K}^n)^m$ is (t, p, ε)*-random probing composable (RPC) *for some $t \in \mathbb{N}$ and $p, \varepsilon \in [0, 1]$ if there exists a deterministic algorithm Sim_1^G and a probabilistic algorithm Sim_2^G such that for every input $\widehat{\boldsymbol{x}} \in (\mathbb{K}^n)^\ell$ and for every set collection $\boldsymbol{J} = (J_1, \ldots, J_m)$ such that $\forall i \in [m]$, $J_i \subseteq [n]$ and $|J_i| \leq t$, the random experiment*

$$
\begin{aligned}
&W \leftarrow \mathsf{LeakingWires}(G, p) \\
&\boldsymbol{I} = (I_1, \ldots, I_\ell) \leftarrow \mathsf{Sim}_1^G(W, \boldsymbol{J}) \\
&\mathit{out} \leftarrow \mathsf{Sim}_2^G\big(\widehat{\boldsymbol{x}}|_{\boldsymbol{I}}\big)
\end{aligned}
$$

yields

$$
\Pr\big((|I_1| > t) \vee \ldots \vee (|I_\ell| > t)\big) \leq \varepsilon \tag{3.4}
$$

and

$$
\mathit{out} \overset{id}{=} \big(\mathsf{AssignWires}(G, W, \widehat{\boldsymbol{x}})\ ,\ \widehat{\boldsymbol{y}}|_{\boldsymbol{J}}\big)
$$

where $\boldsymbol{J} = (J_1, \ldots, J_m)$ *and* $\widehat{\boldsymbol{y}} = G(\widehat{\boldsymbol{x}})$.

In the above definition, the first-pass simulator Sim_1^G determines the necessary input shares (through the returned collection of sets $\boldsymbol{I}$) for the second-pass simulator Sim_2^G to produce a perfect simulation of the leaking wires defined by the set W together with the output shares defined by the collection of sets $\boldsymbol{J}$. Note that there always exists such a collection of sets $\boldsymbol{I}$ since $\boldsymbol{I} = ([n], \ldots, [n])$ trivially allows a perfect simulation whatever W and $\boldsymbol{J}$. However, the goal of Sim_1^G is to return a collection of sets $\boldsymbol{I}$ with cardinals at most t. The idea behind this constraint is to keep the following composition invariant: for each gadget we can achieve a perfect simulation of the leaking wires plus t shares of each output sharing from t shares of each input sharing. We shall call *failure event* the event that at least one of the sets $I_1, \ldots, I_\ell$ output of Sim_1^G has cardinality greater than t. When (t, p, ε)-RPC is achieved, the failure event probability is upper bounded by ε according to (3.4). A failure event occurs whenever Sim_2^G requires more than t shares of one input sharing to be able to produce a perfect simulation of the leaking wires (*i.e.* the wires with label in W) together with the output shares in $\widehat{\boldsymbol{y}}|_{\boldsymbol{J}}$. Whenever such a failure occurs, the composition invariant is broken. In the absence of failure event, the RPC notion implies that a perfect simulation can be achieved for the full circuit composed of RPC gadgets. This is formally stated in the next theorem.

Theorem 1 (Composition)**.** *Let* $t \in \mathbb{N}$, $p, \varepsilon \in [0, 1]$, *and* CC *be a standard circuit compiler with* (t, p, ε)*-RPC base gadgets. For every (randomized) arithmetic circuit* C *composed of* $|C|$ *gadgets, the compiled circuit* $\mathsf{CC}(C)$ *is* $(p, |C| \cdot \varepsilon)$*-random probing secure. Equivalently, the standard circuit compiler* CC *is* (p, ε)*-random probing secure.*

Proof. We suppose that all gadgets have two input and one output sharings for simplicity of notations, but the proof can be trivially generalized to more inputs and outputs.
Let W denote the leaking wires of the randomized circuit $\mathsf{CC}(C)$ with probability p for each wire. We now build a simulator Sim taking as inputs $\mathsf{CC}(C)$ and W, and that perfectly simulates W with probability at least $(1 - |C| \cdot \varepsilon)$ from the simulators of the (t, p, ε)-RPC base gadgets. We start with splitting set W into $|C|$ distinct subsets W_i for $i \in \{1, \ldots, |C|\}$ such that each W_i stands for the output of $\mathsf{LeakingWires}$ when applied to the i^{th} gadget G_i of $\mathsf{CC}(C)$ with probability p.
We first describe how Sim executes $\mathsf{Sim}_1^{G_i}$ for all gadgets in the compiled circuit. Namely, we start with gadgets whose outputs coincide with the circuit's outputs (called end gadgets). We execute their $\mathsf{Sim}_1^{G_i}$ with W_i and $J_i = \emptyset$, to get both sets $I_{i,1}, I_{i,2}$ of required inputs. Then, we target their parents, that are gadgets whose outputs are inputs of end gadgets. For instance, for an end gadget G_i, its first input sharing is connected to the output sharing of a parent gadget G_j, and its second output sharing is connected to the output sharing of a second parent gadget G_k. Then, we execute $\mathsf{Sim}_1^{G_j}$ with W_j and $J_j = I_{1,i}$, and we execute $\mathsf{Sim}_1^{G_k}$ with W_k and $J_k = I_{2,i}$. Both executions give us new sets I of required inputs, which are correspondingly connected to output sharings of higher parent gadgets. The simulation goes through the circuit from bottom to top by applying the Sim_1^G simulators to get the W_i and I/J sets. The simulation fails if at least one set I is of cardinal greater than t. For $|C|$ gadgets, this happens with probability $1-(1-\varepsilon)^{|C|} \leq |C| \cdot \varepsilon$. Otherwise, the final top execution of simulators $\mathsf{Sim}_1^{G_i}$ for gadgets G_i which are directly connected to the input sharings of the circuit $CC(C)$ gives us the last collection of sets $\boldsymbol{I}$ on the input sharings.
Finally, the global simulator Sim runs the Sim_2^G simulators from top to bottom by randomly picking the initial $\widehat{\boldsymbol{x}}|_{\boldsymbol{I}}$. This completes the construction of our global simulator Sim. □

3.3 Expressing ε as a function of p

In this section, we show how all of the previously seen failure probabilities ε can be expressed as functions of the leakage probability p. The functional representation of ε will be instrumental in introducing the idea of Random Probing Expansion, which allows for arbitrarily reducing the failure event probability of any compiled circuit. In addition, this representation will be helpful to automatically verify the security of gadgets and circuits in the random probing model since computing the failure probability then amounts to computing coefficients of a polynomial in p.

We first derive an upper bound on the simulation failure probability as a function of the leakage probability p. We consider a compiled circuit $\widehat{C}$ composed of s wires labeled from 1 to s and a simulator SimAW as defined in Section 3.1. For any sub-set $W \subseteq [s]$ we denote by δ_W the value defined as follows:

$$\delta_W = \begin{cases} 1 & \text{if } \mathsf{SimAW}(\widehat{C}, W) = \perp, \\ 0 & \text{otherwise.} \end{cases}$$

The simulation failure probability ε in (3.3) can then be explicitly expressed as a function of p. Namely, we have $\varepsilon = f(p)$ with f defined for every $p \in [0,1]$ by:

$$f(p) = \sum_{W \subseteq [s]} \delta_W \cdot p^{|W|} \cdot (1-p)^{s-|W|} \ . \tag{3.5}$$

Letting c_i be the number of subsets $W \subseteq [s]$ of cardinality i for which $\delta_W = 1$, namely for which the simulation fails, we have $c_i = \sum_{|W|=i} \delta_W$ and hence (3.5) simplifies to

$$f(p) = \sum_{i=1}^{s} c_i \cdot p^i \cdot (1-p)^{s-i} \ . \tag{3.6}$$

For any circuit $\widehat{C}$ achieving t-probing security, the values δ_W with $|W| \leq t$ are equal to zero. Indeed, a circuit which is t-probing secure means that any set of at most t intermediate computations is independent of the secret values involved in the circuit. In other words, a simulator can perfectly simulate such a set without access to the secret values. Therefore, the corresponding c_i's are equal to zero, which implies the following simplification:

$$f(p) = \sum_{i=t+1}^{s} c_i \cdot p^i \cdot (1-p)^{s-i} \ .$$

Moreover, by definition, the coefficients c_i satisfy:

$$c_i \leqslant \binom{s}{i} \tag{3.7}$$

since we have exactly $\binom{s}{i}$ sets of wire labels of size i from a total of s wire labels. This observation finally leads to the following upper-bound for $f(p)$:

$$f(p) \ \leqslant \sum_{i=t+1}^{s} \binom{s}{i} \cdot p^i \cdot (1-p)^{s-i} \ .$$

An obvious goal is to try to construct circuits (p, ε)-random probing secure for ε as small as possible and p as big as possible. The circuit can then tolerate a large amount of leakage,

which would still be independent of the secret values. From the expression in x, we need to minimize the coefficients c_i in $f(p)$ to get small failure probabilities.

For any compiled circuit $\widehat{C}$ and any simulator defined as in Section 3.1, the computation of the function $f(p)$ for any probability p essentially amounts to computing the values of the coefficients c_i's appearing in (3.6). If no assumption is made on the circuit, this task seems complicated to carry out by hand. It may be checked that exhaustive testing of all the possible tuples of wires for a gadget with s wires has complexity lower bounded by 2^s. For now, we suppose we have access to algorithms that allow us to efficiently and automatically compute the coefficients of the function $\varepsilon = f(p)$. We will discuss automatic verification tools in more detail later in Chapter 7.

Let us take an example to illustrate such a function. We consider the well-known ISW multiplication gadget for $n = 2$ shares. The full description of the gadget is given in Algorithm 1 in Chapter 2.This gadget takes at input the 2-sharings (a_1, a_2) and (b_1, b_2) of a and b respectively, and outputs the 2-sharing

$$(c_1, c_2) = (a_1 \cdot b_1 + r_{1,2}, a_2 \cdot a_2 + r_{1,2} + a_1 \cdot b_2 + a_2 \cdot b_1)$$

where $r_{1,2}$ is a random value. The processing is composed of the following intermediate results, where each variable is assigned a wire:

$$\begin{array}{llll} m_{1,1} = a_1 * b_1 & m_{1,2} = a_1 * b_2 & m_{2,1} = a_2 * b_1 & m_{2,2} = a_2 * b_2 \\ tmp = m_{1,2} + r_{1,2} & r_{2,1} = tmp + m_{2,1} & c_1 = m_{1,1} + r_{1,2} & c_2 = m_{2,2} + r_{2,1} \end{array}$$

Recall that when the same variable is involved as input of several operations, a copy gadget (with 1 input and 2 output wires) is applied to duplicate it. Consequently, each new use of the same variable as input of an operation adds 2 wires to the final count of overall wires.

We can check that the circuit corresponding to the 2-share ISW gadget comprises 21 intermediate wires, excluding the 2 output wires c_1 and c_2. Since the gadget is 1-SNI (*c.f.* [10]) but not 2-SNI, every set with a single wire can be perfectly simulated without knowing the secrets. However, it is only the case for some pairs of wires. For instance, the set composed of two wire labels corresponding to variables $\{a_1, a_2\}$ cannot be perfectly simulated without knowing the secret a since the latter can be obtained from the combination of both share values. 51 such pairs of wires cannot be simulated. If we continue the test for the sets of cardinality from 3 to 21, we get the following list of 21 coefficients, computed with the IronMask verification tool described in Chapter 7:

```
[0, 51, 754, 4827, 18875, 52994, 115520, 203176, 293844, 352702,
352715, 293930, 203490, 116280, 54264, 20349, 5985, 1330, 210, 21, 1].
```

We can directly inject these coefficients in (3.6) to get the expression of $f(p)$ for the 2-share ISW multiplication gadget. This gadget is $(p, f(p))$-random probing secure for the computed function $f(p)$. Figure 3.2 shows the evolution of the value of $f(p)$ for an increasing p. For example, for a leakage rate of $p = 0.5$, we can check that a simulator has almost a 99.7% chance of being unable to perfectly simulate a leaking set of wires during the execution of the circuit. For a leakage rate of $p = 0.05$, this failure chance decreases to about 10%. Clearly, the higher the leakage rate, the more difficult it is to guarantee the independence of the leaking wires from the secret inputs, which is logical since more sensitive leakage gives more information about the secrets. The speed at which the failure probability increases with the increase of the leakage rate depends on the security of the constructed circuit and the underlying ε.

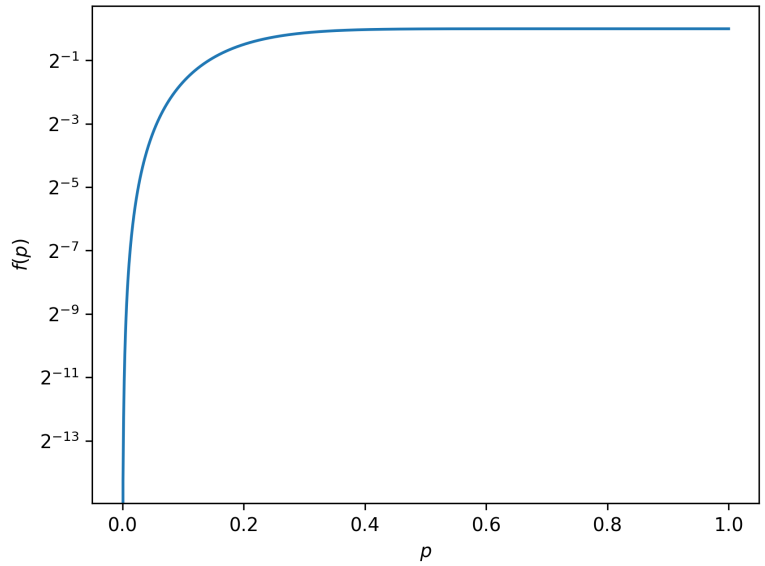

Figure 3.2: Evolution of the failure probability function $f(p)$ for the 2-share ISW multiplication gadget, for an increasing leakage rate p.

3.3.1 Failure function for Random Probing Composability

Recall the definition of (t, p, ε)-random probing composability (Definition 8). In this definition, we can express ε in the same way as a function of the leakage rate p. We need to consider a set of wire labels and a set of t output wires for each circuit output. Namely, for each possible collection of sets $\boldsymbol{J} = (J_1, \ldots, J_m)$ such that $\forall\ i \in [m],\ J_i \subseteq [n]$ and $|J_i| \leq t$, and each set W of internal wire labels, we need to consider the simulation of the wires in W and $\widehat{\boldsymbol{y}}|_{\boldsymbol{J}}$. An additional constraint is that the simulator must be able to perfectly simulate all the wires involved while having access to at most t shares of each input sharing. If we denote $c_i^{\boldsymbol{J}}$ the number of such sets W for which the simulation of W along with $\widehat{\boldsymbol{y}}|_{\boldsymbol{J}}$ fails, then we obtain the expression as follows:

$$f(p) = \sum_{i=1}^{s} \max_{\boldsymbol{J}} c_i^{\boldsymbol{J}} \cdot p^i \cdot (1-p)^{s-i} \ . \tag{3.8}$$

Interestingly, we can demonstrate that n-share gadgets satisfying the t-SNI notion for $t < n$ are also random probing composable for specific values we explicit in the following proposition.

Proposition 1. *Let n, ℓ and t be positive integers and let G be a gadget from $(\mathbb{K}^n)^\ell$ to $\mathbb{K}^n$. If G is t-SNI, then it is also $(\lfloor t/2 \rfloor, p, \varepsilon)$-RPC for any probability p and ε satisfying:*

$$\varepsilon \leq \sum_{i=\lfloor \frac{t}{2}+1 \rfloor}^{s} \binom{s}{i} p^i (1-p)^{s-i} \ , \tag{3.9}$$

where s is the number of wires in G.

Proof. Since G is t-SNI, there exist two simulators Sim_1^G and Sim_2^G satisfying (2.3) and (2.4) for any $J \subseteq [n]$ and any W satisfying $|W| + |J| \leqslant t$. This is in particular true for every W and every J both of cardinality lower than or equal to $\lfloor t/2 \rfloor$. For every set W such that $|W| > \lfloor t/2 \rfloor$ and every set J such that $|J| \leqslant \lfloor t/2 \rfloor$, we consider the worst case. We modify simulator Sim_1^G to return the full set of input indices (*i.e.* $\boldsymbol{I} = [n]^\ell$). Then, the second simulator Sim_2^G is simply augmented to perfectly simulate $(\mathsf{AssignWires}(G, W, \widehat{\boldsymbol{x}})\ ,\ \widehat{\boldsymbol{y}}|_J)$ from the full knowledge of the gadget inputs (which is trivially possible). By construction, for any J with $|J| \leqslant \lfloor t/2 \rfloor$,

the output $\boldsymbol{I} = (I_1, \ldots, I_\ell)$ of $\mathsf{Sim}_1^G(W, J)$ contains at least one I_j with cardinality greater than $\lfloor t/2 \rfloor$ only when W has cardinality strictly greater than $\lfloor t/2 \rfloor$ (and in this case all the I_j's have full cardinality $[n]$). Hence, the probability $\Pr\big((|I_1| > \lfloor t/2 \rfloor) \vee \ldots \vee (|I_\ell| > \lfloor t/2 \rfloor)\big)$, when J is a given set with $|J| \leqslant \lfloor t/2 \rfloor$ and W is the output of $\mathsf{LeakingWires}(G, p)$, satisfies (3.9), which concludes the proof. □

As an illustration of the proposition, let us consider again the 3-share ISW multiplication gadget satisfying 2-SNI. Recall that the gadget has 57 internal wires, excluding the 3 output wires. From Proposition 1, this gadget is also $(1, p, \varepsilon)$-RPC for any probability p and such that

$$\varepsilon \leq \sum_{i=2}^{57} \binom{s}{i} p^i (1-p)^{57-i}.$$

Using a verification tool like IronMask (we will discuss automatic verification tools in more detail later in Chapter 7), we get a tighter representation of the function as

$$\varepsilon = 415 \cdot p^2 \cdot (1-p)^{55} + \mathcal{O}(p^3) \ .$$

Remark 2. *In a follow-up work to our results [32], the authors introduce a tighter composition approach in the random probing model, which splits the failure function $f(p)$ into several failure functions for each possible set of leaking output wires and input wires for the simulation. In a nutshell, their composition strategy relies on using what they call a probe distribution table or PDT. Each entry in the table corresponds to a failure function $f_{\mathcal{I},\mathcal{O}}(p)$ where $\mathcal{O}$ is the set of output shares leaking in addition to the intermediate variables, and $\mathcal{I}$ is the set of input shares necessary for a perfect simulation of the intermediate probes and the output share in $\mathcal{O}$. The PDT for an n-share gadget with ℓ inputs and m outputs has $n \times \ell$ rows and $n \times m$ columns. The authors reason on the nature of the composition, i.e., parallel or sequential composition, and can get better security bounds by composing these PDTs. Meanwhile, the sizes of the PDTs increase exponentially with the circuit size, unlike our RPC notion. In addition, their composition approach is not yet extended to the random probing expansion approach, which we will introduce in the next chapter. An interesting future work would be integrating their composition approach using PDTs with our expansion technique.*

3.4 Conclusion

In this chapter, we recalled the definition of random probing security and proposed a way of composing different RP secure gadgets into a circuit with global RP security. The composition requires that the gadgets satisfy the RP composability notion. In addition, we presented a practical way of computing the failure event probability of RP security as a function of the leakage probability p, which allows for automatic verification of gadgets' security in the random probing model.

While the RP composability ensures global RP security in any circuit, it does not help to increase this security level. Ideally, we would like a circuit to tolerate the highest leakage rate possible p with the lowest simulation failure probability ε. In the next chapter, we tackle this problem by introducing the random probing expansion strategy, which helps achieve any desired security level regarding the failure event probability at the expense of a complexity overhead, which we also analyze.

Chapter 4

Random Probing Expansion

Constructing random probing secure circuits is a challenging task. We want to achieve the highest security level possible, in other words, tolerate the maximum leakage probability p with the lowest failure probability ε. Indeed, for a fixed number of shares n, a secure gadget can only achieve a bounded value for the failure probability. A solution would be to increase the number of shares, meaning that the gadgets' construction should be generic and scale well with the number of shares in the circuit. In addition, we want to quantify the exact failure probability achieved for a given n, which can be done either by getting generic formulas for ε or by using automatic verification tools. Both solutions are difficult to handle when we have complex and large gadgets.

This chapter discusses a powerful tool: the *random probing expanding compiler*. The *expanding compiler* allows for achieving arbitrarily large security levels in the random probing model, starting from small gadgets using the *expansion strategy*. We will see later that small gadgets for small numbers of shares tend to tolerate better leakage probabilities but have higher failure probabilities than larger gadgets. The *expansion strategy* aims to take the best of both worlds: use small gadgets which tolerate high leakage rates and expand them into larger gadgets which tolerate the same leakage rates but have much smaller failure probabilities.

In [3], Ananth, Ishai and Sahai first proposed constructing random probing secure circuit compilers with a gadget *expansion strategy*. Such a strategy was previously used in multi-party computation (MPC) with different but close security goals [37, 58]. Note that such an approach is called *composition* in [3] since it roughly composes a base circuit compiler several times. We prefer the terminology of *expansion* to avoid confusion with the usual notion of composition for gadgets in the masking literature.

Our works introduce an instantiation of the *expansion strategy* relying on atomic gadgets leading to simple and efficient constructions. In contrast, the construction of [3] relies on a t-out-n secure MPC protocol in the passive security model. We introduce a security notion we call *random probing expandability*, allowing gadgets to be used in the expanding compiler. We show that this notion can be easily verified using automatic verification tools, making it convenient to test gadgets for a reasonable order. We provide a detailed analysis and discussion about this strategy, its security in the random probing model and its asymptotic complexity. Finally, we show how we can construct generic gadgets that achieve the best trade-offs between complexity and security levels and instantiate them for a small number of shares to demonstrate actual security levels achieved for some cryptographic algorithms like the AES encryption scheme as a proof-of-concept.

In the following, we mostly focus on gadgets with at most 2 inputs and 2 outputs, which already covers addition, multiplication, copy and random generation gadgets, allowing us to

model any circuit with basic operations. The contributions in this chapter are published in [18, 22].

4.1 Expansion Strategy

The basic principle of the gadget expansion strategy is as follows. Assume we have an arithmetic circuit basis $\mathbb{B} = \{g : \mathbb{K}^\ell \to \mathbb{K}^m\}$ of arithmetic operations composing any circuit (in most cases, we consider $\mathbb{B}$ to consist of addition, multiplication and copy gates). Assume that we also have the corresponding n-share gadgets G_g for $g \in \mathbb{B}$ and denote CC the standard circuit compiler for these base gadgets. We can derive new n^2-share gadgets by simply applying CC to each gadget: $G_g^{(2)} = \mathsf{CC}(G_g)$ for each $g \in \mathbb{B}$. Let us recall that this process simply consists in replacing each gate g in the original gadget by G_g, and by replacing each wire by n wires carrying a sharing of the original wire. Doing so, we obtain n^2-share gadgets for each base gate in $\mathbb{B}$ over $\mathbb{K}$. This process can be iterated an arbitrary number of times, say k, to an input circuit C:

$$C \xrightarrow{\mathsf{CC}} \widehat{C}_1 \xrightarrow{\mathsf{CC}} \cdots \xrightarrow{\mathsf{CC}} \widehat{C}_k \ .$$

The first output circuit $\widehat{C}_1$ is the original circuit in which each gate is replaced by a base gadget G_g. The second output circuit $\widehat{C}_2$ is the original circuit C in which each gate is replaced by an n^2-share gadget $G_g^{(2)}$ as defined above. Equivalently, $\widehat{C}_2$ is the circuit $\widehat{C}_1$ in which each gate is replaced by a base gadget. In the end, the output circuit $\widehat{C}_k$ is hence the original circuit C in which each gate has been replaced by a k-expanded gadget and each wire has been replaced by n^k wires carrying an (n^k)-linear sharing of the original wire. The underlying compiler is called *expanding circuit compiler* which is formally defined hereafter.

Definition 9 (Expanding Circuit Compiler)**.** *Let* CC *be the standard circuit compiler for an arithmetic circuit basis* $\mathbb{B} = \{g : \mathbb{K}^\ell \to \mathbb{K}^m\}$ *with sharing order* n *and base gadgets* $\{G_g\}_{g\in\mathbb{B}}$*. The* expanding circuit compiler *with* expansion level k *and base compiler* CC *is the circuit compiler* $(\mathsf{CC}^{(k)}, \mathsf{Enc}^{(k)}, \mathsf{Dec}^{(k)})$ *satisfying the following:*

1. *The input encoding* $\mathsf{Enc}^{(k)}$ *is an* (n^k)*-linear encoding.*
2. *The output decoding* Dec *is the* (n^k)*-linear decoding mapping.*
3. *The circuit compilation is defined as*

$$\mathsf{CC}^{(k)}(\cdot) = \underbrace{\mathsf{CC} \circ \mathsf{CC} \circ \cdots \circ \mathsf{CC}}_{k \text{ times}}(\cdot)$$

The goal of the expansion strategy in the context of random probing security is to replace the leakage probability p of a wire in the original circuit by the failure event probability ε in the subsequent gadget simulation. If this simulation fails then one needs the full input sharing for the gadget simulation, which corresponds to leaking the corresponding wire value in the base case. The security is thus amplified by replacing the probability p in the base case by the probability ε (assuming that we have $\varepsilon < p$). If the failure event probability ε can be upper bounded by some function of the leakage probability: $\varepsilon < f(p)$ for every leakage probability $p \in [0, p_{\max}]$ for some $p_{\max} < 1$, then the expanding circuit compiler with expansion level k shall result in a security amplification as

$$p = \varepsilon_0 \xrightarrow{f} \varepsilon_1 \xrightarrow{f} \cdots \xrightarrow{f} \varepsilon_k = f^{(k)}(p) \ ,$$

which for an adequate function f (*e.g.* $f : p \mapsto p^2$) provides exponential security. In order to get such a security expansion, the gadgets must satisfy a stronger notion than the composability

notion introduced in Chapter 3, which we call *random probing expandability*; see Section 4.2 below.

Since in our case, we focus on addition, multiplication and copy gadgets, we limit our definitions to $\ell \leq 2$ and $m \leq 2$ for the sake of simplicity, but all notions can be generalized to any ℓ and m.

4.2 Expansion Security

In evaluating random probing composability, let us recall that the failure event in the simulation of a gadget means that more than t shares from one of its inputs are necessary to complete a perfect simulation. For a gadget to be expandable, we need slightly stronger notions than random probing composability. As a first requirement, a two-input gadget should have a failure probability independent for each input. This is because, in the base case, each input wire to a gate leaks independently. On the other hand, in case of a failure event in the child gadget during the expansion, the overall simulator (of the k-expanded gadget) should be able to produce a perfect simulation of the entire output (that is, the entire input for which the failure occurs). Hence, the overall simulator is given the clear output (obtained from the base case simulation) plus any set of $n-1$ output shares. In this case, whenever the set J is of cardinal greater than t, the gadget simulator can replace it with any set J' of cardinal $n-1$. More details are given hereafter.

Definition 10 (Random Probing Expandability for 2-to-1 gadgets)**.** *Let $f : \mathbb{R} \to \mathbb{R}$. An n-share gadget $G : \mathbb{K}^n \times \mathbb{K}^n \to \mathbb{K}^n$ is* (t, f)-random probing expandable (RPE) *if there exists a deterministic algorithm Sim_1^G and a probabilistic algorithm Sim_2^G such that for every input $(\widehat{x}, \widehat{y}) \in \mathbb{K}^n \times \mathbb{K}^n$, for every set $J \subseteq [n]$ and for every $p \in [0, 1]$, the random experiment*

$$
\begin{aligned}
&W \leftarrow \mathsf{LeakingWires}(G, p)\\
&(I_1, I_2, J') \leftarrow \mathsf{Sim}_1^G(W, J)\\
&\mathit{out} \leftarrow \mathsf{Sim}_2^G(W, J', \widehat{x}|_{I_1}, \widehat{y}|_{I_2})
\end{aligned}
$$

ensures that

1. *the failure events $\mathbb{F}_1 \equiv \big(|I_1| > t\big)$ and $\mathbb{F}_2 \equiv \big(|I_2| > t\big)$ verify*

$$\Pr(\mathbb{F}_1) = \Pr(\mathbb{F}_2) = \varepsilon \quad \textit{and} \quad \Pr(\mathbb{F}_1 \wedge \mathbb{F}_2) = \varepsilon^2 \tag{4.1}$$

   with $\varepsilon = f(p)$ (in particular $\mathbb{F}_1$ and $\mathbb{F}_2$ are mutually independent),

2. *J' is such that $J' = J$ if $|J| \leq t$ and $J' \subseteq [n]$ with $|J'| = n - 1$ otherwise,*

3. *the output distribution satisfies*

$$\mathit{out} \overset{id}{=} \big(\mathsf{AssignWires}(G, W, (\widehat{x}, \widehat{y}))\ ,\ \widehat{z}|_{J'}\big) \tag{4.2}$$

where $\widehat{z} = G(\widehat{x}, \widehat{y})$.

The RPE notion can be simply extended to gadgets with 2 outputs: the Sim_1^G simulator takes two sets $J_1 \subseteq [n]$ and $J_2 \subseteq [n]$ as input and produces two sets J_1' and J_2' satisfying the same property as J' in the above definition (w.r.t. J_1 and J_2). The Sim_2^G simulator must then produce an output including $\widehat{z}_1|_{J_1'}$ and $\widehat{z}_2|_{J_1'}$ where $\widehat{z}_1$ and $\widehat{z}_2$ are the output sharings. The RPE notion can also be simply extended to gadgets with a single input: the Sim_1^G simulator produces a single set I so that the failure event ($|I| > t$) occurs with probability lower than ε (and the Sim_2^G simulator is then simply given $\widehat{x}|_I$ where $\widehat{x}$ is the single input sharing). For the sake of completeness, we provide the RPE definition for the 1-to-2 case below, but we only focus on 2-to-1 and 1-to-2 gadgets.

Definition 11 (Random Probing Expandability for 1-to-2 gadgets)**.** *Let $f = \mathbb{R} \to \mathbb{R}$. An n-share gadget $G : \mathbb{K}^n \to \mathbb{K}^n \times \mathbb{K}^n$ is (t, f)-*random probing expandable (RPE) *if there exists a deterministic algorithm Sim_1^G and a probabilistic algorithm Sim_2^G such that for every input $\widehat{x} \in \mathbb{K}^n$, for every pair of sets $J_1 \subseteq [n]$ and $J_2 \subseteq [n]$, and for every $p \in [0, 1]$, the random experiment*

$$\begin{aligned} W &\leftarrow \mathsf{LeakingWires}(G, p) \\ (I, J_1', J_2') &\leftarrow \mathsf{Sim}_1^G(W, J_1, J_2) \\ \mathit{out} &\leftarrow \mathsf{Sim}_2^G(W, J_1', J_2', \widehat{x}|_I) \end{aligned}$$

ensures that

1. *the failure event probability satisfies $\Pr\big(|I| > t\big) \leq \varepsilon$ with $\varepsilon = f(p)$,*
2. *the set J_1' is such that $J_1' = J_1$ if $|J_1| \leq t$ and $J_1' \subseteq [n]$ with $|J_1'| = n - 1$ otherwise,*
3. *the set J_2' is such that $J_2' = J_2$ if $|J_2| \leq t$ and $J_2' \subseteq [n]$ with $|J_2'| = n - 1$ otherwise,*
4. *the output distribution satisfies*

$$\mathit{out} \overset{id}{=} \big(\mathsf{AssignWires}(G, W, \widehat{x})\ ,\ \widehat{y}|_{J_1'}\ ,\ \widehat{z}|_{J_2'}\big) \tag{4.3}$$

where $(\widehat{y}, \widehat{z}) = G(\widehat{x})$.

It is not hard to check that the above expandability notions are stronger than the composability notion introduced in Section 3.2. Formally, we have the following reduction.

Proposition 2. *Let $f = \mathbb{R} \to \mathbb{R}$ and $n \in \mathbb{N}$. Let G be an n-share gadget with $\ell \leq 2$ inputs. If G is (t, f)-RPE then G is (t, f')-RPC, with $f'(\cdot) = \ell \cdot f(\cdot)$.*

Proof. We consider a (t, f)-RPE n-share gadget $G : \mathbb{K}^n \times \mathbb{K}^n \to \mathbb{K}^n$. The $(t, 2 \cdot f)$-random composability property is directly implied by the (t, f)-random probing expandability by making use of the exact same simulators and observing that

$$\Pr\big((|I_1| > t) \vee (|I_2| > t)\big) \leq \Pr(|I_1| > t) + \Pr(|I_2| > t) = 2 \cdot \varepsilon.$$

The case of $\ell = 1$ input gadgets is even more direct. □

Definition 10 (and Definition 11) of random probing expandability is valid for base gadgets. For level-k gadgets $G^{(k)} = \mathsf{CC}^{(k-1)}(G)$ where $G \in \{G_g\}_{g \in \mathbb{B}}$ is a base gadget, we provide a generalized definition of random probing expandability.

Adequate subsets of $[n^k]$. We first define the notion of "adequate" subsets of $[n^k]$, instead of only bounded subsets. For this we define recursively a family $S_k \in \mathcal{P}([n^k])$, where $\mathcal{P}([n^k])$ denotes the set of all subsets of $[n^k]$, as follows:

$$\begin{aligned} S_1 &= \{I \in [n],\ |I| \leq t\} \\ S_k &= \{(I_1, \ldots, I_n) \in (S_{k-1} \cup [n^{k-1}])^n, I_j \in S_{k-1}\ \forall\ j \in [1, n] \text{ except at most } t\} \end{aligned}$$

In other words, a subset I belongs to S_k if among the n subset parts of I, at most t of them are full, while the other ones recursively belong to S_{k-1}; see Figure 4.1 below for an illustration with $n = 3$ and $t = 1$.

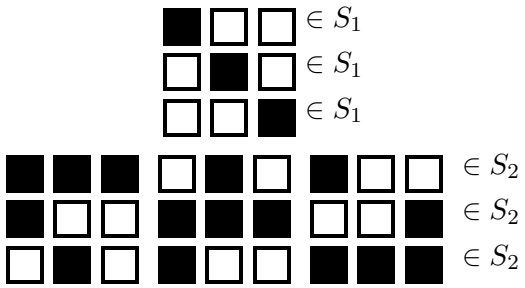

Figure 4.1: Illustration of all elements of S_1 and some elements of S_2, for $n = 3$ and $t = 1$.

Generalized definition of Random Probing Expandability. We generalize Definition 10 as follows. At level k the input sets I_1 and I_2 must belong to S_k, otherwise we have a failure event. As in Definition 10, the simulation is performed for an output subset J' with $J' = J$ if $J \in S_k$, otherwise $J' = [n^k] \setminus \{j^\star\}$ for some $j^\star \in [n^k]$.

Definition 12 (Random Probing Expandability with $\{S_k\}_{k\in\mathbb{N}}$)**.** *Let $f : \mathbb{R} \to \mathbb{R}$ and $k \in \mathbb{N}$. An n^k-share gadget $G : \mathbb{K}^{n^k} \times \mathbb{K}^{n^k} \to \mathbb{K}^{n^k}$ is (S_k, f)-*random probing expandable (RPE) *if there exists a deterministic algorithm Sim_1^G and a probabilistic algorithm Sim_2^G such that for every input $(\widehat{x}, \widehat{y}) \in \mathbb{K}^{n^k} \times \mathbb{K}^{n^k}$, for every set $J \in S_k \cup [n^k]$ and for every $p \in [0, 1]$, the random experiment*

$$\begin{aligned} W &\leftarrow \mathsf{LeakingWires}(G, p) \\ (I_1, I_2, J') &\leftarrow \mathsf{Sim}_1^G(W, J) \\ \mathit{out} &\leftarrow \mathsf{Sim}_2^G(W, J', \widehat{x}|_{I_1}, \widehat{y}|_{I_2}) \end{aligned}$$

ensures that

1. *the failure events $\mathbb{F}_1 \equiv \big(I_1 \notin S_k\big)$ and $\mathbb{F}_2 \equiv \big(I_2 \notin S_k\big)$ verify*

$$\Pr(\mathbb{F}_1) = \Pr(\mathbb{F}_2) = \varepsilon \quad \textit{and} \quad \Pr(\mathbb{F}_1 \wedge \mathbb{F}_2) = \varepsilon^2 \tag{4.4}$$

   with $\varepsilon = f(p)$ (in particular $\mathbb{F}_1$ and $\mathbb{F}_2$ are mutually independent),

2. *the set J' is such that $J' = J$ if $J \in S_k$, and $J' = [n^k] \setminus \{j^\star\}$ for some $j^\star \in [n^k]$ otherwise,*

3. *the output distribution satisfies*

$$\mathit{out} \overset{id}{=} \big(\mathsf{AssignWires}(G, W, (\widehat{x}, \widehat{y}))\ ,\ \widehat{z}|_{J'}\big) \tag{4.5}$$

where $\widehat{z} = G(\widehat{x}, \widehat{y})$.

The notion of random probing expandability from Definition 12 naturally leads to the statement of the following main theorem.

Theorem 2. *Let $n \in \mathbb{N}$ and $f : \mathbb{R} \to \mathbb{R}$. Let $\{G_g\}_{g\in\mathbb{B}}$ be n-share gadgets for $g \in \mathbb{B}$ over $\mathbb{K}$, where each $g \in \mathbb{B}$ is of at most 2 inputs and 2 outputs. Let CC be the standard circuit compiler with sharing order n and base gadgets $\{G_g\}_{g\in\mathbb{B}}$. Let $\mathsf{CC}^{(k)}$ be the expanding circuit compiler with base compiler CC. If the base gadgets $\{G_g\}_{g\in\mathbb{B}}$, are (t, f)-RPE then, $G_g^{(k)} = \mathsf{CC}^{(k-1)}(G_g)$ for $g \in \mathbb{B}$ are $(S_k, f^{(k)})$-RPE, n^k-share gadgets for the addition, multiplication and copy on $\mathbb{K}$.*

Proof. The proof of the theorem relies on what we shall call the *assignment expansion* property. Throughout the proof we shall denote $\varepsilon_k = f^{(k)}(p)$. We call *level-k gadget* a gadget that has been expanded $k-1$ times $G^{(k)} = \mathsf{CC}^{(k-1)}(G)$ where G is a base gadget (or a level-1 gadget) among G_g for $g \in \mathbb{B}$.

We proceed by induction to show that the level-k gadgets are $(S_k, f^{(k)})$-RPE. The base case is one of the theorem hypotheses, namely the base gadgets $\{G_g\}_{g\in\mathbb{B}}$ (*i.e.* the level-1 gadgets) are (t, f)-RPE, which is equivalent to $(\mathcal{S}_1, f)$-RPE. We must then show the induction step: assuming that the level-k gadgets are $(S_k, f^{(k)})$-RPE, show that the level-$(k+1)$ gadgets are $(S_{k+1}, f^{(k+1)})$-RPE. For the sake of simplicity, we depict our proof by assuming that all the gadgets are 2-to-1 gadget (which is actually not the case for copy gadgets). The proof mechanism for the general case is strictly similar but heavier on the form.

In order to show that $G^{(k+1)}$ is $(S_{k+1}, f^{(k+1)})$-RPE we must construct two simulators $\mathsf{Sim}_1^{G^{(k+1)}}$ and $\mathsf{Sim}_2^{G^{(k+1)}}$ that satisfy the conditions of Definition 28 for the set of subsets S_{k+1}. More precisely, we must construct two simulators $\mathsf{Sim}_1^{G^{(k+1)}}$ and $\mathsf{Sim}_2^{G^{(k+1)}}$ such that for every $(\widehat{x}^*, \widehat{y}^*) \in \mathbb{K}^{n^{k+1}} \times \mathbb{K}^{n^{k+1}}$, and for every set $J^* \in S_{k+1} \cup [n^{k+1}]$, the random experiment

$$\begin{aligned} &W^* \leftarrow \mathsf{LeakingWires}(G^{(k+1)}, p) \\ &(I_1^*, I_2^*, J^{*\prime}) \leftarrow \mathsf{Sim}_1^G(W^*, J^*) \\ &\mathit{out} \leftarrow \mathsf{Sim}_2^G(W^*, J^*, \widehat{x}^*|_{I_1^*}, \widehat{y}^*|_{I_2^*}) \end{aligned}$$

ensures that

1. the failure events $\mathbb{F}_1^* \equiv \big(I_1^* \notin S_{k+1}\big)$ and $\mathbb{F}_2^* \equiv \big(I_2^* \notin S_{k+1}\big)$ verify
$$\Pr(\mathbb{F}_1^*) = \Pr(\mathbb{F}_2^*) = \varepsilon_{k+1} \quad \text{and} \quad \Pr(\mathbb{F}_1^* \wedge \mathbb{F}_2^*) = \varepsilon_{k+1}^2 \tag{4.6}$$
2. the set $J^{*\prime}$ is such that $J^{*\prime} = J^*$ if $J^* \in S_{k+1}$ and $J^{*\prime} = [n^{k+1}] \setminus \{\jmath^\star\}$ otherwise,
3. the output distribution satisfies
$$\mathit{out} \overset{\text{id}}{=} \big(\mathsf{AssignWires}(G, W, (\widehat{x}, \widehat{y}))\ ,\ \widehat{z}|_{J^{*\prime}}\big) \tag{4.7}$$

where $\widehat{z} = G^{(k+1)}(\widehat{x}, \widehat{y})$.

We distinguish two cases: either $J^* \in S_{k+1}$ (normal case), or $J^* = [n^{k+1}]$ (saturated case).

Normal case: $J^* \in S_{k+1}$. By definition of the expanding compiler, we have that a level-$(k+1)$ gadget $G^{(k+1)}$ is obtained by replacing each gate of the base gadget by the corresponding level-k gadget and by replacing each wire of the base gadget by n^k wires carrying a (n^k)-linear sharing of the original wire. In particular $G^{(k+1)}$ has n^{k+1} output wires which can be split in n groups of n^k wires, each group being the output of a different $G^{(k)}$ gadget. We split the set J^* accordingly so that $J^* = J_1^* \cup \cdots \cup J_n^*$, where each set J_i^* pertains to the ith group of output wires. By definition of S_k, since $J^* \in S_{k+1}$, we must have $J_i^* \in S_k$ for all $1 \le i \le n$, except at most t of them for which $J_i^* = [n^k]$. We define J_{base} as the set of indexes i such that $J_i^* \notin S_k$. Therefore we must have $|J_{\text{base}}| \le t$.

We first describe the simulator $\mathsf{Sim}_1^{G^{(k+1)}}$ that takes the leaking wires W^* and the output wires $J^* \in S_{k+1}$ to be simulated and produce the sets $I_1^* \subseteq [n^{k+1}]$ and $I_2^* \subseteq [n^{k+1}]$ of required inputs. The simulator $\mathsf{Sim}_1^{G^{(k+1)}}$ starts by defining a set W_{base} which is initialized to $\emptyset$; this will correspond to the set of leaking wires for the base gadget. Then the simulation goes through all the level-k gadgets composing $G^{(k+1)}$ from bottom to top *i.e.* starting with the level-k gadgets producing the output sharing up to the level-k gadgets processing the input sharings.

Let us denote by $\{G_j^{(k)}\}_j$ these level-k gadgets. For each $G_j^{(k)}$, one runs the simulator Sim_1 from the $(S_k, f^{(k)})$-RPE property on input W_j and J_j defined as follows. The set of leaking wires W_j is defined as the subset of W^* corresponding to the wires of $G_j^{(k)}$. For the gadgets $G_j^{(k)}$ on the bottom layer, the set J_j is set to one of the J_i^* (with indices scaled to range in $[n^k]$). For all the other gadgets $G_j^{(k)}$ (which are not on the bottom layer), the set J is defined as the set I_1 or I_2 output from Sim_1 for the child gadget ${G^{(k)}}_{j'}$ (for which Sim_1 has already been run).

Whenever a failure event occurs for a $G_j^{(k)}$ gadget, namely when the set I (either I_1 or I_2) output from Sim_1 is such that $I \notin S_k$, we add the index of the wire corresponding to this input in the base gadget G to the set W_{base}. Once the Sim_1 simulations have been run for all the $G_j^{(k)}$ gadgets, ending with the top layers, we get the final sets I corresponding to the input shares. Each of these sets corresponds to an n^k-sharing as input of a $G_j^{(k)}$ gadget, which corresponds to a wire as input of the base gadget among the $2 \cdot n$ wires carrying the two input n-sharings of the base gadget. We denote by $I_{1,1}^*, \ldots, I_{1,n}^*$ and $I_{2,1}^*, \ldots, I_{2,n}^*$ the corresponding sets so that defining

$$I_1^* = I_{1,1}^* \cup \ldots \cup I_{1,n}^* \quad \text{and} \quad I_2^* = I_{2,1}^* \cup \ldots \cup I_{2,n}^* \ , \tag{4.8}$$

the tuple $\widehat{x}^*|_{I_1^*}$ and $\widehat{y}^*|_{I_2^*}$ contains the shares designated by the final I sets.

At the end of the $\mathsf{Sim}_1^{G^{(k+1)}}$ simulation, the set W_{base} contains all the labels of wires in the base gadget G for which a failure event has occurred in the simulation of the corresponding $G_j^{(k)}$ gadget. Thanks to the $(S_k, f^{(k)})$-RPE property of these gadgets, the failure events happen (mutually independently) with probability ε_k which implies

$$W_{\text{base}} \overset{\text{id}}{=} \mathsf{LeakingWires}(G, \varepsilon_k) \tag{4.9}$$

Recall that $|J_{\text{base}}| \leq t$. We can then run Sim_1^G to obtain:

$$(I_{1,\text{base}}, I_{2,\text{base}}) = \mathsf{Sim}_1^G(W_{\text{base}}, J_{\text{base}}) \ . \tag{4.10}$$

For all $1 \leq i \leq n$, if $i \in I_{1,\text{base}}$, we force $I_{1,i}^* \leftarrow [n^k]$, so that the corresponding i-th input wire of the base gadget can be computed from the corresponding input wires in $I_{1,i}^*$. The simulator $\mathsf{Sim}_1^{G^{(k+1)}}$ then returns (I_1^*, I_2^*) as output.

The (t, f)-RPE property of the base gadget G implies that the *base failure events* $|I_{1,\text{base}}| = n$ and $|I_{2,\text{base}}| = n$ are ε_{k+1}-mutually unlikely, where $\varepsilon_{k+1} = f(\varepsilon_k)$. We argue that for all $1 \leq i \leq n$, $I_{1,i}^* \notin S_k \iff i \in I_{1,\text{base}}$. Namely if a failure event has occurred for a set $I_{1,i}^*$ (*i.e.* $I_{1,i}^* \notin S_k$) then we must have $i \in I_{1,\text{base}}$. Indeed, if a failure event has occurred for a set $I_{1,i}^*$ then the label of the ith input wire (for the first sharing) of the base gadget G has been added to W_{base} and Sim_1^G has no choice but to include this index to the set $I_{1,\text{base}}$ so that Sim_2^G can achieve a perfect simulation of the wire assignment (as required by the RPE property of G). Moreover if $i \in I_{1,\text{base}}$ then by construction we have set $I_{1,i}^* = [n^k]$ and therefore $I_{1,i}^* \notin S_k$. This implies that if $|I_{1,\text{base}}| \leq t$ then $I_1^* \in S_{k+1}$ (and the same happens for I_2^* w.r.t. $I_{2,\text{base}}$). We deduce that the failure events $\mathbb{F}_1^*$ and $\mathbb{F}_2^*$ are also ε_{k+1}-mutually unlikely, as required by the $(S_{k+1}, f^{(k+1)})$-RPE property of $G^{(k+1)}$.

We now describe the simulator $\mathsf{Sim}_2^{G^{(k+1)}}$ that takes as input $\widehat{x}^*|_{I_1^*}$ and $\widehat{y}^*|_{I_2^*}$ and produces a perfect simulation of $\big(\mathsf{AssignWires}(G^{(k+1)}, W^*, (\widehat{x}^*, \widehat{y}^*)), \widehat{z}|_{J^*}\big)$ where $\widehat{z} = G^{(k+1)}(\widehat{x}, \widehat{y})$. Let $\widehat{x}^b$ and $\widehat{y}^b$ denote the n-linear sharings obtained by applying the linear decoding to each group of n^k shares in $\widehat{x}^*$ and $\widehat{y}^*$, so that the elements of $\widehat{x}^b$ and $\widehat{y}^b$ correspond to the input wires in

the base gadget G. The assignment expansion property implies that a perfect assignment of the wires of $G^{(k+1)}$ on input $\widehat{x}^*$ and $\widehat{y}^*$ can be derived from an assignement of the wires of the base gadget G on input $\widehat{x}^b$ and $\widehat{y}^b$. The simulator makes use of this property by first running

$$out_{\text{base}} \leftarrow \mathsf{Sim}_2^G(W_{\text{base}}, J_{\text{base}}, \widehat{x}^b|_{I_{1,\text{base}}}, \widehat{y}^b|_{I_{2,\text{base}}}) \ , \tag{4.11}$$

Note that the input values $\widehat{x}^b|_{I_{1,\text{base}}}$ and $\widehat{y}^b|_{I_{2,\text{base}}}$ can be obtained from the corresponding shares in I_1^* and I_2^*. Thanks to the (t, f)-RPE property of G and by construction of $I_{1,\text{base}}$ and $I_{2,\text{base}}$, this outputs a distribution satisfying

$$out_{\text{base}} \overset{\text{id}}{=} \left(\mathsf{AssignWires}(G, W_{\text{base}}, (\widehat{x}^b, \widehat{y}^b)),\ \widehat{z}^b|_{J_{\text{base}}}\right) \tag{4.12}$$

The simulator then goes through all the $G_j^{(k)}$ gadgets from input to output and for each of them runs the simulator Sim_2 of the RPE property on inputs W_j, J_j, $\widehat{x}|_{I_1}$ and $\widehat{y}|_{I_2}$ where W_j and J_j are the sets from the first phase of the simulation for the gadget $G_j^{(k)}$, I_1 and I_2 are the corresponding sets produced by the Sim_1 simulator for $G_j^{(k)}$, and $\widehat{x}$ and $\widehat{y}$ are the inputs of $G_j^{(k)}$ in the evaluation of $G^{(k+1)}(\widehat{x}^*, \widehat{y}^*)$. Provided that the partial inputs $\widehat{x}|_{I_1}$ and $\widehat{y}|_{I_2}$ are perfectly simulated, this call to Sim_2 produces a perfect simulation of $\left(\mathsf{AssignWires}(G_j^{(k)}, W_j, (\widehat{x}, \widehat{y}), \widehat{z}|_{J_j}\right)$ where $\widehat{z} = G_j^{(k)}(\widehat{x}, \widehat{y})$. In order to get perfect simulations of the partial inputs $\widehat{x}|_{I_1}$ and $\widehat{y}|_{I_2}$, the simulator proceeds as follows. For the top layer of $G^{(k)}$ gadgets (the ones processing the input shares) the shares $\widehat{x}|_{I_1}$ and $\widehat{y}|_{I_2}$ can directly be taken from the inputs $\widehat{x}^*|_{I_1^*}$ and $\widehat{y}^*|_{I_2^*}$. For the next gadgets the shares $\widehat{x}|_{I_1}$ and $\widehat{y}|_{I_2}$ match the shares $\widehat{z}|_J$ output from the call to Sim_2 for a parent gadget. The only exception occurs in case of a failure event.

In that case the simulation needs the full input $\widehat{x} = (x_1, \ldots, x_{n^k})$ (and/or $\widehat{y} = (y_1, \ldots, y_{n^k})$), while we have set $|I_1| = n^k - 1$ (and/or $|I_2| = n^k - 1$) to satisfy the RPE requirements of the parent gadget in the first simulation phase. Nevertheless, for such cases a perfect simulation of the plain value $x = \mathsf{LinDec}(\widehat{x})$ (and/or $y = \mathsf{LinDec}(\widehat{y})$) is included to out_{base} by construction of W_{base}. We can therefore perfectly simulate the missing share from the $n^k - 1$ other shares and the plain value x (or y). We thus get a perfect simulation of $\left(\mathsf{AssignWires}(G_j^{(k)}, W_j, (\widehat{x}, \widehat{y}), \widehat{z}|_{J_j}\right)$ for all the level-k gadgets $G_j^{(k)}$ which gives us a perfect simulation of $\left(\mathsf{AssignWires}(G^{(k+1)}, W^*, (\widehat{x}^*, \widehat{y}^*)), \widehat{z}|_{J^*}\right)$.

Saturated case: $J^* = [n^{k+1}]$. The saturated case proceeds similarly. The difference is that we must simulate all n^{k+1} output shares of the level-$(k+1)$ gadget, except for one share index j^* that can be chosen by the simulator.

The simulator $\mathsf{Sim}_1^{G^{(k+1)}}$ is defined as previously. Since $J^* = [n^{k+1}]$, we must define $J_{\text{base}} = [1, n]$. Moreover we have $J_i^* = [n^k]$ for all $1 \leq i \leq n$. This implies that for the gadgets $G_j^{(k)}$ on the output layer, the sets J_j are all equal to $[n^k]$ as well. The set W_{base} is defined as previously, and the simulator $\mathsf{Sim}_1^{G^{(k+1)}}$ returns (I_1^*, I_2^*) as previously. The failure events $\mathbb{F}_1^*$ and $\mathbb{F}_2^*$ are still ε_{k+1}-mutually unlikely, as required by the $(S_{k+1}, f^{(k+1)})$-RPE property of $G^{(k+1)}$.

The simulator $\mathsf{Sim}_2^{G^{(k+1)}}$ is defined as previously. In particular, from the running of the base gadget simulator Sim_2^G, we obtain a perfect simulation of the output wires $\widehat{z}^b|_{J'_{\text{base}}}$ for some J'_{base} with $|J'_{\text{base}}| = n - 1$. Combined with the perfect simulation of the output wires corresponding to the output sets J'_j from the gadgets $G_j^{(k)}$ on the output layer, with $|J'_j| = n^k - 1$, we obtain a subset J' of output wires for our level-$(k+1)$ gadget with $|J'| = n^{k+1} - 1$ as required. Eventually this gives us a perfect simulation of $\left(\mathsf{AssignWires}(G^{(k+1)}, W^*, (\widehat{x}^*, \widehat{y}^*)), \widehat{z}|_{J'}\right)$. This terminates the proof of Theorem 2. □

The random probing security of the expanding circuit compiler can then be deduced as a corollary of the above theorem together with Proposition 2 (RPE ⇒ RPC reduction) and Theorem 1 (composition theorem).

Corollary 1. *Let $\mathbb{B}$ be an arithmetic circuit basis $\mathbb{B} = \{g : \mathbb{K}^\ell \to \mathbb{K}^m\}$ such that $\ell, m \leq 2$ for any $g \in \mathbb{B}$. Let $n \in \mathbb{N}$ and $f : \mathbb{R} \to \mathbb{R}$. Let $\{G_g\}_{g\in\mathbb{B}}$ be n-share gadgets on $\mathbb{K}$. Let CC be the standard circuit compiler with sharing order n and base gadgets $\{G_g\}_{g\in\mathbb{B}}$. Let $\mathsf{CC}^{(k)}$ be the expanding circuit compiler with base compiler CC. If the base gadgets G_g are (t, f)-RPE then $\mathsf{CC}^{(k)}$ is $(p, 2 \cdot f^{(k)}(p))$-random probing secure.*

4.3 How to compute $f(p)$ for RPE ?

The requirement of the RPE property that the failure events $\mathbb{F}_1$ and $\mathbb{F}_2$ are mutually independent might seem too strong. In practice it might be easier to show or verify that some gadgets satisfy a weaker notion. We say that a gadget is (t, f)-*weak random probing expandable* (wRPE) if the failure events verify $\Pr(\mathbb{F}_1) \leq \varepsilon$, $\Pr(\mathbb{F}_2) \leq \varepsilon$ and $\Pr(\mathbb{F}_1 \wedge \mathbb{F}_2) \leq \varepsilon^2$ instead of (A.3) in Definition 10. Although being easier to achieve and to verify, this notion is actually not much weaker as the original RPE. We have the following reduction of RPE to wRPE.

Proposition 3. *Let $f = \mathbb{R} \to [0, 0.14]$. Let $G : \mathbb{K}^n \times \mathbb{K}^n \to \mathbb{K}^n$ be an n-share gadget. If G is (t, f)-wRPE then G is (t, f')-RPE with $f'(\cdot) = f(\cdot) + \frac{3}{2}f(\cdot)^2$.*

Proof. Let Sim_1^G be the simulator from the (t, f)-wRPE property. This simulator outputs I_1 and I_2 such that

$$\Pr(\mathbb{F}_1) = \varepsilon_1 \leq \varepsilon \ , \quad \Pr(\mathbb{F}_2) = \varepsilon_2 \leq \varepsilon \ \text{ and } \ \Pr(\mathbb{F}_1 \wedge \mathbb{F}_2) = \varepsilon_{12} \leq \varepsilon^2 \ , \tag{4.13}$$

where $\mathbb{F}_1 \equiv (|I_1| > t)$ and $\mathbb{F}_2 \equiv (|I_2| > t)$. We show how to construct $\mathsf{Sim}_1'^G$ which outputs I_1' and I_2' such that

$$\Pr(\mathbb{F}_1') = \Pr(\mathbb{F}_2') = \varepsilon' \ \text{ and } \ \Pr(\mathbb{F}_1' \wedge \mathbb{F}_2') = (\varepsilon')^2 \ \text{ with } \ \varepsilon' = \varepsilon + \frac{3}{2}\varepsilon^2 \tag{4.14}$$

where $\mathbb{F}_1' \equiv (|I_1'| > t)$ and $\mathbb{F}_2' \equiv (|I_2'| > t)$ and such that $I_1 \subseteq I_1'$ and $I_2 \subseteq I_2'$. In particular, the latter implies that we can keep the same Sim_2^G simulator since it is always given the same input shares plus additional input shares to achieve the same simulation as before.

The simulator $\mathsf{Sim}_1'^G$ first calls the simulator Sim_1^G to get I_1 and I_2. Whenever $|I_1|$ and $|I_2|$ are both lower than t, *i.e.* no failure event occurs, which happens with probability $p_{\text{succ}} = 1 - (\varepsilon_1 + \varepsilon_2 - \varepsilon_{12})$, $\mathsf{Sim}_1'^G$ outputs

$$(I_1', I_2') = \begin{cases} ([n], I_2) & \text{with probability } p_1 = \delta_1/p_{\text{succ}} \\ (I_1, [n]) & \text{with probability } p_2 = \delta_2/p_{\text{succ}} \\ ([n], [n]) & \text{with probability } p_{12} = \delta_{12}/p_{\text{succ}} \\ (I_1, I_2) & \text{with probability } 1 - (p_1 + p_2 + p_{12}) \end{cases}$$

for some $\delta_1, \delta_2, \delta_{12} \geq 0$ such that $\delta_1 + \delta_2 + \delta_{12} \leq p_{\text{succ}}$. We hence get

$$\begin{aligned} \Pr(\mathbb{F}_1') &= \varepsilon_1 + \delta_1 + \delta_{12} \\ \Pr(\mathbb{F}_2') &= \varepsilon_2 + \delta_2 + \delta_{12} \\ \Pr(\mathbb{F}_1' \wedge \mathbb{F}_2') &= \varepsilon_{12} + \delta_{12} \end{aligned}$$

We must now fix $\delta_1, \delta_2, \delta_{12} \geq 0$ to satisfy Equation 4.14, with $\varepsilon' = \varepsilon + 3 \cdot \varepsilon^2/2$ and $\delta_1 + \delta_2 + \delta_{12} \leq p_{\text{succ}} = 1 - (\varepsilon_1 + \varepsilon_2 - \varepsilon_{12})$. We fix $\delta_{12} = (\varepsilon')^2 - \varepsilon_{12}$; this gives $\Pr(\mathbb{F}'_1 \wedge \mathbb{F}'_2) = (\varepsilon')^2$, and from Equation 4.13 we obtain $\delta_{12} \geq 0$ as required. We let:

$$\delta_1 := \varepsilon' - \varepsilon_1 - \delta_{12}$$

which gives $\Pr(\mathbb{F}'_1) = \varepsilon'$ as required. Moreover we obtain using Equation 4.13:

$$\begin{aligned}\delta_1 &= \varepsilon + \frac{3}{2}\varepsilon^2 - \varepsilon_1 - \left((\varepsilon + \frac{3}{2}\varepsilon^2)^2 - \varepsilon_{12}\right) \geq \frac{3}{2}\varepsilon^2 - \left(\varepsilon^2 + 3\varepsilon^3 + \frac{9}{4}\varepsilon^4\right) \\ &\geq \varepsilon^2 \cdot \left(\frac{1}{2} - 3\varepsilon - \frac{9}{4}\varepsilon^2\right) \geq 0 \quad \text{for } \varepsilon \leq 0.14.\end{aligned}$$

We obtain similar conditions for $\delta_2 := \varepsilon' - \varepsilon_2 - \delta_{12}$. Finally, we have

$$\begin{aligned}\delta_1 + \delta_2 + \delta_{12} &= \varepsilon' - \varepsilon_1 - \delta_{12} + \varepsilon' - \varepsilon_2 - \delta_{12} + \delta_{12} \\ &= 2\varepsilon' - \varepsilon_1 - \varepsilon_2 - (\varepsilon')^2 + \varepsilon_{12} = p_{\text{succ}} + 2\varepsilon' - (\varepsilon')^2 - 1 \\ &\leq p_{\text{succ}} + 2\varepsilon' - 1 \leq p_{\text{succ}} \quad \text{for } \varepsilon \leq 0.14.\end{aligned}$$

as required. Note that the upper bound $\varepsilon \leq 0.14$ is reasonable and is satisfied by all of the constructions we consider which have a much lower failure event probability. □

Assume that we can show or verify that a gadget is wRPE with the following failure event probabilities

$$\Pr(\mathbb{F}_1) = f_1(p) \ , \quad \Pr(\mathbb{F}_2) = f_2(p) \quad \text{and} \quad \Pr(\mathbb{F}_1 \wedge \mathbb{F}_2) = f_{12}(p) \ ,$$

for every $p \in [0, 1]$. Then the above proposition implies that the gadget is (p, f)-RPE with

$$f : p \mapsto f_{\max}(p) + \frac{3}{2} f_{\max}(p)^2 \quad \text{with} \quad f_{\max} = \max(f_1, f_2, \sqrt{f_{12}}) \ .$$

The computation of f for wRPE can be split into two steps that we first describe for the case of addition and multiplication gadgets with fan-in 2 and fan-out 1.

In a first step, we compute f to check the (t, f)-wRPE property for output sets of shares of cardinal at most t. For 2-input gadgets, this step leads to the computation of coefficients c_i corresponding to three failure events $\mathbb{F}_1$, $\mathbb{F}_2$, and $\mathbb{F}_1 \wedge \mathbb{F}_2$ as defined above but restricted to output sets of shares of cardinal less than t. The process is very similar to the verification of random probing composability but requires to separate the failure events counter into failure events for the first input ($|\mathcal{I}_1| > t$), for the second input ($|\mathcal{I}_2| > t$) or for both ($(|\mathcal{I}_1| > t) \wedge (|\mathcal{I}_2| > t)$). In the following, we denote the three functions formed from the corresponding coefficients as $f_1^{(1)}$, $f_2^{(1)}$, and $f_{12}^{(1)}$.

Then, in a second step, we check that there exists at least one set of $n-1$ shares for each output, such that the simulation failure is limited by $f(p)$ for some probability $p \in [0, 1]$. In that case, we still loops on the possible output sets of shares (of cardinal $n-1$) but instead of computing the maximum coefficients, we determine whether the simulation succeeds for at least one of such sets. A failure event is recorded for a given tuple if no output sets of cardinal $n-1$ can be simulated together with this tuple from at most t shares of each input. We record the resulting coefficients for the three failure events to obtain functions $f_1^{(2)}$, $f_2^{(2)}$, and $f_{12}^{(2)}$.

From these two steps, we can deduce f such that the gadget is (t, f)-wRPE:

$$\forall p \in [0, 1], f(p) = \max(f_1(p), f_2(p), \sqrt{f_{12}(p)})$$

with

$$f_\alpha(p) \quad = \quad \max(f_\alpha^{(1)}(p), f_\alpha^{(2)}(p)) \qquad \text{for} \quad \alpha \in \{1, 2, 12\}$$

The computation of f for a gadget to satisfy (t, f)-weak random probing expandability is a bit trickier for copy gadgets which produce two outputs. Instead of two verification steps considering both possible ranges of cardinals for the output set of shares J, we need to consider four scenarios for the two possible features for output sets of shares J_1 and J_2. In a nutshell, the idea is to follow the first verification step described above when both J_1 and J_2 have cardinal equal or less than t and to follow the second verification step described above when both J_1 and J_2 have greater cardinals. This leads to functions $f^{(1)}$ and $f^{(2)}$. Then, two extra cases are to be considered, namely when $(|J_1| \leq t)$ and $(|J_2| > t)$ and the reverse when $(|J_1| > t)$ and $(|J_2| \leq t)$. To handle these scenarios, our tool loops over the output sets of shares of cardinal equal or less than t for the first output, and it determines whether there exists a set of $n-1$ shares of the second output that a simulator can perfectly simulate with the leaking wires and the former set. This leads to function $f^{(12)}$ and reversely to function $f^{(21)}$. From these four verification steps, we can deduce f such that the copy gadget is (t, f)-wRPE:

$$\forall p \in [0, 1], f(p) = \max(f^{(1)}(p), f^{(2)}(p), f^{(12)}(p), f^{(21)}(p)).$$

Once gadgets have been proven (t, f)-weak RPE, they are also proven to be (t, f')-RPE from Proposition 3 with $f' : p \mapsto f(p) + \frac{3}{2} f(p)^2$.

Similarly to RP and RPC, we suppose we have access to algorithms that allow us to efficiently and automatically compute the coefficients of the function $f(p)$ for (w)RPE. We will discuss automatic verification tools in more detail later in Chapter 7, and provide results of such computation in the following sections. But before, let us the discuss the complexity of the expansion strategy.

4.4 Expansion Complexity

In this section we show that the asymptotic complexity of a compiled circuit $\widehat{C} = \mathsf{CC}^{(k)}(C)$ is $|\widehat{C}| = \mathcal{O}\big(|C| \cdot \kappa^e\big)$ for a security parameter κ and some constant e that we explicit below.

Let us denote by $\boldsymbol{N} = (N_{g_1}, \ldots, N_{g_{|\mathbb{B}|}})^\mathsf{T}$ the column vector of gate counts for some base gadget G, where N_{g_i} stands for the number of g_i gates for $g_i \in \mathbb{B}$. We have $\mathbb{B}$ different such vectors

$$\boldsymbol{N}_{G_{g_i}} \quad = \quad (N_{G_{g_1}, g_1}, \ldots, N_{G_{g_{|\mathbb{B}|}}, g_{|\mathbb{B}|}})^\mathsf{T}$$

for the gate counts respectively in the base gadget G_{g_i} for each $g_i \in \mathbb{B}$. Let us define the $|\mathbb{B}| \times |\mathbb{B}|$ square matrix $\boldsymbol{M}$ as

$$\boldsymbol{M} = \big(\boldsymbol{N}_{G_{g_1}} \mid \ldots \mid \boldsymbol{N}_{G_{g_{|\mathbb{B}|}}}\big)$$

It can be checked that applying the standard circuit compiler with base gadgets $\{G_g\}_{g \in \mathbb{B}}$ to some circuit C with gate-count vector $\boldsymbol{N}_C$ gives a circuit $\widehat{C}$ with gate-count vector $\boldsymbol{N}_{\widehat{C}} = \boldsymbol{M} \cdot \boldsymbol{N}_C$. It follows that the k^{th} power of the matrix $\boldsymbol{M}$ gives the gate counts for the level-k gadgets as:

$$\boldsymbol{M}^k = \underbrace{\boldsymbol{M} \cdot \boldsymbol{M} \cdots \boldsymbol{M}}_{k \text{ times}} = \big(\boldsymbol{N}^{(k)}_{G_{g_1}} \mid \ldots \mid \boldsymbol{N}^{(k)}_{G_{g_{|\mathbb{B}|}}}\big)$$

where $\boldsymbol{N}_i^{(k)}$ are the gate-count vectors for the level-k gadgets $G_{g_i}^{(k)}$ respectively. Let us denote the eigen decomposition of $\boldsymbol{M}$ as $\boldsymbol{M} = \boldsymbol{Q} \cdot \boldsymbol{\Lambda} \cdot \boldsymbol{Q}^{-1}$, we get

$$\boldsymbol{M}^k = \boldsymbol{Q} \cdot \boldsymbol{\Lambda}^k \cdot \boldsymbol{Q}^{-1} \quad \text{with} \quad \boldsymbol{\Lambda}^k = \begin{pmatrix} \lambda_1^k & & \\ & \ddots & \\ & & \lambda_{|\mathbb{B}|}^k \end{pmatrix}$$

where λ_i are the eigenvalues of $\boldsymbol{M}$. We then obtain an asymptotic complexity of

$$|\widehat{C}| = \mathcal{O}\big(|C| \cdot (\lambda_1^k + \ldots + \lambda_{|\mathbb{B}|}^k)\big) = \mathcal{O}\big(|C| \cdot N_{\max}^k\big) \text{ with } N_{\max} = \max_i(\lambda_i) \ . \tag{4.15}$$

for a compiled circuit $\widehat{C} = \mathsf{CC}^{(k)}(C)$ (where the constant in the $\mathcal{O}(\cdot)$ depends on $\boldsymbol{Q}$ and shall be fairly small).

Specialization to addition, copy, multiplication and random gates. We now consider $\mathbb{B}$ to contain 4 gates that we consider in our circuits: addition, copy and multiplication gates, as well as randomness gates which simply generate a fresh uniform random value (*i.e.* with zero input and one output). In this case, we have the 4 gate count vectors

$$\begin{aligned} \boldsymbol{N}_{G_{\text{add}}} &= (N_{G_{\text{add}},\text{a}}, N_{G_{\text{add}},\text{c}}, N_{G_{\text{add}},\text{m}}, N_{G_{\text{add}},\text{r}})^\mathsf{T} \\ \boldsymbol{N}_{G_{\text{mult}}} &= (N_{G_{\text{mult}},\text{a}}, N_{G_{\text{mult}},\text{c}}, N_{G_{\text{mult}},\text{m}}, N_{G_{\text{mult}},\text{r}})^\mathsf{T} \\ \boldsymbol{N}_{G_{\text{copy}}} &= (N_{G_{\text{copy}},\text{a}}, N_{G_{\text{copy}},\text{c}}, N_{G_{\text{copy}},\text{m}}, N_{G_{\text{copy}},\text{r}})^\mathsf{T} \end{aligned}$$

for the three base gadgets G_{add}, G_{mult} and G_{copy} respectively (where a, c, m, r stand for addition, copy, multiplication and random gates respectively). We also have

$$\boldsymbol{N}_{\text{rand}} = (0, 0, 0, n)^\mathsf{T}$$

which holds from the fact that the standard circuit compiler simply replaces each random gate by n random gates. Then, we can construct the matrix $\boldsymbol{M}$ as

$$\boldsymbol{M} = \big(\boldsymbol{N}_{G_{\text{add}}} \mid \boldsymbol{N}_{G_{\text{copy}}} \mid \boldsymbol{N}_{G_{\text{mult}}} \mid \boldsymbol{N}_{\text{rand}}\big)$$

Interestingly, if multiplication gates are solely used in the multiplication gadget (*i.e.* $N_{G_{\text{add}},m} = N_{G_{\text{copy}},m} = 0$) which is the case in the constructions we usually consider, it can be checked that (up to some permutation) the eigenvalues satisfy

$$(\lambda_1, \lambda_2) = \mathsf{eigenvalues}(\boldsymbol{M}_{ac}) \ , \quad \lambda_3 = N_{G_{\text{mult}},m}^k \quad \text{and} \quad \lambda_4 = n^k$$

where $\boldsymbol{M}_{ac}$ is the top left 2×2 block matrix of $\boldsymbol{M}$ *i.e.*

$$\boldsymbol{M}_{ac} = \begin{pmatrix} N_{G_{\text{add}},a} & N_{G_{\text{copy}},a} \\ N_{G_{\text{add}},c} & N_{G_{\text{copy}},c} \end{pmatrix} \ . \tag{4.16}$$

We finally get

$$|\widehat{C}| = \mathcal{O}\big(|C| \cdot N_{\max}^k\big) \quad \text{with} \quad N_{\max} = \max(\mathsf{eigenvalues}(\boldsymbol{M}_{ac}), N_{G_{\text{mult}},m}) \ . \tag{4.17}$$

Complexity and security level. In order to reach some security level $\varepsilon = 2^{-\kappa}$ for some target security parameter κ and assuming that we have a security expansion $p \to f^{(k)}(p)$, the expansion level k must be chosen so that $f^{(k)}(p) \leq 2^{-\kappa}$. In practice, the function f is of the form

$$f : p \mapsto \sum_{i \geq d} c_i \, p^i \leq (c_d + \mathcal{O}(p)) \, p^d \ .$$

where $\mathcal{O}(p)$ is to be interpreted as p tends to 0. We shall say that such a function has *amplification order* d.

Definition 13 (Amplification Order)**.** *We define the amplification order of a function and the amplification order of a gadget.*

- *Let $f : \mathbb{R} \to \mathbb{R}$ which satisfies*

$$f(p) = c_d \, p^d + \mathcal{O}(p^{d+\varepsilon})$$

  as p tends to 0, for some $c_d > 0$ and $\varepsilon > 0$. Then d is called the amplification order of f.

- *Let $t > 0$ and G a gadget. Let d be the maximal integer such that G achieves (t, f)-RPE for $f : \mathbb{R} \to \mathbb{R}$ of amplification order d. Then d is called the amplification order of G (with respect to t).*

We stress that the amplification order of a gadget G is defined with respect to the RPE threshold t. Namely, different RPE thresholds t are likely to yield different amplification orders d for G (or equivalently d can be thought of as a function of t).

The upper bound $f(p) \leq c'_d \, p^d$ with $c'_d = c_d + \mathcal{O}(p)$ implies $f^{(k)}(p) < (c'_d \, p)^{d^k}$. Hence, to satisfy the required security $f^{(k)}(p) \leq 2^{-\kappa}$ while assuming $c'_d p < 1$, the number k of expansions must satisfy:

$$k \geqslant \log_d(\kappa) - \log_d(-\log_2(c'_d \, p)) \ .$$

We can then rewrite (4.15) (and also (4.17)) as

$$|\widehat{C}| = \mathcal{O}\big(|C| \cdot \kappa^e\big) \quad \text{with} \quad e = \frac{\log N_{\max}}{\log d} \ . \tag{4.18}$$

In addition, we recall that for the expansion strategy to be useful, we must have $\varepsilon = f(p) < p$. We call the *maximum tolerated leakage probability* of a gadget, the maximal value $0 \leq p_{\max} \leq 1$ such that $f(p_{\max}) < p_{\max}$.

Two crucial parameters determine the complexity of the expanding compiler: $N_{\max}$ and d. The value of $N_{\max}$ depends on the number of gates in the gadgets (*i.e.* complexity of the gadgets), while the amplification order depends on the associated failure functions for RPE property. In order to get the best expanding compilers in terms of asymptotic complexity, we need to find gadgets with the least gates and the highest amplification order.

4.4.1 Bounding the Amplification Order

As recalled above, the amplification order of a gadget is a crucial parameter of its random probing expandability. The higher the amplification order, the lower the asymptotic complexity of the expanding compiler, *ceteris paribus*. A natural question is to determine the best amplification order that can be hoped for given the different parameters of a gadget. In this section, we exhibit concrete upper bounds on the amplification order that can be achieved by

a gadget depending on its input-output dimensions (ℓ, m), its number of shares n, and its RPE threshold t.

Before giving the bounds let us make a key observation on the amplification order of a gadget. Let G be a 2-to-1 n-share gadget achieving (t, f)-RPE. A subset W of the wires of G is said to be a *failure set* with respect to the first input (resp. the second input) if there exists a set $J \subseteq [n]$ such that $(I_1, I_2, J') \leftarrow \mathsf{Sim}_1^G(W, J)$ implies $|I_1| > t$ (resp. $|I_2| > t$), namely if a leaking set W implies the failure event $\mathcal{F}_1$ (resp. $\mathcal{F}_2$) in the definition of RPE (Definition 10). One can check that G has amplification order $d \leq d_{up}$ if one of the two following events occurs:

1. there exists a failure set W w.r.t. the first input *or* the second input such that $|W| = d_{up}$,
2. there exists a failure set W w.r.t. the first input *and* the second input such that $|W| = 2d_{up}$.

In the former case, the existence of the failure set implies that the function $f(p)$ has a non-zero coefficient in $p^{d_{up}}$ and hence $d \leq d_{up}$. In the latter case, the existence of the double failure set implies that the function $f^2(p)$ has a non-zero coefficient in $p^{2d_{up}}$ and hence $d \leq d_{up}$. The case of a single-input gadget is simpler: it has amplification order $d \leq d_{up}$ if there exists a failure set W (w.r.t. its single input) such that $|W| = d_{up}$.

We start by exhibiting a generic upper bound for the amplification order and then look at the particular case of what we shall call a *standard* multiplication gadget.

Generic Upper Bound. In the following we will say that a function $g : \mathbb{K}^\ell \to \mathbb{K}^m$ is *complete* if at least one of its m outputs is functionally dependent on the ℓ inputs. Similarly, we say that a gadget G is complete if its underlying function g is complete. The following lemma gives our generic upper bound on the amplification order.

Lemma 1. *Let $f : \mathbb{R} \to \mathbb{R}$, $n \in \mathbb{N}$ and $\ell, m \in \{1, 2\}$. Let $G : (\mathbb{K}^n)^\ell \to (\mathbb{K}^n)^m$ be an ℓ-to-m n-share complete gadget achieving (t, f)-RPE. Then its amplification order d is upper bounded by*

$$\min((t + 1), (3 - \ell) \cdot (n - t)).$$

Proof. The first part of the bound on the amplification order $d \leq (t+1)$ is immediate since by probing $t + 1$ shares of any input, the considered set will be a failure set of cardinality $t + 1$. We then consider two cases depending on the number of inputs:

1. *1-input gadgets ($\ell = 1$):* We show that we can exhibit a failure set of size $2(n - t)$. Let us denote the output shares $z_1, \ldots, z_n$ (for two-output gadgets, *i.e.* $m = 2$, $z_1, \ldots, z_n$ can be any of the output sharings). In the evaluation of the (t, f)-RPE property, t shares among the z_i's (corresponding to the set J) must be simulated. Without loss of generality, let $z_1, \ldots, z_t$ be those shares (*i.e.* $J = [t]$). By including both input gates of each of the remaining output shares $z_{t+1}, \ldots, z_n$ in the set W, the distribution to be simulated requires the knowledge of the full input (by completeness of the gadget). The set W is thus a failure set with $2(n - t)$ elements.

2. *2-input gadgets ($\ell = 2$):* Considering the same failure set as in the above case, the simulation of *out* in the RPE definition requires the full two input sharings. Hence W is a failure set of size $2(n - t)$ with respect to the two inputs, and so the amplification order satisfies $d \leq (n - t)$.

We hence conclude that $d \leq \min((t + 1), 2(n - t))$ for one-input gadgets, and $d \leq \min((t + 1), (n - t))$ for two-input gadgets. □

Corollary 2 (One-input gadget)**.** *The amplification order d of a one-input gadget achieving (t, f)-RPE is upper bounded by*

$$d \leq \frac{2(n+1)}{3} \ .$$

The above corollary directly holds from Lemma 1 for a RPE threshold $t = \lfloor \frac{2n-1}{3} \rfloor$ (which balances the two sides of the min).

Corollary 3 (Two-input gadget)**.** *The amplification order d of a two-input gadget achieving (t, f)-RPE is upper bounded by*

$$d \leq \frac{n+1}{2} \ .$$

The above corollary directly holds from Lemma 1 for a RPE threshold $t = \lfloor \frac{n-1}{2} \rfloor$ (which balances the two sides of the min).

We deduce from the two above corollaries that for a circuit composed of addition, multiplication and copy gadgets, the amplification order is upper bounded

$$d \leq \min\left(\frac{2(n+1)}{3}, \frac{n+1}{2}\right) = \frac{n+1}{2} \ ,$$

which can only be achieved for an odd number of shares by taking $t = \lfloor \frac{n-1}{2} \rfloor$ as RPE threshold.

Upper Bound for Standard Multiplication Gadgets. The generic bound exhibited above is not tight in the special case of a standard multiplication gadget which computes cross products between the input shares, such as the ISW multiplication gadget [59]. We exhibit hereafter a tighter bound for such gadgets.

Formally, a n-share multiplication gadget G is a *standard multiplication gadget*, if on input $(\widehat{x}, \widehat{y}) \in (\mathbb{K}^n)^2$, G computes the cross products $x_i \cdot y_j$ for $1 \leq i, j \leq n$. Our upper bound on the amplification order for such gadgets is given in the following lemma.

Lemma 2. *Let $f : \mathbb{R} \to \mathbb{R}$ and $n \in \mathbb{N}$. Let G be an n-share standard multiplication gadget achieving (t, f)-RPE. Then its amplification order d is upper bounded by*

$$d \leq \min\left(\frac{t+1}{2}, (n-t)\right) .$$

Proof. The second part of the bound $(n-t)$ holds directly from Lemma 1. We now prove the bound $(t+1)/2$ by exhibiting a failure set of size $t+1$ with t output shares, which will be a failure on both inputs. Let $\{m_{ij}\}_{0 \leq i,j \leq n}$ denote the cross products such that $m_{ij} = x_i \cdot y_j$. Consider a set W made of $t+1$ such variables $\{m_{ij}\}$ for which the indices i and j are all distinct. Specifically, $W = \{x_{i_1} \cdot y_{j_1}, \ldots, x_{i_{t+1}} \cdot y_{j_{t+1}}\}$ such that $\{i_\ell\}_{1 \leq \ell \leq t+1}$ and $\{j_\ell\}_{1 \leq \ell \leq t+1}$ are both sets of $(t+1)$ distinct indices. Clearly, such a set is a failure set for both inputs $\boldsymbol{x}$ and $\boldsymbol{y}$ since it requires $t+1$ shares of each of them to be perfectly simulated (even without considering the output shares to be also simulated). We hence have a double failure set of cardinality $t+1$ which implies the $(t+1)/2$ upper bound on the amplification order. □

The above lemma implies that the highest amplification order for standard multiplication gadgets might be achieved for a RPE threshold $t = \lfloor \frac{2n-1}{3} \rfloor$ which yields the following maximal upper bound:

$$d \leq \frac{n+1}{3} \ ,$$

which is lower than the generic upper bound for 2-to-1 gadgets exhibited in Corollary 3. This loss suggests that better amplification orders could be achieved for multiplication gadgets that do not compute direct cross products of the input shares. We actually provide such constructions in later chapters (*c.f.* Chapter 5).

About 2-share gadgets. Thanks to Corollary 3, we know that the highest amplification order achievable for 2 inputs 2-share gadgets is equal to 1. For such a gadget, the failure function for RPE would then be equal to $f(p) = c_1 \cdot p + \mathcal{O}(p^2)$, with $c_1 > 0$. Recall that an amplification order strictly greater than one guarantees that there exists a probability $p_{max} \in [0, 1]$ such that $\forall p \leq p_{max}, f(p) \leq p$, which is necessary to benefit from the expansion. It is not the case for 2-share expanding compilers that use 2 inputs gadgets such as addition and multiplication. We thus restrict our investigation in the following to n-share gadgets, with $n \geq 3$ to instantiate the expanding compiler.

4.4.2 RPE Compiler Parameters

When constructing an expanding compiler as in Definition 9, the complexity depends on the parameters of the worst case gadget. In other words, the amplification order and the tolerated leakage rate will be associated to the worst failure function $f(p)$ among the functions associated to the base RPE gadgets. We can give the following general definition of the expanding compiler including the way we define the amplification order and the tolerated leakage rate associated to it.

Definition 14 (RPE Compiler)**.** *Let $\mathbb{B} = \{g : \mathbb{K}^\ell \to \mathbb{K}^m\}$ be an arithmetic circuit basis. Let $n, t \in \mathbb{N}$, and let $\{G_g\}_{g\in\mathbb{B}}$ be a family of (t, f_{G_g})-RPE n-share gadgets for the gate functionalities in $\mathbb{B}$. The* RPE compiler CC *associated to $\{G_g\}_{g\in\mathbb{B}}$ is the expanding circuit compiler which consists in replacing each gate from a circuit over $\mathbb{B}$ by the corresponding expanded gadget $G_g^{(k)}$ given an expansion level k as defined in Definition 9. Moreover,*

- *the* expanding function *of* CC *is the function f defined as*

$$f : p \mapsto \max_g f_{G_g}(p)$$

- *the* amplification order *of* CC *is the integer d defined as*

$$d = \min_g d_{G_g}$$

  where d_{G_g} is the amplification order of f_{G_g},

- *the* gadget complexity *of* CC *is the integer s defined as*

$$s = \max_g |G_g|$$

  where $|G_g|$ denotes the number of wires in the gadget G_g,

- *the* tolerated leakage rate *of* CC *is the real number $q \in [0, 1)$ such that $f(p) < p$ for every $p < q$.*

4.5 A First 3-share RPE Construction

In this section, we exhibit and analyze small $(1, f)$-wRPE gadgets for the addition, multiplication, and copy (on any base field $\mathbb{K}$) with $n = 3$ shares to instantiate the RPE circuit compiler. These gadgets are sound in the sense that their function f has amplification order strictly greater than one. For 2-input gadgets, f is defined as the maximum between f_1, f_2, and $\sqrt{f_{12}}$ (*c.f.* Section 4.3). Therefore, the constraint on the amplification order also applies to the functions f_1, f_2, and $\sqrt{f_{12}}$. For the function f_{12}, this means that the amplification order should be strictly greater than two.

The constructions in this section do not achieve the best parameters regarding security and complexity. The addition and copy constructions achieve the maximal amplification order for 3 shares (*i.e.* $d = 2$), while the multiplication construction only achieves an amplification order of $d = 3/2$. These constructions were the first attempt at the expanding compiler construction from [18]. Nevertheless, this section aims to present a first working example with concrete security and complexity bounds. In addition, Section 4.6 shows that this non-optimal construction still outperforms previous constructions in the state-of-the-art. In Chapter 5 and Chapter 6, our primary goal will be to achieve the best security and complexity bounds while discussing the benefits and limitations of the expansion strategy.

In the upcoming gadget descriptions, notice that variables r_i are fresh random values, operations are processed with the usual priority rules and the number of implicit copy gates can be deduced from the occurrences of each intermediate variable such that s occurrences require $s-1$ implicit copy gates (*i.e.* a total of $2 \cdot s - 1$ wires assigned with the same variable expression). Also, the function expression below each gadget corresponds to the function obtained from IronMask, computed as discussed in Section 4.3. It implies that the gadget is (t, f)-wRPE for $t = 1$ except when defined otherwise.

Addition gadget. The most classical masked addition schemes are share-wise additions which satisfy the simpler probing security property. Basically, given two input n-sharings $\widehat{x}$ and $\widehat{y}$, such an addition computes the output n-sharing $\widehat{z}$ as $z_1 \leftarrow x_1 + y_1$, $z_2 \leftarrow x_2 + y_2$, $\ldots$, $z_n \leftarrow x_n + y_n$. Unfortunately, such elementary gadgets do not work in our setting. Namely, consider an output set of shares J of cardinality t. Then, for any n, there exists sets W of leaking wires with $|W| = 1$ such that no sets (I_1, I_2) (on inputs $\widehat{x}$ and $\widehat{y}$ respectively) both of cardinalities $\leq t$ can point to input shares that are enough to simulate both the leaking wire and the output shares of indices in J. For instance, given a set $J = \{1, \ldots, t\}$, if W contains x_{t+1}, then no set I_1 (on input $\widehat{x}$) of cardinal $\leq t$ can define a set of input shares from which we can simulate both the leaking wire and $z_1, \ldots, z_t$. Indeed, each z_i for $1 \leq i \leq t$ requires both input shares x_i and y_i for its simulation. Thus, I_1 and I_2 would contain at least $\{1, \ldots, t\}$, and I_1 additionally contains $t + 1$ for the simulation of the leaking wire. I_1 would thus be of cardinal $t + 1$, representing a failure event in the random probing expandability definition. Consequently, such an n-share addition gadget could only be (t, f)-RPE, where f has a first coefficient c_1 strictly positive. In other words, f would be of amplification order one such that $\forall p \in [0, 1]$, $f(p) \geq p$.

In the following, we introduce a 3-share addition gadget, which is $(1, f)$-wRPE with f of amplification order strictly greater than one. Basically, in our addition gadget, both inputs are refreshed with a circular refreshing gadget as originally introduced in [11], while simply rearranging the order of the refreshing variables:

$$
\begin{array}{rcll}
G_{\text{add}} : z_1 & \leftarrow & x_1 + r_1 + r_5 + y_1 + r_2 + r_4 & \\
z_2 & \leftarrow & x_2 + r_2 + r_6 + y_2 + r_3 + r_5 & f_{max}(p) = \sqrt{69}p^2 + \mathcal{O}(p^3) \\
z_3 & \leftarrow & x_3 + r_3 + r_4 + y_3 + r_1 + r_6 & \log_2(p_{\max}) = -4,41
\end{array}
$$

$\widehat{x}$ and $\widehat{y}$ are the input sharings and $\widehat{z}$ the output sharing; f_{max} additionally reports the maximum of the first non-zero coefficient of the three functions f_1, f_2, and f_{12}, as defined in Section 4.3. $p_{\max}$ reports the maximum tolerated leakage rate as defined in Section 4.4. Both are computed using IronMask. Note that G_{add} is built with 15 addition gates and 6 implicit copy gates.

Copy gadget. We exhibit a 3-share $(1, f)$-wRPE copy gadget with f of amplification order strictly greater than one. The gadget also relies on two calls to the circular refreshing from [11]

on the input:

$$\begin{array}{rcll} w_1 & \leftarrow & x_1 + r_1 + r_2;\ z_1 \leftarrow x_1 + r_4 + r_5 & \\ w_2 & \leftarrow & x_2 + r_2 + r_3;\ z_2 \leftarrow x_2 + r_5 + r_6 & f_{max}(p) = 33p^2 + \mathcal{O}(p^3) \\ w_3 & \leftarrow & x_3 + r_3 + r_1;\ z_3 \leftarrow x_3 + r_6 + r_4 & \log_2(p_{\max}) = -5,88 \end{array}$$

This gadget is made of 12 addition gates and 9 implicit copy gates. $\widehat{x}$ is the input sharing, while $\widehat{w}$ and $\widehat{z}$ are the output sharings.

Multiplication gadget. As shown in Lemma 2, the maximum amplification order for multiplication gadgets performing a direct products between input shares, is equal to one for $n = 3$ shares. Hence, *standard* 3-share multiplication gadgets cannot be used as base gadgets of our compiler. To circumvent this issue, we build a 3-share multiplication gadget, by first refreshing both input sharings, before any multiplication is performed:

$$\begin{array}{rcllll} u_1 & \leftarrow & x_1 + r_6 + r_7; & u_2 \leftarrow x_2 + r_7 + r_8; & u_3 \leftarrow x_3 + r_8 + r_6 \\ v_1 & \leftarrow & y_1 + r_9 + r_{10}; & v_2 \leftarrow y_2 + r_{10} + r_{11}; & v_3 \leftarrow y_3 + r_{11} + r_9 \end{array}$$

$$\begin{array}{rcl} z_1 & \leftarrow & \big(u_1 \cdot v_1 + r_1\big) + \big(u_1 \cdot v_2 + r_2\big) + \big(u_1 \cdot v_3 + r_3\big) \\ z_2 & \leftarrow & \big(u_2 \cdot v_1 + r_2\big) + \big(u_2 \cdot v_2 + r_5\big) + \big(u_2 \cdot v_3 + r_4\big) \\ z_3 & \leftarrow & \big(u_3 \cdot v_1 + r_3\big) + \big(u_3 \cdot v_2 + r_4\big) + \big(u_3 \cdot v_3 + r_1\big) + r_5 \end{array}$$

$$\begin{array}{rcl} f_{max}(p) & = & \sqrt{32}p^{3/2} + \mathcal{O}(p^2) \\ \log_2(p_{\max}) & = & -7.09 \end{array}$$

Complexity and tolerated probability. Following the asymptotic analysis of Section 4.4, our construction yields the following instantiation of the matrix $\boldsymbol{M}$

$$\boldsymbol{M} = \begin{pmatrix} 15 & 12 & 28 & 0 \\ 6 & 9 & 23 & 0 \\ 0 & 0 & 9 & 0 \\ 6 & 6 & 11 & 3 \end{pmatrix} \tag{4.19}$$

with

$$\boldsymbol{M}_{ac} = \begin{pmatrix} 15 & 12 \\ 6 & 9 \end{pmatrix} \quad \text{and} \quad N_{G_{\text{mult}},m} = 9\ .$$

The eigenvalues of $\boldsymbol{M}_{ac}$ are 3 and 21, which gives $N_{\max} = 21$. We also have a random probing expandability with function f of amplification order $d = \frac{3}{2}$ (by taking the worst case between G_{add},G_{copy} and G_{mult}). Hence we get

$$e = \frac{\log N_{\max}}{\log d} = \frac{\log 21}{\log 1.5} \approx 7.5$$

which gives a complexity of $|\widehat{C}| = \mathcal{O}\big(|C| \cdot \kappa^{7.5}\big)$. Finally, our construction tolerates a leakage probability up to

$$p_{\max} \approx 2^{-7.09}\ .$$

This corresponds to the maximum value p for which we have $f(p) < p$ which is a necessary and sufficient condition for the expansion strategy to apply with (t, f)-RPE gadgets.

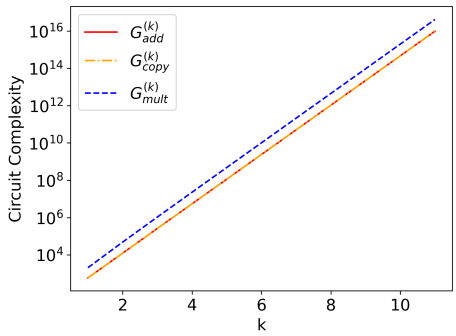

Figure 4.2: Number of gates for $G^{(k)}_{\text{add}}$, $G^{(k)}_{\text{copy}}$, $G^{(k)}_{\text{mult}}$ circuits with respect to the level k.

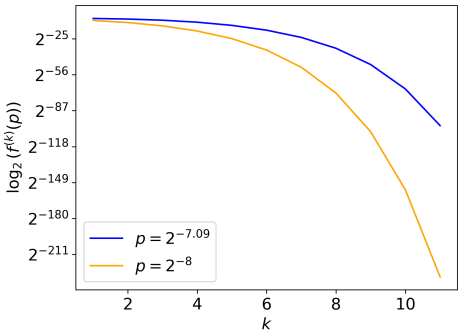

Figure 4.3: Values taken by $f^{(k)}(p)$ for different starting leakage rates p with respect to the level k.

As explained in Section 4.4, we can compute the new gate count vectors for each of the compiled gadgets $G^{(k)}_{\text{add}}$, $G^{(k)}_{\text{copy}}$, $G^{(k)}_{\text{mult}}$ by computing the matrix $\boldsymbol{M}^k$. In Figure 4.2, we plot the total number of gates $(N_{\text{a}} + N_{\text{c}} + N_{\text{m}} + N_{\text{r}})$ in each of the compiled gadgets as a function of the level k. In Figure 4.3, we plot the values taken by $f^{(k)}(p)$ as a function of the expansion level k and starting from two different leakage rates: $p = 2^{-7.09}$ which is the maximum tolerated leakage rate by our constructed compiler and another lower leakage rate of $p = 2^{-8}$.

For instance, for level $k = 9$ the number of gates in the compiled gadgets is around 10^{13}. For the latter level and assuming a leakage probability of $p = 2^{-7.09}$ (which is the maximum we can tolerate), we achieve a security of $\varepsilon \approx 2^{-47}$. If the considered leakage rate is slightly lower, for instance $p = 2^{-8}$, we achieve a security $\varepsilon \approx 2^{-10(}$ for the same expansion level $k = 9$.

4.6 Comparison with Previous Works

In this section, we compare our scheme to previous constructions. Specifically, we first compare it to the well-known Ishai-Sahai-Wagner (ISW) construction and discuss the instantiation of our scheme from the ISW multiplication gadget. Then we exhibit the asymptotic complexity (and tolerated leakage probability) of the Ananth-Ishai-Sahai compiler and compare their results to our instantiation.

4.6.1 Comparison with ISW

The classical ISW construction [59] is secure in the t-probing model when the adversary can learn any set of t intermediate variables in the circuit, for $n = 2t + 1$ shares. This can be extended to t probes per gadget, where each gadget corresponds to a AND or XOR gate in the original circuit. Using Chernoff bound, security in the t-probing model per gadget implies security in the p-random probing model, where each wire leaks with probability p, with $p = \mathcal{O}(t/|G|)$, where $|G|$ is the gadget size. Since in ISW each gadget has complexity $\mathcal{O}(t^2)$, this gives $p = \mathcal{O}(1/t)$. Therefore, in the p-random probing model, the ISW construction is only secure against a leakage probability $p = \mathcal{O}(1/n)$, where the number of shares n must grow linearly with the security parameter κ in order to achieve security $2^{-\kappa}$. This means that ISW does not achieve security under a constant leakage probability p; this explains why ISW is actually vulnerable to horizontal attacks [13], in which the adversary can combine information from a constant fraction of the wires.

ISW-based instantiation of the expanding compiler. In our instantiation, we choose to construct a new 3-share multiplication gadget instead of using the ISW multiplication gadget

from [59]. In fact, ISW first performs a direct product of the secret shares before adding some randomness, while we proved in Section 4.4.1 that no such 3-share multiplication gadget made of direct products could satisfy $(1, f)$-RPE with amplification order strictly greater than one (*c.f.* Lemma 2). Therefore the ISW gadget is not adapted for our construction with 3 shares.

Table 4.1 displays the output of the VRAPS verification tool [18] when run on the ISW multiplication gadget for up to 7 shares with different values for t. It can be seen that an amplification order strictly greater than one is only achieved for $t > 1$, with 4 or more shares. And an order of 3/2 is only achieved with a minimum of 4 shares for $t = 2$, whereas we already reached this order with our 3-share construction for $t = 1$. Actually, we observe through the table that the ISW multiplication gadget achieves the maximal amplification order for a multiplication gadget performing direct cross-products of input shares (*c.f.* Lemma 2). We prove generic bounds on the amplification order achievable by the ISW multiplication gadget and study linear constructions based on the ISW refresh gadget later in Chapter 5.

If we use the 4 share ISW gadget with appropriate 4-share addition and copy gadgets instead of our instantiation, the overall complexity of the compiler would be greater, while the amplification order would remain the same, and the tolerated leakage probability would be worse (recall that our instantiation tolerates a maximum leakage probability $p \approx 2^{-7.09}$, while the 4-share ISW multiplication tolerates $p \approx 2^{-9.83}$). Clearly, the complexity of the 4-share ISW gadget $(N_a, N_c, N_m, N_r) = (24, 30, 16, 6)$ is higher than that of our 3-share multiplication gadget $(N_a, N_c, N_m, N_r) = (28, 23, 9, 11)$. In addition, using 3-share addition and copy gadgets (as in our case) provides better complexity than 4-share gadgets. Hence to reach an amplification order of 3/2, a 4-share construction with the ISW gadget would be more complex and would offer a lower tolerated leakage probability.

Table 4.1: Complexity, amplification order and maximum tolerated leakage probability of the ISW multiplication gadgets. Some leakage probabilities were not computed accurately by VRAPS [18] for performances reasons. An interval on these probabilities is instead given by evaluating lower and upper bound functions f_{inf} and f_{sup} of $f(p)$.

# shares	Complexity $(N_{\text{a}}, N_{\text{c}}, N_{\text{m}}, N_{\text{r}})$	t	Amplification order	$\log_2$ of maximum tolerated leakage probability
3	(12, 15, 9, 3)	1	1	$-$
4	(24, 30, 16, 6)	1	1	$-$
		2	3/2	-9.83
5	(40, 50, 25, 10)	1	1	$-$
		2	3/2	-11.00
		3	2	-8.05
6	(60, 75, 36, 15)	1	1	$-$
		2	3/2	-13.00
		3	2	$[-9.83, -7.87]$
		4	2	$[-9.83, -5.92]$
7	(84, 105, 49, 21)	1	1	$-$
		2	3/2	$[-16.00, -14.00]$
		3	2	$[-12.00, -7.87]$
		4	5/2	$[-12.00, -2.27]$
		5	2	$[-12.00, -3.12]$

4.6.2 Complexity of the Ananth-Ishai-Sahai Compiler

The work from [3] provides a construction of circuit compiler (the AIS compiler) based on the expansion strategy described in Section 4.1 with a (p, ε)-composable security property, analogous to our (t, f)-RPE property. To this purpose, the authors use an (m, c)-multi-party computation (MPC) protocol Π. Such a protocol allows to securely compute a functionality shared among m parties and tolerating at most c corruptions. In a nutshell, their composable circuit compiler consists of multiple layers: the bottom layer replaces each gate in the circuit by a circuit computing the (m, c)-MPC protocol for the corresponding functionality (either Boolean addition, Boolean multiplication, or copy). The next $k-1$ above layers apply the same strategy recursively to each of the resulting gates. As this application can eventually have exponential complexity if applied to a whole circuit C directly, the top layer of compilation actually applies the k bottom layers to each of the gates of C independently and then stitches the inputs and outputs using the correctness of the XOR-encoding property. Hence the complexity is in

$$\mathcal{O}(|C| \cdot N_{\mathrm{g}}^{k}) \ , \tag{4.20}$$

where $|C|$ is the number of gates in the original circuit and N_{g} is the number of gates in the circuit computing Π. The authors of [3] prove that such compiler satisfies (p, ε)-composition security property, where p is the tolerated leakage probability and ε is the simulation failure probability. Precisely:

$$\varepsilon = N_{\mathrm{g}}^{c+1} \cdot p^{c+1} \tag{4.21}$$

Equations (4.20) and (4.21) can be directly plugged into our asymptotic analysis of Section 4.4, with N_g replacing our $N_{\max}$ and where $c+1$ stands for our amplification order d. The obtained asymptotic complexity for the AIS compiler is

$$\mathcal{O}\big(|C| \cdot \kappa^{e}\big) \quad \text{with} \quad e = \frac{\log N_{\mathrm{g}}}{\log c + 1} \ . \tag{4.22}$$

This is to be compared to $e = \frac{\log N_{\max}}{\log d}$ in our scheme. Moreover, this compiler can tolerate a leakage probability

$$p = \frac{1}{N_{\mathrm{g}}^{2}} \ .$$

The authors provide an instantiation of their construction using an existing MPC protocol due to Maurer [67]. From their analysis, this protocol can be implemented with a circuit of $N_{\mathrm{g}} = (4m - c) \cdot \left(\binom{m-1}{c}^2 + 2m\binom{m}{c}\right)$ gates. They instantiate their compiler with this protocol for parameters $m = 5$ parties and $c = 2$ corruptions, from which they get $N_{\mathrm{g}} = 5712$. From this number of gates, they claim to tolerate a leakage probability $p = \frac{1}{5712^2} \approx 2^{-25}$ and our asymptotic analysis gives a complexity of $\mathcal{O}\big(|C| \cdot \kappa^{e}\big)$ with $e \approx 7.87$ according to (4.22).

Complexity [67] of the Maurer MPC protocol. In the following we compute the complexity and the value of N_{g} in the instantiation of the AIS compiler [3]. First, using this compiler, given a circuit C to compile, each gate G is implemented using a functionality F associated to the MPC protocol. Such a functionality F receives m shares of each input and then reconstructs them to obtain original values. This reconstruction can be done with $2(m-1)$ addition gates. Then after computing the gate G, m additive shares of the output are computed twice. This step consists of one gate for G, and $2(m-1)$ gates for the additive sharing along with $2(m-1)$ random gates.[1] So each gate G to compile is replaced by $6m-5$ gates, each computed jointly by the m parties in the MPC protocol. Next, we state the complexity

[1] In [3], the authors only consider $2(m-1)$ for the cost of this step, not counting the number of random gates necessary to compute the additive sharing of the output.

of the protocol from [67]. Each gate in a functionality F is jointly computed by all m parties. In the beginning, each party holds one share of each input.

The first step consists in a k-secret sharing of each input share where $k = \binom{m}{c}$. For an input of m shares, each party will hold a total of $m\binom{m-1}{c}$ shares. For two inputs, this step has a complexity of $m(2k-2)$.

The second step is either performing an addition or a multiplication, depending on the gate G associated to the functionality. An addition simply means each party locally adding all its shares, holding a complexity of $m\binom{m-1}{c}$. In case of a multiplication gate, each party will locally compute the sum of the product of the shares of both inputs, and then share its local result using a secret sharing scheme as in the first step. This procedure holds a complexity of $\binom{m-1}{c}^2$ for computing the result, $m(2k-2)$ for the secret sharing, and $2k^2$ copy gates. Clearly, the cost of the second step is more important for the multiplication and can be upper bounded by[2]

$$\binom{m-1}{c}^2 + m \cdot (2k-2) + 2k^2.$$

In the final step, every party broadcasts its shares to all other parties, and then adds all the shares it receives. The complexity of this step is $\binom{m}{c}$.

Considering the cost of replacing each gate G in the circuit to compile by $6m-5$ gates, and the cost to compute each of these gates using the protocol Π, the total number of gates N_{g} is upper bounded by

$$(6m-5) \cdot \left(\binom{m-1}{c}^2 + m(2k-2) + 2k^2 \right).$$

Using the parameters $m = 5$ and $c = 2$ from the AIS compiler instantiation [3], we get $N_{\mathrm{g}} = 8150$. This yields a tolerated leakage probability of $p \approx 2^{-26}$ and an exponent $e = \log(8150)/\log(3) \approx 8.19$ in the asymptotic complexity $\mathcal{O}\big(|C| \cdot \kappa^e\big)$ of the AIS compiler.

These results are to be compared to the $p \approx 2^{-7.09}$ and $e \approx 7.5$ achieved by our construction. In either case ($N_{\mathrm{g}} = 5712$ as claimed in [3] or $N_{\mathrm{g}} = 8150$ according to our analysis), our construction achieves a slightly better complexity while tolerating a much higher leakage probability. We stress that further instantiations of the AIS scheme (based on different MPC protocols) or of our scheme (based on different gadgets) could lead to better asymptotic complexities and/or tolerated leakage probabilities.

4.7 A Proof-of-Concept Secure AES Implementation

In this section, we describe and report the performances of a proof-of-concept implementation of the RPE compiler with the base gadgets from Section 4.5 as well as a protected AES implementation. The source code of these implementations is publicly available at:

`https://github.com/CryptoExperts/poc-expanding-compiler`

All implementations were run on a laptop computer (Intel(R) Core(TM) i7-8550U CPU, 1.80GHz with 4 cores) using Ubuntu operating system and various C, python and sage libraries.

[2]The authors claim in their paper a complexity of $\binom{m-1}{c}^2 + 2mk$, since they do not take into account the copy gates needed to compute the product of input shares.

4.7.1 Circuit Compiler

First, we developed an implementation in python of a compiler CC, that given three n-share gadgets G_{add}, G_{mult}, G_{copy} and an expansion level k, outputs the compiled gadgets $G^{(k)}_{\text{add}}$, $G^{(k)}_{\text{copy}}$,$G^{(k)}_{\text{mult}}$, each as a C function. The variables' type is given as a command line argument. Table 4.2 shows the complexity of the compiled gadgets from Section 4.5 using the compiler with several expansion levels k, as well as their execution time in milliseconds when run in C on randomly generated 8-bit integers. For the generation of random variables, we consider that an efficient external random number generator is available in practice, and so we simply use the values of an incremented counter variable to simulate random gates.

Table 4.2: Complexity and execution time (in ms, on an Intel i7-8550U CPU) for compiled gadgets $G^{(k)}_{\text{add}}$, $G^{(k)}_{\text{copy}}$, $G^{(k)}_{\text{mult}}$ from Section 4.5 implemented in C.

k	# shares	Gadget	Complexity $(N_{\text{a}}, N_{\text{c}}, N_{\text{m}}, N_{\text{r}})$	Execution time
1	3	$G^{(1)}_{\text{add}}$	(15, 6, 0, 6)	$1,69.10^{-4}$
		$G^{(1)}_{\text{copy}}$	(12, 9, 0, 6)	$1,67.10^{-4}$
		$G^{(1)}_{\text{mult}}$	(28, 23, 9, 11)	$5,67.10^{-4}$
2	9	$G^{(2)}_{\text{add}}$	(297, 144, 0, 144)	$2,21.10^{-3}$
		$G^{(2)}_{\text{copy}}$	(288, 153, 0, 144)	$2,07.10^{-3}$
		$G^{(2)}_{\text{mult}}$	(948, 582, 81, 438)	$9,91.10^{-3}$
3	27	$G^{(3)}_{\text{add}}$	(6183, 3078, 0, 3078)	$9,29.10^{-2}$
		$G^{(3)}_{\text{copy}}$	(6156, 3105, 0, 3078)	$9,84.10^{-2}$
		$G^{(3)}_{\text{mult}}$	(23472, 12789, 729, 11385)	$3,67.10^{-1}$

It can be observed that both the complexity and running time grow by almost the same factor with the expansion level, with multiplication gadgets being the slowest as expected. Base gadgets with $k = 1$ roughly take 10^{-4} ms, while these gadgets expanded 2 times ($k = 3$) take between 10^{-2} and 10^{-1} ms. The difference between the linear cost of addition and copy gadgets, and the quadratic cost of multiplication gadgets can also be observed through the gadgets' complexities.

4.7.2 AES Implementation

We describe hereafter a proof-of-concept AES implementation protected with our instantiation of the RPE compiler. We start by describing the underlying AES circuit (over $\mathbb{K} = \text{GF}(256)$), followed by an analysis of the implementation in C of the complete algorithm.

AES circuit. We first describe the non-linear part of the AES, namely the sbox computation. For the field exponentiation ($x \mapsto x^{254}$ over GF(256)), we use the circuit representation of the processing proposed in [43] and presented in Figure 4.4. It corresponds to the *addition chain* $(1, 2, 4, 8, 9, 18, 19, 36, 55, 72, 127, 254)$ and it has been chosen due to its optimality regarding the number of multiplications (11 in total). Each time an intermediate result had to be reused, a copy gate (marked with $\|$) has been inserted.

For the second part of the sbox, the affine function is implemented according to the following equation:

$$\text{Affine}(x) = ((((((207x)^2 + 22x)^2 + 1x)^2 + 73x)^2 + 204x)^2 + 168x)^2 + 238x)^2 + 5x + 99$$

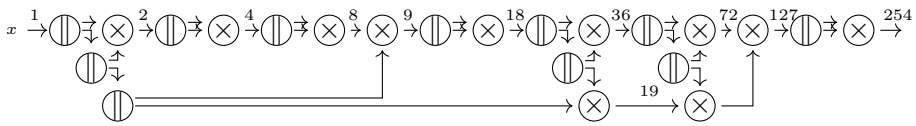

Figure 4.4: Circuit for the exponentiation $x \mapsto x^{254}$.

with the necessary copy gates. Similarly, the inverse of the affine function is implemented for the sbox inversion as follows:

$$\text{Affine}^{-1}(x) = (((((((147x)^2 + 146x)^2 + 190x)^2 + 41x)^2 + 73x)^2 + 139x)^2 + 79x)^2 + 5x + 5$$

The rest of the operations (MixColumns, ShiftRows, AddRoundKey) are considered as in a standard AES, while adding the necessary copy gates.

Gate count. Table 4.3 displays the gate count vectors for AES-128 encryption/decryption procedures as well as for their building blocks. The sbox (resp. sbox inversion) gate count vector was computed as the sum of the gate count vectors of both the exponentiation and affine (resp. affine inversion) functions. We recall that N_{a}, N_{c}, N_{m}, N_{r} stand for the number of addition gates, copy gates, multiplication gates, and random gates, respectively.

Table 4.3: AES operations complexity.

AES Operation	**Complexity** $(N_{\mathbf{a}}, N_{\mathbf{c}}, N_{\mathbf{m}}, N_{\mathbf{r}})$
AddRoundKey (for 1 byte)	$(1, 0, 0, 0)$
SubBytes (for 1 byte)	$(8, 25, 26, 0)$
MixColumns (for all columns)	$(60, 60, 16, 0)$
ShiftRows (for all rows)	$(0, 0, 0, 0)$
AES-128 encryption	$(1996, 4540, 4304, 0)$
SubBytes Inversion (for 1 byte)	$(8, 25, 26, 0)$
MixColumns Inversion (for all columns)	$(104, 104, 36, 0)$
ShiftRows Inversion (for all rows)	$(0, 0, 0, 0)$
AES-128 decryption	$(2392, 4936, 4484, 0)$

Using the gadgets G_{add}, G_{mult} and G_{copy} proposed in Section 4.5 for the compilation of the AES algorithm, we obtain the instantiation given in Equation (4.19) of the matrix $\boldsymbol{M}$ introduced in Section 4.4. Applying the same complexity analysis done previously on the gate count vectors, we display in Figure 4.5 the total number of gates in the AES-128 encryption/decryption procedures as functions of the level k. For instance, for the same security level of 2^{-76} exhibited in Section 4.5 for the gadgets of Figure 4.2, the AES-128 would have to be compiled at a level $k = 9$, and would count around 10^{16} gates.

Implementation in C. An n-share AES-128 implementation was developed in C from the above description. Compiled gadgets from Section 4.7.1 were used for basic operations (addition, multiplication, copy), as generated using our circuit compiler described in Section 4.7.1. We chose the C 8-bit unsigned integer type, and considered operations in $\mathsf{GF}(256)$. For the generation of random values, we assume the availability of an efficient (pseudo)random number generator, and so we simply considered the values of an incremented counter variable to simulate the cost.

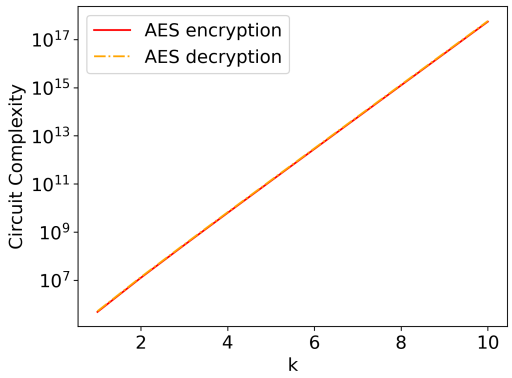

Figure 4.5: Number of gates after compilation of AES-128 encryption/decryption circuits with respect to the level k.

Table 4.4 shows the AES-128 execution time on a 16-byte message with 10 pre-computed sub-keys, using compiled gadgets $G_{\text{add}}^{2(k)}$, $G_{\text{copy}}^{1(k)}$, $G_{\text{mult}}^{1(k)}$, with respect to the expansion level k and sharing order $n = 3^k$. It can be seen that the execution time increases with the expansion level with a similar growth as in Table 4.2. This is because the complexity of the AES circuit strongly depends on the gadgets that are used to replace each gate in the original arithmetic circuit. For example, with our 3-share gadgets that tolerate a leakage probability of $p \approx 2^{-8}$, a 27-share ($k = 3$) AES-128 takes almost 200 milliseconds to encrypt or decrypt a message.

Table 4.4: Standard and n-share AES-128 execution time (in ms, on an Intel i7-8550U CPU) using compiled gadgets $G_{\text{add}}^{2(k)}$, $G_{\text{copy}}^{1(k)}$, $G_{\text{mult}}^{1(k)}$.

AES Version	**Execution Time (in ms)**	
	Encryption	**Decryption**
Standard (no sharing)	0.06	0.05
3-share ($k = 1$)	1.08	1.07
9-share ($k = 2$)	11.71	10.26
27-share ($k = 3$)	200.29	197.70

4.8 Conclusion

In this chapter, we introduced the powerful concept of random probing expansion and provided security-complexity trade-offs on practical implementations inspired by the original work on expansion using multi-party computation. We compared our constructions to the previous ones and showed that we achieved better results in terms of security and complexity. We also showed the result of the expansion on a real-life AES circuit to exhibit the the approach. Indeed, the obtained performance is not usable in practice yet. The implementation is a proof-of-concept to show the current advantages and limitations of constructing circuits with proven security levels in the random probing model. This construction opens the door for many future works that try to optimize constructions in the random probing model until we can construct real-life implementations with proven security levels and good performance.

In the following chapters, we exhibit many improvements that offer better trade-offs. We dive more profoundly into the random probing expansion analysis to test its limits and achieve higher security levels with better tolerated leakage rates and lower complexities.

Chapter 5

Generic RPE Constructions

In this chapter, we provide an in-depth analysis of the random probing expandability security notion discussed in the previous chapter. We show that the RPE notion can be made tighter, and we exhibit strong relations between RPE and the strong non-interference (SNI) composition notion for probing-secure gadgets.

From these results, we introduce the first generic constructions of gadgets achieving RPE for any number of shares and with nearly optimal amplification orders. These generic gadgets are derived from the widely known Ishai-Sahai-Wagner (ISW) construction. We show that the obtained expanding compiler can approach a quadratic complexity depending on the leakage probability that must be tolerated: the smaller the leakage probability, the closer the complexity to $\mathcal{O}(\kappa^2)$. We further introduce a new multiplication gadget achieving the optimal amplification order, which allows us to improve the convergence to a quadratic complexity.

Finally, we provide new concrete constructions of copy, addition, and multiplication gadgets achieving maximal amplification orders for small numbers of shares. These gadgets yield much more efficient instantiations than all the previous schemes (including the analysed ISW-based constructions). While slightly decreasing the tolerated leakage probability to $p = 2^{-7.5}$, our 3-share instantiation achieves a complexity of $\mathcal{O}(\kappa^{3.9})$. Moreover, our 5-share instantiation drops the complexity to $\mathcal{O}(\kappa^{3.2})$. The contributions in this chapter are published in [22].

5.1 A Closer Look at Random Probing Expandability

In this section, we give a closer look at the RPE notion. We first show that it naturally splits into two different notions, that we shall call RPE1 and RPE2, and further introduce a tighter variant which will be useful for our purpose. We then study the relations between (tight) RPE and the *Strong Non-Interference* (SNI) notion used for probing security. We exhibit strong connections between (tight) RPE1 and SNI, which will be very useful for our construction results depicted in later sections.

5.1.1 Splitting RPE

From Definition 10, we can define two sub-properties which are jointly equivalent to RPE. In the first one, designated by RPE1, the set J is constrained to satisfy $|J| \leq t$ and $J' = J$ (the simulator does not choose J').

Definition 15 (Random Probing Expandability 1)**.** *Let $f : \mathbb{R} \to \mathbb{R}$. An n-share gadget $G : \mathbb{K}^n \times \mathbb{K}^n \to \mathbb{K}^n$ is (t, f)-RPE1 if there exists a deterministic algorithm Sim_1^G and a probabilistic algorithm Sim_2^G such that for every input $(\widehat{x}, \widehat{y}) \in \mathbb{K}^n \times \mathbb{K}^n$, for every set $J \subseteq [n]$,*

such that $|J| \leq t$*, and for every* $p \in [0,1]$*, the random experiment*

$$\begin{aligned} &W \leftarrow \mathsf{LeakingWires}(G,p) \\ &(I_1, I_2) \leftarrow \mathsf{Sim}_1^G(W, J) \\ &\mathit{out} \leftarrow \mathsf{Sim}_2^G(W, J, \widehat{x}|_{I_1}, \widehat{y}|_{I_2}) \end{aligned}$$

ensures that

1. *the failure events* $\mathbb{F}_1 \equiv \big(|I_1| > t\big)$ *and* $\mathbb{F}_2 \equiv \big(|I_2| > t\big)$ *verify*

$$\Pr(\mathbb{F}_1) = \Pr(\mathbb{F}_2) = \varepsilon \quad \textit{and} \quad \Pr(\mathbb{F}_1 \wedge \mathbb{F}_2) = \varepsilon^2 \tag{5.1}$$

   with $\varepsilon = f(p)$ *(in particular* $\mathbb{F}_1$ *and* $\mathbb{F}_2$ *are mutually independent),*

2. *the output distribution satisfies*

$$\mathit{out} \overset{id}{=} \big(\mathsf{AssignWires}(G, W, (\widehat{x}, \widehat{y}))\ ,\ \widehat{z}|_J\big) \tag{5.2}$$

where $\widehat{z} = G(\widehat{x}, \widehat{y})$*.*

In the second definition, designated by RPE2, J' is chosen by the simulator such that $J' \subseteq [n]$ with $|J'| = n - 1$ (and J does not matter anymore).

Definition 16 (Random Probing Expandability 2)**.** *Let* $f : \mathbb{R} \to \mathbb{R}$*. An* n*-share gadget* $G : \mathbb{K}^n \times \mathbb{K}^n \to \mathbb{K}^n$ *is* (t, f)*-RPE2 if there exists a deterministic algorithm* Sim_1^G *and a probabilistic algorithm* Sim_2^G *such that for every input* $(\widehat{x}, \widehat{y}) \in \mathbb{K}^n \times \mathbb{K}^n$*, for every* $p \in [0,1]$*, the random experiment*

$$\begin{aligned} &W \leftarrow \mathsf{LeakingWires}(G,p) \\ &(I_1, I_2, J) \leftarrow \mathsf{Sim}_1^G(W) \\ &\mathit{out} \leftarrow \mathsf{Sim}_2^G(W, J, \widehat{x}|_{I_1}, \widehat{y}|_{I_2}) \end{aligned}$$

ensures that

1. *the failure events* $\mathbb{F}_1 \equiv \big(|I_1| > t\big)$ *and* $\mathbb{F}_2 \equiv \big(|I_2| > t\big)$ *verify*

$$\Pr(\mathbb{F}_1) = \Pr(\mathbb{F}_2) = \varepsilon \quad \textit{and} \quad \Pr(\mathbb{F}_1 \wedge \mathbb{F}_2) = \varepsilon^2 \tag{5.3}$$

   with $\varepsilon = f(p)$ *(in particular* $\mathbb{F}_1$ *and* $\mathbb{F}_2$ *are mutually independent),*

2. J *is such that* $J \subseteq [n]$ *with* $|J| = n - 1$

3. *the output distribution satisfies*

$$\mathit{out} \overset{id}{=} \big(\mathsf{AssignWires}(G, W, (\widehat{x}, \widehat{y}))\ ,\ \widehat{z}|_J\big) \tag{5.4}$$

where $\widehat{z} = G(\widehat{x}, \widehat{y})$*.*

This split is somehow a partition of the RPE notion since we have:

$$G \text{ is } (t,f)\text{-RPE} \iff G \text{ is } (t,f)\text{-RPE1 } \textit{and } G \text{ is } (t,f)\text{-RPE2}$$

for any gadget G. As a result of the above equivalence, we can show that a gadget achieves RPE1 and RPE2 independently in order to obtain RPE for this gadget. Formally, we use the following lemma.

Lemma 3. *An* n*-share gadget* $G : \mathbb{K}^n \times \mathbb{K}^n \to \mathbb{K}^n$ *which is* (t, f_1)*-RPE1 and* (t, f_2)*-RPE2 is also* (t, f)*-RPE with* $f(p) \geq \max(f_1(p), f_2(p))$ *for every* $p \in [0,1]$*.*

We can refine the upper bounds on the amplification order introduced in Section 4.4.1 with respect to this split. In Lemma 1, the bound $d \leq t + 1$ applies to both RPE1 and RPE2, while the bound $d \leq (3 - \ell) \cdot (n - t)$ only applies to RPE1. Similarly, in Lemma 2, the bound $d \leq (t+1)/2$ applies to both RPE1 and RPE2, while the bound $d \leq (n - t)$ only applies to RPE1.

5.1.2 Tightening RPE

We introduce a tighter version of the RPE security property. The so-called *tight random probing expandability* (TRPE) is such that a failure occurs when the simulation requires more than t input shares (as in the original RPE notion) but also whenever this number of shares is greater than the size of the leaking set W. Formally, the failure event $\mathbb{F}_j$ is defined as

$$\mathbb{F}_j \equiv \big(|I_j| > \min(t, |W|)\big)$$

for every $j \in [\ell]$. We give a formal definition in Definition 17. The text in blue points to the differences from the original RPE definition (*i.e.* Definition 10).

Definition 17 (Tight Random Probing Expandability)**.** *Let $f : \mathbb{R} \to \mathbb{R}$. An n-share gadget $G : \mathbb{K}^n \times \mathbb{K}^n \to \mathbb{K}^n$ is (t, f)-*tight random probing expandable (TRPE) *if there exists a deterministic algorithm Sim_1^G and a probabilistic algorithm Sim_2^G such that for every input $(\widehat{x}, \widehat{y}) \in \mathbb{K}^n \times \mathbb{K}^n$, for every set $J \subseteq [n]$ and for every $p \in [0, 1]$, the random experiment*

$$\begin{aligned} W &\leftarrow \mathsf{LeakingWires}(G, p) \\ (I_1, I_2, J') &\leftarrow \mathsf{Sim}_1^G(W, J) \\ \mathit{out} &\leftarrow \mathsf{Sim}_2^G(W, J', \widehat{x}|_{I_1}, \widehat{y}|_{I_2}) \end{aligned}$$

ensures that

1. *the failure events $\mathbb{F}_1 \equiv \big(|I_1| > \min(t, |W|)\big)$ and $\mathbb{F}_2 \equiv \big(|I_2| > \min(t, |W|)\big)$ verify*

$$\Pr(\mathbb{F}_1) = \Pr(\mathbb{F}_2) = \varepsilon \quad \textit{and} \quad \Pr(\mathbb{F}_1 \wedge \mathbb{F}_2) = \varepsilon^2 \tag{5.5}$$

   with $\varepsilon = f(p)$ (in particular $\mathbb{F}_1$ and $\mathbb{F}_2$ are mutually independent),

2. *J' is such that $J' = J$ if $|J| \leq t$ and $J' \subseteq [n]$ with $|J'| = n - 1$ otherwise,*

3. *the output distribution satisfies*

$$\mathit{out} \overset{id}{=} \big(\mathsf{AssignWires}(G, W, (\widehat{x}, \widehat{y}))\ ,\ \widehat{z}|_{J'}\big) \tag{5.6}$$

   where $\widehat{z} = G(\widehat{x}, \widehat{y})$,

This tighter security property will be instrumental in the following to obtain generic RPE constructions. Similarly to the original RPE property, the TRPE property can be split into two intermediate properties, namely TRPE1 and TRPE2.

Definition 18 (Tight Random Probing Expandability 1)**.** *Let $f : \mathbb{R} \to \mathbb{R}$. An n-share gadget $G : \mathbb{K}^n \times \mathbb{K}^n \to \mathbb{K}^n$ is (t, f)-*tight random probing expandable (TRPE) *if there exists a deterministic algorithm Sim_1^G and a probabilistic algorithm Sim_2^G such that for every input $(\widehat{x}, \widehat{y}) \in \mathbb{K}^n \times \mathbb{K}^n$, for every set $J \subseteq [n]$, such that $|J| \leq t$, and for every $p \in [0, 1]$, the random experiment*

$$\begin{aligned} W &\leftarrow \mathsf{LeakingWires}(G, p) \\ (I_1, I_2) &\leftarrow \mathsf{Sim}_1^G(W, J) \\ \mathit{out} &\leftarrow \mathsf{Sim}_2^G(W, J, \widehat{x}|_{I_1}, \widehat{y}|_{I_2}) \end{aligned}$$

ensures that

1. *the failure events* $\mathbb{F}_1 \equiv \big(|I_1| > \min(t, |W|)\big)$ *and* $\mathbb{F}_2 \equiv \big(|I_2| > \min(t, |W|)\big)$ *verify*

$$\Pr(\mathbb{F}_1) = \Pr(\mathbb{F}_2) = \varepsilon \quad \textit{and} \quad \Pr(\mathbb{F}_1 \wedge \mathbb{F}_2) = \varepsilon^2 \tag{5.7}$$

with $\varepsilon = f(p)$ *(in particular* $\mathbb{F}_1$ *and* $\mathbb{F}_2$ *are mutually independent),*

2. *the output distribution satisfies*

$$out \overset{id}{=} \big(\mathsf{AssignWires}(G, W, (\widehat{x}, \widehat{y}))\ ,\ \widehat{z}|_J\big) \tag{5.8}$$

where $\widehat{z} = G(\widehat{x}, \widehat{y})$*,*

Definition 19 (Tight Random Probing Expandability 2)**.** *Let* $f : \mathbb{R} \to \mathbb{R}$*. An* n*-share gadget* $G : \mathbb{K}^n \times \mathbb{K}^n \to \mathbb{K}^n$ *is* (t, f)-tight random probing expandable (TRPE) *if there exists a deterministic algorithm* Sim_1^G *and a probabilistic algorithm* Sim_2^G *such that for every input* $(\widehat{x}, \widehat{y}) \in \mathbb{K}^n \times \mathbb{K}^n$*, for every* $p \in [0, 1]$*, the random experiment*

$$\begin{aligned} W &\leftarrow \mathsf{LeakingWires}(G, p) \\ (I_1, I_2, J) &\leftarrow \mathsf{Sim}_1^G(W) \\ out &\leftarrow \mathsf{Sim}_2^G(W, J, \widehat{x}|_{I_1}, \widehat{y}|_{I_2}) \end{aligned}$$

ensures that

1. *the failure events* $\mathbb{F}_1 \equiv \big(|I_1| > \min(t, |W|)\big)$ *and* $\mathbb{F}_2 \equiv \big(|I_2| > \min(t, |W|)\big)$ *verify*

$$\Pr(\mathbb{F}_1) = \Pr(\mathbb{F}_2) = \varepsilon \quad \textit{and} \quad \Pr(\mathbb{F}_1 \wedge \mathbb{F}_2) = \varepsilon^2 \tag{5.9}$$

with $\varepsilon = f(p)$ *(in particular* $\mathbb{F}_1$ *and* $\mathbb{F}_2$ *are mutually independent),*

2. J *is such that* $J \subseteq [n]$ *with* $|J| = n - 1$

3. *the output distribution satisfies*

$$out \overset{id}{=} \big(\mathsf{AssignWires}(G, W, (\widehat{x}, \widehat{y}))\ ,\ \widehat{z}|_J\big) \tag{5.10}$$

where $\widehat{z} = G(\widehat{x}, \widehat{y})$*,*

Lemma 3 also applies to the case of TRPE. Moreover the upper bounds on the amplification order for RPE in Lemma 1 and Lemma 2 further apply to the amplification order for TRPE (which holds by definition). We show hereafter that the TRPE notion is actually equivalent to the RPE notion if and only if the function f is of maximal amplification order $t + 1$.

Lemma 4. *Let* $t \in \mathbb{N}$*, let* $f : \mathbb{R} \to \mathbb{R}$ *of amplification order* d*. Let* G *be a gadget.*

1. *If* G *achieves* (t, f)*-TRPE, then it achieves* (t, f')*-RPE for some* $f' : \mathbb{R} \to \mathbb{R}$ *of amplification order* $d' \geq d$*.*

2. *If* G *is of amplification order* d *with respect to* t *(i.e.* d *is the max amplification order of a function* f *for which* G *is* (t, f)*-RPE), then for all* $f' : \mathbb{R} \to \mathbb{R}$ *for which* G *achieves* (t, f')*-TRPE,* f' *is of amplification order* $d' \leq d$*.*

3. *If* $d = t + 1$*, then* G *achieves* (t, f)*-TRPE if and only if* G *achieves* (t, f)*-RPE.*

Proof. The proof for the first two points is easy. In particular, for the first point, if G achieves TRPE with an amplification order of d, then G achieves RPE with amplification order at least d, since a failure in the TRPE setting *i.e.* $|I_j| > \min(t, |W|)$ does not necessarily imply a failure in the RPE setting *i.e.* $|I_j| > t$, meanwhile if there is no failure for TRPE for a leaking set of wires W, then this implies that $|I_j| \leq \min(t, |W|) \leq t$ so there is no failure in the RPE setting either.

As for the second point, the proof is similar: if G achieves an amplification of d in the RPE setting, then it achieves an amplification order of at most d in the TRPE setting, since a failure in the RPE setting *i.e.* $|I_j| > t$ immediately implies a failure in the TRPE setting $|I_j| > \min(t, |W|)$. But also, even if there is no failure for a leaking set of wires W in the RPE setting we might still have a failure in the TRPE setting for the same set W. This is mainly the case where W can be simulated with sets of input shares I_j such that $|W| < |I_j| \leq t$, so we have $|I_j| \leq t$ (*i.e.* no failure for RPE) and $|I_j| > \min(t, |W|) = |W|$ (*i.e.* failure on TRPE). This concludes the proof for the second point.

We will now prove the third point. Let $d = t + 1$. We will show that for every set $J' \subseteq [n]$ of output shares and every leaking set of wires W, a failure occurs in the TRPE setting if and only if a failure also occurs in the RPE setting. If $|W| \geq t$, then the two settings are equivalent since $\min(t, |W|) = t$. We will thus only focus on the case $|W| < t$. Clearly, a failure in the RPE setting, *i.e.* $|I_j| > t$, implies a failure in the TRPE setting, *i.e.* $|I_j| > \min(t, |W|)$. Let us now show that the converse is also true.

We assume by contradiction that there exists J' and W implying a TRPE failure which is not an RPE failure, that is a set I_j satisfying $|W| < |I_j| \leq t$. We then show that there exists a leaking set W' of size $|W'| < t+1$ for which an RPE failure always occurs, which implies an amplification order strictly lower than $t+1$ and hence contradicts the lemma hypothesis. This set W' is constructed as $W' = W \cup I'_j$ for some set $I'_j \subset [n] \setminus I_j$ such that $|I'_j| = t + 1 - |I_j|$. The simulation of W' and J' then requires the input shares from $I_j \cup I'_j$. However, we have

$$|I_j \cup I'_j| = |I_j| + |I'_j| = t + 1$$

implying an RPE failure, and

$$|W'| = |W \cup I'_j| \leq |W| + |I'_j| = |W| + t + 1 - |I_j| < |W| + t + 1 - |W| = t + 1.$$

Thus, we have built a failure set W' of size strictly less than the amplification order $t + 1$, which contradicts the hypothesis and hence concludes the proof. □

The above proof also applies to the case of the split notions, specifically for ((t, f)-RPE1, (t, f)-TRPE1) and for ((t, f)-RPE2, (t, f)-TRPE2).

5.1.3 Unifying (Tight) RPE and SNI

Strong non-interference (SNI) is a widely used notion to compose probing-secure gadgets [10]. In Section 3.3.1, we exhibit a relation between the SNI and the *random probing composability* (RPC) property (Proposition 1). Here, we study the relation between SNI and (T)RPE. We state hereafter some equivalence results between the (T)RPE1 and SNI notions, up to some constraints on the parameters. We first formally show that (T)RPE1 implies SNI.

Lemma 5. *Let $t \in \mathbb{N}$ and $f : \mathbb{R} \to \mathbb{R}$ of amplification order $t + 1$. Let G be a gadget which achieves (t, f)-TRPE1. Then G is also t-SNI.*

Proof. By definition of TRPE1 and by hypothesis on the amplification order, there exist input sets $I_1, \ldots, I_\ell$ which can perfectly simulate any leaking wires set W such that $|W| \leq t$ and

any set of output shares J such that $|J| \leq t$, satisfying $|I_1|, \ldots, |I_\ell| \leq |W|$. Consequently, there exist input sets $I_1, \ldots, I_\ell$ which can perfectly simulate any leaking wires set W such that $|W| = t_i \leq t$ and any set of output shares J such that $|W| + |J| \leq t$ with $|I_1|, \ldots, |I_\ell| \leq t_i$. G is thus t-SNI. □

We now show that SNI implies TRPE1 up to some constraints on the parameters.

Lemma 6. *Let $\tau, \ell \in \mathbb{N}$. Let G be an ℓ-to-1 gadget which achieves τ-SNI. Then G satisfies (t, f)-TRPE1 for some $f : \mathbb{R} \to \mathbb{R}$ with an amplification order of*

$$d \geq \frac{1}{\ell} \min(t + 1, \tau - t + 1) \ .$$

Proof. Since G is τ-SNI, then for any set of leaking wires W and output shares J such that $|W| + |J| \leq \tau$, the wires indexed by W and the output shares indexed by J can be perfectly simulated from input shares indexed by I_1, …, I_ℓ such that $|I_j| \leq |W|$ for every $1 \leq j \leq \ell$. In the TRPE1 property, the set J of output shares can be any set of size $|J| \leq t$ so we can assume $|J| = t$ without loss of generality.

For a leaking set W of size $|W| < \min(t+1, \tau-t+1)$ no failure event occurs. Indeed τ-SNI and $|W| < \tau - t + 1$ implies $|W| + |J| \leq \tau$ and hence the existence of the sets I_1, …, I_ℓ allowing the simulation with $|I_j| \leq |W|$. And $|W| < t + 1$ implies $|I_j| \leq \min(t, |W|)$ for every j which implies the absence of failure. Then for a leaking set W of size $|W| \geq \min(t + 1, \tau - t + 1)$, no condition remains to rule out simulation failures and one could actually get a failure for every input. In the latter case, the amplification order would equal $\frac{1}{\ell} \min(t + 1, n - t)$, but in all generality it could be higher (*i.e.* this value is a lower bound). □

We give an illustrative summary of the relations between RPE1, TRPE1 and SNI in Figure 5.1 (d denotes the amplification order of the function f). We hence observe an equivalence between the three notions up to some constraints on the parameters t, d, τ and ℓ.

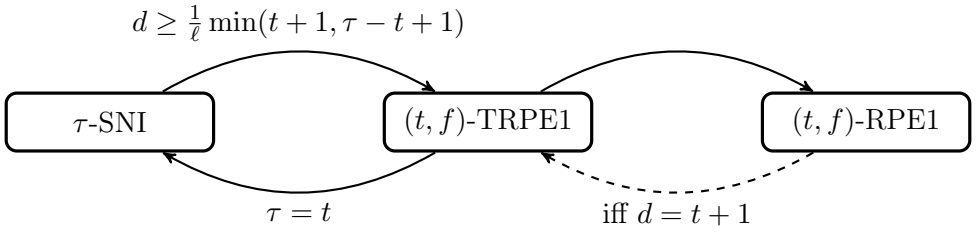

Figure 5.1: Summary of relations between the different notions.

Relation and separation between (T)RPE2 and SNI. For a given n-share gadget G, the (T)RPE2 notion exclusively focuses on the simulation of a set of leaking intermediate variables together with a chosen set of $(n-1)$ output shares. If G is τ-SNI for $\tau < n - 1$, then nothing can be claimed on the simulation of the latter sets. But if G is $(n-1)$-SNI, then any set of $(n-1)$ output shares can be perfectly simulated without the knowledge of any input share. Concretely, it implies that G is (t, f)-(T)RPE2 of amplification order at least 1 as a chosen output set of $(n-1)$ shares alone can be perfectly simulated without any additional knowledge on the input shares. Namely, we have

$$(n-1)\text{-SNI} \Rightarrow (t, f)\text{-(T)RPE2 of amplification order at least 1.}$$

Nevertheless, there is no relation from τ-SNI to (t, f)-(T)RPE2 for amplification orders strictly greater than 1 as (T)RPE2 would then consider leaking sets of size larger than or equal to n (for n-share gadgets, $\tau < n$). On the other side, there is no direct implication either from (t, f)-(T)RPE2 to τ-SNI since the former property does not consider all possible output sets of size $(n-1)$, but only a chosen one.

5.2 Generic Constructions: Addition and Copy Gadgets

In Section 4.5, we exhibited the first construction of RPE gadgets explicitly designed for a small number of shares, namely $n = 3$. The constructed addition and copy gadgets achieved the maximal amplification order for 3-shares construction, while the multiplication gadget only achieved an amplification order of 3/2. A natural open question is the definition of RPE gadgets with good amplification orders, typically achieving or approaching the upper bounds exhibited in Section 4.4.1, for *any* number of shares n.

As intuitively proposed in [18] for small gadgets, copy and addition gadgets can be naturally derived from a refresh gadget. Such a gadget takes one sharing as input and outputs a new refreshed sharing of the same value. In this section, we exhibit generic constructions of addition and copy gadgets based on refresh gadgets and show that these gadgets can achieve maximal amplification orders if the underlying refresh gadget satisfies specific properties that we detail later. We also prove that the well-known ISW [59] and $\mathcal{O}(n \log n)$ [14] refresh gadgets can be used to construct addition and copy gadgets with maximal amplification orders for any number of shares.

5.2.1 Generic Copy Gadgets

Algorithm 4 displays the generic construction for the copy gadget from a refresh gadget. It simply consists in refreshing the input sharing twice to obtain two fresh copies.

Algorithm 4: Copy gadget G_{copy}

Input : $(a_1, \ldots, a_n)$ input sharing
Output: $(e_1, \ldots, e_n)$, $(f_1, \ldots, f_n)$ fresh copies of $(a_1, \ldots, a_n)$
1 $(e_1, \ldots, e_n) \leftarrow G_{\text{refresh}}(a_1, \ldots, a_n)$;
2 $(f_1, \ldots, f_n) \leftarrow G_{\text{refresh}}(a_1, \ldots, a_n)$;

We have the following lemma.

Lemma 7. *Let $G_{refresh}$ be an n-share (t, f)-TRPE refresh gadget of amplification order d. Then, the copy gadget G_{copy} displayed in Algorithm 4 is (t, f')-TRPE of amplification order $d' = d$.*

Proof. To prove that G_{copy} is TRPE achieving the same amplification order d as the underlying refresh gadget G_{refresh}, we need to prove that any set of leaking wires W such that $|W| \leq d-1$ can be perfectly simulated together with any sets of outputs wires $J_1, J_2 \subseteq [n]$ (such that J_1 refers to the first output e and J_2 to the second output f) from a set of input wires I such that $|I| \leq \min(t, |W|)$. In addition, we know from Lemma 1 that the maximal amplification order achievable in the TRPE setting is $d_{\max} \leq \min(t+1, 2(n-t))$. Since we consider sets W of size at most $|W| \leq d-1 \leq \min(t+1, 2(n-t)) - 1 \leq t$ then we need to prove that $|I| \leq \min(t, |W|) = |W|$.

The leaking set W can be split into two distinct subsets W_1 and W_2 such that $W = W_1 \cup W_2$ where W_1 (resp. W_2) is the set of leaking wires of G_{refresh} for the output e (resp. f). Let $J_1, J_2 \subseteq [n]$. We consider four cases:

- $|J_1| \leq t, |J_2| \leq t$: since $|W| \leq d-1$, then $|W_1|, |W_2| \leq d-1$. Since G_{refresh} achieves an amplification order d, then by definition of TRPE, the sets W_1 and J_1 can be simulated with a set of input shares I_1 such that $|I_1| \leq \min(|W_1|, t)$. Similarly, the sets W_2 and J_2 can be simulated with a set of input shares I_2 such that $|I_2| \leq \min(|W_2|, t)$. As a consequence, set I defined as $I = I_1 \cup I_2$ is enough to simulate $W = W_1 \cup W_2$ and both output shares J_1 and J_2. Furthermore, we have

$$|I| \leq |I_1| + |I_2| \leq \min(|W_1|, t) + \min(|W_2|, t) \leq |W| = \min(|W|, t)$$

- $|J_1| > t, |J_2| > t$: in this case, we need to prove the existence of a set of input shares I such that $|I| \leq \min(t, |W|) = |W|$ (since $|W| \leq d-1 \leq t$) for which we can perfectly simulate W together with two chosen output sets J_1' and J_2' such that $|J_1'| = |J_2'| = n-1$. Since we have $W = W_1 \cup W_2$ such that $|W_1| \leq d-1, |W_2| \leq d-1$, then by definition of TRPE, there exists $J_1', |J_1'| = n-1$ such that W_1 and J_1' can be perfectly simulated from a set of inputs shares I_1 such that $|I_1| \leq \min(|W_1|, t)$. Similarly, there exists $J_2', |J_2'| = n-1$ such that W_2 and J_2' can be perfectly simulated from a set of inputs shares I_2 such that $|I_2| \leq \min(|W_2|, t)$. By choosing such sets J_1', J_2', the overall simulation of G_{copy} can be done with the set of input shares $I = I_1 \cup I_2$, and we have

$$|I| \leq |I_1| + |I_2| \leq \min(|W_1|, t) + \min(|W_2|, t) \leq |W| = \min(|W|, t)$$

- $|J_1| \leq t, |J_2| > t$: Since $|J_1| \leq t$, by definition of TRPE, W_1 and J_1 can be perfectly simulated from a set of input shares I_1 such that $|I_1| \leq \min(|W_1|, t)$. In addition, for $|J_2| > t$, we also know that we can choose a set J_2' such that $|J_2'| = n-1$ that can be perfectly simulated with W_2 from a set of input shares I_2 with $|I_2| \leq \min(|W_2|, t)$. By choosing such a set J_2', the overall simulation of G_{copy} can be achieved with the set of input wires $I = I_1 \cup I_2$, and we have

$$|I| \leq |I_1| + |I_2| \leq \min(|W_1|, t) + \min(|W_2|, t) \leq |W| = \min(|W|, t)$$

- $|J_1| > t, |J_2| \leq t$: the proof is exactly the reflect of the previous one.

Hence in the four cases, there is no failure tuple W of size $|W| < d$. In addition, since the refresh gadget is of amplification order d, then we know that a failure of size d exists. Then the gadget G_{copy} achieves an amplification order of d, which concludes the proof. □

As a consequence of this result, a TRPE refresh gadget directly yields a TRPE copy gadget achieving with the same amplification order. Both gadgets can then reach the upper bound for 1-input gadgets whenever $t + 1 = 2(n - t)$ implying an amplification order $d = \frac{2(n+1)}{3}$.

5.2.2 Generic Addition Gadgets

Algorithm 5 displays the generic construction for the addition gadget from a refresh gadget. It simply consists in refreshing both input sharings before adding them.

Algorithm 5: Addition Gadget G_{add}

Input : $(a_1, \ldots, a_n), (b_1, \ldots, b_n)$ input sharings
Output: $(c_1, \ldots, c_n)$ sharing of $a + b$

1 $(e_1, \ldots, e_n) \leftarrow G_{\text{refresh}}(a_1, \ldots, a_n)$;
2 $(f_1, \ldots, f_n) \leftarrow G_{\text{refresh}}(b_1, \ldots, b_n)$;
3 $(c_1, \ldots, c_n) \leftarrow (e_1 + f_1, \ldots, e_n + f_n)$;

We have the following lemma.

Lemma 8. *Let $G_{refresh}$ be an n-share refresh gadget and let G_{add} be the corresponding addition gadget displayed in Algorithm 5. Then if $G_{refresh}$ is (t, f)-RPE (resp. (t, f)-TRPE) of amplification order d, then G_{add} is (t, f')-RPE (resp. (t, f')-TRPE) for some f' of amplification order $d' \geq \lfloor \frac{d}{2} \rfloor$.*

Proof. We need to prove that when G_{refresh} is (t, f)-RPE (resp. (t, f')-TRPE) of amplification order d, then G_{add} is (t, f')-RPE (resp. (t, f')-TRPE) of amplification order at least $\lfloor \frac{d}{2} \rfloor$. We will prove the property for the RPE setting, and the proof for the TRPE setting will be exactly the same except for the notion of failure event which changes. This amounts to proving that:

1. Any set of leaking wires W such that $|W| < \lfloor \frac{d}{2} \rfloor$ can be simulated together with any set of outputs wires $J \subseteq [n]$ from sets of input wires I_1 on a and I_2 on b such that $|I_1| \leq t$ and $|I_2| \leq t$ (for TRPE we would have $|I_1| \leq \min(t, |W|)$ and $|I_2| \leq \min(t, |W|)$).
2. Any set of leaking wires such that $\lfloor \frac{d}{2} \rfloor \leq |W| < d$ can be simulated together with any set of outputs wires $J \subseteq [n]$ from sets of input wires I_1, I_2 such that $|I_1| \leq t$ or $|I_2| \leq t$ (because of the double failure, *i.e* failure on both inputs) (for TRPE we would have $|I_1| \leq \min(t, |W|)$ or $|I_2| \leq \min(t, |W|)$).

We proceed by building the necessary simulators for G_{add} from the simulators that already exist for G_{refresh}. Concretely, we split each set W of leaking wires, into four subsets $W = W_1^r \cup W_1^a \cup W_2^r \cup W_2^a$ where W_1^r (resp. W_2^r) is the set of leaking wires during the computation of $G_{\text{refresh}}(a_1, \ldots, a_n)$ (resp. $G_{\text{refresh}}(b_1, \ldots, b_n)$), and W_1^a (resp. W_2^a) is the set of leaking wires of $(e_1, \ldots, e_n)$ (resp. $(f_1, \ldots, f_n)$).

From these notations, we build a leaking set W' which contains W_1^r and W_2^r and also each input or pair of inputs of gates whose output is a wire in W_1^a or W_2^a. We have that

$$|W'| \leq |W_1^r| + |W_2^r| + 2|W_1^a| + 2|W_2^a| \leq 2|W|.$$

The new set W' can be split into two subsets W_1' and W_2' such that W_1' (resp. W_2') contains only leaking wires during the computation of $G_{\text{refresh}}(a_1, \ldots, a_n)$ (resp. $G_{\text{refresh}}(b_1, \ldots, b_n)$). We now demonstrate how we can simulate W' when the output set J is of size less that t ((T)RPE1) and when it is of size strictly more than t ((T)RPE2).

- if $|J| \leq t$ ((T)RPE1): we prove both properties 1 and 2:

  1. we assume that $|W| < \lfloor \frac{d}{2} \rfloor$. Then we consider the set $W' = W_1' \cup W_2'$ (as previously defined) such that

     $$|W'| \leq 2|W| < 2\lfloor \frac{d}{2} \rfloor \leq d$$

     and hence,

     $$|W_1'| < d \quad \text{and} \quad |W_2'| < d.$$

     From the (t, f)-RPE property of G_{refresh} and its amplification order, there exists an input set of shares of a I_1 such that $|I_1| \leq t$ (for TRPE we would have $|I_1| \leq \min(t, |W|)$) and I_1 perfectly simulates W_1' and any set J_1 of up to t variables e_i. Similarly, there exists an input set of shares of b I_2 such that $|I_2| \leq t$ (for TRPE we would have $|I_2| \leq \min(t, |W|)$) and I_2 perfectly simulates W_2' and any set J_2 of up to t variables f_i. J_1 and J_2 are chosen as the inputs e_i and f_i respectively of wires $e_i + f_i$ in set J. Namely $|J_1| = |J_2| = |J|$.

     From these definitions, I_1 and I_2 together perfectly simulate W' and J and are both of size less than t (less than $\min(t, |W|)$ for TRPE), which proves the first property in this scenario.

2. we now assume that $\lfloor \frac{d}{2} \rfloor \leq |W| < d$. Then we consider the set $W' = W'_1 \cup W'_2$ such that

$$|W'| \leq 2|W| < 2d$$

and hence,

$$|W'_1| < d \quad \text{or} \quad |W'_2| < d.$$

Without loss of generality, let us consider that $|W'_1| < d$ (the proof is similar in the opposite scenario). From the (t, f)-RPE property of G_{refresh} and its amplification order, there exists an input set of shares of a I_1 such that $|I_1| \leq t$ (for TRPE we would have $|I_1| \leq \min(t, |W|)$) and I_1 perfectly simulates W'_1 and any set J_1 of up to t variables e_i. There also exists an input set of shares of b I_2 which perfectly simulates W'_2 and any set J_2 of up to t variables f_i but not necessarily of size less than t (less than $\min(t, |W|)$ for TRPE). If J_1 and J_2 are chosen as the inputs e_i and f_i respectively of wires $e_i + f_i$ in set J, then sets I_1 and I_2 together perfectly simulate W' and J. In this case, we only have a failure on at most one of the inputs (b in this case), which concludes the proof for the second property.

At this point, we proved that G_{add} achieves an amplification order greater than or equal to $\lfloor \frac{d}{2} \rfloor$ for RPE1 (for TRPE1 in the TRPE setting).

- if $|J| > t$ ((T)RPE2): we prove both properties 1 and 2:

1. we assume that $|W| < \lfloor \frac{d}{2} \rfloor$. Then we consider the set $W' = W'_1 \cup W'_2$ (as previously defined) such that

$$|W'| \leq 2|W| < d.$$

W'_1 and W'_2 both point to leaking wires in instances of G_{refresh}. We denote by W''_1 the set of leaking wires on the first instance of G_{refresh} (on input a) such that W''_1 contains W'_1 and all the wires that are leaking within the second instance of G_{refresh} (designated by W'_2 in this second instance). Hence, we have that $|W''_1| \leq |W'_1 \cup W'_2| < d$. From the (t, f)-RPE ((t, f)-TRPE in the TRPE setting) property of G_{refresh} and its amplification order, there exists an input set of shares of a I_1 such that $|I_1| \leq t$ (for TRPE we would have $|I_1| \leq \min(t, |W|)$) and a set of output shares e_i J'_1 of size $n-1$ such that the input shares of I_1 perfectly simulate the wires designated by W''_1 and J'_1. Similarly, as both instances of G_{refresh} are identical, the same set I_2 but of input shares b perfectly simulates W''_2 (defined as the equivalent of W''_1 on the second instance) and J'_2 which points to the same output shares than J_1 but on f_i instead of e_i. We thus have two input sets I_1 and I_2 of size less than t (less than $\min(t, |W|)$ for TRPE2) whose shares perfectly simulate the wires W' and the elements $e_i + f_i$ of a set J' of size $n - 1$ with $i \in J'_1 = J'_2$. That concludes the proof for the first property.

2. we now assume that $\lfloor \frac{d}{2} \rfloor \leq |W| < d$. Then we consider the set $W' = W'_1 \cup W'_2$ such that

$$|W'| \leq 2|W| < 2d.$$

Without loss of generality, let us consider that $|W'_1| < d$ (the proof is similar in the opposite scenario). From the (t, f)-RPE ((t, f)-TRPE in the TRPE setting) property of G_{refresh} and its amplification order, there exists a set J'_1 such that $|J'_1| = n - 1$ and a set of input shares I_1 such that I_1 perfectly simulates W'_1 and J'_1 and $|I_1| \leq t$ (for TRPE we would have $|I_1| \leq \min(t, |W|)$). Thus, we can select a set J' of outputs of G_{add} such that J' corresponds to the outputs of J_1 (for each element e_i designated by J_1, $e_i + f_i$ is pointed by J). Then, by choosing $I_2 = [n]$, we have two input sets I_1 and I_2 which perfectly simulate W' and an output set J'

of size $n-1$ such that $|I_1| \leq t$ (for TRPE we would have $|I_1| \leq \min(t, |W|)$). That concludes the proof for the second property.

We thus proved that G_{add} achieves an amplification order greater than or equal to $\lfloor \frac{d}{2} \rfloor$ for RPE2 (for TRPE2 in the TRPE setting).

Since G_{add} has an amplification order greater than or equal to $\lfloor \frac{d}{2} \rfloor$ for RPE1 and RPE2 (resp. TRPE1 and TRPE2), then G_{add} is a (t, f')-RPE (resp. (t, f')-TRPE) addition gadget for some function f' of amplification order $d' \geq \lfloor \frac{d}{2} \rfloor$, which concludes the proof. □

The above lemma shows that a (T)RPE refresh gadget of amplification order d directly yields a (T)RPE addition gadget of amplification order at least $\lfloor \frac{d}{2} \rfloor$. If the refresh gadget achieves the optimal $d = \frac{2(n+1)}{3}$, then the generic addition gadget has an amplification order at least $\lfloor \frac{n}{3} \rfloor$, which is not far from the upper bound for two-input gadgets of $\frac{n+1}{2}$. In order to fill this gap, we introduce a new property which, when satisfied by its inherent refresh gadget G_{refresh}, makes the addition gadget TRPE with the same amplification order as G_{refresh}. We refer to this property as *strong TRPE2*.

Definition 20 (t-Strong TRPE2)**.** *Let G be an n-share 1-input gadget. Then G is t-Strong TRPE2 (abbreviated t-STRPE2) if and only if for any set J' of output shares indices and any set W of internal wires of G such that $|W| + |J'| \leq t$, there exists a set J of output share indices such that $J' \subseteq J$ and $|J| = n - 1$ and such that the assignment of the wires indexed by W together with the output shares indexed by J can be perfectly simulated from the input shares indexed by a set I of cardinality satisfying $|I| \leq |W| + |J'|$.*

This new property directly implies the TRPE2 property with maximal amplification order. Recall that G is t-TRPE2 with maximal amplification order if and only if for any set W of probed wires such that $|W| < t + 1$ (recall that the bound $n - t$ on the amplification order only applies to the TRPE1 property), there exists a set J of output shares indices such that $|J| = n - 1$ and such that an assignment of the wires indexed by W and the output shares indexed by J can be jointly perfectly simulated from input shares indexed in a set I such that $|I| \leq |W|$. This is recalled in the following corollary for the sake of completeness.

Corollary 4. *Let G be an n-share 1-input gadget. If G is t-Strong TRPE2 (abbreviated t-STRPE2), then G is (t, f)-TRPE2 for some f of amplification order $t + 1$.*

Having a refresh gadget which satisfies the property from Definition 20 results in tighter constructions for generic addition gadgets as stated in Lemma 9.

Lemma 9. *Let $G_{refresh}$ be an n-share refresh gadget and let G_{add} be the addition gadget described in Algorithm 5. Then for any $t \leq n - 1$, if $G_{refresh}$ is (t, f)-TRPE of amplification order $d \geq \min(t + 1, n - t)$ and $G_{refresh}$ is $(n - 1)$-STRPE2, then G_{add} is (t, f')-TRPE for any $t \leq n - 1$ for some f' of amplification order* $\min(t + 1, n - t)$.

Proof. Let G_{refresh} be a (t, f)-TRPE refresh gadget for any $t \leq n - 1$ with amplification order $d \geq \min(t + 1, n - t)$ and which satisfies Definition 20. We will prove that the construction of G_{add} using G_{refresh} described in Algorithm 5 is (t, f)-TRPE for any $t \leq n - 1$ of amplification order $\min(t + 1, n - t)$. This amounts to proving that:

1. Any set of leaking wires W such that $|W| < \min(t + 1, n - t)$ can be simulated together with any set of outputs wires $J \subseteq [n]$ from sets of input wires I_1 on a and I_2 on b such that $|I_1| \leq \min(t, |W|)$ and $|I_2| \leq \min(t, |W|)$.

2. Any set of leaking wires such that $\min(t+1, n-t) \leq |W| < 2\min(t+1, n-t)$ can be simulated together with any set of outputs wires $J \subseteq [n]$ from sets of input wires I_1, I_2 such that $|I_1| \leq \min(t, |W|)$ or $|I_2| \leq \min(t, |W|)$ (because of the double failure, *i.e* failure on both inputs).

Indeed, this amplification order being the maximum one achievable by 2-input addition gadgets, it would conclude the proof.

We will denote $(e_1, \ldots, e_n) = G_{\text{refresh}}(a_1, \ldots, a_n)$ and $(f_1, \ldots, f_n) = G_{\text{refresh}}(b_1, \ldots, b_n)$. Then the gadget G_{add} consists in the sharewise addition $(e_1 + f_1, \ldots, e_n + f_n)$ as described in Algorithm 5. We proceed by building the necessary simulators for G_{add} from the simulators that already exist for G_{refresh}. Concretely, we split each set W of leaking wires, into four subsets $W = W_1^r \cup W_1^a \cup W_2^r \cup W_2^a$ where W_1^r (resp. W_2^r) is the set of leaking wires during the computation of $G_{\text{refresh}}(a_1, \ldots, a_n)$ (resp. $G_{\text{refresh}}(b_1, \ldots, b_n)$), and W_1^a (resp. W_2^a) is the set of leaking wires of $(e_1, \ldots, e_n)$ (resp. $(f_1, \ldots, f_n)$). We can see that $W_1^r \cup W_1^a$ (resp. $W_2^r \cup W_2^a$) contains only leaking wires during the computation of $G_{\text{refresh}}(a_1, \ldots, a_n)$ (resp. $G_{\text{refresh}}(b_1, \ldots, b_n)$). We now demonstrate how we can simulate W when the output set J is of size less that t ((T)RPE1) and when it is of size strictly more than t ((T)RPE2).

- if $|J| \leq t$ ((T)RPE1): we prove both properties 1 and 2:

  1. we assume that $|W| < \min(t+1, n-t)$. We construct a new set of probes on $(e_1, \ldots, e_n)$ that we denote J_e such that $J_e = W_1^a \cup \{e_i \mid i \in J\}$. Similarly, we construct the set of probes on $(f_1, \ldots, f_n)$, $J_f = W_2^a \cup \{f_i \mid i \in J\}$. It is clear that if we can perfectly simulate W_1^r, W_2^r, J_e and J_f, then we can perfectly simulate W, and J (for each $i \in J$, we can perfectly simulate e_i in J_e and f_i in J_f so we can perfectly simulate $e_i + f_i$). We denote $|W_1^a| = m$ and $|W_2^a| = m'$. We have

  $$|W_1^r| \leq \min(t+1, n-t) - 1 - m \quad , \quad |J_e| \leq t + m$$

  and

  $$|W_2^r| \leq \min(t+1, n-t) - 1 - m', \quad , \quad |J_f| \leq t + m'$$

  From the (t, f)-TRPE property of G_{refresh} for any $t \leq n-1$ and specifically for $t' = t+m$ with amplification order at least $d' = \min(t+1+m, n-t-m)$, and since $|W_1^r| \leq \min(t+1, n-t) - 1 - m \leq d' - 1$, then there exists an input set of shares of a I_1 such that $|I_1| \leq \min(t+m, |W_1^r|) = |W_1^r| \leq |W|$ and I_1 perfectly simulates W_1^r and J_e.
  Similarly, there exists an input set of shares of b I_2 such that $|I_2| \leq \min(t + m', |W_2^r|) = |W_2^r| \leq |W|$ and I_2 perfectly simulates W_2^r and J_f.
  From these definitions, I_1 and I_2 together perfectly simulate W and J and are both of size less than $\min(t, |W|)$, which proves the first property in this scenario.

  2. we now assume that $\min(t+1, n-t) \leq |W| < 2\min(t+1, n-t)$. Without loss of generality, let us consider that $|W_1^r \cup W_1^a| < \min(t+1, n-t) \leq t$ (the proof is similar in the opposite scenario). As in the first property, we construct a new set of probes on $(e_1, \ldots, e_n)$ that we denote J_e such that $J_e = W_1^a \cup \{e_i \mid i \in J\}$. We fix the set of input shares I_2 on b as $I_2 = [n]$, so we can perfectly simulate all probes in W_2^r and W_2^a using the full input b. Next, we need to prove that we can perfectly simulate all probes in W_1^r and J_e similarly as before. We denote $|W_1^a| = m$. We have

  $$|W_1^r| \leq \min(t+1, n-t) - 1 - m \quad , \quad |J_e| \leq t + m$$

  From the (t, f)-TRPE property of G_{refresh} for any $t \leq n-1$ and specifically for $t' = t+m$ with amplification order at least $d' = \min(t+1+m, n-t-m)$, and since

$|W_1^r| \leq \min(t+1, n-t) - 1 - m \leq d' - 1$, then there exists an input set of shares of a I_1 such that $|I_1| \leq \min(t+m, |W_1^r|) = |W_1^r| \leq |W|$ and I_1 perfectly simulates W_1^r and J_e.

From these definitions, I_1 and I_2 together perfectly simulate W and J (J is simulated by perfectly simulating each $i \in J$ by using e_i in J_e and simulating f_i using the full input b), and we only have a failure on at most one of the inputs (b in this case). This concludes the proof for the second property.

At this point, we proved that G_{add} achieves an amplification order greater than or equal to $\min(t+1, n-t)$ for TRPE1. Since this amplification order is the maximum achievable by 2-input addition gadgets, then G_{add} achieves an amplification order exactly equal to $\min(t+1, n-t)$.

- if $|J| > t$ ((T)RPE2): we prove both properties 1 and 2:

  1. we assume that $|W| < \min(t+1, n-t)$. As before, we split W as $W = W_1^r \cup W_1^a \cup W_2^r \cup W_2^a$. We consider $J' = \{i \mid e_i \in W_1^a\} \cup \{i \mid f_i \in W_2^a\}$ so we have $|J'| \leq |W_1^a| + |W_2^a|$. We also construct the set W^r which contains the set of leaking wires on the first instance of G_{refresh} (on input a) in W_1^r, and all the wires that are leaking within the second instance of G_{refresh} in W_2^r. Hence, we have that $|W^r| \leq |W_1^r \cup W_2^r| < \min(t+1, n-t)$. Hence, we have $|W^r| + |J'| \leq \min(t+1, n-t) \leq n-1$, so by Definition 20 satisfied by G_{refresh}, there exists a set of output shares indices J such that $J' \subseteq J$ and $|J| = n-1$ such that W^r and J can be perfectly simulated from a set of input shares indices I such that $|I| \leq |W^r| + |J'|$. Thus, we can fix I_1 on a and I_2 on b such that $I_1 = I_2 = I$ and we fix the set of $n-1$ output shares indices on G_{add} as the same indices in J. Hence, we can perfectly simulate all wires in W^r and J, so we can perfectly simulate all wires in W_1^r and W_2^r and W_1^a and W_2^a as well as $n-1$ output shares of G_{add} using I_1 and I_2 such that $|I_1| = |I_2| \leq |W^r| + |J'| \leq |W| = \min(t, |W|)$. That concludes the proof for the first property.
  2. we now assume that $\min(t+1, n-t) \leq |W| < 2\min(t+1, n-t)$. Without loss of generality, let us consider that $|W_1^r \cup W_1^a| < \min(t+1, n-t)$ (the proof is similar in the opposite scenario).

     We fix $I_2 = [n]$ on input b, which allows us to perfectly simulate all wires and output shares on G_{refresh} instance with input sharing $(b_1, \ldots, b_n)$, including W_2^a and W_2^r. Next, we set $J' = \{i \mid e_i \in W_1^a\}$. Since $|W_1^r| + |J'| \leq n-1$, by Definition 20 satisfied by G_{refresh}, there exists a set of output shares indices J such that $J' \subseteq J$ and $|J| = n-1$ such that W_1^r and J can be perfectly simulated from a set of input shares indices I_1 on a such that $|I_1| \leq |W_1^r| + |J'| \leq |W_1^r| + |W_1^a| \leq |W|$. Thus, we can fix the set of $n-1$ output shares indices on G_{add} as the same indices in J. We can perfectly simulate all output shares indexed in J since for each $i \in J$, we can perfectly simulate e_i using I_1 and f_i using the full input b in I_2, so we can perfectly simulate $e_i + f_i$. Hence, we can perfectly simulate all wires in W as well as $n-1$ output shares of G_{add} using I_1 and I_2 such that $|I_1| \leq |W_1^r| + |W_1^a| \leq \min(t, |W|)$ and with a failure on input b with $I_2 = [n]$. That concludes the proof for the second property.

  We thus proved that G_{add} achieves an amplification order greater than or equal to $\min(t+1, n-t)$ for TRPE2. Since $\min(t+1, n-t)$ is the maximum order achievable for TRPE2 for a 2-input gadget, then G_{add} achieves exactly the order $\min(t+1, n-t)$.

Since G_{add} has an amplification order equal to $\min(t+1, n-t)$ for TRPE1 and TRPE2, then G_{add} is a (t, f')-TRPE addition gadget for some function f' of amplification order $\min(t+1, n-t)$, which concludes the proof. □

Hence, we provided generic constructions of addition and copy gadgets from a single building block, a refresh gadget. The amplification orders achieved can be maximal depending on the properties verified by the refresh gadget. These constructions allow us to look for efficient refresh gadgets with good security properties and plug them directly into our copy and addition gadgets. In the next section, we show that the well-known ISW refresh can be used to instantiate our constructions and both the ISW-based addition and copy gadgets achieve maximal amplification orders.

5.3 ISW-based Instantiation of the RPE Compiler

This section describes a complete instantiation of the expanding compiler solely based on the ISW scheme. Namely, we present instantiations of addition and copy gadgets from the ISW refresh gadget, and we directly use the ISW multiplication gadget for our compiler.

ISW addition and copy gadgets. The widely-used ISW refresh gadget can be seen as an ISW multiplication between the input sharing and the n-tuple $(1, 0, \ldots, 0)$. We formally depict it in the preliminaries in Algorithm 2. We demonstrate through Lemma 10 that the ISW refresh gadget satisfies TRPE with an optimal close to the optimal one.

Lemma 10. *Let $n \in \mathbb{N}$. For every $t \leq n-1$, the n-share ISW refresh gadget is (t, f_1)-TRPE1 for some function $f_1 : \mathbb{R} \to \mathbb{R}$ of amplification orders $d_1 = \min(t+1, n-t)$. The gadget is also $(n-1)$-STRPE2.*

Proof. We start by proving the first property of the lemma, i.e the amplification order d_1 for TRPE1. The n-share ISW refresh gadget was proven to be $(n-1)$-SNI [10], hence it follows from Lemma 6 that $d_1 \geq \min(t+1, n-t)$. In addition, we know from the proof of Lemma 1 that $d_1 \leq t+1$. It remains to show that $d_1 \leq n-t$. We thus have to exhibit a simulation failure by carefully choosing $n-t$ leaking variables (the leaking set W) together with t leaking output variables (indexed by the set J). Consider the set of output shares indexed by $J = \{1, \ldots, t\}$, which corresponds to the first t shares $c_1, \ldots, c_t$ of the output. Next, we construct the set of leaking wires W of size $n-t$. First, observe that the partial sums of the output shares are of the form

$$c_{i,j} = \begin{cases} a_i + r_{1,i} + \cdots + r_{j,i} & \text{if } j < i \\ a_i + r_{1,i} + \cdots + r_{i-1,i} + r_{i,i+1} + \cdots + r_{i,j} & \text{otherwise.} \end{cases}$$

Then, let $W = \{c_{t+1,t}, \ldots, c_{n,t}\}$. We can prove that the constructed set W along with the set of indexes of output shares $J = \{1, \ldots, t\}$ cannot be perfectly simulated with at most $\min(t, |W|)$ input shares. For this, we consider a variable $s = c_1 + \cdots + c_t + c_{t+1,t} + \ldots + c_{n,t}$, the sum of the t output shares indexed in J, and the leaking variables from W. Each of the output shares $\{c_i\}_{1 \leq i \leq t}$ is the sum of exactly one input share a_i and $n-1$ random values. Each of the leaking variables $\{c_{i,t}\}_{t+1 \leq i \leq n}$ is the sum of exactly one input share a_i and t random values. In addition, it can be observed that each random value appears exactly twice in the set of expressions of the variables $\{c_i\}_{1 \leq i \leq t} \cup \{c_{i,t}\}_{t+1 \leq i \leq n}$, so all of the random values are eliminated in the expression of the variable s, which is the sum of all of these variables. Since each of the variables has one input share a_i appearing in its expression, then we have $s = a_1 + \ldots + a_n = a$. Thus, simulating the variable s requires the knowledge of the full input, and hence the leaking variables indexed by W and J cannot be perfectly simulated without the knowledge of the full input. Hence, the set W of size $n-t$ represents a failure set with

respect to TRPE1, and so the function f_1 cannot be of amplification order higher than $n-t$, that is $d_1 \leq n-t$. From the three inequalities $d_1 \geq \min(t+1, n-t)$, $d_1 \leq t+1$ and $d_1 \leq n-t$, we obtain $d_1 = \min(t+1, n-t)$.

Next, we demonstrate that the gadget is $(n-1)$-STRPE2. Let W be a set of leaking wires and J' a set of output shares indices such that $|W| + |J'| \leq n-1$. We aim to prove that there exists a set J indexing $n-1$ output shares such that $J' \subseteq J$ and the leaking variables indexed by both W and J can be perfectly simulated with the input shares indexed by a set I such that $|I| \leq |W| + |J'|$. First, we observe that the leaking wires in W are of the following forms:

1. input share a_i
2. random variable r_{ij} $(i < j)$
3. partial sum $c_{ij} = \begin{cases} a_i + r_{1,i} + \cdots + r_{j,i} & \text{if } j < i \\ a_i + r_{1,i} + \cdots + r_{i-1,i} + r_{i,i+1} + \cdots + r_{i,j} & \text{otherwise.} \end{cases}$

We then build I from an empty set as follows. For every wire in W of the first or third form, we add index i to I. For every wire in W of the second form (r_{ij}), if $i \in I$, we add j to I, otherwise we add i to I. Finally, for each $i \in J'$, we add i to I. By construction we have $|I| \leq |W| + |J'| \leq n-1$. Moreover, the wires indexed in W and J' only depends of the input shares a_i with $i \in I$ which implies that we can perfectly simulate them from the input shares indexed by I (an output wire i in J' can be simulated with $a_i$and $n-1$ uniformly random variables).

We then build the set J as the union of two subsets J_1 and J_2 such that $J_1 = I$ and J_2 is any set satisfying $|J_2| = n - 1 - |I|$ and $J_1 \cap J_2 = \emptyset$. We also denote o as the single index in the set $[n] \setminus J$. Now, we aim to show that the output shares determined by the indexes in $J = J_1 \cup J_2$ can be further perfectly simulated from the input shares indexed by I (namely given the previous simulation of the variables from W). The simulation works as follows:

- each output share c_i such that $i \in J_1$ can be perfectly simulated with a_i (since $i \in I$) and $n-1$ uniformly random variables (the same generated r_{ij} can be reused for several c_i);
- for each output share c_i such that $i \in J_2$, we have $i \notin I$ and hence a_i is not available. Let us suppose without loss of generality that $i < o$ and consider the random variable $r_{i,o}$. Since $i \notin I$ and $o \notin I$, then $r_{i,o}$ does not appear in any probed expression by construction of the set I from W and J'. Hence, we can use $r_{i,o}$ to mask the expression of c_i, which can then be simulated as a fresh random value without the need for the input share a_i. This produces a perfect simulation of all output wires indexed in J_2.

We thus obtain a perfect simulation of the output shares indexed by $J = J_1 \cup J_2$, such that $|J| = n-1$, together with the variables indexed by W, from the input shares indexed by a set I of size $|I| \leq |W| + |J'| \leq n-1$. Hence the ISW refresh gadget is $(n-1)$-STRPE2, which concludes the proof. □

Corollary 5 then directly follows from Lemma 3 applied to TRPE, Lemma 10, and Corollary 4.

Corollary 5. *Let $n \in \mathbb{N}$. For every $t \leq n-1$, the n-share ISW refresh gadget is (t, f)-TRPE of amplification order*

$$d = \min(t+1, n-t).$$

The copy gadget G_{copy} that uses the n-share ISW refresh gadget as a building block in Algorithm 4 achieves the same amplification order as the ISW refresh for the TRPE setting, *i.e.* $d = \min(t+1, n-t)$. This is a direct implication from Lemma 7. Then, from Lemma 4, we have that ISW-based G_{copy} also achieves (t, f')-RPE with amplification order $d' \geq d$. We can actually prove that ISW-based G_{copy} achieves (t, f')-RPE with amplification order d' exactly equal to the amplification order in the TRPE setting, *i.e.* $d' = d = \min(t+1, n-t)$. This is stated in the following lemma.

Lemma 11 (ISW copy gadget)**.** *Let G_{copy} be the n-share copy gadget displayed in Algorithm 4 and instantiated with the ISW refresh gadget. Then for every $t \leq n-1$, G_{copy} achieves (t, f)-RPE with amplification order $d = \min(t+1, n-t)$.*

Proof. In order to prove that the amplification order d of G_{copy} instantiated with the ISW refresh gadget is equal to $\min(t+1, n-t)$, we first demonstrate that $d \geq \min(t+1, n-t)$ and then we show the existence of failure tuples to argue that $d \leq \min(t+1, n-t)$.

In fact, we already know that the ISW refresh gadget is (t, f_1)-TRPE of amplification order $d_1 = \min(t+1, n-t)$. Then from Lemma 7, we know that G_{copy} instantiated with ISW refresh is also (t, f_2)-TRPE of amplification order $d_2 = d_1 = \min(t+1, n-t)$. Then, from Lemma 4 we have that G_{copy} is (t, f)-RPE of amplification order $d \geq d_2 = \min(t+1, n-t)$. Next, we need to prove that $d \leq \min(t+1, n-t)$. In addition, we know from Lemma 1 that $d \leq t+1$. Hence, it remains to show that it is also upper bounded by $n-t$.

We know from the proof of Lemma 10 that, for the ISW refresh gadget, we can construct a set of leaking wires W of size $n-t$ along with a set of t indexes of output shares J such that a perfect simulation of both sets W and J requires the knowledge of the full input sharing *i.e.* $I = [n]$. Then, in the case of the copy gadget G_{copy}, let W be the set of leaking wires and $J_1, J_2 \subseteq [n]$ be the sets of output shares on the outputs e and f respectively. Then, we can split W into two distinct subsets W_1 and W_2 such that $W = W_1 \cup W_2$, where W_1 (resp. W_2) is the set of leaking wires of ISW G_{refresh} for the output e (resp. f). Then, in the case where $|J_1| \leq t$, we can construct the set $W = W_1$ of size $n-t$ ($W_2 = \emptyset$) in the exact same way as in the proof of Lemma 10, such that we have simulation failure of W_1 along with the output shares indexed in J_1 on the input of the gadget. Otherwise, in the case where $|J_2| \leq t$, we can construct the set $W = W_2$ of size $n-t$ ($W_1 = \emptyset$) in the exact same way, such that we have simulation failure of W_2 along with the output shares indexed in J_2 on the input of the gadget. Hence, the amplification order d of G_{copy} is upper bounded by $n-t$.

From the three inequalities $d \geq \min(t+1, n-t)$, $d \leq t+1$ and $d \leq n-t$, we conclude that the copy gadget instantiated with ISW refresh is (t, f)-RPE of amplification order $d = \min(t+1, n-t)$. □

The addition gadget G_{add} that uses the n-share ISW refresh gadget as a building block in Algorithm 5 achieves the same amplification order as the ISW refresh gadget, which is tighter than the bound from Lemma 8. This directly follows from Lemma 10 and Lemma 9.

Corollary 6 (ISW addition gadget)**.** *Let $n \in \mathbb{N}$. For every $t \leq n-1$, the n-share gadget G_{add} displayed in Algorithm 5 and instantiated with the ISW refresh gadget is (t, f)-RPE of amplification order $d = \min(t+1, n-t)$.*

ISW Multiplication Gadget In contrast to the copy and addition gadgets that are built from generic schemes with a refresh gadget as a building block, the multiplication gadget can be directly defined as the standard ISW multiplication, recalled in Algorithm 1. We have the following lemma.

Lemma 12. *Let $n \in \mathbb{N}$. For every $t \leq n-2$, the n-share ISW multiplication gadget displayed in Algorithm 1 is (t, f_1)-RPE1 and (t, f_2)-RPE2 for some functions $f_1, f_2 : \mathbb{R} \to \mathbb{R}$ of amplification orders d_1, d_2 which satisfy:*

- $d_1 = \dfrac{\min(t+1, n-t)}{2}$,
- $d_2 = \dfrac{t+1}{2}$.

Proof. We start by proving the first property of the lemma. Since the n-share ISW multiplication gadget is $(n-1)$-SNI [10], then we know from Lemma 6 that

$$d_1 \geq \frac{\min(t+1, n-t)}{2}.$$

In addition, we know from the proof of Lemma 2 that

$$d_1 \leq \frac{t+1}{2}.$$

It remains to show that $d_1 \leq (n-t)/2$. In this purpose, we exhibit a simulation failure on both inputs by carefully choosing $n-t$ leaking variables, with t output variables. Consider the set of indexes of output shares $J = \{1, \ldots, t\}$, which corresponds to the first t output shares $c_1, \ldots, c_t$. Next, we construct the set of leaking wires W of size $n-t$. First, observe that the partial sums of the output shares are of the form

$$c_{i,j} = \begin{cases} a_i \cdot b_i + r_{i,1} + \cdots + r_{i,j} & \text{if } j < i \\ a_i \cdot b_i + r_{i,1} + \cdots + r_{i,i-1} + r_{i,i+1} + \cdots + r_{i,j} & \text{otherwise.} \end{cases}$$

Then, let $W = \{c_{t+1,t}, \ldots, c_{n,t}\}$. We can prove that the constructed set W along with the set of output shares $J = \{1, \ldots, t\}$ cannot be perfectly simulated with at most t input shares. For this, we consider a variable $s = c_1 + \cdots + c_t + c_{t+1,t} + \ldots + c_{n,t}$, the sum of the t output shares indexed in J, and the leaking variables from W. Each of the output shares $\{c_i\}_{1 \leq i \leq t}$ is the sum of

- one product of input shares $a_i \cdot b_i$
- $n-1$ random values,
- at most $n-1$ pairs of input shares products: $(a_i \cdot b_j,\ a_j \cdot b_i)$ with $i \neq j$.

Each of the leaking variables $\{c_{i,t}\}_{t+1 \leq i \leq n}$ is the sum of

- one product of input shares $a_i \cdot b_i$,
- t random values,
- at most t pairs of input shares products: $(a_i \cdot b_j,\ a_j \cdot b_i)$ with $i \neq j$.

In addition, each random value appears exactly twice in the set of expressions of the variables $\{c_i\}_{1 \leq i \leq t} \cup \{c_{i,t}\}_{t+1 \leq i \leq n}$, so all the random values are eliminated from the expression of the variable s, which is the sum of all of these variables. Hence, $s = a_1 \cdot b_1 + \ldots + a_n \cdot b_n + C$ where C is a variable containing other products of input shares of the form $a_i \cdot b_j$ and $a_j \cdot b_i$ with $i \neq j$. Thus, simulating the variable s requires the knowledge of the full inputs a and b. Since s is constructed from the set of leaking wires W and the output shares indexed in J, then W and J cannot be perfectly simulated without the knowledge of the full inputs a and b. Hence, the set W of size $n-t$ represents a failure tuple on both inputs, and so the function

f_1 for RPE1 cannot be of amplification order higher than $(n-t)/2$. Thus, $d_1 \leq (n-t)/2$. From the three inequalities $d_1 \geq \dfrac{\min(t+1, n-t)}{2}$, $d_1 \leq (t+1)/2$ and $d_1 \leq (n-t)/2$, we conclude that

$$d_1 = \frac{\min(t+1, n-t)}{2}.$$

Next, we demonstrate the second part of the lemma. Let W be a set of leaking wires such that $|W| \leq t$. We aim to prove that there exists a set J of $n-1$ output wires such that W and J can be perfectly simulated with sets of input shares I_1 on a and I_2 on b such that $|I_1| \leq t$, $|I_2| \leq t$. First, observe that the leaking wires in W are of the following forms :

1. input shares a_i, b_i, product of shares $a_i \cdot b_i$.
2. partial sum $c_{i,j} = \begin{cases} a_i \cdot b_i + r_{i,1} + \cdots + r_{i,j} & \text{if } j < i \\ a_i \cdot b_i + r_{i,1} + \cdots + r_{i,i-1} + r_{i,i+1} + \cdots + r_{i,j} & \text{otherwise.} \end{cases}$
3. random variable r_{ij} for $i < j$, variable $r_{ji} = a_i \cdot b_j + r_{ij} + a_j \cdot b_i$ for $j > i$.
4. product of shares $a_i \cdot b_j$, or variable $a_i \cdot b_j + r_{ij}$ with $i \neq j$.

We build sets I_1 and I_2 from empty sets as follows. For every wire in W of the first or second form, we add index i to I_1 and I_2. For every wire in W of the third or fourth form, if $i \in I_1$, we add j to I_1, otherwise we add i to I_1, and if $i \in I_2$, we add j to I_2, otherwise we add i to I_2. Since W is of size at most t, then $|I_1| \leq t$ and $|I_2| \leq t$. Following the t-SNI property proof from [10], we can show that W is perfectly simulated using shares of indexes in I_1 and I_2. We now build the set J of $n-1$ indexes of output shares from two subsets J_1 and J_2. We define $J_1 = \{i \mid c_{i,j} \text{ is observed in } W\}$. Next, we define J_2 as any set such that $|J_2| = n - 1 - |J_1|$ and $J_1 \cap J_2 = \emptyset$. Now, we show that the output shares determined by the indexes in $J = J_1 \cup J_2$ can be perfectly simulated from I_1 and I_2:

- First consider the output wires indexed in J_1, which have a partial sum observed. For each such variable c_i, the biggest partial sum which is observed is already simulated. For the remaining r_{ij} in c_i, if $i < j$, then r_{ij} is assigned to a fresh random value. Otherwise, if r_{ji} enters in the computation of any other internal observation, then $i, j \in I_1$ and $i, j \in I_2$, and so r_{ji} can be perfectly simulated from the input shares. If not, then r_{ji} is replaced by the random value r_{ij}. So all output wires indexed in J_1 are perfectly simulated from I_1 and I_2.
- Now consider the output wires indexed in J_2. None of the c_i indexed in J_2 has a partial sum observed. Meanwhile, each c_i indexed in J_2 is composed of $n-1$ random values, and at most one of them can enter in the expression of each other output wire c_j. Since by construction of J_1, all the variables observed through the set W are included in the set of variables observed through J_1, and since $|J_1| \leq |W| \leq t \leq n-2$ and $|J_2| = n - 1 - |J_1|$, then each output wire c_i indexed in J_2 has at least one random value that does not appear in any other observation from W or J_1, so c_i can be assigned to a fresh random value. This produces a perfect simulation of all output wires indexed in J_2.

We conclude that the set J of $n-1$ wires is perfectly simulated along with W from the constructed sets I_1 and I_2 of sizes $|I_1| \leq |W| \leq t$ and $|I_2| \leq |W| \leq t$. So there is no failure set of observations of size at most t for RPE2 on any of the inputs. Hence $d_2 \geq (t+1)/2$. In addition, we know from the proof of Lemma 2 and as explained in Section 5.1 that $d_2 \leq (t+1)/2$. Hence, $d_2 = (t+1)/2$, which concludes the proof for RPE2. □

Corollary 7 then directly follows from Lemma 12 by applying Lemma 3 (RPE1 ∩ RPE2 ⇒ RPE).

Corollary 7. *Let $n \in \mathbb{N}$. For every $t \leq n-2$, the n-share ISW multiplication gadget displayed in Algorithm 1 is (t, f)-RPE of amplification order*

$$d = \frac{\min(t+1, n-t)}{2} .$$

According to Lemma 2, the upper bound on the amplification order of a standard multiplication gadget (*i.e.* which starts with the cross-products of the input shares) is $d \leq \min((t+1)/2, (n-t))$ which gives $d \leq (n+1)/3$ for $t = (2n-1)/3$. In contrast, the ISW multiplication gadget reaches $d = \lfloor \frac{n+1}{4} \rfloor$ by taking $t = \lceil \frac{n-1}{2} \rceil$.

Application to the RPE compiler As described in Section 4.4, instantiating the RPE compiler with three RPE base gadgets gives a $(p, 2^{-\kappa})$-random probing secure compiler (*i.e.* achieving κ bits of security against a leakage probability p) with a complexity blowup of $\mathcal{O}(\kappa^e)$ for an exponent e satisfying

$$e = \frac{\log N_{\max}}{\log d}$$

with $N_{\max}$ as defined in (4.17) and where d is the minimum amplification order of the three base gadgets as in Definition 14.

We can instantiate the RPE compiler using the ISW-based gadgets. Specifically, we use the ISW multiplication for the multiplication gadget G_{mult}, and the generic constructions of addition and copy gadgets based on the ISW refresh. The maximum amplification order achievable by the compiler is the minimum of the three gadgets, which is the order of the ISW multiplication gadget:

$$d = \frac{\min(t+1, n-t)}{2} .$$

Hence, for a given number of shares n, the maximum amplification order achievable is

$$d_{\max} = \left\lfloor \frac{n+1}{4} \right\rfloor$$

which is obtained for $t = \lceil \frac{n-1}{2} \rceil$. On the other hand, the value of $N_{\max}$ can be characterized in terms of the number of shares n. Recall from Section 4.4 that

$$N_{\max} = \max \left(N_{G_{\text{mult}},m} \, , \, \mathsf{eigenvalues} \left(\begin{pmatrix} N_{G_{\text{add}},a} & N_{G_{\text{copy}},a} \\ N_{G_{\text{add}},c} & N_{G_{\text{copy}},c} \end{pmatrix} \right) \right) .$$

For the ISW multiplication gadget, we have $N_{G_{\text{mult}},m} = n^2$. Then, for the ISW-based addition and copy gadgets, we have

$$\begin{pmatrix} N_{G_{\text{add}},a} & N_{G_{\text{copy}},a} \\ N_{G_{\text{add}},c} & N_{G_{\text{copy}},c} \end{pmatrix} = \begin{pmatrix} n(2n-1) & 2n(n-1) \\ n(n-1) & n^2 \end{pmatrix} .$$

The eigenvalues of the above matrix are $\lambda_1 = n$ and $\lambda_2 = 3n^2 - 2n$, implying

$$N_{\max} = 3n^2 - 2n \, . \tag{5.11}$$

Thus, the RPE compiler purely based on ISW constructions has a complexity blowup $\mathcal{O}(\kappa^e)$ with exponent

$$e = \frac{\log(3n^2 - 2n)}{\log(\lfloor (n+1)/4 \rfloor)} .$$

Figure 5.2 shows the evolution of the value of this exponent with respect to the number of shares n (where we assume an odd n). The blue curve corresponds to the construction based on ISW for multiplication, addition and copy. The value of e clearly decreases as the number of shares grows, and this decrease is faster for a small number of shares ($5 \leq n \leq 10$). The exponent value reaches $e \approx 4$ for a number of shares around 25 and then slowly converges towards $e = 2$ as n grows. This is to be compared with the $\mathcal{O}(\kappa^{7.5})$ complexity achieved in Section 4.5.

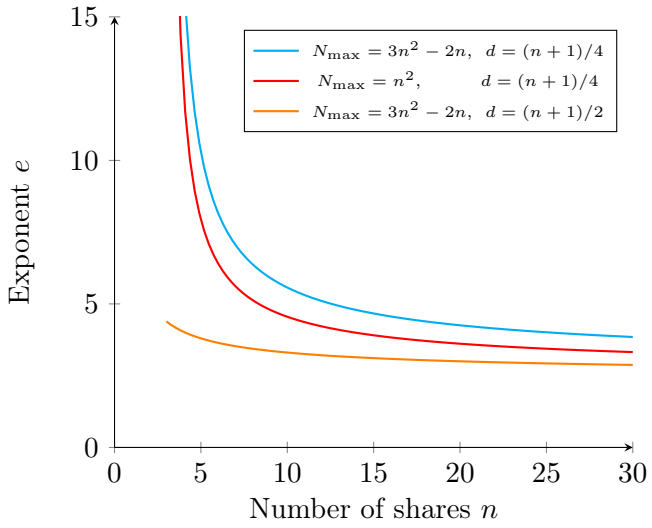

Figure 5.2: Evolution of the complexity exponent $e = \log(N_{\max})/\log(d)$ with respect to the number of shares n. The blue curve matches the instantiation with the ISW-based gadgets; the orange curve assumes the optimal amplification order (*i.e.* an improvement of the multiplication gadget); the pink curve assumes a better complexity for addition and copy gadgets (so that $N_{\max}$ matches $N_{\mathrm{m,m}} = n^2$).

The red and orange curves provide a comparison in the cases where we can achieve better values for $N_{\max}$ and d. The red curve assumes a lower value for $N_{\max}$. For instance, constructing addition and copy gadgets with better complexity so that $N_{\max}$ matches the value of $N_{\mathrm{m,m}} = n^2$. The orange curve, on the other hand, assumes that we achieve the maximal amplification. While the three curves converge towards $e = 2$, improving the values of $N_{\max}$ and d increases the convergence speed. Decreasing the value of $N_{\max}$ allows for a faster convergence (*e.g.* $e \approx 3.6$ for $n = 20$ shares compared to $e \approx 4.25$ on the blue curve) while achieving the maximal amplification order allows achieving much lower complexity blowups for constructions with small numbers of shares (*e.g.* $e < 5$ for $n = 3$ shares).

Towards a Better Complexity. As we saw in Figure 5.2, there are two directions for improvement. The first one is choosing gadgets which attain the upper bound $\min(t+1, n-t)$ on the amplification order from Lemma 1, allowing the compiler to have the maximum amplification order $d = (n+1)/2$ and thus have the lowest complexity blowup. Our ISW-based copy and addition gadgets achieve this bound while the ISW multiplication gadget is limited to $(n+1)/4$ (Lemma 12). To reach the optimal amplification order, one would need a different multiplication gadget and, in particular, a multiplication gadget which does not perform a direct product of shares (because of the bound from Lemma 2). We tackle this problem in Section 5.4, where we introduce a new multiplication gadget achieving the upper bound on the amplification order $\min(t+1, n-t)$ by avoiding a direct product of shares using a refresh on the input sharings. The orange curve in Figure 5.2 shows the evolution

of the value of the exponent when instantiating the expanding compiler with our previous addition and copy gadgets and this new multiplication gadget. For such an instantiation, the complexity exponent still slowly converges towards $e = 2$, but, as we can see from Figure 5.2, the exponent value is much better for small values of n.

Another possible direction for improvement would be to lower the complexity of the addition and copy gadgets, which is mainly dominated by the complexity of the refresh gadget. Assume that we can design a (T)RPE refresh gadget in sub-quadratic complexity, *e.g.* as the refresh gadgets proposed in [72, 14, 47], then the eigenvalues of the matrix $\boldsymbol{M}_{ac}$ as defined in Section 4.4 would also be sub-quadratic and the value of $N_{\max}$ from equation (4.17) would drop to $N_{\mathrm{m,m}} = n^2$ (if the multiplication gadget still requires n^2 multiplication gates). The pink curve in Figure 5.2 depicts the evolution of the exponent value under this assumption. We still have a slow convergence towards $e = 2$, but the exponent value is yet better for small values of n. Such asymptotic complexities are interesting for bigger numbers of shares. For this reason, we tackle this direction for improvement in the next chapter (Chapter 6), where we introduce the idea of dynamic random probing expansion. In a nutshell, the dynamic random probing expansion allows benefiting from the best tolerated leakage rates of constructions for a small number of shares and the optimal asymptotic complexity of constructions for a bigger number of shares. We show that we can use the $\mathcal{O}(n \log n)$ refresh gadget from [14] to instantiate the addition and copy gadgets and construct a new multiplication gadget, which performs a linear number of multiplications (*i.e.* $N_{\mathrm{m,m}} = \mathcal{O}(n)$) under some constraints on the base field. Indeed, this construction is asymptotic and works better for more shares. Eventually, this approach allows us to have $N_{\max} = \mathcal{O}(n \log n)$.

The above analysis shows that the RPE compiler can theoretically approach a quadratic complexity at the cost of increasing the number of shares in the base gadgets. The downside is that the tolerated leakage probability will likely decrease as the number of shares grows. For instance, the ISW construction is known only to tolerate a leakage probability $p = \mathcal{O}(1/n)$ [45]. The number of shares hence offers multiple trade-offs between the tolerated probability and the asymptotic complexity of the compiler. Starting from a target leakage probability p, one could determine the highest number of shares admissible from a generic construction and thus deduce the best complexity exponent achievable. In Section 5.5, we exhibit concrete trade-offs that can be reached for small values of n.

5.4 Multiplication Gadget with Maximal Amplification Order

Constructing a multiplication gadget which achieves the upper bound on the amplification order from Lemma 1 is tricky. First, as a standard multiplication gadget (*i.e.* which computes the cross products of the input shares), the ISW multiplication cannot achieve the maximal amplification order (see Lemma 2). In order to reach the upper bound for two-input gadgets (see Corollary 3), we need a non-standard multiplication gadget, *i.e.* which does not perform a direct product between the input shares. As an additional observation, the addition, copy, and random gates are *virtually free* in a multiplication gadget since they do not impact the final complexity of the RPE compiler (see Section 4.4). This suggests that we can be greedy in terms of randomness to reach the maximal amplification order.

In the following, we will describe the construction of a new multiplication gadget which achieves the maximum amplification order $\min(t+1, n-t)$. We first describe our standard n-share multiplication gadget and then explain how we avoid the initial cross products of shares.

First, the gadget constructs the matrix of the cross product of input shares:

$$M = \begin{pmatrix} a_1 \cdot b_1 & a_1 \cdot b_2 & \cdots & a_1 \cdot b_n \\ a_2 \cdot b_1 & a_2 \cdot b_2 & \cdots & a_2 \cdot b_n \\ \vdots & \vdots & \ddots & \vdots \\ a_n \cdot b_1 & a_n \cdot b_2 & \cdots & a_n \cdot b_n \end{pmatrix}$$

Then, it picks n^2 random values which define the following matrix:

$$R = \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,n} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{n,1} & r_{n,2} & \cdots & r_{n,n} \end{pmatrix}$$

It then performs an element-wise addition between the matrices M and R:

$$P = M + R = \begin{pmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,n} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n,1} & p_{n,2} & \cdots & p_{n,n} \end{pmatrix}$$

At this point, the gadget randomizes each product of input shares from the matrix M with a single random value from R. In order to generate the correct output, the gadget adds all the columns of P into a single column V of n elements, and adds all the columns of the transpose matrix R^{T} into a single column X of n elements:

$$V = \begin{pmatrix} p_{1,1} + \cdots + p_{1,n} \\ p_{2,1} + \cdots + p_{2,n} \\ \vdots \\ p_{n,1} + \cdots + p_{n,n} \end{pmatrix}, \qquad X = \begin{pmatrix} r_{1,1} + \cdots + r_{n,1} \\ r_{1,2} + \cdots + r_{n,2} \\ \vdots \\ r_{1,n} + \cdots + r_{n,n} \end{pmatrix}$$

The n-share output is finally defined as $(c_1, \ldots, c_n) = V + X$.

In order to further increase the maximum amplification order attainable by the gadget, we need to avoid performing a direct product of shares (because of the bound proved in Lemma 2). For this, we add a pre-processing phase to the gadget using a refresh gadget G_{refresh}. Specifically, we refresh the input $(b_1, \ldots, b_n)$ each time it is used. In other terms, each row of the matrix M uses a fresh copy of $(b_1, \ldots, b_n)$ produced using the considered refresh gadget. This amounts to performing n independent refreshes of the input $(b_1, \ldots, b_n)$. The matrix M is thus defined as

$$M = \begin{pmatrix} a_1 \cdot b_1^{(1)} & a_1 \cdot b_2^{(1)} & \cdots & a_1 \cdot b_n^{(1)} \\ a_2 \cdot b_1^{(2)} & a_2 \cdot b_2^{(2)} & \cdots & a_2 \cdot b_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ a_n \cdot b_1^{(n)} & a_n \cdot b_2^{(n)} & \cdots & a_n \cdot b_n^{(n)} \end{pmatrix}$$

where $(b_1^{(j)}, \ldots, b_n^{(j)})$, $j \in [n]$, are the n independent refreshings of the input $(b_1, \ldots, b_n)$.

With this refreshing scheme, we avoid using the same share more than once for one of the two input sharings. As a consequence, the double failure set of size $t + 1$ which is the reason behind the bound $(t+1)/2$ in Lemma 2, becomes a simple failure set (*i.e.* provoking a failure on a single input sharing). In addition, the computational overhead of these additional

n refreshes is negligible compared to the joint contribution of the copy and addition gadgets to the complexity of the RPE compiler.

For the sake of completeness, we present the full algorithm for this multiplication gadget in Algorithm 6.

Algorithm 6: Our multiplication gadget

Input : $(a_1, \ldots, a_n)$,$(b_1, \ldots, b_n)$ input sharings, $\{r_{ij}\}_{1 \leq i \leq n, 1 \leq j \leq n}$ random values, refresh gadget G_{refresh}

Output: $(c_1, \ldots, c_n)$ sharing of $a \cdot b$

for $i \leftarrow 1$ **to** n **do**
  $(b_1^{(i)}, \ldots, b_n^{(i)}) \leftarrow G_{\text{refresh}}(b_1, \ldots, b_n)$;
end
for $i \leftarrow 1$ **to** n **do**
  for $j \leftarrow 1$ **to** n **do**
    $p_{i,j} \leftarrow a_i \times b_j^{(i)} + r_{i,j}$;
  end
end
$(v_1, \ldots, v_n) \leftarrow (0, \ldots, 0)$;
$(x_1, \ldots, x_n) \leftarrow (0, \ldots, 0)$;
for $i \leftarrow 1$ **to** n **do**
  for $j \leftarrow 1$ **to** n **do**
    $v_i \leftarrow v_i + p_{i,j}$;
    $x_i \leftarrow x_i + r_{i,j}$;
  end
end
for $i \leftarrow 1$ **to** n **do**
  $c_i \leftarrow v_i + x_i$;
end
return $(c_1, \ldots, c_n)$;

In the following lemma, we show that if the refresh gadget G_{refresh} achieves the TRPE1 property with the amplification order at least $d = \min(t+1, n-t)$ for any t, then the multiplication gadget depicted in Algorithm 6 achieves TRPE with the maximum amplification orders. The proof of the Lemma is given in Section A.1.

Lemma 13. *Let $t \leq n-1$. Let $G_{refresh}$ be a (t, f')-TRPE1 refresh gadget for some function $f' : \mathbb{R} \to \mathbb{R}$, and G_{mult} the n-share multiplication gadget from Algorithm 6. If f' is of amplification order $d' \geq d = \min(t+1, n-t)$, then G_{mult} achieves (t, f)-TRPE for some function $f : \mathbb{R} \to \mathbb{R}$ of amplification order $d = \min(t+1, n-t)$.*

Corollary 8 then directly follows from Lemma 13 by applying Lemma 4 (TRPE $\Rightarrow$ RPE).

Corollary 8. *Let $t \leq n-1$. Let $G_{refresh}$ be a (t, f')-TRPE1 refresh gadget for some function $f' : \mathbb{R} \to \mathbb{R}$, and G_{mult} the n-share multiplication gadget from Algorithm 6. If f' is of amplification order $d' \geq d = \min(t+1, n-t)$, then G_{mult} achieves (t, f)-RPE for some function $f : \mathbb{R} \to \mathbb{R}$ of the same amplification order $d = \min(t+1, n-t)$.*

5.5 Efficient Small Gadgets

This section displays our new constructions of small gadgets for copy, addition, and multiplication operations with a low number of shares. As explained earlier, the maximal amplification

order achievable for 2-input n-share gadgets is $n+1/2$, hence it is equal to 1 for $n = 2$ gadets. We hence focus on $n \geq 3$. Then, as explained in Section 4.4.1, the highest amplification orders can only be achieved for gadgets with an odd number of shares. We therefore omit 4-share gadgets and display our best trade-offs in terms of RPE security and complexity for 3-share and 5-share gadgets. Each one of these gadgets is experimentally verified using the IronMask verification tool.

Addition and Copy Gadgets. For the construction of small 3-share and 5-share addition and copy gadgets, we use the generic constructions depicted in Algorithm 4 and Algorithm 5 (in Section 5.2) which naturally use a refresh gadget as a building block. We hence start by looking for refresh gadgets that have a good complexity in terms of gates count, and achieve the upper bound on the amplification order for the specific case of 3-share and 5-share constructions (but not necessarily for a higher number of shares).

Multiplication gadget. For the construction of small 3-share and 5-share multiplication gadgets, we use the generic construction depicted in Algorithm 6 from Section 5.4 which, to the best of our knowledge, is the only multiplication gadget which achieves the maximum amplification order for any number of shares, and specifically for 3-share and 5-share constructions. As for the refresh gadget G_{refresh} which is used to perform n refreshes on the second input, we use the same scheme as for the construction of small addition and copy gadgets (and which shall satisfy the necessary condition on G_{refresh} from Corollary 8).

While the multiplication gadget from Section 5.4 achieves the desired amplification order, we add another pre-processing phase to the gadget in order to further improve the tolerated leakage probability. In addition to the n refreshes performed on the second input b (see Algorithm 6), we add another single refresh of the input $(a_1, \ldots, a_n)$ before computing the cross-products, using the same refresh gadget G_{refresh}. Refreshing the input $(a_1, \ldots, a_n)$ before usage experimentally shows a further increase in the maximum tolerated leakage probability, by adding more randomness to the input shares before computing the cross-product matrix M in Algorithm 6. And since the refresh gadget G_{refresh} achieves the maximum amplification order, the amplification order achieved by G_{mult} is not affected by adding another refresh to the first input a.

The above construction achieves the maximum amplification order for 3-share ($d = 2$) and 5-share ($d = 3$) gadgets based on natural refresh gadgets detailed hereafter.

5.5.1 3-share Gadgets

We start with the construction of 3-share gadgets for our three base operations.

Copy and Addition Gadgets. We build our copy and addition gadgets from the instantiation of the generic constructions of Section 5.2 (Algorithm 4 and Algorithm 5) with 3 shares. However, instead of using the ISW refresh gadget, we use the following more efficient construction with only two random values:

$$\begin{aligned} G_{\text{refresh}} : c_1 &\leftarrow r_1 + a_1 \\ c_2 &\leftarrow r_2 + a_2 \\ c_3 &\leftarrow (r_1 + r_2) + a_3. \end{aligned}$$

This refresh is sufficient to reach the upper bounds on the amplification orders (from Lemma 1). From this basis, we obtain the following 3-share addition gadget with four random values:

$$\begin{aligned} G_{\text{add}} : c_1 &\leftarrow (r_1 + a_1) + (r_3 + b_1) \\ c_2 &\leftarrow (r_2 + a_2) + (r_4 + b_2) \\ c_3 &\leftarrow \big((r_1 + r_2) + a_3\big) + \big((r_3 + r_4) + b_3\big) \end{aligned}$$

and the following 3-share copy gadget with also four random values:

$$\begin{aligned} G_{\text{copy}} : c_1 &\leftarrow r_1 + a_1; & d_1 &\leftarrow r_3 + a_1 \\ c_2 &\leftarrow r_2 + a_2; & d_2 &\leftarrow r_4 + a_2 \\ c_3 &\leftarrow (r_1 + r_2) + a_3; & d_3 &\leftarrow (r_3 + r_4) + a_3. \end{aligned}$$

Multiplication Gadget. The following construction is a 3-share instantiation of the multiplication gadget described in Section 5.4. For the input refreshing, we use the 3-share refresh gadget described above with two uniformly random values. The construction achieves the bound on the amplification order from Lemma 1 with 17 random values:

$$\begin{aligned} G_{\text{mult}} : i_{1,1} &\leftarrow r_1 + b_1; & i_{1,2} &\leftarrow r_2 + b_2; & i_{1,3} &\leftarrow (r_1 + r_2) + b_3 \\ i_{2,1} &\leftarrow r_3 + b_1; & i_{2,2} &\leftarrow r_4 + b_2; & i_{2,3} &\leftarrow (r_3 + r_4) + b_3 \\ i_{3,1} &\leftarrow r_5 + b_1; & i_{3,2} &\leftarrow r_6 + b_2; & i_{3,3} &\leftarrow (r_5 + r_6) + b_3 \\ a'_1 &\leftarrow r_7 + a_1; & a'_2 &\leftarrow r_8 + a_2; & a'_3 &\leftarrow (r_7 + r_8) + a_3 \end{aligned}$$

$$\begin{aligned} c_1 &\leftarrow (a'_1 \cdot i_{1,1} + r_{1,1}) + (a'_1 \cdot i_{1,2} + r_{1,2}) + (a'_1 \cdot i_{1,3} + r_{1,3}) + (r_{1,1} + r_{2,1} + r_{3,1}) \\ c_2 &\leftarrow (a'_2 \cdot i_{2,1} + r_{2,1}) + (a'_2 \cdot i_{2,2} + r_{2,2}) + (a'_2 \cdot i_{2,3} + r_{2,3}) + (r_{1,2} + r_{2,2} + r_{3,2}) \\ c_3 &\leftarrow (a'_3 \cdot i_{3,1} + r_{3,1}) + (a'_3 \cdot i_{3,2} + r_{3,2}) + (a'_3 \cdot i_{3,3} + r_{3,3}) + (r_{1,3} + r_{2,3} + r_{3,3}). \end{aligned}$$

Results. Table 5.1 displays the results for the above gadgets obtained through the IronMask tool. The second column gives the complexity, where N_a, N_c, N_m, N_r stand for the number of addition gates, copy gates, multiplication gates and random gates respectively. The third column provides the amplification order of the gadget. And the last column gives the maximum tolerated leakage probability. The last row gives the global complexity, amplification order, and maximum tolerated leakage probability for the RPE compiler using these three gadgets.

Table 5.1: Results for the 3-share gadgets for $(t = 1, f)$-RPE, achieving the bound on the amplification order.

Gadget	**Complexity** (N_a, N_c, N_m, N_r)	**Amplification order**	$\log_2$ **of maximum tolerated proba**
G_{refresh}	$(4, 2, 0, 2)$	2	-5.14
G_{add}	$(11, 4, 0, 4)$	2	-4.75
G_{copy}	$(8, 7, 0, 4)$	2	-7.50
G_{mult}	$(40, 29, 9, 17)$	2	-7.41
Compiler	$\mathcal{O}(\lvert C\rvert \cdot \kappa^{\mathbf{3.9}})$	**2**	$\mathbf{-7.50}$

Remark 3. *The copy gadget G_{copy} instantiated in Section 4.5 which uses a refresh scheme with 3 randoms for each output, also reaches the amplification order 2. It tolerates a better leakage probability (i.e. $2^{-5.9}$) than the one provided here, but with a higher complexity of $(12, 9, 0, 6)$. If it is used to replace the 3-share copy gadget, the maximum tolerated leakage probability by the compiler from Table 5.1 would be of $2^{-7.4}$ slightly better than the current value of $2^{-7.5}$ but with a higher complexity of $\mathcal{O}(|C|\cdot\kappa^{4.08})$ instead of $\mathcal{O}(|C|\cdot\kappa^{3.9})$. Another copy gadget can be constructed by using the refresh scheme with 3 random values from Section 4.5 for one of the outputs, and the refresh scheme presented in this section with 2 random values for the second output. This gadget tolerates a maximum leakage probability of around $2^{-7.1}$ with a complexity of $(10, 8, 0, 5)$. Using it would bring the complexity of the compiler from Table 5.1 to $\mathcal{O}(|C| \cdot \kappa^4)$, while tolerating a leakage probability of $2^{-7.4}$, the same as that of the used multiplication gadget.*

5.5.2 5-share Gadgets

We now present our 5-share gadgets for our three base operations, which reach the optimal amplification order from Lemma 1.

Copy and Addition Gadgets. As for the 3-share case, we use the generic constructions from Section 5.2. Instead of using the ISW refresh gadget, we use the *circular refresh gadget* described in [7, 11] which requires less randomness (a.k.a. *block refresh gadget*):

$$\begin{aligned} G_{\text{refresh}} : c_1 &\leftarrow (r_1 + r_2) + a_1 \\ c_2 &\leftarrow (r_2 + r_3) + a_2 \\ c_3 &\leftarrow (r_3 + r_4) + a_3 \\ c_4 &\leftarrow (r_4 + r_5) + a_4 \\ c_5 &\leftarrow (r_5 + r_1) + a_5. \end{aligned}$$

This gadget only uses n randoms for an n-share construction, and while it does not achieve enough security in the generic case (unless the refresh block is iterated on the input a certain number of times [7, 11]), it proves to be more than enough to achieve the necessary amplification order for our 5-share constructions. We use a variant of the original version (also suggested in [7]): we choose to sum the random values first (thus obtaining a sharing of 0) before adding them to the input shares. The idea is to avoid using the input shares in any of the intermediate variables, so that input shares only appear in the input variables $\{a_i\}_{1\leq i\leq n}$ and the final output variables $\{c_i\}_{1\leq i\leq n}$. Intuitively, this trick allows to have less failure tuples in the gadget because there are less variables that could leak information about the input. This is validated experimentally where we obtain better results in terms of amplification order and tolerated leakage probability for small gadgets.

From this circular refresh, we obtain an addition gadget with ten random values which reaches the upper bound on the amplification order:

$$\begin{aligned} G_{\text{add}} : c_1 &\leftarrow \big((r_1 + r_2) + a_1\big) + \big((r_6 + r_7) + b_1\big) \\ c_2 &\leftarrow \big((r_2 + r_3) + a_2\big) + \big((r_7 + r_8) + b_2\big) \\ c_3 &\leftarrow \big((r_3 + r_4) + a_3\big) + \big((r_8 + r_9) + b_3\big) \\ c_4 &\leftarrow \big((r_4 + r_5) + a_4\big) + \big((r_9 + r_{10}) + b_4\big) \\ c_5 &\leftarrow \big((r_5 + r_1) + a_5\big) + \big((r_{10} + r_6) + b_5\big) \end{aligned}$$

and a copy gadget with also ten random values and which also reaches the upper bound on

the amplification order:

$$
\begin{aligned}
G_{\text{copy}} : c_1 &\leftarrow (r_1 + r_2) + a_1; & d_1 &\leftarrow (r_6 + r_7) + a_1 \\
c_2 &\leftarrow (r_2 + r_3) + a_2; & d_2 &\leftarrow (r_7 + r_8) + a_2 \\
c_3 &\leftarrow (r_3 + r_4) + a_3; & d_3 &\leftarrow (r_8 + r_9) + a_3 \\
c_4 &\leftarrow (r_4 + r_5) + a_4; & d_4 &\leftarrow (r_9 + r_{10}) + a_4 \\
c_5 &\leftarrow (r_5 + r_1) + a_5; & d_5 &\leftarrow (r_{10} + r_6) + a_5.
\end{aligned}
$$

Multiplication Gadget. The following construction is a 5-share instantiation of the multiplication gadget described in Section 5.4. For the input refreshing, we use the 5-share circular refresh gadget described above. The gadget advantageously achieves the optimal amplification order (given by Lemma 1) with 55 random values:

$$
\begin{aligned}
G_{\text{mult}} : i_{1,1} &\leftarrow (r_1 + r_2) + b_1; \quad i_{1,2} \leftarrow (r_2 + r_3) + b_2; \quad i_{1,3} \leftarrow (r_3 + r_4) + b_3; \\
i_{1,4} &\leftarrow (r_4 + r_5) + b_4; \quad i_{1,5} \leftarrow (r_5 + r_1) + b_5 \\
\\
i_{2,1} &\leftarrow (r_6 + r_7) + b_1; \quad i_{2,2} \leftarrow (r_7 + r_8) + b_2; \quad i_{2,3} \leftarrow (r_8 + r_9) + b_3; \\
i_{2,4} &\leftarrow (r_9 + r_{10}) + b_4; \quad i_{2,5} \leftarrow (r_{10} + r_6) + b_5 \\
\\
i_{3,1} &\leftarrow (r_{11} + r_{12}) + b_1; \quad i_{3,2} \leftarrow (r_{12} + r_{13}) + b_2; \quad i_{3,3} \leftarrow (r_{13} + r_{14}) + b_3; \\
i_{3,4} &\leftarrow (r_{14} + r_{15}) + b_4; \quad i_{3,5} \leftarrow (r_{15} + r_{11}) + b_5 \\
\\
i_{4,1} &\leftarrow (r_{16} + r_{17}) + b_1; \quad i_{4,2} \leftarrow (r_{17} + r_{18}) + b_2; \quad i_{4,3} \leftarrow (r_{18} + r_{19}) + b_3; \\
i_{4,4} &\leftarrow (r_{19} + r_{20}) + b_4; \quad i_{4,5} \leftarrow (r_{20} + r_{16}) + b_5 \\
\\
i_{5,1} &\leftarrow (r_{21} + r_{22}) + b_1; \quad i_{5,2} \leftarrow (r_{22} + r_{23}) + b_2; \quad i_{5,3} \leftarrow (r_{23} + r_{24}) + b_3; \\
i_{5,4} &\leftarrow (r_{24} + r_{25}) + b_4; \quad i_{5,5} \leftarrow (r_{25} + r_{21}) + b_5 \\
\\
a'_1 &\leftarrow (r_{26} + r_{27}) + a_1; \quad a'_2 \leftarrow (r_{27} + r_{28}) + a_2; \quad a'_3 \leftarrow (r_{28} + r_{29}) + a_3; \\
a'_4 &\leftarrow (r_{29} + r_{30}) + a_4; \quad a'_5 \leftarrow (r_{30} + r_{26}) + a_5
\end{aligned}
$$

$$
\begin{aligned}
c_1 \leftarrow\ & (a'_1 \cdot i_{1,1} + r_{1,1}) + (a'_1 \cdot i_{1,2} + r_{1,2}) + (a'_1 \cdot i_{1,3} + r_{1,3}) + (a'_1 \cdot i_{1,4} + r_{1,4}) \\
& + (a'_1 \cdot i_{1,5} + r_{1,5}) + (r_{1,1} + r_{2,1} + r_{3,1} + r_{4,1} + r_{5,1}) \\
c_2 \leftarrow\ & (a'_2 \cdot i_{2,1} + r_{2,1}) + (a'_2 \cdot i_{2,2} + r_{2,2}) + (a'_2 \cdot i_{2,3} + r_{2,3}) + (a'_2 \cdot i_{2,4} + r_{2,4}) \\
& + (a'_2 \cdot i_{2,5} + r_{2,5}) + (r_{1,2} + r_{2,2} + r_{3,2} + r_{4,2} + r_{5,2}) \\
c_3 \leftarrow\ & (a'_3 \cdot i_{3,1} + r_{3,1}) + (a'_3 \cdot i_{3,2} + r_{3,2}) + (a'_3 \cdot i_{3,3} + r_{3,3}) + (a'_3 \cdot i_{3,4} + r_{3,4}) \\
& + (a'_3 \cdot i_{3,5} + r_{3,5}) + (r_{1,3} + r_{2,3} + r_{3,3} + r_{4,3} + r_{5,3}) \\
c_4 \leftarrow\ & (a'_4 \cdot i_{4,1} + r_{4,1}) + (a'_4 \cdot i_{4,2} + r_{4,2}) + (a'_4 \cdot i_{4,3} + r_{4,3}) + (a'_4 \cdot i_{4,4} + r_{4,4}) \\
& + (a'_4 \cdot i_{4,5} + r_{4,5}) + (r_{1,4} + r_{2,4} + r_{3,4} + r_{4,4} + r_{5,4}) \\
c_5 \leftarrow\ & (a'_5 \cdot i_{5,1} + r_{5,1}) + (a'_5 \cdot i_{5,2} + r_{5,2}) + (a'_5 \cdot i_{5,3} + r_{5,3}) + (a'_5 \cdot i_{5,4} + r_{5,4}) \\
& + (a'_5 \cdot i_{5,5} + r_{5,5}) + (r_{1,5} + r_{2,5} + r_{3,5} + r_{4,5} + r_{5,5}).
\end{aligned}
$$

Results. Table 5.2 gives the results for the above gadgets obtained through the IronMask tool.

Table 5.2: Results for the 5-share gadgets for $(t = 2, f)$-RPE, achieving the bound on the amplification order.

Gadget	**Complexity**	**Amplification order**	$\log_2$ **of maximum tolerated proba**
G_{refresh}	$(10, 5, 0, 5)$	3	-4.83
G_{add}	$(25, 10, 0, 10)$	3	$[-4.67, -4.42]$
G_{copy}	$(20, 15, 0, 10)$	3	-6.17
G_{mult}	$(130, 95, 25, 55)$	3	$[-9.67, -7.66]$
Compiler	$\mathcal{O}(\lvert C\rvert \cdot \kappa^{\mathbf{3.23}})$	**3**	$[\mathbf{-9.67}, \mathbf{-7.66}]$

From Table 5.1 and Table 5.2, we observe that the asymptotic complexity is better for the instantiation based on 5-share gadgets as they provide a better amplification order with limited overhead. While this result can seem to be counterintuitive, it actually comes from the fact that each gadget will be expended less in the second scenario. We stress that we could only obtain an interval $[2^{-9.67}, 2^{-7.66}]$ for the tolerated leakage probability because it was computationally too expensive to obtain a tighter interval. Meanwhile, we can consider that our best complexity $\mathcal{O}(|C| \cdot \kappa^{3.2})$ comes at the price of a lower tolerated leakage probability of $2^{-9.67}$ (5-share gadget) compared to the $\mathcal{O}(|C| \cdot \kappa^{3.9})$ complexity and $2^{-7.5}$ tolerated leakage probability obtained for our 3-share instantiation.

In comparison, the previous instantiation of the RPE compiler in Section 4.5 could only achieve a complexity of $\mathcal{O}(|C| \cdot \kappa^{7.5})$ for maximum tolerated probabilities of $2^{-7.09}$, and the instantiation of the expanding approach with a multi-party computation protocol [3], could only achieve a complexity of $\mathcal{O}(|C| \cdot \kappa^{8.2})$ for maximum tolerated probabilities of 2^{-26}.

5.6 Conclusion

This chapter provided an in-depth analysis of the RPE security notion and showed that it could be made tighter, with a connection to the strong non-interference (SNI) composition notion. We also introduced generic RPE constructions achieving the maximal amplification order for any number of shares and instantiated small constructions with better asymptotic complexities.

As previously discussed, we want to target high tolerated leakage rates and good asymptotic complexities. While constructions with small numbers of shares tend to tolerate better leakage rates, some constructions have better asymptotic complexities for higher numbers of shares. In the next chapter, we try to get the best of both worlds with dynamic random probing expansion. The goal is to start the expansion with a few iterations of a compiler with a small number of shares to benefit from its tolerated leakage probability. Then, we can continue the expansion with other compilers with optimal asymptotic complexities, bringing us a step closer to optimal expansion strategies.

Chapter 6

Dynamic RPE and Optimal Asymptotic Complexities

In this chapter, we further push the random probing expansion strategy by analyzing the base gadgets' dynamic choice. While the expanding compiler considered in previous chapters consists in applying a compiler CC composed of base RPE gadgets a given number of times, say k, to the input circuit: $\widehat{C} = \mathsf{CC}^{(k)}(C)$, we consider a dynamic approach in which a new compiler is selected at each step of the expansion from a family of base compilers $\{\mathsf{CC}_i\}_i$. This approach is motivated by the generic gadget constructions introduced in Chapter 5, which achieve the RPE property for any number of shares n. While the asymptotic complexity of the expanding compiler decreases with n, the tolerated leakage probability p also gets smaller with n, which makes those constructions only practical for small values of n. We show that using our dynamic approach, we can get the best of both worlds: our dynamic expanding compiler enjoys the best tolerated probability and the best asymptotic complexity from the underlying family of RPE compilers $\{\mathsf{CC}_i\}_i$. We further illustrate how this approach can reduce the complexity of a random probing secure AES implementation by a factor of 10 using a dynamic choice of the gadgets from Section 5.5 in Chapter 5.

The dynamic random probing expansion motivates us to design asymptotic RPE gadgets achieving better complexity. While the earlier asymptotic constructions achieve a quadratic complexity, we introduce new constructions achieving quasi-linear complexity, using the quasi-linear refresh gadget from Battistello, Coron, Prouff, and Zeitoun [14]. With such linear gadgets, the complexity bottleneck of the expanding compiler becomes the number of multiplications in the multiplication gadget, which is quadratic in known RPE constructions. We then provide a new generic construction of an RPE multiplication gadget featuring a linear number of multiplications. We obtain this construction by tweaking the probing-secure multiplication gadget from Belaïd, Benhamouda, Passelègue, Prouff, Thillard, and Vergnaud [17]. As in the original construction, our RPE gadget imposes some constraints on the underlying finite field. We demonstrate that for any number of shares, there exists a (possibly large) finite field on which our construction can be instantiated, and we provide some concrete instantiations for small numbers of shares.

Using our new asymptotic gadget constructions with the dynamic expansion approach, we obtain random probing security for a leakage probability of $2^{-7.5}$ with asymptotic complexity of $\mathcal{O}(\kappa^2)$. Moreover, assuming that the constraint on the finite field from our multiplication gadget is satisfied, we can make this asymptotic complexity arbitrarily close to $\mathcal{O}(\kappa)$, which is optimal. Hence, securing circuits defined on a large field against random probing leakage can be achieved at a sub-quadratic nearly-linear complexity. The contributions in this chapter are published in [23].

6.1 Dynamic Random Probing Expansion

Recall that the principle of the expanding compiler is to apply a base circuit compiler CC which is composed of base gadgets –one per gate type in the circuit– several times, say k, to the input circuit: $\widehat{C} = \mathsf{CC}^{(k)}(C)$. The level of expansion k is chosen in order to achieve a certain desired security level κ such that $f^{(k)}(p) \leq 2^{-\kappa}$.

In this section, we generalize this approach to choose the circuit compiler dynamically at the different steps of the expansion. Let $\{\mathsf{CC}_i\}_i$ be a family of circuit compilers, the *dynamic expanding compiler* for this family with respect to the expansion sequence $k_1, \ldots k_\mu$, is defined as

$$\widehat{C} = \mathsf{CC}_\mu^{k_\mu} \circ \mathsf{CC}_{\mu-1}^{k_{\mu-1}} \circ \ldots\ldots \circ \mathsf{CC}_1^{k_1}(C) \ . \tag{6.1}$$

The idea behind this generalization is to make the most from a family of RPE compilers $\{\mathsf{CC}_i\}_i$ which is defined with respect to the number of shares n_i in the base gadgets. If we assume that each compiler CC_i with n_i shares achieves the maximum amplification order $d_i = \frac{n_i+1}{2}$, then the benefit of using a compiler with higher number of shares is to increase the amplification order and thus reduce the number of steps necessary to achieve the desired security level κ. On the other hand, the tolerated leakage rate of existing constructions decreases with n_i. As we show hereafter, a dynamic increase of n_i can ensure both, the tolerated leakage rate of a small n_i and the better complexity of a high n_i.

6.1.1 Dynamic Expanding Compiler

In the following, we state the security and asymptotic complexity of the dynamic expanding compiler. We will consider a family of different RPE compilers where each compiler is indexed by an index i, *i.e.* a family of different RPE compilers is denoted as $\{\mathsf{CC}_i\}_i$ for different number of shares $\{n_i\}_i$. We start with a formal definition of the dynamic compiler:

Definition 21 (Dynamic Expanding Compiler)**.** *Let $\{\mathsf{CC}_i\}_i$ be a family of RPE compilers with numbers of shares $\{n_i\}_i$. The* dynamic expanding compiler *for $\{\mathsf{CC}_i\}_i$ with* expansion levels *$k_1, \ldots, k_\mu$, is the circuit compiler $(\mathsf{CC}, \mathsf{Enc}, \mathsf{Dec})$ where*

1. *The input encoding Enc is a $\left(\prod_{i=1}^{\mu} n_i^{k_i}\right)$-linear encoding.*
2. *The output decoding Dec is the $\left(\prod_{i=1}^{\mu} n_i^{k_i}\right)$-linear decoding mapping.*
3. *The circuit compilation is defined as*

$$\mathsf{CC}(\cdot) = \mathsf{CC}_\mu^{k_\mu} \circ \mathsf{CC}_{\mu-1}^{k_{\mu-1}} \circ \ldots\ldots \circ \mathsf{CC}_1^{k_1}(\cdot) \ .$$

The following theorem states the random probing security of the dynamic expanding compiler. The proof of the theorem is very similar to the proof of RPE security (Theorem 2) from Section 4.2 The main difference is that at each level of the expansion, we can use a different expanding compiler with different sharing orders. Besides that, the proof follows the same baselines as in the original proof of Theorem 2. For the sake of completeness, we give the full proof in Section A.2.

Theorem 3 (Security)**.** *Let $\{\mathsf{CC}_i\}_i$ be a family of RPE compilers with expanding functions $\{f_i\}_i$. The dynamic expanding compiler for $\{\mathsf{CC}_i\}_i$ with expansion levels $k_1, \ldots, k_\mu$ is (p, ε)-random probing secure with*

$$\varepsilon = f_\mu^{k_\mu} \circ \cdots \circ f_1^{k_1}(p) \ .$$

We now state the asymptotic complexity of the dynamic expanding compiler in the next theorem.

Theorem 4 (Asymptotic Complexity)**.** *Let* $\{\mathsf{CC}_i\}_i$ *be a family of circuit compilers with complexity matrices* $\{M_{\mathsf{CC}_i}\}_i$*. For any input circuit* C*, the output circuit* $\widehat{C} = \mathsf{CC}_\mu^{k_\mu} \circ \cdots\cdots \circ \mathsf{CC}_1^{k_1}(C)$ *is of size*

$$|\widehat{C}| = |C| \cdot \mathcal{O}\Big(\prod_{i=1}^{\mu} |\lambda_i|^{k_i}\Big) \quad \textit{with} \quad \lambda_i \quad \textit{such that} \quad |\lambda_i| := \max \ |\mathsf{eigenvalues}(M_{\mathsf{CC}_i})| \ . \tag{6.2}$$

Proof. Let $\{\mathsf{CC}_i\}_i$ be a family of circuit compilers with complexity matrices $\{M_{\mathsf{CC}_i}\}_i$. Given a circuit C with its complexity vector N_C as described in Section 4.4, it can be verified that the complexity of the compiled circuit $\widehat{C} = \mathsf{CC}_\mu^{k_\mu} \circ \cdots\cdots \circ \mathsf{CC}_1^{k_1}(C)$ satisfies

$$N_{\widehat{C}} = M_{\mathsf{CC}_\mu}^{k_\mu} \cdot \ldots \cdot M_{\mathsf{CC}_1}^{k_1} \cdot N_C$$

If we denote $M_{\mathsf{CC}_i} = Q_i \cdot \Lambda_i \cdot Q_i^{-1}$ to be the eigen decomposition of the matrix M_{CC_i}, then we get

$$N_{\widehat{C}} = Q_\mu \cdot \Lambda_\mu^{k_\mu} \cdot Q_\mu^{-1} \cdot \ldots \cdot Q_1 \cdot \Lambda_1^{k_1} \cdot Q_1^{-1} \cdot N_C \tag{6.3}$$

We consider in the theorem that the expansion levels $\{k_i\}_i$ are the main parameters. We can also see from (6.3) that the complexity of the compiled circuit is expressed in terms of the eigen matrices to the powers k_i as $\Lambda_i^{k_i}$. The parameters $\{k_i\}_i$ do not affect the matrices $\{Q_i, Q_i^{-1}\}_i$. Then, if we denote $\lambda_i := \max \ \mathsf{eigenvalues}(M_{\mathsf{CC}_i})$ *i.e.* the maximum of the eigenvalues in Λ_i, then we get that in terms of the parameters $\{k_i\}_i$, the complexity of the compiled circuit $\widehat{C}$ can be expressed as

$$N_{\widehat{C}} = \mathcal{O}\Big(|\lambda_\mu|^{k_\mu} \cdot \ldots \cdot |\lambda_1|^{k_1}\Big) \cdot N_C$$

which gives

$$|\widehat{C}| = |C| \cdot \mathcal{O}\Big(\prod_{i=1}^{\mu} |\lambda_i|^{k_i}\Big)$$

which concludes the proof of Theorem 4. □

In the following, we shall call λ_i as defined above, the *eigen-complexity* of the compiler CC_i. We shall further call the product $\prod_{i=1}^{\mu} |\lambda_i|^{k_i}$ the *complexity blowup* of the dynamic expanding compiler. We note that minimizing the complexity blowup is equivalent to minimizing the log complexity blowup, which is

$$\sum_{i=1}^{\mu} k_i \cdot \log_2(|\lambda_i|) \ . \tag{6.4}$$

6.1.2 General Bounds for Asymptotic Constructions

The following theorem introduces general bounds on the tolerated leakage rate and the expanding function of an RPE compiler with respect to its amplification order and gadget complexity.

Theorem 5. *Let* CC_i *be an RPE circuit compiler of amplification order* d_i *and gadget complexity* s_i*. The tolerated leakage rate* q_i *of* CC_i *is lower bounded by*

$$q_i \geq \bar{q}_i := \frac{1}{\mathrm{e}} \left(\frac{1}{2\,\mathrm{e}}\right)^{\frac{1}{d_i - 1}} \left(\frac{d_i}{s_i}\right)^{1 + \frac{1}{d_i - 1}} \tag{6.5}$$

For any $p < \bar{q}_i$*, the expanding function* f_i *of* CC_i *is upper bounded by*

$$f_i(p) \leq 2\binom{s_i}{d_i} p^{d_i} \leq 2\left(\frac{\mathrm{e} \cdot s_i}{d_i}\right)^{d_i} p^{d_i} \ . \tag{6.6}$$

Proof. To prove Theorem 5, we introduce the following lemma.

Lemma 14. *Let CC_i be an RPE circuit compiler of amplification order d_i and complexity s_i. For any probability*

$$p \leq \frac{1}{2} \cdot \frac{d_i + 1}{s_i - d_i} \tag{6.7}$$

the expanding function f_i of CC_i is upper bounded by

$$f_i(p) \leq 2\binom{s_i}{d_i} p^{d_i} \ . \tag{6.8}$$

Proof. (Lemma 14) Let us first recall the following general bound on f_i:

$$f_i(p) \leq \sum_{j=d_i}^{s_i} \binom{s_i}{j} p^j \ , \tag{6.9}$$

for any $p \in [0, 1)$. From (6.7), for any $j \in [s_i]$, we get:

$$\binom{s_i}{j+1} p^{j+1} \leq \frac{1}{2}\binom{s_i}{j} p^j$$

which gives

$$f_i(p) \leq \sum_{j=d_i}^{s_i} \binom{s_i}{d_i} \left(\frac{1}{2}\right)^{j-d_i} p^{d_i} = \binom{s_i}{d_i} p^{d_i} \sum_{j=0}^{s_i-d_i} \left(\frac{1}{2}\right)^j \leq 2\binom{s_i}{d_i} p^{d_i} \ .$$

□

We are now ready to prove Theorem 5. We show that for every p satisfying

$$p < \frac{1}{\mathrm{e}} \left(\frac{1}{2\,\mathrm{e}}\right)^{\frac{1}{d_i - 1}} \left(\frac{d_i}{s_i}\right)^{1 + \frac{1}{d_i - 1}} \tag{6.10}$$

we have $f_i(p) < p$. Let us define

$$\bar{f}_i : p \mapsto 2\binom{s_i}{d_i} p^{d_i} \ .$$

(the upper bound on f_i from Lemma 14). The equation $\bar{f}_i(\gamma) = \gamma$ has the following solution

$$\gamma = \left(\frac{1}{2\binom{s_i}{d_i}}\right)^{\frac{1}{d_i - 1}}$$

which, from

$$\binom{s_i}{d_i} \leq \left(\frac{s_i \cdot \exp(1)}{d_i}\right)^{d_i} \ ,$$

further satisfies

$$\gamma \geq \frac{1}{\mathrm{e}} \left(\frac{1}{2\,\mathrm{e}}\right)^{\frac{1}{d_i - 1}} \left(\frac{d_i}{s_i}\right)^{1 + \frac{1}{d_i - 1}}$$

We deduce that (6.10) implies $p < \gamma$ which further implies $\bar{f}_i(p) < p$. Moreover (6.10) implies

$$p < \frac{1}{2} \left(\frac{d_i}{s_i}\right)^{1 + \frac{1}{d_i - 1}} < \frac{1}{2} \cdot \frac{d_i}{s_i} < \frac{1}{2} \cdot \frac{d_i + 1}{s_i - d_i} \ ,$$

which, by Lemma 14, further implies $f_i(p) \leq \bar{f}_i(p)$. We hence deduce that (6.10) implies $f_i(p) < p$ which concludes the proof. □

The lower bound $\bar{q}_i$ on the tolerated leakage rate quickly converges to the ratio $\mathrm{e}^{-1} \cdot d_i/s_i$ as d_i grows. In other words, an RPE compiler family $\{\mathsf{CC}_i\}_i$ indexed by the number of shares n_i of its base gadgets tolerates a leakage probability which is linear in the ratio between its amplification order d_i and its complexity s_i. For known families of RPE compilers from Section 5.5, this ratio is in $\mathcal{O}(1/n_i)$.

From Theorem 5, we obtain the following bound for the composition $f_i^{(k)}$.

Corollary 9. *Let CC_i be an RPE compiler of expanding function f_i, amplification order d_i and gadget complexity s_i. For any $p < \bar{q}_i$ as defined in (6.5), we have*

$$f_i^{(k)}(p) \;\leq\; \left[2\binom{s_i}{d_i}\right]^{\left(1+\frac{1}{d_i-1}\right)d_i^{k-1}} p^{d_i^k} \;\leq\; \left[\left(\frac{2^{\frac{1}{d_i}}\mathrm{e}s_i}{d_i}\right)^{\left(1+\frac{1}{d_i-1}\right)} p\right]^{d_i^k} .$$

Proof. For any function $f(p) = c \cdot p^d$, we have

$$f^{(k)}(p) = c^{(d^{k-1}+d^{k-2}+\cdots+1)} \cdot p^{d^k} \leq c^{\left(1+\frac{1}{d-1}\right)d^{k-1}} p^{d^k}.$$

When $c_i = 2\binom{s_i}{d_i}$, Equation (6.6) from Theorem 5 gives the first and the second inequalities. □

The following lemma gives an explicit lower bound on the expansion level $\{k_i\}_i$ to reach some arbitrary target probability $p_{out} = 2^{-\kappa_{out}}$ from a given input probability $p_{in} = 2^{-\kappa_{in}}$ by applying $\mathsf{CC}_i^{(k_i)}$.

Lemma 15. *Let $p_{in} = 2^{-\kappa_{in}} < q_i$ and $p_{out} = 2^{-\kappa_{out}} \in (0, 1]$. For any integer k_i satisfying*

$$k_i \geq \log_{d_i}(\kappa_{out}) - \log_{d_i}(\kappa_{in} - \Delta_i)$$

with

$$\Delta_i := \left(1 + \frac{1}{d_i - 1}\right)\left(\frac{1}{d_i} + \log_2\left(\frac{\mathrm{e}s_i}{d_i}\right)\right)$$

we have

$$f_i^{(k_i)}(p_{in}) \leq p_{out} = 2^{-\kappa_{out}} .$$

In the above lemma, Δ_i represents a lower bound for κ_{in} which matches the upper bound $\bar{q}_i$ of $p_{in} = 2^{-\kappa_{in}}$. Assuming that s_i and d_i are both monotonically increasing with i, we get that the threshold Δ_i tends towards $\log_2\left(\frac{\mathrm{e}s_i}{d_i}\right)$.

From Lemma 15, we further get that the cost induced by the choice of the compiler CC_i to go from an input probability p_{in} to a target output probability p_{out} is

$$k_i \cdot \log_2(|\lambda_i|) \geq \frac{\log_2(|\lambda_i|)}{\log_2(d_i)}\big(\log_2(\kappa_{out}) - \log_2(\kappa_{in} - \Delta_i)\big) \tag{6.11}$$

(in terms of the log complexity blowup (6.4)). Note that this lower bound is tight: it could be replaced by an equality at the cost of ceiling the term between parentheses (*i.e.* the term corresponding to k_i). We further note that the above equation is consistent with the complexity analysis of the expanding compiler provided in Section 4.4. Indeed going from a constant leakage probability $p_{in} = p$ to a target security level $p_{out} = 2^{-\kappa}$ by applying k_i times a single RPE compiler CC_i, we retrieve a complexity of $\mathcal{O}(\kappa^e)$ with $e = \frac{\log_2(|\lambda_i|)}{\log_2(d_i)}$.

Equation (6.11) shows that using CC_i to go from input probability p_{in} to output probability p_{out} induces a log complexity cost close to

$$\frac{\log_2(|\lambda_i|)}{\log_2(d_i)}\big(\log_2(\kappa_{out}) - \log_2(\kappa_{in})\big)$$

provided that κ_{in} is sufficiently greater than Δ_i. So given the latter informal condition, it appears that the parameter i minimizing the ratio $\frac{\log_2(|\lambda_i|)}{\log_2(d_i)}$ gives the best complexity.

Application. For the asymptotic construction introduced in Section 5.2 and Section 5.3, the RPE compiler CC_i features

- an amplification order $d_i = \mathcal{O}(n_i)$,
- a gadget complexity $s_i = \mathcal{O}(n_i^2)$,
- an eigen-complexity $|\lambda_i| = \mathcal{O}(n_i^2)$.

For such a construction, the ratio $\frac{\log_2(|\lambda_i|)}{\log_2(d_i)}$ is decreasing and converging towards 2 as n_i grows. On the other hand, Δ_i tends to $\log_2(n_i)$ which implies that CC_i should only be applied to an input probability lower than $\frac{1}{n_i}$.

6.1.3 Selection of the Expansion Levels

In this section, we investigate the impact of the choice of the expansion levels k_i on the complexity of the dynamic expanding compiler. We first assess the asymptotic complexity obtained from a simple approach and then provide some application results for some given gadgets.

In the following CC_0 shall denote an RPE compiler with constant parameters while $\{\mathsf{CC}_i\}_{i\geq 1}$ shall denote a family of RPE compilers indexed by a parameter i. We do this distinction since the goal of the CC_0 compiler shall be to tolerate the highest leakage rate and to transit from a (possibly high) leakage probability p to some lower failure probability p_i which is in turn tolerated by at least one compiler from $\{\mathsf{CC}_i\}_i$.

A Simple Approach. We consider a simple approach in which the compiler CC_0 is iterated k_0 times and then a single compiler CC_i is iterated k_i times. The complexity blowup of this compiler is $|\lambda_0|^{k_0}|\lambda_i|^{k_i}$. The first expansion level k_0 is chosen to ensure that the intermediate probability $p_i := f_0^{(k_0)}(p)$ is lower than $\bar{q}_i$ (the lower bound on the tolerated leakage rate of CC_i from Theorem 5). Then k_i is chosen so that $f_i^{(k_i)} \leq 2^{-\kappa}$.

Concretely, we set $\kappa_i := \Delta_i + 1$ which, by Lemma 15, gives

$$k_0 = \left\lceil \log_{d_0}(\Delta_i + 1) - \log_{d_0}(\log_2(p) - \Delta_0) \right\rceil \ , \tag{6.12}$$

and

$$k_i = \left\lceil \log_{d_i}(\kappa) \right\rceil = \mathcal{O}\big(\log_{d_i}(\kappa)\big) \ . \tag{6.13}$$

For some constant leakage probability p and some start compiler CC_0 with constant parameters, we get $k_0 = \mathcal{O}\big(\log_{d_0}(\Delta_i)\big)$ giving an asymptotic complexity blowup of

$$\mathcal{O}\big(|\lambda_0|^{k_0}|\lambda_i|^{k_i}\big) = \mathcal{O}\big(\Delta_i^{e_0}\kappa^{e_i}\big) \quad \text{with} \quad e_0 = \frac{\log_2(|\lambda_0|)}{\log_2(d_0)} \quad \text{and} \quad e_i = \frac{\log_2(|\lambda_i|)}{\log_2(d_i)} \ . \tag{6.14}$$

Then for any choice of i we get an asymptotic complexity blowup of $\mathcal{O}\big(\kappa^{e_i}\big)$ which is the same asymptotic complexity as the standard expanding compiler with base compiler CC_i. On the other hand, our simple dynamic compiler $\mathsf{CC}_i^{(k_i)} \circ \mathsf{CC}_0^{(k_0)}$ tolerates the same leakage rate as CC_0.

Using this simple approach we hence get the best of both worlds:

- a possibly inefficient RPE compiler CC_0 tolerating a high leakage rate q_0,
- a family of RPE compilers $\{\mathsf{CC}_i\}_i$ with complexity exponent $e_i = \frac{\log_2(|\lambda_i|)}{\log_2(d_i)}$ decreasing with i.

We stress that for monotonously increasing $|\lambda_i|$ and d_i, the asymptotic complexity of our simple approach is $\mathcal{O}(\kappa^e)$ where e can be made arbitrary close to $\lim_{i\to\infty} \frac{\log_2(|\lambda_i|)}{\log_2(d_i)}$.

Application. To illustrate the benefits of our dynamic approach, we simply get back to the experimentations on the AES implementation from Section 4.7. We can apply either a 3-share or 5-share compiler (from the constructions in Section 5.5) repeatedly until they reach their targeted security level. While using the 5-share compiler reduces the tolerated probability, we demonstrate that we can use both compilers to get the best tolerated probability as well as a better complexity.

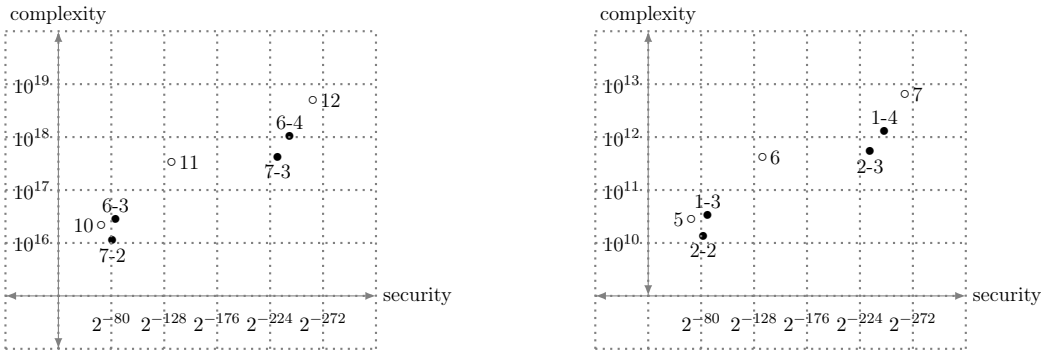

Figure 6.1: Complexity of random probing AES for different security levels for a tolerated probability of $2^{-7.6}$ (left) or $2^{-9.5}$ (right).

Figure 6.1 illustrates the trade-offs in terms of achieved security level and complexity of the expansion strategy when using different compilers at each iteration of the expansion. Starting from a tolerated leakage probability p ($2^{-7.6}$ on the left and $2^{-9.5}$ on the right), the empty bullets ($\circ$) give this trade-off when only the 3-share compiler is iterated. In this case, the final security function ε from Theorem 3 is equal to $f_3^{(k_3)}(p)$ if we consider f_3 to be the failure function of the 3-share compiler, for a certain number of iterations k_3 which is written next to each empty bullet on the figure. On the other hand, the black bullets ($\bullet$) represent the trade-offs achieved in terms of complexity and security levels while combining both compilers with different numbers of iterations. In this case, we start the expansion with a certain number of iterations k_3 of the 3-share compiler, and then we continue with k_5 iterations of the 5-share compiler of failure function f_5, the final compiled circuit is then random probing secure with $\varepsilon = f_5^{(k_5)}(f_3^{(k_3)}(p))$ for $p \in \{2^{-7.6}, 2^{-9.5}\}$. The number of iterations of the compilers is written next to each black bullet in the format k_3-k_5.

For instance, starting from the best tolerated probability $2^{-7.6}$, the static compiler requires 11 applications of the 3-share compiler to achieve a security level of at least 80 bits. This effort comes with an overall complexity of $10^{17.52}$. Using our dynamic approach, we can combine the 3-share and the 5-share to achieve this 80 bits security level for the same tolerated probability but with a complexity of $10^{16.04}$. That would require 7 iterations of the 3-share compiler and 2 iterations of the 5-share compiler. Starting from the same leakage probability, a security level of at least 128 bits is achieved also with 11 applications of the 3-share compiler with a complexity of $10^{17.52}$. In order to achieve at least the same security, we would need more iterations of both compilers in the dynamic approach. With 7 iterations of the 3-share compiler and 3 iterations of the 5-share compiler, we get a complexity of $10^{17.62}$ which is very close to the complexity of the 3-share application alone, while achieving a security level of 231 bits. That is, we almost double the security level achieved using 11 iterations of the 3-share compiler with an almost equal complexity. For a tolerated probability of $2^{-7.6}$ and at least 128 bits of security, note that 11 applications of the 3-share compiler yield a security order of 2^{-135} while both other trade-offs directly yield security orders of 2^{-242} (6 iterations of 3-share and 4 iterations of 5-share) and 2^{-231} (7 iterations of 3-share and 3 iterations of 5-share), with one

less iteration they would be below 128 bits, which explains their more important complexity. The same behavior can be observed with a starting tolerated leakage probability of $2^{-9.5}$ on the right.

We showed this far that the tolerated leakage probability decreases with an increasing number of shares n. So if we want to tolerate the best leakage probability, we would start with a few iterations of a compiler with a small number of shares and which tolerates a good leakage probability (which can be computed, for instance, with the verification tool VRAPS [18]), typically a 3-share construction. Meanwhile, after a few constant number of iterations, we can change to a different compiler which benefits from a better asymptotic complexity (as explained above with our simple approach).

The above results motivate finding RPE compilers which achieve the maximal amplification orders and which benefit from good asymptotic complexity (*i.e.* gadgets defined for any number of shares n with amplification order increasing with n) in order to optimize the security-efficiency trade-off and to tolerate the best possible leakage probability. We showed this far that the tolerated leakage probability decreases with an increasing number of shares n. So if we want to tolerate the best leakage probability, we would start with a few iterations of a compiler with a small number of shares which tolerates a good leakage probability (which can be computed, for instance, with verification tools like IronMask), typically a 3-share construction. Meanwhile, after a few constant number of iterations, we can change to a different compiler which benefits from a better asymptotic complexity (as explained above with our simple approach).

In Section 5.3 and Section 5.4, we instantiate the expanding compiler with ISW-based addition and copy gadgets, and a new multiplication gadget, which all achieve the maximal amplification order. The bottleneck of these constructions in terms of asymptotic complexity was from the linear gadgets (addition and copy) with $N_{\max} = 3n^2 - 2n$. In the next section, we show that the $\mathcal{O}(n \log n)$ refresh gadget from [14] can be used to instantiate addition and copy gadgets with maximal amplification order, hence achieving quasi-linear complexity in $\mathcal{O}(n \log n)$. The bottleneck then becomes the multiplication gadget (with n^2 multiplications), which we also improve in the following sections under some conditions on the base field.

6.2 Linear Gadgets with Quasi-Linear Complexity

In a first attempt, we aim to reduce the complexity of the linear gadgets that are to be used in our dynamic compiler.

A refresh gadget with $\mathcal{O}(n \log n)$ complexity was introduced in [14]. In a nutshell, the idea is to add a linear number of random values on the shares at each step, to split the shares in two sets to apply the recursion, and then to add a linear number of random values again. We recall the algorithmic description of the gadget in Algorithm 3.

The $\mathcal{O}(n \log n)$ refresh gadget was proven to be $(n-1)$-SNI in [14]. In Lemma 16, we show that this gadget is also (t, f)-TRPE of amplification order $\min(t+1, n-t)$ and that it satisfies $(n-1)$-STRPE2. Hence, we can instantiate the generic copy and addition gadgets described in Algorithm 4 and Algorithm 5 from Section 5.2 using the above refresh gadget as G_{refresh}. We thus obtain RPE gadgets G_{add} and G_{copy} enjoying optimal amplification order in quasi-linear complexity $\mathcal{O}(n \log n)$.

Lemma 16. *Let $G_{refresh}$ be the n-share refresh gadget described above from [14]. Then $G_{refresh}$ is (t, f)-TRPE for some function $f : \mathbb{R} \to \mathbb{R}$ of amplification order $d \geq \min(t+1, n-t)$. $G_{refresh}$ is additionally $(n-1)$-STRPE2.*

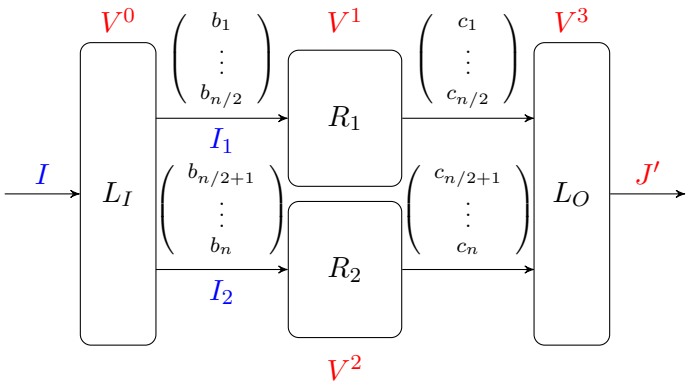

Figure 6.2: $\mathcal{O}(n \log n)$ refresh gadget from [13]

Proof. We will prove that the gadget from Algorithm 3 is (t, f)-TRPE for any $t \leq n-1$ of amplification order $d \geq \min(t+1, n-t)$. For this, we will prove both properties TRPE1 and TRPE2.

Proof for TRPE1.

The gadget is proven to be $(n-1)$-SNI in [13], thus it is (t, f)-TRPE1 of amplification order $d \geq \min(t+1, n-t)$ thanks to Lemma 6. Note that we can find failure sets of wires of size $t+1$ which require the knowledge of $t+1$ input shares (simply consider the leaking wires $\{a_1, \ldots, a_{t+1}\}$ on input a for instance), so $d \leq t+1$.

Proof for $(n-1)$-STRPE2 (which implies TRPE2 of amplification order $t+1$, *c.f.* Corollary 4).

We will first start by recalling the result of Lemma 5 in [13] which will be useful for our proof.

Lemma 5 from [13]. Let $a_1, a_2 \in \mathbb{K}$ be inputs, and let $r \xleftarrow{\$} \mathbb{K}$. Let V be a subset of the variables $\{a_1, a_2, r\}$ and $O \in \{\emptyset, \{a_1 + r\}\}$. Then the variables in $V \cup O \cup \{a2 - r\}$ can be perfectly simulated from $I \subset \{a_1, a_2\}$, with $|I| \leq |V| + 2 \cdot |O|$.

Proof of Lemma 5 from [13]. If $|O| = 1$ or $|V| \geq 2$, we can take $I = \{a_1, a_2\}$. If $|O| = 0$ and $|V| = 0$, we can simulate $a_2 - r$ with a random value. If $|O| = 0$ and $|V| = 1$, if $V = \{a_1\}$ we let $I = \{a_1\}$ and we can again simulate $a_2 - r$ with a random value; if $V = \{r\}$ or $V = \{a_2\}$ then we let $I = \{a_2\}$. □

We are now ready to prove our main result. For TRPE2, we will prove the slightly stronger property $(n-1)$-STRPE2. We can clearly see that $(n-1)$-STRPE2 implies TRPE2 of amplification order $d = t+1$ as shown in Corollary 4. We will prove $(n-1)$-STRPE2 by recurrence on the number of shares $n \geq 2$.

The gadget in the base case $(n = 2)$ gives the following output sharing:

$$\begin{aligned} d_1 &\leftarrow a_1 + r \\ d_2 &\leftarrow a_2 + r \end{aligned}$$

The proof in this case is easy. Mainly, if $J' = \emptyset$, it is easy to see that we can choose J of size 1 such that we can perfectly simulate W and J from a set of input shares I on a such that $|I| \leq |W| \leq 1$. Otherwise, if $|J'| = 1$, then $|W| = 0$, and we choose $J = J'$ and in this case we have $|I| = 0$, since we can perfectly simulate any of the output shares alone by simply

generating a freshly random value. This concludes the proof for the base case.

Next we suppose that the gadget is $(n'-1)$-STRPE2 for any number of shares $n' < n$, and we prove the property for n shares.

To prove this, we split the gadget into four subgadgets as in Figure 6.2, where gadget L_I corresponds to the first loop in Algorithm 3 which adds $\lfloor n/2 \rfloor$ random values to the sharing, R_1 and R_2 gadgets correspond to the two recursive calls respectively, and L_O gadget corresponds to the second loop which also add $\lfloor n/2 \rfloor$ random values to the output sharing. We split any set of probes W on G_{refresh} into $W = V^0 \cup V^1 \cup V^2 \cup V^3$ on each of the subgadgets L_I, R_1, R_2 and L_O respectively. The gadget R_1 is a $\left\lfloor \frac{n}{2} \right\rfloor$-share gadget while R_2 is $\left\lceil \frac{n}{2} \right\rceil$-share gadget. We consider that there are no probes on the output shares of R_1 and R_2 as they can be probed through V^3. Similarly, we consider no output probes on L_I, since they can be probed through V^1 and V^2.

Let W bet the set of probes on G_{refresh} and J' be the set of output shares indices such that $|W| + |J'| \leq n - 1$. We will construct the sets J'_1 and J'_2 for output shares of the gadgets R_1 and R_2 as follows:

- for each $i \in J' \cap \left[\left\lfloor \frac{n}{2} \right\rfloor \right]$, add i to J'_1
- for each $i \in J' \cap \left[\left\lfloor \frac{n}{2} \right\rfloor + 1 : n \right]$, add i to J'_2
- for each $i \in \left[\left\lfloor \frac{n}{2} \right\rfloor \right]$ such that the input probe c_i to L_O is probed in V^3, add i to J'_1
- for each $i \in \left[\left\lfloor \frac{n}{2} \right\rfloor + 1 : n \right]$ such that the input probe c_i to L_O is probed in V^3, add i to J'_2

It can be seen that if we can perfectly simulate J'_1 and J'_2, then we can perfectly simulate J' and all probes in V^3 (V^3 is composed of input probes c_i and random variables r_i, since probes of the form $c_i + r_j$ are probed in J'). Observe that we also have $|J'_1| + |J'_2| \leq |V^3| + |J'|$.
In order for the recurrence hypothesis to hold, we need the following condition to hold for the gadget R_1:

$$|V^1| + |J'_1| \leq \left\lfloor \frac{n}{2} \right\rfloor - 1 \tag{6.15}$$

and the following for the gadget R_2:

$$|V^2| + |J'_2| \leq \left\lceil \frac{n}{2} \right\rceil - 1 \tag{6.16}$$

We consider three cases based on the sizes of the sets of probes:

- $|\mathbf{V^2}| + |\mathbf{J'_2}| \geq \left\lceil \frac{\mathbf{n}}{\mathbf{2}} \right\rceil$. Then we must have $|V^1| + |J'_1| \leq \left\lfloor \frac{n}{2} \right\rfloor - 1$, because we have that $|W| + |J'| \leq n - 1$ and $|J'_1| + |J'_2| \leq |V^3| + |J'|$.
  Since (6.15) holds, by the recurrence hypothesis on R_1, we can choose a set J_1 of size $\left\lfloor \frac{n}{2} \right\rfloor - 1$ such that $J'_1 \subseteq J_1$ and we can perfectly simulate J_1 and V^1 from a set of input shares I_1 on $(b_1, \ldots, b_{\lfloor n/2 \rfloor})$ such that $|I_1| \leq |V^1| + |J'_1|$. Since (6.16) does not hold for R_2, we can set $J_2 = \left[\left\lceil \frac{n}{2} \right\rceil : n \right]$ and $I_2 = \left[\left\lceil \frac{n}{2} \right\rceil : n \right]$, and finally set $J = J_1 \cup J_2$ of $n-1$ output shares on G_{refresh}. We can see that $J'_2 \subseteq J_2$ and J_2 and V^2 can be perfectly simulated from I_2 trivially (full input).
  Next, we show how to perfectly simulate the sets I_1, I_2 on intermediate variable b, and V^0. In fact, thanks to the properties of the L_I gadget, we can apply Lemma 5 from [13]

for all $1 \leq i \leq \lfloor n/2 \rfloor$ on each set of intermediate variables $\{a_i, a_{\lfloor n/2 \rfloor + i}, r_i\}$ and output variable $b_i = a_i + r_i$, where all output variables $b_{\lfloor n/2 \rfloor + i} = a_{\lfloor n/2 \rfloor + i} - r_i$ must be simulated (since we fixed $I_2 = \left[\left\lceil \frac{n}{2} \right\rceil : n \right]$), and by summing the inequalities, we construct $I \subset [n]$ on n-share input a to perfectly simulate I_1, I_2 on intermediate variable b, and V^0 such that

$$|I| \leq |V^0| + 2|I_1| + (n \bmod 2) \leq |V^0| + 2(|V^1| + |J_1'|) + (n \mod 2)$$

where $(n \bmod 2)$ comes from the fact that we need to perfectly simulate all shares of $(b_{\lceil n/2 \rceil}, \ldots, b_n)$ and if $n \mod 2 = 1$, then $b_n = a_n$ by construction of the gadget L_I.
From (6.15) which holds in this case, observe that we have

$$|V^1| + |J_1'| + (n \bmod 2) \leq \left\lfloor \frac{n}{2} \right\rfloor \leq \left\lceil \frac{n}{2} \right\rceil \leq |V^2| + |J_2'|,$$

then we have

$$|I| \leq |V^0| + 2(|V^1| + |J_1'|) + (n \mod 2) \leq |V^0| + |V^1| + |J_1'| + |V^2| + |J_2'|$$

which gives

$$|I| \leq |W| + |J'|$$

and using the input shares in I, we can perfectly simulate probes in V^0, I_1 and I_2, and using I_1 and I_2 we proved that we can perfectly simulate probes in V^1, V^2, J_1 and J_2, and so we can also perfectly simulate the chosen set of $n-1$ output shares J and probes in V^3. So we can perfectly simulate all internal probes plus the chosen set J of $n-1$ output shares from I. This proves the recurrence step in this case.

- $|\mathbf{V^1}| + |\mathbf{J_1'}| \geq \left\lfloor \frac{\mathbf{n}}{\mathbf{2}} \right\rfloor$. Then we must have $|V^2| + |J_2'| \leq \left\lceil \frac{n}{2} \right\rceil - 1$, because we have that $|W| + |J'| \leq n - 1$ and $|J_1'| + |J_2'| \leq |V^3| + |J'|$.
Since (6.16) holds, by the recurrence hypothesis on R_2, we can choose a set J_2 of size $\left\lceil \frac{n}{2} \right\rceil - 1$ such that $J_2' \subseteq J_2$ and we can perfectly simulate J_2 and V^2 from a set of input shares I_2 on $(b_{\lceil n/2 \rceil}, \ldots, b_n)$ such that $|I_2| \leq |V^2| + |J_2'|$. Since (6.15) does not hold for R_1, we can set $J_1 = \left[\left\lfloor \frac{n}{2} \right\rfloor \right]$ and $I_1 = \left[\left\lfloor \frac{n}{2} \right\rfloor \right]$, and finally set $J = J_1 \cup J_2$ of $n-1$ output shares on G_{refresh}. We can see that $J_1' \subseteq J_1$ and J_1 and V^1 can be perfectly simulated from I_1 trivially (full input).
Next, we show how to perfectly simulate the sets I_1, I_2 on intermediate variable b, and V^0. In fact, thanks to the properties of the L_I gadget, we can apply Lemma 5 from [13] for all $1 \leq i \leq \lfloor n/2 \rfloor$ on each set of intermediate variables $\{a_i, a_{\lfloor n/2 \rfloor + i}, r_i\}$ and output variable $b_{\lfloor n/2 \rfloor + i} = a_{\lfloor n/2 \rfloor + i} - r_i$, where all output variables $b_i = a_i + r_i$ must be simulated (since we fixed $I_1 = \left[\left\lfloor \frac{n}{2} \right\rfloor \right]$), and by summing the inequalities, we construct $I \subset [n]$ on n-share input a to perfectly simulate I_1, I_2 on intermediate variable b, and V^0 such that

$$|I| \leq |V^0| + 2|I_2| \leq |V^0| + 2(|V^2| + |J_2'|)$$

(in this case, we don't have the term $(n \mod 2)$ anymore because we do not need the full input sharing $(b_{\lceil n/2 \rceil}, \ldots, b_n)$ for the simulation as before). Since (6.16) holds and (6.15) does not hold, we observe that

$$|V^2| + |J_2'| \leq \left\lceil \frac{n}{2} \right\rceil - 1 \leq \left\lfloor \frac{n}{2} \right\rfloor \leq |V^1| + |J_1'|$$

so we get

$$|I| \leq |V^0| + 2(|V^2| + |J_2'|) \leq |V^0| + |V^2| + |J_2'| + |V^1| + |J_1'|$$

which gives

$$|I| \leq |W| + |J'|$$

and using the input shares in I, we can perfectly simulate probes in V^0, I_1 and I_2, and using I_1 and I_2 we proved that we can perfectly simulate probes in V^1, V^2, J_1 and J_2, and so we can also perfectly simulate the chosen set of $n-1$ output shares J and probes in V^3. So we can perfectly simulate all internal probes plus the chosen set J of $n-1$ output shares from I. This proves the recurrence step in this case.

- $|\mathbf{V^1}| + |\mathbf{J'_1}| \leq \left\lfloor \frac{\mathbf{n}}{\mathbf{2}} \right\rfloor - \mathbf{1}$ and $|\mathbf{V^2}| + |\mathbf{J'_2}| \leq \left\lceil \frac{\mathbf{n}}{\mathbf{2}} \right\rceil - \mathbf{1}$. This case can be treated in the exact same way as the above cases. Namely, if we have $|V^1| + |J'_1| + (n \mod 2) \leq |V^2| + |J'_2|$, then we can consider the first case and treat it in the same way (by appyling the recursion hypothesis on gadget R_1 and setting $J_2 = \left[\left\lceil \frac{n}{2} \right\rceil : n \right]$ and $I_2 = \left[\left\lceil \frac{n}{2} \right\rceil : n \right]$.
  Otherwise, if we have $|V^2| + |J'_2| \leq |V^1| + |J'_1| + (n \mod 2)$, then we can consider the second case and treat it in the same way (by appyling the recursion hypothesis on gadget R_2 and setting $J_1 = \left[\left\lfloor \frac{n}{2} \right\rfloor \right]$ and $I_1 = \left[\left\lfloor \frac{n}{2} \right\rfloor \right]$.
  This also concludes the proof in this case.

By treating all possible cases on the probed wires, we conclude the recursive proof. This proves that for any n shares such that $|W| + |J'| \leq n-1$, we can choose a set J of $n-1$ output shares such that $J' \subseteq J$ and we can perfectly simulate J and W from a set of input shares I such that $|I| \leq |W| + |J'|$. Thus, we conclude that the gadget G_{refresh} is $(n-1)$-STRPE2. Thus, it is also (t, f)-TRPE2 of amplification order $d = t+1$. This concludes the proof. □

Hence, we can instantiate the generic copy and addition gadgets described in Section 5.2 using the above refresh gadget as G_{refresh}. We thus obtain RPE gadgets G_{add} and G_{copy} in quasi-linear complexity $\mathcal{O}(n \log n)$. In addition, G_{add} and G_{copy} achieve the maximal amplification order according to the following corollary, which directly follows from Lemma 16, Lemma 9 and Lemma 7.

Corollary 10. *Let $n \in \mathbb{N}$. Let G_{add} and G_{copy} be the gadgets described in Algorithm 4 and Algorithm 5, and instantiated with the $\mathcal{O}(n \log n)$ refresh gadget described in Algorithm 3. Then for every $t \leq n-1$, G_{add} if (t, f)-RPE and G_{copy} is (t, f')-RPE, both of amplification orders $d = \min(t+1, n-t)$.*

Regarding the asymptotic complexity of the expanding compiler, the eigenvalues λ_1, λ_2 of $\boldsymbol{M}_{ac}$ (4.16) from Section 4.4 are hence now both in $\mathcal{O}(n \log n)$. Namely, we have

$$\boldsymbol{M}_{ac} = \begin{pmatrix} N_{G_{\text{add}},a} & N_{G_{\text{copy}},a} \\ N_{G_{\text{add}},c} & N_{G_{\text{copy}},c} \end{pmatrix} = \begin{pmatrix} 2n(2\log(n)-1)+n & n(2\log(n)-1) \\ 2n(2\log(n)-1) & n(2\log(n)-1)+n \end{pmatrix}.$$

The eigenvalues of the above matrix are $\lambda_1 = n$ and $\lambda_2 = 6n\log(n) - 2n$, implying

$$N_{\max} = \max(6n\log(n) - 2n, N_{G_{\text{mult}},m}) \tag{6.17}$$

At this point, only the quadratic number of multiplications in the multiplication gadget still separates us from a compiler of quasi-linear complexity. We tackle this issue in the next section by constructing a generic multiplication gadget. We finally end up with a full expanding compiler with quasi-linear asymptotic complexity.

6.3 Towards Asymptotically Optimal Multiplication Gadgets

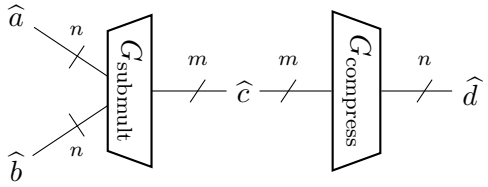

Figure 6.3: n-share multiplication gadget G_{mult} from two subgadgets G_{submult} and G_{compress}

In what follows we should distinguish two types of multiplication gates: regular two-operand multiplications on $\mathbb{K}$, that we shall call bilinear multiplications, and multiplications by constant (or scalar multiplications) which have a single input operand and the constant scalar is considered as part of the gate description.

In the previous constructions, the number of bilinear multiplications is the prominent term of the expanding compiler's complexity. While the most deployed multiplication gadgets (*e.g.*, [59]) require a quadratic number of bilinear multiplications in the masking order, the authors of [17] exhibited a probing secure higher-order masking multiplication with only a linear number of bilinear multiplications. Their construction, which applies on larger fields, is built from the composition of two subgadgets G_{submult} and G_{compress}, as described in Figure 6.3. In a nutshell, on input sharings $\widehat{a}$ and $\widehat{b}$, the subgadget G_{submult} performs multiplications between the input shares of $\widehat{a}$ and $\widehat{b}$ as well as linear combinations of these products and it outputs a m-sharing $\widehat{c}$ of the product $a \cdot b$ where $m \geq n$ [1]. Next, the compression gadget G_{compress} compresses the m-sharing $\widehat{c}$ back into an n-sharing $\widehat{d}$ of the product $a \cdot b$.

The authors of [17] instantiate this construction with a sub-multiplication gadget which performs only $\mathcal{O}(n)$ bilinear multiplications and with the compression gadget from [31]. In addition to bilinear multiplications, their sub-multiplication gadget additionally requires a quadratic number of linear operations (*i.e.*, addition, copy, multiplications by a constant) and random generation gates.

In the following, we rely on the construction [17] with its gadget G_{submult} which offers a linear number of bilinear multiplications to build a more efficient RPE multiplication gadget. In order to use it in our expanding compiler, we integrate an additional gate for the multiplication by a constant and discuss the resulting asymptotic complexity. We additionally demonstrate that the compression gadget of [17] is not $(n-1)$-SNI as claimed in the paper, and show that we can rely on other simple and more efficient compression gadgets which satisfy the expected properties.

6.3.1 Global Multiplication Gadget

We first define two new properties that G_{submult} and G_{compress} will be expected to satisfy to form a (t, f)-RPE multiplication gadget with the maximum amplification order (*c.f.* Lemma 1).

Contrary to the usual simulation notions, the first *partial*-NI property distinguishes the number of probes on the gadget, and the number of input shares that must be used to simulate them. It additionally tolerates a *simulation failure* on at most one of the inputs (*i.e.*, no limitation on the number of shares for the simulation).

[1] In case of a sharewise multiplication for instance, we would have $m = n^2$.

Definition 22 ((s,t)-partial NI)**.** *Let G be a gadget with two input sharings $\widehat{a}$ and $\widehat{b}$. Then G is* (s,t)-partial NI *if and only if any the assignment of any t wires of G can be perfectly simulated from shares $(a_i)_{i\in I_1}$ of $\widehat{a}$ and $(b_i)_{i\in I_2}$ of $\widehat{b}$ such that $|I_1| \leq s$ **or** $|I_2| \leq s$.*

The second property is a variant of the classical TRPE property that we refer to as *comp-TRPE*.

Definition 23 ((t,f)-comp-TRPE)**.** *Let G be a 1-to-1 gadget with m input shares and n output shares such that $m > n$. Let $t \leq n-1$ and $d = \min(t+1, n-t)$. Then G is* (t,f)-comp-TRPE *if and only if for all sets of internal wires W of G with $|W| \leq 2d-1$, we have:*

1. *$\forall\ J, |J| \leq t$ a set of output share indices of G, the assignment of the wires indexed by W and the output shares indexed by J can be jointly perfectly simulated from the input shares of G indexed by a set I, such that $|I| \leq |W|$.*

2. *$\exists\ J', |J'| = n-1$ a set of output share indices of G, such that the assignment of the wires indexed by W and the output shares indexed by J' can be jointly perfectly simulated from the input shares of G indexed by a set I, such that $|I| \leq |W|$.*

Similarly to what was done in [17] for the SNI property, we can prove that the composition of a gadget G_{submult} and G_{compress} which satisfy well chosen properties results in an overall multiplication gadget which is (t,f)-RPE specifically for any $t \leq n-1$ achieving the maximum amplification order $d = \min(t+1, n-t)$. This is formally stated in the following Lemma.

Lemma 17. *Consider the n-share multiplication gadget of Figure 6.3 formed by a 2-to-1 multiplication subgadget $G_{submult}$ of m output shares and a 1-to-1 compression gadget $G_{compress}$ of m input shares such that $m > n$. Let $t \leq n-1$ and $d = \min(t+1, n-t)$. If*

- *$G_{submult}$ is $(d-1)$-NI and $(d-1, 2d-1)$-partial NI,*
- *$G_{compress}$ is (t,f)-comp-TRPE,*

then the multiplication gadget G_{mult} is (t,f)-RPE of amplification order d.

Proof. First let us fix $t \leq n-1$. We will be splitting a set of probe W on the multiplication gadget into two sets of probes $W = W_m \cup W_c$ where W_m are probes on G_{submult} (internal and output wires) and W_c are probes G_{compress} (on internal wires only).

We start by proving RPE1. Let J bet a set of output shares such that $|J| \leq t$.

- Let W be a set of probes on the multiplication gadget such that $|W| = |W_m \cup W_c| \leq d-1$. We know in particular from the comp-TRPE property on G_{compress} that all wires in J and W_c can be simulated from a set of input shares I_c on the intermediate result c such that $|I_c| \leq |W_c|$ (since $|W_c| \leq d-1 < 2d$). Then, we have a set of probes $W'_m = W_m \cup I_c$ on G_{submult} which is of size $|W'_m| \leq |W_m| + |I_c| \leq |W_m| + |W_c| \leq d-1$, then from $(d-1)$-NI property of G_{submult} we know that all the probes in W'_m can be simulated from sets of input shares I_a and I_b such that $|I_a| \leq d-1 \leq t$ and $|I_b| \leq d-1 \leq t$. This proves that we can simulate all probes in the overall set of probes W and in J from at most t shares of a and t shares of b. this proves the first property for RPE1.

- Next let W be a set of probes on the multiplication gadget such that $d \leq |W| = |W_m \cup W_c| \leq 2d-1$. We need to show that we can simulate W and J with at most a failure on one of the inputs a or b. We know in particular from the comp-TRPE property on G_{compress} that all wires in J and W_c can be simulated from a set of input shares I_c on the intermediate result c such that $|I_c| \leq |W_c|$ (since $|W_c| \leq 2d-1 < 2d$). Then, we

have a set of probes $W'_m = W_m \cup I_c$ on G_{submult} which is of size $|W'_m| \leq |W_m| + |I_c| \leq |W_m| + |W_c| \leq 2d-1$. Hence from $(d-1, 2d-1)$-partial NI property of G_{submult}, all the probes in W'_m can be simulated from sets of input shares I_a and I_b such that $|I_a| \leq d-1$ or $|I_b| \leq d-1 \leq t$. Since $d = \min(t+1, n-t)$, then this implies that we have a failure on at most one of the inputs.

This proves that we can simulate all probes in the overall set of probes W and in J from at most t shares of at least one of the inputs a or b (in other words, if we need more than t shares of a, then we need at most t shares of b). This proves the second property for RPE1.

From the above two cases, we conclude that the multiplication gadget is (t, f_1)-RPE1 with amplification order $d = \min(t+1, n-t)$.

Next we prove the property RPE2.

- Let W be a set of probes on the multiplication gadget such that $|W| = |W_m \cup W_c| \leq d-1$. We know in particular from the comp-TRPE property on G_{compress} that there exists a set J of $n-1$ output shares such that all wires in W_c and J can be simulated from a set of input shares I_c on the intermediate result c such that $|I_c| \leq |W_c|$ (since $|W_c| \leq d-1 < 2d$). Then, we have a set of probes $W'_m = W_m \cup I_c$ on G_{submult} which is of size $|W'_m| \leq |W_m| + |I_c| \leq |W_m| + |W_c| \leq d-1$, then from $(d-1)$-NI property of G_{submult} we know that all the probes in W'_m can be simulated from sets of input shares I_a and I_b such that $|I_a| \leq d-1 \leq t$ and $|I_b| \leq d-1 \leq t$. This proves that there exists a set J of $n-1$ output shares such that we can simulate all probes in the overall set of probes W and in J from at most t shares of a and t shares of b. This proves the first property for RPE2.

- Next let W be a set of probes on the multiplication gadget such that $d \leq |W| = |W_m \cup W_c| \leq 2d-1$. We know in particular from the comp-TRPE property on G_{compress} that there exists a set J of $n-1$ output shares such that all wires in W_c and J can be simulated from a set of input shares I_c on the intermediate result c such that $|I_c| \leq |W_c|$ (since $|W_c| \leq 2d-1 < 2d$). Then, we have a set of probes $W'_m = W_m \cup I_c$ on G_{submult} which is of size $|W'_m| \leq |W_m| + |I_c| \leq |W_m| + |W_c| \leq 2d-1$. Hence as for RPE1, from $(d-1, 2d-1)$-partial NI property of G_{submult}, we have that all the probes in W'_m can be simulated from sets of input shares I_a and I_b such that $|I_a| \leq d-1$ or $|I_b| \leq d-1 \leq t$. Since $d = \min(t+1, n-t)$, then this implies that we have a failure on at most one of the inputs.

  This proves that there exists a set J of $n-1$ output shares such that we can simulate all probes in the overall set of probes W and in J from at most t shares of at least one of the inputs a or b (in other words, if we need more than t shares of a, then we need at most t shares of b). This proves the second property for RPE2.

From the above two cases, we conclude that the multiplication gadget is (t, f_2)-RPE2 with amplification order $d = \min(t+1, n-t)$.

Combining both properties RPE1 and RPE2 with the same amplification order d, we conclude that the multiplication gadget is (t, f)-RPE with $f = \max(f_1, f_2)$ and of amplification order $d = \min(t+1, n-t)$. This concludes the proof of Lemma 17. □

6.3.2 Construction of G_{compress}

In a first attempt, we analyze the compression function that was introduced in [31] and used to build a multiplication gadget in [17]. As it turns out not to be SNI or meet our requirements

for the expanding compiler, we exhibit a new and also more efficient construction in a second attempt.

G_{compress} from [17, 31]. The authors of [17] use the $[m : n]$-compression gadget introduced in [31] for any input sharing m, using a $[2n : n]$-compression subgadget as a building block. In a nutshell, it first generates an *ISW*-refresh of the zero n-sharing $(w_1, \ldots, w_n)$. Then, these shares are added to the input ones $(c_1, \ldots, c_n)$ to produce the sequence of output shares $(c_1 + w_1, \ldots, c_n + w_n)$.

The compression gadget is claimed to be $(n-1)$-SNI in [17]. However, we demonstrate that it is not with the following counterexample. Let $n > 2$ and $i \in [n]$. We consider the set composed of a single output share of the compression procedure $J = \{(c_i + w_i) + c_{n+i}\}$ and the set of probes on the internal wires $W = \{w_i\}$. For the compression to be 2-SNI, we must be able to perfectly simulate both the wires in W and J with at most $|W| = 1$ share of the input $\widehat{c}$. However, we can easily observe that $(c_i + w_i) + c_{n+i} - w_i = c_i + c_{i+n}$ requires the two input shares c_i and c_{i+n} to be simulated, which does not satisfy the 2-SNI property. In conclusion, the above gadget is actually not SNI, and interestingly it is not sufficient either for our construction, *i.e.* it does not satisfy Definition 23. This observation motivates our need for a new compression gadget which satisfies the necessary property for our construction.

New Construction for G_{compress}. In Algorithm 7, we exhibit a new $[m : n]$-compression technique using an m-share refresh gadget G_{refresh} as a building block. We demonstrate in Lemma 18 that this new compression gadget satisfies the necessary properties for our construction as long as $m \geq 2n$.

Algorithm 7: $[m : n]$-compression gadget

Input : $(c_1, \ldots, c_m)$ such that $m \geq 2n$, m-share refresh gadget G_{refresh}
Output: $(d_1, \ldots, d_n)$ such that $\sum_{i=1}^{n} d_i = \sum_{i=1}^{m} c_i$

$K \leftarrow \lfloor m/n \rfloor$;
$(c'_1, \ldots, c'_m) \leftarrow G_{\text{refresh}}(c_1, \ldots, c_m)$;
$(d_1, \ldots, d_n) \leftarrow (c'_1, \ldots, c'_n)$;
for $i = 1$ **to** $K - 1$ **do**
  $(d_1, \ldots, d_n) \leftarrow (d_1 + c'_{1+i \cdot n}, \ldots, d_n + c'_{n+i \cdot n})$;
end
for $i = 1$ **to** $m - K \cdot n$ **do**
  $d_i \leftarrow d_i + c'_{i+K \cdot n}$;
end
return $(d_1, \ldots, d_n)$;

Lemma 18. *Let $G_{compress}$ be the $[m : n]$-compression gadget from Algorithm 7 such that $m \geq 2n$. If $G_{refresh}$ is $(m-1)$-SNI and $(m-1)$-STRPE2 (c.f. Definition 20), then $G_{compress}$ is (t, f)-comp-TRPE (Definition 23).*

Proof. Let G_{compress} be the $[m : n]$-compression gadget from Algorithm 7 such that $m \geq 2n$ and let G_{refresh} be the m-share refresh gadget such that G_{refresh} is $(m-1)$-SNI and $(m-1)$-STRPE2. We will prove that G_{compress} $[m : n]$-compression gadget constructed with such G_{refresh} is (t, f)-*comp-TRPE*. Let us denote $(c_1, \ldots, c_m)$ the input shares of G_{compress}, $(d_1, \ldots, d_n)$ its output shares, and $(c'_1, \ldots, c'_m)$ the refreshed shares of $(c_1, \ldots, c_m)$ using G_{refresh}. We write m as $m = K.n + \ell$ for $K, \ell \in \mathbb{N}$ such that $K = \lfloor m/n \rfloor$. For each $1 \leq i \leq \ell$,

we have $d_i = c'_i + \ldots + c'_{i+K.n}$, and for $\ell+1 \leq i \leq n$, we have $d_i = c'_i + \ldots + c'_{i+(K-1).n}$. We will prove that G_{compress} is (t, f)-comp-TRPE. This amounts to proving that $\forall\ W, |W| \leq 2d-1$ a set of probes on the internal wires of G_{compress} where $d = \min(t+1, n-t)$:

1. $\forall\ J, |J| \leq t$ a set of output shares of G_{compress}, J and W can be simulated from a set of input shares I of the input c of G_{compress}, such that $|I| \leq |W|$.
2. $\exists\ J', |J'| = n-1$ a set of output shares of G_{compress}, such that J' and W can be simulated from a set of input shares I of the input c of G_{compress}, such that $|I| \leq |W|$.

We will prove both points separately

1. Let J be a set of output shares indices on G_{compress} such that $|J| \leq t$ for a $t \leq n-1$ and let $d = \min(t+1, n-t)$. Let W be a set of probes on G_{compress} such that $|W| \leq 2d-1$. We need to prove that we can perfectly simulate W and J from input shares indices in I such that $|I| \leq |W|$. For this, We will simulate W and J using probes on G_{refresh}. First let us consider $J^\star$ the set of probes such that $J^\star = \{i \mid c'_i \in W \cap \{c'_1, \ldots, c'_m\}\}$.
   We construct the set W' of probes on G_{refresh} as follows:

$$W' = \{p \mid p \in W \setminus \{c'_1, \ldots, c'_m\}\} \tag{6.18}$$

   In addition, we construct the set J' of output shares on G_{refresh} as follows:

$$J' = J^\star \cup \bigcup_{\substack{i \in J \\ i \leq \ell}} \{i, \ldots, i+K.n\} \cup \bigcup_{\substack{i \in J \\ i > \ell}} \{i, \ldots, i+(K-1).n\} \tag{6.19}$$

   It is easy to see that if we can perfectly simulate W' and J', then we can perfectly simulate W and J since $W = W' \cup \{c'_i \mid i \in J^\star\}$ and by perfectly simulating $(c'_i, \ldots, c'_{i+K.n})$ for $i \in J$ such that $i \leq \ell$, then we can perfectly simulate $d_i = c'_i + \ldots + c'_{i+K.n}$ and by perfectly simulating $(c'_i, \ldots, c'_{i+(K-1).n})$ for $i \in J$ such that $i > \ell$, then we can perfectly simulate $d_i = c'_i + \ldots + c'_{i+(K-1).n}$; thus all output shares in J are perfectly simulate using shares in J'. Hence, we need to prove that we can perfectly simulate W' and J' using the G_{refresh} m-share gadget.
   Observe that since $|J^\star| \leq |W \setminus W'|$, then

$$|J'| \leq |W \setminus W'| + K.|J| + \min(t, \ell) \leq K.t + \min(t, \ell) \tag{6.20}$$

   where the term $\min(t, \ell)$ comes from the worst case where all output shares $i \in J$ are such that $i \leq \ell$, because in this case we add to J' all the indices $(i, \ldots, i+K.n)$ instead of $(i, \ldots, i+(K-1).n)$ according to (6.19). Also, according to (6.18), we have $|W'| \leq |W|$. Hence, we have

$$|W'| + |J'| \leq |W'| + |W \setminus W'| + K.|J| + \min(t, \ell) \leq |W| + K.|J| + \min(t, \ell) \leq 2d - 1 + K.t + \min(t, \ell)$$

   , so

$$|W'| + |J'| \leq 2\min(t+1, n-t) - 1 + K.t + \min(t, \ell)$$

   , then

$$|W'| + |J'| \leq 2(n-t) + K.t + \ell - 1 \leq 2n + (K-2).t + \ell - 1$$

   , and from $t \leq n-1$ we get

$$|W'| + |J'| \leq K.n + \ell - 1 - (K-2)\ .$$

Since by hypothesis we have $m \geq 2n$, so $K \geq 2$ and $(K-2) \geq 0$, hence

$$|W'| + |J'| \leq K.n + \ell - 1 \leq m - 1$$

Then by the $(m-1)$-SNI property of m-share G_{refresh}, we can perfectly simulate the set of probes W' and output shares in J' from a set of input shares I such that $|I| \leq |W'|$, hence we have

$$|I| \leq |W'| \leq |W|$$

which completes the proof for the first point of comp-TRPE on gadget G_{compress}.

2. Let $t \leq n-1$ and let $d = \min(t+1, n-t)$. Let W be a set of probes on G_{compress} such that $|W| \leq 2d-1$. We need to prove that we can perfectly simulate W and a chosen set J of $n-1$ output shares from input shares indices in I such that $|I| \leq |W|$. For this, we will simulate W and choose the set J using probes on G_{refresh}. First let us consider $J^\star$ the set of probes such that $J^\star = \{i \mid c'_i \in W \cap \{c'_1, \ldots, c'_m\}\}$.
We construct the set W' of probes on G_{refresh} as follows:

$$W' = \{p \mid p \in W \setminus \{c'_1, \ldots, c'_m\}\} \tag{6.21}$$

In addition, we construct the set J' of output shares on G_{refresh} as follows:

$$J' = J^\star \tag{6.22}$$

Observe that

$$|W'| + |J'| \leq |W| \leq 2d - 1 \leq 2\min(t+1, n-t) - 1$$

, so

$$|W'| + |J'| \leq 2n - 1 \leq m - 1$$

Then by the $(m-1)$-STRPE2 property of m-share G_{refresh}, there exists a set J'' such that $J' \subseteq J''$ and $|J''| = m-1$ and W' and J'' can be perfectly simulated from input shares indexed in I such that $|I| \leq |W'| + |J'|$. Since $W = W' \cup \{c'_i \mid i \in J'\}$ then $|I| \leq |W|$.
By perfectly simulating W' and J'', we can perfectly simulate W since $W = W' \cup \{c'_i \mid i \in J'\}$. In addition, we choose the set J of $n-1$ output shares on G_{compress} as follows:

$$J = \{i \mid i \leq \ell \text{ and } \{i, \ldots, i + K.n\} \subseteq J''\} \cup \{i \mid i > \ell \text{ and } \{i, \ldots, i + (K-1).n\} \subseteq J''\}$$

Since $|J''| = m-1$, then we are sure that $|J| = n-1$ since there is only 1 share of $(c'_1, \ldots, c'_m)$ missing from J''. And since we can perfectly simulate J'' then we can also perfectly simulate J like before.
This proves that we can choose a set J of $n-1$ output shares on G_{compress} using probes on the internal gadget G_{refresh} such that W and J can be perfectly simulated from input shares in I such that $|I| \leq |W'| + |J'| \leq |W|$ for any $|W| \leq 2d-1$. This concludes the proof for the second point of comp-TRPE on gadget G_{compress}.

Thus, we proved that G_{compress} from Algorithm 7 is (t, f)-STRPE2. This concludes the proof for Lemma 18. □

As shown in Section 6.2 (Lemma 16), the $\mathcal{O}(n \log n)$ refresh gadget from [13] is actually $(m-1)$-SNI and $(m-1)$-STRPE2 for any sharing order m. This gadget can then be used as a building block for the $[m:n]$-compression gadget, giving it a complexity of $\mathcal{O}(m \log m)$ and satisfying the necessary properties. In addition, this further provides an improvement over the complexity of the proposed gadget in [17] which has a complexity of $\mathcal{O}(\lfloor \frac{m}{n} \rfloor n^2)$ (because it performs a n-share ISW-refreshing $\lfloor \frac{m}{n} \rfloor$ times, see [17] for more details on the algorithm).

6.3.3 Construction of $G_{\mathbf{submult}}$

To complete the construction of the overall multiplication gadget, we now exhibit relevant constructions for G_{submult}. We first rely on the construction from [17] which happens to achieve the desired goal in some settings. While all the cases are not covered by the state-of-the-art proposal, we then slightly modify the construction to meet all our requirements. Both constructions rely on linear multiplications that are not included yet in the expanding compiler. We thus start with a construction for this additional linear gadget that we further denote G_{cmult}.

Construction for $G_{\mathbf{cmult}}$. We give a natural construction for G_{cmult} in Algorithm 8 which simply multiplies each input share by the underlying constant value and then applies a (t, f)-RPE refresh gadget G_{refresh}. Basically, with a (T)RPE refresh gadget G_{refresh}, we obtain a (T)RPE linear multiplication gadget G_{cmult} as stated in Lemma 19.

Algorithm 8: n-share multiplication by a constant

Input : sharing $(a_1, \ldots, a_n)$, constant value $\widehat{c}$, n-share refresh gadget G_{refresh}
Output: sharing $(d_1, \ldots, d_n)$ such that $d_1 + \cdots + d_n = c.(a_1 + \ldots + a_n)$

1 $(b_1, \ldots, b_n) \leftarrow (c.a_1, \ldots, c.a_n)$;
2 $(d_1, \ldots, d_n) \leftarrow G_{\text{refresh}}((b_1, \ldots, b_n))$;
3 **return** $(d_1, \ldots, d_n)$;

Lemma 19. *Let $G_{refresh}$ be a (t, f)-(T)RPE n-share refresh gadget of amplification order d. Then G_{cmult} instantiated with $G_{refresh}$ is (t, f')-(T)RPE of amplification order d.*

Proof. G_{cmult} has the exact same wires as the underlying G_{refresh} except for the extra input wires $\{a_1, \ldots, a_n\}$ (the wires multiplied by the constant i.e $\{c \cdot a_1, \ldots, c \cdot a_n\}$ are the input wires to G_{refresh}). So to simulate probes on G_{cmult}, we use the simulator of G_{refresh}. Each probe which is in the set $\{a_1, \ldots, a_n\}$ will be replaced by the corresponding input share multiplied by the constant c, in the set of probes on G_{refresh}, which would lead to a probe on an input share of G_{refresh} of the form $c \cdot a_i$. It is clear that if we can perfectly simulate $c \cdot a_i$ in G_{refresh}, then we can perfectly simulate the input share a_i in G_{cmult}. Thus any set of probes on G_{cmult} is simulated using the simulator of G_{refresh} with the exact same number of probes. Hence, if G_{refresh} is (t, f)-(T)RPE n-share refresh gadget of amplification order d, then the gadget G_{cmult} is also (t, f')-(T)RPE of amplification order d. This concludes the proof. □

Relying on an additional gate for the linear multiplication does not impact the security analysis and the application of the compilation, but it modifies the complexity analysis of the expanding compiler. From the analysis given in Section 4.4, a complexity vector is associated to each base gadget $N_G = (N_a, N_c, N_{cm}, N_m, N_r)^{\mathsf{T}}$ where N_a, N_c, N_{cm}, N_m, N_r stand for the number of addition gates, copy gates, constant multiplication gates, (bilinear) multiplication gates and random gates respectively in the corresponding gadget. The matrix M_{CC} is now a 5×5 square matrix defined as

$$M = \left(N_{G_{\text{add}}} \mid N_{G_{\text{copy}}} \mid N_{G_{\text{cmult}}} \mid N_{G_{\text{mult}}} \mid N_{G_{\text{random}}}\right)$$

including, for each vector, the number of linear multiplications. Five eigenvalues λ_1, λ_2, λ_3, λ_4, λ_5 are to be computed, *i.e.*, one more compared to the expanding compiler in the original setting.

We can consider as before that bilinear multiplication gates are solely used in G_{mult} ($N_{G_{\text{add}},m} = N_{G_{\text{copy}},m} = N_{G_{\text{cmult}},m} = 0$) and that constant multiplication gates are eventually solely used in G_{cmult} and G_{mult} ($N_{G_{\text{add}},cm} = N_{G_{\text{copy}},cm} = 0$) which is the case in the constructions we consider in this paper. It can be checked that (up to some permutation) the eigenvalues satisfy

$$(\lambda_1, \lambda_2) = \mathsf{eigenvalues}(M_{ac}) \ , \quad \lambda_3 = N_{G_{\text{cmult}},cm} \ , \quad \lambda_4 = N_{G_{\text{mult}},m} \quad \text{and} \quad \lambda_5 = n$$

where M_{ac} is the top left 2×2 block matrix of M_{CC}

$$M_{ac} = \begin{pmatrix} N_{G_{\text{add}},a} & N_{G_{\text{copy}},a} \\ N_{G_{\text{add}},c} & N_{G_{\text{copy}},c} \end{pmatrix} \ .$$

We get two complexity expressions for the expansion strategy

$$|\widehat{C}| = \mathcal{O}\big(|C| \cdot N_{\max}^k\big) \tag{6.23}$$

with $N_{\max} = \max(|\mathsf{eigenvalues}(M_{ac})|, N_{G_{\text{cmult}},cm}, N_{G_{\text{mult}},m}, n)$ and with the security parameter κ

$$|\widehat{C}| = \mathcal{O}\big(|C| \cdot \kappa^e\big) \quad \text{with} \quad e = \frac{\log N_{\max}}{\log d} \ .$$

Note that the exhibited construction for the linear multiplication gadget requires $N_{G_{\text{cmult}},cm} = n$ linear multiplications. Hence $\lambda_3 = N_{G_{\text{cmult}},cm} = \lambda_5 = N_{G_{\text{random}},r} = n$ and the global complexity (6.23) can be rewritten as

$$|\widehat{C}| = \mathcal{O}\big(|C| \cdot N_{\max}^k\big) \quad \text{with} \quad N_{\max} = \max(|\mathsf{eigenvalues}(M_{ac})|, N_{G_{\text{mult}},m})$$

if the number of multiplications is greater than n. The asymptotic complexity of the RPE compiler is thus not affected by our new base gadget G_{cmult}. We now describe our constructions of G_{submult}.

G_{submult} from [17]. The authors of [17] provide a $(n-1)$-NI construction for G_{submult} which outputs $2n-1$ shares while consuming only a linear number of bilinear multiplications in the masking order. We first recall their construction which relies on two square matrices of $(n-1)^2$ coefficients in the working field. As shown in [17], these matrices are expected to satisfy some condition for the compression gadget to be $(n-1)$-NI. Since we additionally want the compression gadget to be $(d-1, 2d-1)$-partial NI, we introduce a stronger condition and demonstrate the security of the gadget in our setting.

Let $\mathbb{F}_q$ be the finite field with q elements. Let $\boldsymbol{\gamma} = (\gamma_{i,j})_{1 \le i,j < n} \in \mathbb{F}_q^{(n-1)\times(n-1)}$ be a constant matrix, and let $\boldsymbol{\delta} = (\delta_{i,j})_{1 \le i,j < n} \in \mathbb{F}_q^{(n-1)\times(n-1)}$ be the matrix defined by $\delta_{i,j} = 1 - \gamma_{j,i}$ for all $1 \le i, j < n-1$. G_{submult} takes as input two n-sharings $\widehat{a}$ and $\widehat{b}$ and outputs a $(2n-1)$-sharing $\widehat{c}$ such that:

- $c_1 = \Big(a_1 + \sum_{i=2}^{n}(r_i + a_i)\Big) \cdot \Big(b_1 + \sum_{i=2}^{n}(s_i + b_i)\Big)$
- $c_i = -r_i \cdot \Big(b_1 + \sum_{j=2}^{n}(\delta_{i-1,j-1}s_j + b_j)\Big)$ for $i = 2, \ldots, n$
- $c_{i+n-1} = -s_i \cdot \Big(a_1 + \sum_{j=2}^{n}(\gamma_{i-1,j-1}r_j + a_j)\Big)$ for $i = 2, \ldots, n$

where r_i and s_i are randomly generated values for all $2 \leq i \leq n$. It can be easily checked that G_{submult} performs $2n-1$ bilinear multiplications, and that it is correct, *i.e.* $\sum_{i=1}^{2n-1} c_i = \sum_{i=1}^{n} a_i \cdot \sum_{i=1}^{n} b_i$.

In [17], the authors prove that a gadget is $(n-1)$-NI if one cannot compute a linear combination of any set of $n-1$ probes which can reveal all of the n secret shares of the inputs and which does not include any random value in its algebraic expression. We refer to [17] for more details on this result. Based on this result, the authors demonstrate in [17], that G_{submult} is $(n-1)$-NI if the matrices $\boldsymbol{\gamma}$ and $\boldsymbol{\delta}$ satisfy Condition 1 that we recall below.

Condition 1. ***(from [17])*** *Let $\ell = 2 \cdot (n+1) \cdot (n-1) + 1$. Let $\boldsymbol{I}_{n-1} \in \mathbb{F}_q^{(n-1)\times(n-1)}$ be the identity matrix, $\mathbf{0}_{x\times y} \in \mathbb{F}_q^{x\times y}$ be a matrix of zeros (when $y = 1$, $\mathbf{0}_{x\times y}$ is also written $\mathbf{0}_x$), $\mathbf{1}_{x\times y} \in \mathbb{F}_q^{x\times y}$ be a matrix of ones, $\boldsymbol{D}_{\boldsymbol{\gamma},j} \in \mathbb{F}_q^{(n-1)\times(n-1)}$ be the diagonal matrix such that $D_{\boldsymbol{\gamma},j,i,i} = \gamma_{j,i}$, $\boldsymbol{T}_{n-1} \in \mathbb{F}_q^{(n-1)\times(n-1)}$ be the upper-triangular matrix with just ones, and $\boldsymbol{T}_{\boldsymbol{\gamma},j} \in \mathbb{F}_q^{(n-1)\times(n-1)}$ be the upper-triangular matrix for which $T_{\boldsymbol{\gamma},j,i,k} = \gamma_{j,i}$ for $i \leq k$:*

$$\boldsymbol{I}_{n-1} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \dots & 0 & 1 \end{pmatrix} \qquad \boldsymbol{D}_{\boldsymbol{\gamma},j} = \begin{pmatrix} \gamma_{j,1} & 0 & \dots & 0 \\ 0 & \gamma_{j,2} & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \dots & 0 & \gamma_{j,n-1} \end{pmatrix}$$

$$\boldsymbol{T}_{n-1} = \begin{pmatrix} 1 & 1 & \dots & 1 \\ 0 & 1 & & 1 \\ \vdots & & \ddots & \vdots \\ 0 & \dots & 0 & 1 \end{pmatrix} \qquad \boldsymbol{T}_{\boldsymbol{\gamma},j} = \begin{pmatrix} \gamma_{j,1} & \gamma_{j,1} & \dots & \gamma_{j,1} \\ 0 & \gamma_{j,2} & & \gamma_{j,2} \\ \vdots & & \ddots & \vdots \\ 0 & \dots & 0 & \gamma_{j,n-1} \end{pmatrix}$$

We define the following matrices (with $n' = n-1$):

$$\boldsymbol{L} = \left(\begin{array}{c|c|c|c|c|c|c|c|c|c} 1 & \mathbf{0}_{1\times n'} & \mathbf{0}_{1\times n'} & \mathbf{0}_{1\times n'} & \mathbf{0}_{1\times n'} & \dots & \mathbf{0}_{1\times n'} & \mathbf{1}_{1\times n'} & \mathbf{1}_{1\times n'} & \dots & \mathbf{1}_{1\times n'} \\ \mathbf{0}_{n'} & \boldsymbol{I}_{n'} & \mathbf{0}_{n'\times n'} & \boldsymbol{I}_{n'} & \boldsymbol{I}_{n'} & \dots & \boldsymbol{I}_{n'} & \boldsymbol{T}_{n'} & \boldsymbol{T}_{n'} & \dots & \boldsymbol{T}_{n'} \end{array}\right)$$

$$\boldsymbol{M} = \left(\begin{array}{c|c|c|c|c|c|c|c|c|c} \mathbf{0}_{n'} & \mathbf{0}_{n'\times n'} & \boldsymbol{I}_{n'} & \boldsymbol{I}_{n'} & \boldsymbol{D}_{\boldsymbol{\gamma},1} & \dots & \boldsymbol{D}_{\boldsymbol{\gamma},n'} & \boldsymbol{T}_{n'} & \boldsymbol{T}_{\boldsymbol{\gamma},1} & \dots & \boldsymbol{T}_{\boldsymbol{\gamma},n'} \end{array}\right)$$

Condition 1 is satisfied for a matrix $\boldsymbol{\gamma}$ *if for any vector $\boldsymbol{v} \in \mathbb{F}_q^{\ell}$ of Hamming weight $\mathsf{hw}(\boldsymbol{v}) \leq n-1$ such that $\boldsymbol{L} \cdot \boldsymbol{v}$ contains no coefficient equal to 0 then $\boldsymbol{M} \cdot \boldsymbol{v} \neq \mathbf{0}_{n-1}$.*

In the above condition, the matrices $\mathbf{L}$ and $\mathbf{M}$ represent the vectors of dependencies for each possible probe. All the probes involving shares of $\widehat{a}$ for matrix $\boldsymbol{\gamma}$ (and symmetrically shares of $\widehat{b}$ for matrix $\boldsymbol{\delta}$) are covered in the columns of $\mathbf{L}$ and $\mathbf{M}$. Namely, the first column represents the probe a_1. As it does not involve any random, it results in a zero column in $\mathbf{M}$. The next columns represents the probes a_i, then the probes r_i. They are followed by columns for the probes $(a_i + r_i)$, then $(a_i + \gamma_{j-1,i-1} r_i)$ (for $2 \leq j \leq n$), then $a_1 + \sum_{i=2}^{k}(r_i + a_i)$ (for $2 \leq k \leq n$), and finally then $a_1 + \sum_{j=2}^{k}(\gamma_{i-1,j-1} r_j + a_j)$ (for $2 \leq i \leq n$ and $2 \leq k \leq n$). The above condition means that there is no linear combination of $(n-1)$ probes which can include the expression of all of the input shares, and no random variable.

From this result and by the equivalence between non-interference and tight non-interference developed in [17], we conclude that G_{submult} is $(d-1)$-NI for $d = \min(t+1, n-t)$ for any $t \leq n-1$. Lemma 17 also requires G_{submult} to be $(d-1, 2d-1)$-partial NI to get an overall RPE multiplication gadget. For G_{submult} to satisfy this second property, we need to rely on a stronger condition for matrices $\boldsymbol{\gamma}$ and $\boldsymbol{\delta}$ that we present in Condition 2.

Condition 2. *Let* $z = 2 \cdot (n+1) \cdot (n-1) + 1$. *Let* $\boldsymbol{I}_{n-1} \in \mathbb{F}_q^{(n-1)\times(n-1)}$, $\boldsymbol{0}_{\ell\times n} \in \mathbb{F}_q^{\ell\times n}$, $\boldsymbol{1}_{\ell\times n} \in \mathbb{F}_q^{\ell\times n}$, $\boldsymbol{D}_{\gamma,j} \in \mathbb{F}_q^{(n-1)\times(n-1)}$, $\boldsymbol{T}_{n-1} \in \mathbb{F}_q^{(n-1)\times(n-1)}$, $\boldsymbol{T}_{\gamma,j} \in \mathbb{F}_q^{(n-1)\times(n-1)}$ *and* $\boldsymbol{L}$ *and* $\boldsymbol{M}$ *the same matrices as defined in Condition 1.*

Condition 2 is satisfied for a matrix $\boldsymbol{\gamma}$ *if and only if for any vector* $\boldsymbol{v} \in \mathbb{F}_q^z$ *of Hamming weight* $\mathsf{hw}(\boldsymbol{v}) \leq n-1$, *and for any* $i_1, \ldots, i_K \in [z]$ *such that* $v_{i_1} \neq 0, \ldots, v_{i_K} \neq 0$ *and the corresponding columns* $i_1, \ldots, i_K$ *in* $\boldsymbol{L}$ *and in* $\boldsymbol{M}$ *have no zero coefficient (i.e there are* K *probes of the form* $a_1 + \sum_{i=2}^{n}(r_i + a_i)$ *or* $a_1 + \sum_{j=2}^{n}(\gamma_{i-1,j-1}r_j + a_j)$ *for any* $i \in \{2, \ldots, n\}$*), if* $\mathbf{M}.v = 0$, *then we have* $\mathsf{hw}(\boldsymbol{L} \cdot \boldsymbol{v}) \leq \mathsf{hw}(\boldsymbol{v}) - K$.

Based on this new condition, we can prove our second property G_{submult}, as stated in Lemma 20.

Lemma 20. *Let* $t \leq n-1$ *such that either* n *is even or* $t \neq \lfloor \frac{n-1}{2} \rfloor$ *and let* $d = \min(t+1, n-t)$. *Let* $G_{submult}$ *the multiplication subgadget introduced in [17]. If both matrices* $\boldsymbol{\gamma}$ *and* $\boldsymbol{\delta}$ *satisfy Condition 2, then* $G_{submult}$ *is* $(d-1)$*-NI and* $(d-1, 2d-1)$*-partial NI.*

Proof. Let $t \leq n-1$ where n is the number of shares such that $(n,t) \neq (2k+1, \lfloor \frac{n-1}{2} \rfloor)$ for $k \in \mathbb{N}$ (i.e n is even or $t \neq \lfloor \frac{n-1}{2} \rfloor$), and let $d = \min(t+1, n-t)$. We will prove that if both matrices γ and δ satisfy Condition 2, then G_{submult} from Lemma 20 is $(d-1)$-NI and $(d-1, 2d-1)$-partial NI.

Proof for $(d-1)$-NI:
If the matrices γ and δ satisfy Condition 2, then they also sastisfy Condition 1, since Condition 2 is stronger. Then, in [17], the authors prove that we have that if γ and δ satisfy Condition 1, then the gadget G_{submult} is $(n-1)$-NI. In addition, if G_{submult} is $(n-1)$-NI, then in particular it is also $(d-1)$-NI for any $t \leq n-1$ and $d = \min(t+1, n-t)$. This implies that if the matrices satisfy Condition 2, then the gadget G_{submult} is $(d-1)$-NI thanks to the proof from [17]. This concludes the proof for the first point of Lemma 20.

Proof for $(d-1, 2d-1)$-partial NI:
We need to prove that G_{submult} is $(d-1, 2d-1)$-partial NI where $d = \min(t+1, n-t)$. In other words, we need to consider a set of probes W of size $|W| \leq 2d-1 \leq n-1$ and show that W can be simulated from inputs shares I_a and I_b such that $|I_a| \leq d-1$ or $|I_b| \leq d-1$. For this, we will split the set W into 3 distinct subsets $W = W_1 \cup W_2 \cup W_3$ with respect to the form of the probes in W. In fact, The authors from [17] show that G_{submult} is $(n-1)$-NI if the matrices γ and δ satisfy certain conditions. In fact, all of the probes on the sub-gadget G_{submult} are of a form in one of the following sets:

Set 1: $a_1, a_i, r_i, r_i + a_i, \gamma_{j-1,i-1}r_i, \gamma_{j-1,i-1}r_i + a_i$ (for $2 \leq i \leq n$ and $2 \leq j \leq n$)

Set 2: $a_1 + \sum_{i=2}^{k}(r_i + a_i)$ (for $2 \leq k \leq n$)

Set 3: $a_1 + \sum_{i=2}^{k}(\gamma_{j-1,i-1}r_i + a_i)$ (for $2 \leq j \leq n$ and $2 \leq k \leq n$)

Set 4: $b_1, b_i, s_i, s_i + b_i, \delta_{j-1,i-1}s_i, \delta_{j-1,i-1}s_i + b_i$ (for $2 \leq i \leq n$ and $2 \leq j \leq n$)

Set 5: $b_1 + \sum_{i=2}^{k}(s_i + b_i)$ (for $2 \leq k \leq n$)

Set 6: $b_1 + \sum_{i=2}^{k}(\delta_{j-1,i-1}s_i + b_i)$ (for $2 \leq j \leq n$ and $2 \leq k \leq n$)

Set 7: $-r_i \times (b_1 + \sum_{j=2}^{n}(\delta_{i-1,j-1}s_j + b_j))$ (for $2 \leq i \leq n$)

Set 8: $-s_i \times (a_1 + \sum_{j=2}^{n}(\gamma_{i-1,j-1}r_j + a_j))$ (for $2 \leq i \leq n$)

Set 9: $(a_1 + \sum_{i=2}^{n}(r_i + a_i)) \times (b_1 + \sum_{i=2}^{n}(s_i + b_i))$

The matrix γ would be related to probes of the form 1,2 and 3, while the matrix δ is directly related to probes of the form 4,5 and 6.
So we split the set W into $W = W_1 \cup W_2 \cup W_3$ with respect to the form of each probe as follows:

- W_1 contains probes of the forms in the sets 1, 2 and 3.
- W_2 contains probes of the forms in the sets 4, 5 and 6.
- W_3 contains probes of the forms in the sets 7, 8 and 9.

This split means that the set W_1 only contains probes involving the input shares of a and the randoms r_i, while W_2 only contains probes involving the input shares of b and the randoms s_i. W_3 contains products of both of the probes of W_1 and W_2.
Next, we will construct two subsets of probes W_a and W_b from the set W and prove that we can simulate all probes in W from W_a and W_b. In other terms, we start with $W_a = W_1$ and $W_b = W_2$.

Suppose first that $W_3 = \emptyset$. Then we consider the sets $W_a = W_1$ and $W_b = W_2$ as before. Suppose that to simulate W_a, we need sets of input shares I_a such that $|I_a| \geq d$, and let M be the number of probes of the form in sets 2 and 3 in the set of probes W_a. Then from Condition 2 on matrix γ we know that $|I_a| \leq |W_a| - M \leq |W_a|$ (because $|W_a| \leq 2d-1 \leq n-1$ since $t \leq n-1$ such that $\Big((n = 2k) \vee (t \neq \frac{n-1}{2})\Big)$), then in order to have $|I_a| \geq d$, we must have:

$$d \leq |I_a| \leq |W_a|$$

Hence, since $|W| \leq 2d-1$, then we must have $|W_b| \leq d-1$ (because $|W_a| + |W_b| \leq 2d-1$), then from Condition 2 on matrix δ, we can perfectly simulate W_b from I_b such that $|I_b| \leq |W_b| - M' \leq |W_b| \leq d-1$ where M' is the number of probes of the form in sets 5 and 6 in the set of probes W_b. Thus we showed that we can perfectly simulate W with $|W| \leq 2d-1 \leq n-1$ from W_a and W_b using I_a and I_b such that if $|I_a| \geq d$, then $|I_b| \leq d-1$, so we have $|I_a| \leq d-1$ or $|I_b| \leq d-1$. This concludes the proof in the case where $W_3 = \emptyset$.

Next, we suppose that $W_3 \neq \emptyset$ so there is at least one probe of one of the sets 7, 8 or 9 in W_3. We construct sets W_a and W_b as before starting with $W_a = W_1$ and $W_b = W_2$, and for each probe in W_3:

- If the probe is of the form $-r_i \times (b_1 + \sum_{j=2}^{n}(\delta_{i-1,j-1}s_j + b_j))$, then we do $W_a = W_a \cup \{-r_i\}$, $W_b = W_b \cup \{(b_1 + \sum_{j=2}^{n}(\delta_{i-1,j-1}s_j + b_j))\}$. We denote the set of these probes in W_3 as W_3^7.
- If the probe is of the form $-s_i \times (a_1 + \sum_{j=2}^{n}(\gamma_{i-1,j-1}r_j + a_j))$, then we do $W_a = W_a \cup \{(a_1 + \sum_{j=2}^{n}(\gamma_{i-1,j-1}r_j + a_j))\}$, $W_b = W_b \cup \{-s_i\}$. We denote the set of these probes in W_3 as W_3^8.
- if the probe is of the form $(a_1 + \sum_{i=2}^{n}(r_i + a_i)) \times (b_1 + \sum_{i=2}^{n}(s_i + b_i))$, then we do $W_a = W_a \cup \{(a_1 + \sum_{i=2}^{n}(r_i + a_i))\}$, $W_b = W_b \cup \{(b_1 + \sum_{i=2}^{n}(s_i + b_i))\}$. We denote the set of these probes in W_3 as W_3^9.

Suppose that in order to simulate W_a, we need the set I_a such that $|I_a| \geq d$. In addition, since $|W_a| \leq |W| \leq 2d-1 \leq n-1$ (because $t \leq n-1$ such that $\Big((n = 2k) \vee (t \neq \frac{n-1}{2})\Big)$), then we know from Condition 2 on γ that W_a can be perfectly simulated from I_a such that $|I_a| \leq |W_a| - M$ where M is the number of probes in W_a of the form $(a_1 + \sum_{j=2}^{n}(\gamma_{i-1,j-1}r_j + a_j))$

or $(a_1 + \sum_{i=2}^{n}(r_i + a_i))$. Then, since probes in the sets W_3^8 and W_3^9 add to W_a probes of these forms, then we have $|I_a| \leq |W_a| - |W_3^8| - |W_3^9|$. Hence, in order to have $|I_a| \geq d$, we must have

$$d \leq |I_a| \leq |W_a| - |W_3^8| - |W_3^9| \leq |W_1| + |W_3^7|$$

Similarly, suppose that to simulate W_b we need $|I_b| \geq d$, then we also must have

$$d \leq |I_b| \leq |W_b| - |W_3^7| - |W_3^9| \leq |W_2| + |W_3^8|$$

Hence, in order to have $|I_a| \geq d$ and $|I_b| \geq d$ at the same time, we must have

$$2d \leq |I_a| + |I_b| \leq |W_1| + |W_3^7| + |W_2| + |W_3^8| \leq |W|$$

which holds a contradiction with the fact that $|W| \leq 2d - 1$. Hence, we cannot have at the same time $|I_a| \geq d$ and $|I_b| \geq d$. So G_{submult} is $(d-1, 2d-1)$-partial NI in the case where $W_3 \neq \emptyset$.
Hence, we conclude that G_{submult} is $(d-1, 2d-1)$-partial NI after proving the property in both cases $W_3 = \emptyset$ and $W_3 \neq \emptyset$.

We conclude that G_{submult} satisfies both $(d-1)$-NI and $(d-1, 2d-1)$-partial NI, which conludes the proof for Lemma 20. □

The condition on t and n on Lemma 20 implies that the maximum amplification order for the multiplication gadget cannot be achieved for an odd number of shares (since the maximum order is reached when $t = \lfloor \frac{n-1}{2} \rfloor$). This is not a proof artifact but a limitation of the gadget G_{submult} with respect to the new $(d-1, 2d-1)$-partial NI property. We can easily show that under this extreme conditions on t and n, we have $2d - 1 = n$. If we consider the instantiation of G_{submult} for $n = 3$ input shares, we obtain the following $2n - 1 = 5$ output shares:

$$\begin{aligned}
c_1 &= (a_1 + (r_2 + a_2) + (r_3 + a_3)) \cdot (b_1 + (s_2 + b_2) + (s_3 + b_3)) \\
c_2 &= -r_2 \cdot (b_1 + (\delta_{1,1} \cdot s_2 + b_2) + (\delta_{1,2} \cdot s_3 + b_3)) \\
c_3 &= -r_3 \cdot (b_1 + (\delta_{2,1} \cdot s_2 + b_2) + (\delta_{2,2} \cdot s_3 + b_3)) \\
c_4 &= -s_2 \cdot (a_1 + (\gamma_{1,1} \cdot r_2 + a_2) + (\gamma_{1,2} \cdot r_3 + a_3)) \\
c_5 &= -s_3 \cdot (a_1 + (\gamma_{2,1} \cdot r_2 + a_2) + (\gamma_{2,2} \cdot r_3 + a_3))
\end{aligned}$$

To prove the $(d-1, 2d-1)$-partial NI property, we need to ensure that any set of at most $2d - 1 = 3$ probes can be perfectly simulated from at most $d - 1 = 1$ shares of one of the inputs and any number of shares from the other one. However, the three probes on c_1, c_3, c_4 reveal information on each of their sub-product. In particular, $(a_1 + (r_2 + a_1) + (r_3 + a_3))$ (from c_1), r_3 (from c_3) and $(a_1 + (\gamma_{1,1} \cdot r_2 + a_2) + (\gamma_{1,2} \cdot r_3 + a_3))$ (from c_4) would reveal $\widehat{a}$. Similarly, $(b_1 + (s_2 + b_2) + (s_3 + b_3))$ (from c_1), $(b_1 + (\delta_{2,1} \cdot s_2 + b_2) + (\delta_{2,2} \cdot s_3 + b_3))$ (from c_3) and s_2 (from c_4) would reveal $\widehat{b}$. Hence, the gadget is not $(d-1, 2d-1)$-partial NI. This counterexample with 3 shares can be directly extended to any odd number of shares.

This counterexample motivates a new construction for G_{submult} which would cover all values for n and t. In the following, we slightly modify the construction from [17] to achieve the maximum amplification order in any setting.

Remark 4. *The current construction of $G_{submult}$ outputs $m = 2n - 1$ shares, which does not satisfy the requirement $m \geq 2n$ shares for the compression gadget. Nevertheless, it is enough to add an artificial extra share c_{2n-1} equal to zero between both building blocks. In particular, the compression gadget (and subsequently the refresh gadget) does not expect the input sharing to be uniform to achieve the stated security properties.*

New Construction for G_{submult}. As stated earlier, Lemma 20 does not hold for G_{submult} in the case where n is odd and $t = (n-1)/2$. In order to cover this case, we propose a slightly modified version of G_{submult} with two extra random values r_1 and s_1. In this version, we let $\boldsymbol{\gamma} = (\gamma_{i,j})_{1 \leq i,j \leq n} \in \mathbb{F}_q^{n \times n}$ be a constant matrix, and let $\boldsymbol{\delta} \in \mathbb{F}_q^{n \times n}$ be the matrix defined by $\delta_{i,j} = 1 - \gamma_{i,j}$. The sub-gadget G_{submult} outputs $2n+1$ shares:

- $c_1 = \Big(\sum_{i=1}^{n} (r_i + a_i) \Big) \cdot \Big(\sum_{i=1}^{n} (s_i + b_i) \Big)$
- $c_{i+1} = -r_i \cdot \Big(\sum_{j=1}^{n} (\delta_{i,j} s_j + b_j) \Big)$ for $i = 1, \ldots, n$
- $c_{i+n+1} = -s_i \cdot \Big(\sum_{j=1}^{n} (\gamma_{i,j} r_j + a_j) \Big)$ for $i = 1, \ldots, n$

where r_i and s_i are randomly generated values. It can be easily checked that G_{submult} now performs $2n+1$ bilinear multiplications, and that it is correct, *i.e.* $\sum_{i=1}^{2n+1} c_i = \sum_{i=1}^{n} a_i \cdot \sum_{i=1}^{n} b_i$.

We now need the following slightly modified version of Condition 2 on $\boldsymbol{\gamma}$ and on $\boldsymbol{\delta}$, which instead of considering a linear combination of at most $n-1$ probes as in Condition 2, considers up to n probes.

Condition 3. *Let $z = (2n+4) \cdot n$. Let $\boldsymbol{I}_n \in \mathbb{F}_q^{n \times n}$ be the identity matrix, $\boldsymbol{0}_{\ell \times n} \in \mathbb{F}_q^{\ell \times n}$ be the matrix of zeros, $\boldsymbol{1}_{\ell \times n} \in \mathbb{F}_q^{\ell \times n}$ be the matrix of ones, $\boldsymbol{D}_{\gamma,j} \in \mathbb{F}_q^{n \times n}$ be the diagonal matrix such that $\boldsymbol{D}_{\gamma,j,i,i} = \gamma_{j,i}$, $\boldsymbol{T}_n \in \mathbb{F}_q^{n \times n}$ be the upper triangular matrix with just ones, $\boldsymbol{T}_{\gamma,j} \in \mathbb{F}_q^{n \times n}$ be the upper triangular matrix such that $\boldsymbol{T}_{\gamma,j,i,k} = \gamma_{j,i}$ for $i \leq k$. We define the following matrices:*

$$\boldsymbol{L} = \left[\begin{array}{c|c|c|c|c|c|c|c|c} \boldsymbol{I}_n & \boldsymbol{0}_{n \times n} & \boldsymbol{I}_n & \boldsymbol{I}_n & \ldots & \boldsymbol{I}_n & \boldsymbol{T}_n & \boldsymbol{T}_n & \ldots & \boldsymbol{T}_n \end{array} \right]$$
$$\boldsymbol{M} = \left[\begin{array}{c|c|c|c|c|c|c|c|c} \boldsymbol{0}_{n \times n} & \boldsymbol{I}_n & \boldsymbol{I}_n & \boldsymbol{D}_{\gamma,1} & \ldots & \boldsymbol{D}_{\gamma,n} & \boldsymbol{T}_n & \boldsymbol{T}_{\gamma,1} & \ldots & \boldsymbol{T}_{\gamma,n} \end{array} \right]$$

Then we say that $\boldsymbol{\gamma}$ satisfies Condition 3 if and only if

- *for any vector $\boldsymbol{v} \in \mathbb{F}_q^z$ of Hamming weight $\mathsf{hw}(\boldsymbol{v}) \leq n$,*
- *for any $i_1, \ldots, i_K \in [z]$ such that $v_{i_1} \neq 0, \ldots, v_{i_K} \neq 0$ and the corresponding columns $i_1, \ldots, i_K$ in $\boldsymbol{L}$ and in $\boldsymbol{M}$ have no zero coefficient (i.e there are K probes of the form $\sum_{i=1}^{n}(r_i + a_i)$ or $\sum_{j=1}^{n}(\gamma_{i,j} r_j + a_j)$ for any $i = 1, \ldots, n$),*

if $\boldsymbol{M} \cdot \boldsymbol{v} = 0$, then we have $\mathsf{hw}(\boldsymbol{L} \cdot \boldsymbol{v}) \leq \mathsf{hw}(\boldsymbol{v}) - K$.

Under this new condition, we obtain the following result.

Lemma 21. *Let $t \leq n-1$ and $d = \min(t+1, n-t)$. Let $G_{submult}$ as defined above with n-share inputs. If both matrices $\boldsymbol{\gamma}$ and $\boldsymbol{\delta}$ satisfy Condition 3, then $G_{submult}$ is $(d-1)$-NI and $(d-1, 2d-1)$-partial NI.*

Proof. The proof of the Lemma is in fact the same as the proof of Lemma 20. The only difference is that in this lemma, we also cover the special case of an odd value for the number of shares n and $t = \lfloor \frac{n-1}{2} \rfloor = \frac{n-1}{2}$. In the latter case, we consider in the proof up to n probes on the gadget G_{submult}, while in Lemma 20, we could only have up to $n-1$ probes on the gadget. Since Condition 3 covers the case of having up to n probes on G_{submult}, then we can follow the exact same procedure of the proof of Lemma 20 to prove the Lemma by considering the new condition. □

Remark 5. *The number of output shares $m = 2n+1$ of $G_{submult}$ satisfies the constraint required by $G_{compress}$ in Algorithm 7 ($m \geq 2n$). We can thus use the compression gadget $G_{compress}$ exactly as described in the algorithm on the input sharing $(c_1, \ldots, c_{2n+1})$, instantiated with the $\mathcal{O}(n \log n)$ refresh gadget. Since the multiplication sub-gadget $G_{submult}$ requires $\mathcal{O}(n)$ random values and $G_{compress}$ requires $\mathcal{O}(n \log n)$ random values from the refresh gadget, the overall multiplication gadget G_{mult} also requires a quasi-linear number of random values $\mathcal{O}(n \log n)$.*

6.3.4 Instantiations

We first state the existence of a matrix $\boldsymbol{\gamma}$ which satisfies Condition 3 over any finite field $\mathbb{F}_q$ for q large enough (with $\log(q) = \Omega(n \log n)$)[2]. The proof technique follows closely the proof of [17, Theorem 4.5] and makes use of the non-constructive "probabilistic method". Specifically, it states that if one chooses $\boldsymbol{\gamma}$ uniformly at random in $\mathbb{F}_q^{n \log n}$, the probability that the matrix $\boldsymbol{\gamma}$ satisfies Condition 3 is strictly positive, when q is large enough. It is important to note that the proof relies on probability but the existence of a matrix $\boldsymbol{\gamma}$ which satisfies Condition 3 (for q large enough) is guaranteed without any possible error.

Theorem 6. *For any $n \geq 1$, for any prime power q, if $\boldsymbol{\gamma}$ is chosen uniformly in $\mathbb{F}_q^{n \times n}$, then*

$$\Pr[\boldsymbol{\gamma} \textit{ satisfies Condition 3}] \geq 1 - 2 \cdot (12n)^n \cdot n \cdot q^{-1} \ .$$

In particular, for any $n \geq 1$, there exists an integer $Q = \mathcal{O}(n)^{n+1}$, such that for any prime power $q \geq Q$, there exists a matrix $\boldsymbol{\gamma} \in {\mathbb{F}_q}^{n \times n}$ satisfying Condition 3.

As when $\boldsymbol{\gamma}$ is uniformly random, so is $\boldsymbol{\delta}$, Theorem 6 immediately follows from the following proposition and the union bound.

Proposition 4. *For any $n \geq 1$, for any prime power q, if $\boldsymbol{\gamma}$ is chosen uniformly in $\mathbb{F}_q^{n \times n}$, then*

$$\Pr[\boldsymbol{\gamma} \textit{ satisfies Condition 3}] \geq 1 - (12n)^n \cdot n \cdot q^{-1} \ .$$

In particular, for any $n \geq 1$, there exists an integer $Q = \mathcal{O}(n)^{n+1}$, such that for any prime power $q \geq Q$, there exists a matrix $\boldsymbol{\gamma} \in \mathbb{F}_q^{n \times n}$ satisfying Condition 3.

The proof of this proposition is very technical but follows essentially the proof of the analogous [17, Proposition 4.6]. It is provided in Section A.3.

In [17], Belaïd *et al.*. presented examples of matrices which satisfy their condition for 2 shares and 3 shares. Karpman and Roche [61] proposed afterwards new explicit instantiations up to order $n = 6$ over large finite fields and up to $n = 4$ over practically relevant fields such as $\mathbb{F}_{256}$. It is worth mentioning that the matrices proposed in [61] are actually incorrect (due to a sign error) but this can be easily fixed and we check that matrices obtained following [61] also achieve our Condition 3.

A first matrix for 3 shares can be used over the finite field $\mathbb{F}_{2^5}$ represented as $\mathbb{F}_2[X]/(X^5 + X^2 + 1)$:

$$\boldsymbol{\gamma} = \begin{pmatrix} X+1 & X & X^2+1 \\ X & X^2+1 & X+1 \\ X^2+1 & X+1 & X \end{pmatrix}$$

Another matrix for 3 shares (denoted in hexadecimal by evaluating each polynomial at $X = 2$ and writing the result in base 16) can be used over the finite field $\mathbb{F}_{2^6}$ represented as

[2]Such large finite fields may actually be useful to build efficient symmetric primitives (see for instance MiMC [2]).

$\mathbb{F}_2[X]/(X^6+X+1)$:

$$\gamma = \begin{pmatrix} \texttt{36} & \texttt{30} & \texttt{1d} \\ \texttt{21} & \texttt{05} & \texttt{1a} \\ \texttt{35} & \texttt{31} & \texttt{1b} \end{pmatrix}$$

Another example for 4 shares can be instantiated using the following matrix (also denoted in hexadecimal) over the (AES) finite field $\mathbb{F}_{2^8}$ represented as $\mathbb{F}_2[X]/(X^8+X^4+X^3+X+1)$:

$$\gamma = \begin{pmatrix} \texttt{2d} & \texttt{f5} & \texttt{2e} & \texttt{23} \\ \texttt{e1} & \texttt{c3} & \texttt{ac} & \texttt{30} \\ \texttt{bd} & \texttt{f6} & \texttt{fa} & \texttt{8a} \\ \texttt{e6} & \texttt{4a} & \texttt{4d} & \texttt{ab} \end{pmatrix}$$

Eventually, we present a matrix for 5 shares over the finite field $\mathbb{F}_{2^{10}}$ represented as $\mathbb{F}_2[X]/(X^{10}+X^3+1)$:

$$\gamma = \begin{pmatrix} \texttt{225} & \texttt{2a9} & \texttt{0d0} & \texttt{224} & \texttt{2dd} \\ \texttt{254} & \texttt{11b} & \texttt{325} & \texttt{3a6} & \texttt{219} \\ \texttt{3d2} & \texttt{2bc} & \texttt{2bf} & \texttt{3a2} & \texttt{2a1} \\ \texttt{2af} & \texttt{311} & \texttt{295} & \texttt{26b} & \texttt{11d} \\ \texttt{16c} & \texttt{124} & \texttt{158} & \texttt{319} & \texttt{0b8} \end{pmatrix}$$

6.4 Improved Asymptotic Complexity

In the previous sections, we exhibit the construction of a multiplication gadget G_{mult} which performs a linear number of multiplications between variables, and a quadratic number of multiplications by a constant operations. Using the results of Lemma 18, Lemma 21 and Lemma 17, the constructed multiplication gadget is RPE and achieves the maximum amplification order $\lfloor \frac{n+1}{2} \rfloor$ for any number of shares n.

Using the linear gadgets (G_{add}, G_{copy}, G_{cmult}) instantiated with the $\mathcal{O}(n \log n)$ refresh gadget (which all achieve the maximal amplification order for any number of shares n), and the proposed construction of the multiplication gadget G_{mult}, we get an expanding compiler with a complexity matrix M_{CC} of eigenvalues:

$$(\lambda_1, \lambda_2) = (n, 6n\log(n) - 2n)\ , \quad \lambda_3 = n\ , \quad \lambda_4 = 2n+1 \quad \text{and} \quad \lambda_5 = n.$$

Hence we have $N_{\max} = 6n\log(n) - 2n = \mathcal{O}(n \log n)$.

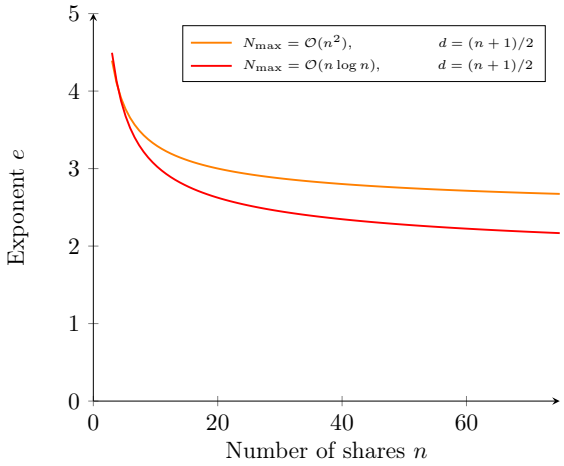

Figure 6.4: Evolution of the complexity exponent $e = \log(N_{\max})/\log(d)$ with respect to the number of shares n. The orange curve matches the instantiation from Section 5.3 using the ISW refresh gadget with quadratic asymptotic complexity and the multiplication gadget from Section 5.4; the red curve matches the new construction (including the new multiplication gadget) with quasi-linear asymptotic complexity ($N_{\max} = \mathcal{O}(n \log n)$).

Figure 6.4 illustrates the evolution of the complexity exponent with respect to the number of shares n, for the best asymptotic construction provided in Chapter 5 with quadratic complexity (as in (5.11)) for an expanding compiler (orange curve), and our new construction with quasi-linear complexity (red curve). While the best construction from Section 5.3 yields a complexity in $\mathcal{O}(|C| \cdot \kappa^e)$ for e close to 3 for reasonable numbers of shares, the new expanding compiler quickly achieves a sub-quadratic complexity in the same settings.

6.5 Conclusion

In this chapter, we have proposed a dynamic expansion strategy for random probing security, which can make the most of different RPE gadgets regarding tolerated leakage probability and asymptotic complexity. We also introduced generic constructions of gadgets achieving RPE for any number of shares n. When the finite base field of the circuit meets the requirement of our multiplication gadget, the asymptotic complexity of the obtained expanding compiler becomes arbitrarily close to linear, which is optimal.

As for concrete instantiations, our small example on the AES demonstrates the benefits of our dynamic approach. Namely, it provides the best tolerated probability (from the best-suited compiler) while optimizing the complexity using higher numbers of shares. Using two compilers with 3 and 5 shares instead of a single one already reduces the complexity by a factor of 10.

Future works could exhibit explicit constructions of matrices with (quasi)constant field size for our multiplication gadget. One could also investigate different RPE multiplication gadget designs with linear number of multiplications for arbitrary fields. Another interesting direction is to optimize the tolerated leakage probability for a set of (possibly inefficient) small gadgets to be used as the starting point of the expansion in our dynamic approach before switching to more (asymptotically) efficient RPE gadgets.

This chapter concludes the theoretical analysis done during the PhD on the random probing expandability and achieving arbitrary levels of security in the random probing model. However, the random probing expansion has proven to be an exciting and promising strategy, and many

future works can be done to push it even further to be of actual practical use in real-life implementations.

Chapter 7

Automatic Verification Tools

As discussed in Chapter 1 (Section 1.7), the manual verification of security properties in the (random) probing models is very error-prone [40]. Hence, automatic tools are regularly built to apply formal verification on software and hardware masked implementations.

Many tools already exist to verify properties in the probing model. For instance, maskVerif [9, 10, 6] includes the verification of most probing-like security notions, including properties in the robust probing model in the presence of glitches. maskVerif allows for efficient verification and good accuracy for circuits of reasonable sizes. Bordes and Karpman [28] also try to improve accuracy with their tool matverif. SILVER [62] also verifies the classical probing-like security properties for hardware implementations with a method based on the analysis of probability distributions. For a more detailed list of existing tools, we refer to Chapter 1, Section 1.7.

It is worth noting that many of the tools discussed above that verify probing-like security properties do not provide *completeness* (*i.e.*, they can falsely deem a set of leaking variables as insecure with respect to the secret). As for tools that achieve completeness, we only count SILVER [62], which suffers from low performance, and matverif [28], which is restricted to specific gadgets only.

On the other hand, before this thesis, no verification tools existed for the random probing model. Since the few past years, the community has made a significant effort to provide designs in this model, see *e.g.* [1, 45, 3, 18, 22, 32]. The random probing expandability (RPE) approach introduced in the previous chapters currently gives the best complexity to achieve arbitrary random probing security with a constant (and quantifiable) leakage probability.

During our work in [18] where we introduced the RPE notion, we also introduced VRAPS, the first tool to verify random probing properties. It was followed by STRAPS [32], which additionally provides a probabilistic mode to boost the performance and checks another random probing security property based on what the authors call a Probe Distribution Table (see Remark 2), but still does not provide a complete verification method since it uses a set of verification rules from maskVerif which is not complete.

In terms of efficiency, VRAPS makes it challenging to verify even small gadgets at reasonable orders, and STRAPS only manages to do it using verification rules from the underlying tool maskVerif [9, 10, 6] (which only verifies probing security properties), and is not complete.

This chapter introduces IronMask (published in [21]), a new versatile verification tool for masking security. IronMask is the first to verify standard simulation-based security notions in the probing model and recent composition and expandability notions in the random probing model. It supports any masking gadgets with linear randomness (*e.g.* addition, copy, and refresh gadgets) as well as quadratic gadgets (*e.g.* multiplication gadgets) that might include non-linear randomness (*e.g.* by refreshing their inputs) while providing complete verification

results for both types of gadgets. We achieve this complete verifiability by introducing a new algebraic characterization for such quadratic gadgets and exhibiting a complete method to determine the sets of input shares necessary and sufficient to perform a perfect simulation of any set of probes. We report various benchmarks which show that IronMask is competitive with state-of-the-art verification tools in the probing model (maskVerif, scVerif, SILVER, matverif). IronMask is also several orders of magnitude faster than VRAPS –the only previous tool verifying random probing composability and expandability– as well as SILVER –the only previous tool providing complete verification for quadratic gadgets with non-linear randomness.

Remark 6. *We do not discuss VRAPS in this manuscript since IronMask is intended to be a more advanced and efficient version of VRAPS. For more details on the original implementation, we refer the readers to the corresponding work [18] and the associated open-source project.*

The organization of this chapter is as follows.

In Section 7.1, we formalize all of the probing and random probing properties from the state of the art from a single building block, a function we call SIS, and show that all of the security properties can be verified using a unique instantiation of that function. In a nutshell, SIS receives as input a set of probes on a gadget (and the description of the corresponding gadget) and performs some operations on the algebraic expressions of the probes in order to determine the exact sets of input shares which are necessary and sufficient to perform a perfect simulation of these probes. While SIS partially uses some properties from the state of the art, it was not exhibited before, and the unification of all the (random) probing-like security notions for this function was not explicitly well-defined.

Then, in Section 7.2, we extend the algebraic characterization introduced in [16, 17] of gadgets with linear randomness (*i.e.* all random values are additive on the wires of the gadget) to more general gadgets with non-linear randomness which perform quadratic operations on input shares mixed with randomness. Our extended characterization notably captures recent gadget designs such as the one from [13] or the multiplication gadget achieving maximal amplification order from Section 5.4. This characterization provides a complete verification method that applies to most (if not all) masking gadgets for standard operations (addition, multiplication, refreshing, etc.). In comparison, the only previously existing complete verification method for such general gadgets would rely on exhausting the truth table of tuples of intermediate variables for the inputs and the randomness, which is highly inefficient.

Finally, in Section 7.3, we present IronMask, a new versatile verification tool for all probing and random probing-like properties in the state of the art. IronMask supports the verification of traditional gadgets with linear randomness and newly formalized gadgets with non-linear randomness, along with a complete verification method for both types of gadgets based on our extended algebraic characterization. IronMask implements several optimizations to make the verification faster. We benchmark the performance of our new tool in Section 7.4 and show that it is competitive with state-of-the-art verification tools in the probing model (maskVerif, scVerif, SILVER, matverif) and is also several orders of magnitude faster than VRAPS, the only previous tool verifying random probing composability and expandability, and SILVER, the only previous tool providing complete verification for gadgets with non-linear randomness. IronMask uses complete methods for the verification, unlike VRAPS and STRAPS which are the only verification tools in the random probing model. IronMask is open-source and publicly available at:

`https://github.com/CryptoExperts/IronMask`

7.1 Characterization of Security Notions for Masking Gadgets

In the following, we shall consider the symbolic expression of a probed wire of a circuit or gadget. The symbolic expression of a circuit's wire is expressed in terms of the circuit inputs (shares) and the generated randoms (outputs of random gates). We denote as before the input sharings of the circuit as $\widehat{x}_1, \ldots, \widehat{x}_\ell$ of n shares each, and we group the random variables used as input to the circuit in a vector $\boldsymbol{r}$ of size ρ.

We also denote in the rest of this chapter a set of probes on a gadget as a vector or a tuple $\boldsymbol{P}$, slightly different from the previous chapters where we consider sets of probes denoted by W. This is because we consider the vector representation of variables to perform linear algebra operations on tuples of probes, such as Gaussian Elimination.

Although many different security notions have been introduced to build proofs of gadgets in the (random) probing model (*e.g.* NI, SNI in the probing model, and RPC, RPE in the random probing model), we show that they can almost all be defined on top of a single building block: the *set of input shares* (SIS) function. The latter takes as input a set of probes on the gadget's internal wires and a set of output shares and returns a set of input shares necessary (and sufficient) to perfectly simulate these internal probes and output shares. We formalize the SIS primitive in Definition 24 before showing how to use it to express state-of-the-art properties. For clarity, we restrain the following definitions to the case of single-output gadgets (the most common case), but the extension to multi-output gadgets is straightforward. We denote SIS_G to be SIS with input gadget G.

Definition 24. *Let G be an (n-share, ℓ-to-1) gadget mapping ℓ input sharings $(\widehat{x}_1, \ldots, \widehat{x}_\ell) \in (\mathbb{K}^n)^\ell$ to an output sharing $\widehat{y} \in \mathbb{K}^n$. Let $\boldsymbol{P}$ be a tuple of probes on G and $O \subseteq [n]$ a set of output shares indices. The function SIS_G maps $\boldsymbol{P}$ and O to the unique smallest sets of input indices $I_1, \ldots, I_\ell$ such that $(\boldsymbol{P}, \widehat{y}|_O)$ can be perfectly simulated from $\widehat{x}_1|_{I_1}, \ldots, \widehat{x}_\ell|_{I_\ell}$.*

Note that for any gadget G, the *smallest* set of input shares returned by SIS_G is *uniquely* defined from the result [16, Lemma 7.5], which demonstrates that if a set of probes can be simulated from different sets of inputs shares, then it can also be simulated by the intersection of these sets.

7.1.1 Probing Security Notions

We now formalize the *probing-like* security notions (*i.e.*, to achieve security and secure composition in the probing model) for any n-share, ℓ-to-1 gadget G (all these notions can be generalized for the case of multiple outputs). Definition 25 reformulates Definition 4 of the NI notion from [10], using the SIS building block.

Definition 25 (t-NI). *A gadget G is t-NI if for any tuple $\boldsymbol{P}$ of t_1 internal probes and any set O of t_2 output share indices such that $t_1 + t_2 \leq t$, the sets $(I_1, \ldots I_\ell) := \mathsf{SIS}_G(\boldsymbol{P}, O)$ satisfy $|I_i| \leq t, \ \forall i \in [\ell]$.*

Other common probing-like properties can be defined in a similar way by changing the condition on the sets $(I_1, \ldots, I_\ell)$ in the output of the SIS_G primitive. We list these conditions in Table 7.1 for some of the most common probing-like properties with respect to t and O. While most of the properties are interesting in the context of composing secure gadgets to achieve global security, directly verifying the probing security of a complete implementation is useful in some cases, such as analyzing a complete 2-share AES implementation. To represent this case, we denote PS* the SIS-based probing security definition. PS* is actually equivalent to the case of $(n-1)$-NI with $O = \emptyset$ and $t = n - 1$. We list this property in Table 7.1 as well.

Table 7.1: Probing-like security notions from the basic primitive SIS_G. t_1 indicates the number of probes on input and intermediate variables, while t_2 indicates the number of probes on output shares, with $t = t_1 + t_2$.

Notion	Condition
t-NI [9]	$\lvert I_i\rvert \leq t,\ \forall i \in [\ell]$
t-SNI [10]	$\lvert I_i\rvert \leq t_1,\ \forall i \in [\ell]$
t-TNI [16]	$\lvert I_i\rvert \leq t_1 + t_2,\ \forall i \in [\ell]$
(t, f)-NI [8]	$\lvert I_i\rvert \leq f(t_1, t_2),\ \forall i \in [\ell]$
t-PINI [33]	$\lvert(\cup_i I_i) \setminus O\rvert \leq t_1$
PS*	$\lvert I_i\rvert \leq n-1,\ \forall i \in [\ell]$

7.1.2 Random Probing Security Notions.

We rely on the LeakingWires procedure formalized in Section 3.1, Chapter 3 to formalize security notions in the random probing model. We recall that LeakingWires outputs a tuple of probes $\boldsymbol{P}$ on the gadget G such that each wire of G is added to $\boldsymbol{P}$ independently with probability p. Definition 26 reformulates Definition 8 of the random probing composability (RPC) notion from Section 3.2, based on the SIS primitive.

Definition 26. *Let $p, \varepsilon \in [0,1]$ and $n, t \in [0, n]$. Let G be a n-share ℓ-to-1 gadget and let $\boldsymbol{P}$ be the random vector defined as $\boldsymbol{P} = \mathsf{LeakingWires}(G, p)$. Then G is (t, p, ε)-RPC if for every $O \subseteq [n]$ with $\lvert O\rvert = t$, the sets $(I_1, \ldots, I_\ell) = \mathsf{SIS}_G(\boldsymbol{P}, O)$ satisfy $\Pr[(\lvert I_1\rvert > t) \vee \ldots \vee (\lvert I_\ell\rvert > t)] \leq \varepsilon$, where the probability is taken over all tuples of probes $\boldsymbol{P}$ obtained through $\mathsf{LeakingWires}(G, p)$.*

Recall that the event $\big((\lvert I_1\rvert > t) \vee \ldots \vee (\lvert I_\ell\rvert > t)\big)$ is called a *failure event* (failure of a perfect simulation) and ε is the *failure probability* or the probability of a failure event to occur.

The *random probing expandability* (RPE) notion introduced and analyzed in Chapter 4, Chapter 5 and Chapter 6 can also be defined similarly. Like in the previous chapters, we restrict its definition to 2-input circuits for clarity but recall that the extension is straightforward. We have that G is (t, p, ε)-RPE1 (resp. RPE2) if for every $O \subseteq [n]$ with $\lvert O\rvert = t$ (resp. if there exists $O \subseteq [n]$ with $\lvert O\rvert = n-1$), the sets $(I_1, I_2) = \mathsf{SIS}_G(\boldsymbol{P}, O)$ satisfy

$$\big(\Pr[\lvert I_1\rvert > t] \leq \varepsilon\big) \ \wedge\ \big(\Pr[\lvert I_2\rvert > t] \leq \varepsilon\big) \wedge\ \big(\Pr[(\lvert I_1\rvert > t) \wedge (\lvert I_2\rvert > t)] \leq \varepsilon^2\big)\ .$$

Table 7.2 summarizes the three random probing notions. As in the probing security case earlier, it can be helpful to verify the random probing security of a complete implementation directly. To represent this case, we denote (p, ε)-RPS* the SIS-based definition of random probing security. This notion is similar to the RPC definition, except that we do not consider probes on the outputs, *i.e.* $O = \emptyset$, and a failure occurs when all the shares (of one input) are necessary to simulate the probes perfectly, *i.e.* the failure event is

$$\Pr[(\lvert I_1\rvert = n) \vee \ldots \vee (\lvert I_\ell\rvert = n)] \leq \varepsilon\ .$$

In the previous chapters, we introduced a method to verify random probing properties by computing the failure probability ε as a function $f(p)$ of the leakage probability p. We briefly recall the principle hereafter using SIS. For more details, we refer to Section 3.3 and Section 4.3. For (p, ε)-RPS* of an n-share gadget of s wires for example, $\varepsilon = f(p)$ is computed as

$$f(p) = \sum_{\substack{\boldsymbol{P} \text{ s.t. } (I_1,\ldots,I_\ell)=\mathsf{SIS}(\boldsymbol{P},\emptyset) \\ \lvert I_1\rvert=n \ \vee\ldots\vee\ \lvert I_\ell\rvert=n}} p^{\lvert\boldsymbol{P}\rvert}(1-p)^{s-\lvert\boldsymbol{P}\rvert}\ . \tag{7.1}$$

Table 7.2: Random probing-like security notions from the basic primitive SIS_G.

Notion	Output O	Condition(s)
(t,p,ε)-RPC [18]	$\forall\, O, \lvert O\rvert = t$	$\Pr[(\lvert I_1\rvert > t) \vee \ldots \vee (\lvert I_\ell\rvert > t)] \leq \varepsilon$
(t,p,ε)-RPE1 [18]	$\forall\, O, \lvert O\rvert = t$	$\big(\forall i,\ \Pr[(\lvert I_i\rvert > t)] \leq \varepsilon\big) \wedge \big(\Pr[(\lvert I_1\rvert > t) \wedge (\lvert I_2\rvert > t)] \leq \varepsilon^2\big)$
(t,p,ε)-RPE2 [18]	$\exists O, \lvert O\rvert = n-1$	$\big(\forall i,\ \Pr[(\lvert I_i\rvert > t)] \leq \varepsilon\big) \wedge \big(\Pr[(\lvert I_1\rvert > t) \wedge (\lvert I_2\rvert > t)] \leq \varepsilon^2\big)$
(p,ε)-RPS*	$O = \emptyset$	$\Pr[(\lvert I_1\rvert = n) \vee \ldots \vee (\lvert I_\ell\rvert = n)] \leq \varepsilon$

In the above equation, we consider that each tuple of probes $\boldsymbol{P}$ on a gadget can exactly leak with probability $p^{|\boldsymbol{P}|}(1-p)^{s-|\boldsymbol{P}|}$ since each of the wires in $\boldsymbol{P}$ is added independently with probability p, and each of the remaining wires does not leak with probability $1-p$. Then, out of all such possible tuples of wires, $f(p)$ represents the sum over the probabilities of obtaining tuples of probes only for which we get a failure event using SIS (the failure event being $(|I_1| = n \ \vee \ldots \vee \ |I_\ell| = n)$ in this context). For a gadget with a total of s wires, computing $f(p)$ then amounts to computing

$$f(p) = \sum_{i=1}^{s} c_i \ p^i (1-p)^{s-i} \ , \tag{7.2}$$

where we group the probabilities with respect to the size of the tuples of probes. In other words, c_i is the number of tuples of i wires, for which we obtain a failure event using SIS. For instance, if there are exactly 2 tuples of probes $\boldsymbol{P_1}, \boldsymbol{P_2}$ for which we get a failure event and such that $|\boldsymbol{P_1}| = |\boldsymbol{P_2}| = 3$, then we get $c_3 = 2$ in equation (7.2). For other random probing properties, the computation is similar, considering the correct failure event with the correct t and the condition on the output set of shares O, which is not empty anymore (*e.g.* RPC, RPE).

The recent Probe Distribution Table (PDT) of Cassiers *et al.* [32] for tighter random probing composition (see Remark 2) can also be expressed in terms of our basic function SIS_G. The PDT is a two-dimensional table indexed by all possible sets of input indices $\boldsymbol{I} = (I_1, \ldots, I_\ell)$ where $I_i \subseteq [n]$ and by all possible sets of output indices $O \subseteq [n]$, defined as

$$\mathrm{PDT}[\boldsymbol{I}\,][O] := \sum_{\boldsymbol{P}\text{ s.t. }\boldsymbol{I}=\mathsf{SIS}(\boldsymbol{P},O)} p^{|\boldsymbol{P}|}(1-p)^{s-|\boldsymbol{P}|} \tag{7.3}$$

where s is the number of wires in G. In other words, each entry in the PDT is a different function $f(p)$ as in equations (7.1), (7.2). Computing the PDT amounts to considering each possible tuple of probes $\boldsymbol{P}$ on the gadget and computing $\mathsf{SIS}_G(\boldsymbol{P}, O) = \boldsymbol{I} = (I_1, \ldots, I_\ell)$ for each possible set of output shares indices $O \subseteq [n]$. Then, update the corresponding function in the PDT indexed by $\boldsymbol{I}$ and O as $\mathrm{PDT}[\boldsymbol{I}\,][O] = \mathrm{PDT}[\boldsymbol{I}\,][O] \ + \ p^{|\boldsymbol{P}|}(1-p)^{s-|\boldsymbol{P}|}$. When exploring all the possible sets of internal probes, $\boldsymbol{P}$, and all the sets of output indices O, the output of SIS_G shall serve as a basis to compute the expected distributions.

We showed how standard probing and random probing security notions can be expressed using the SIS_G function. In the next section, we focus on the algebraic characterization of masking gadgets and the concrete evaluation of the SIS_G function.

7.2 Algebraic Characterization of Masking Gadgets

In this section, we recall and extend the algebraic characterization of masking gadgets and the subsequent security results. Previous works [16, 17] considered gadgets with linear randomness, *i.e.* all random values are additive on the wires of the gadget. We refer to these gadgets

as LR-gadgets (see, for instance, ISW multiplication or refresh gadget recalled in Chapter 2). In this work, we extend the characterization to gadgets with non-linear randomness, *i.e.* gadgets performing non-linear operations on input shares mixed with randomness. We denote these gadgets as NLR-gadgets. Our extended characterization notably captures recent gadget designs; see *e.g.* [13] or the multiplication gadget from Section 5.4. We also show how to verify the security of masking gadgets using this algebraic characterization by a concrete evaluation of the SIS primitive, which will be the core primitive of IronMask.

7.2.1 Characterization of Gadgets with Linear Randomness

We call an LR-gadget any ℓ-to-m gadget $G : \widehat{\boldsymbol{x}} = (\widehat{x}_1, \ldots, \widehat{x}_\ell) \mapsto \widehat{\boldsymbol{y}} = (\widehat{y}_1, \ldots, \widehat{y}_m)$ with the output of the form (recall that for the description of SIS, we consider $m = 1$ but we can generalize it to any number of outputs):

$$\widehat{\boldsymbol{y}} := R\big(F(\widehat{\boldsymbol{x}}),\, \boldsymbol{r}\big) \ ,$$

where F is any arithmetic circuit, R is a linear arithmetic circuit (*i.e.* computing a linear function) and $\boldsymbol{r}$ is a vector of internal randomness uniformly drawn from $\mathbb{K}^\rho$. Formally, each coordinate of $\boldsymbol{r}$ is the output of a randomness gate of G, and F and R are composed solely of operation gates. Note that this characterization is more general than the one from [16, 17], which only considers quadratic circuits for F. We show hereafter that we can still obtain an efficient and complete evaluation of SIS for those gadgets, yielding an efficient verification of the considered security notions.

By definition, any probe on an LR-gadget can be written as

$$p = f_p(\widehat{\boldsymbol{x}}) + \boldsymbol{r}^\mathsf{T} \cdot \boldsymbol{s}_p \tag{7.4}$$

for some arithmetic function $f_p : (\mathbb{K}^n)^\ell \to \mathbb{K}$ and some constant vector $\boldsymbol{s}_p \in \mathbb{K}^\rho$.

Given a tuple of probes $\boldsymbol{P} = (p_1, \ldots, p_d)$ on the gadget G, we are interested in determining the sets of input shares necessary for a perfect simulation of all probes in $\boldsymbol{P}$. In particular, if $\boldsymbol{P}$ can be simulated with at most $n-1$ shares of each input sharing, then we know that $\boldsymbol{P}$ is independent of the secret inputs. Belaïd *et al.* [16] showed how to use a Gaussian elimination technique in order to determine the simulatability of a tuple of probes for gadgets with linear randomness over the binary field. This technique was later extended to any finite field in [17]. We base our tool's verification procedure for LR-gadgets on this technique.

We start by stating the result with Gaussian elimination from [16, 17] in a different, more convenient formulation for our purposes. For this, we first define a simple function shares(.), which takes as input a tuple of symbolic expressions $(e_1, \ldots, e_d)$ of the input shares, *i.e.* $e_i = f_{e_i}(\widehat{\boldsymbol{x}})$ for some algebraic function f_{e_i}, and which outputs the (smallest) sets of indices $\boldsymbol{I} = (I_1, \ldots, I_\ell)$ such that $(e_1, \ldots, e_d)$ functionally depend on $\widehat{\boldsymbol{x}}|_{\boldsymbol{I}}$. Notice that evaluating shares(.) consists in extracting the indices of the input shares that are contained in the symbolic expressions $(e_1, \ldots, e_d)$. We stress that the input shares $\widehat{\boldsymbol{x}}|_{\boldsymbol{I}}$ where $\boldsymbol{I} = (I_1, \ldots, I_\ell) :=$ shares$(e_1, \ldots, e_d)$ are necessary and sufficient for a perfect simulation of $(e_1, \ldots, e_d)$. Note that shares(.) is executed on the tuple of expressions, the output tuple of the Gaussian elimination technique. In fact, after executing the Gaussian elimination, we are guaranteed that the remaining expressions cannot be simplified any further in the given field $\mathbb{K}$, and they are solely formed of operations between input shares (they do not include any random variables). In this case, to perfectly simulate the resulting tuple (which is equivalent to perfectly simulating the tuple given before Gaussian elimination), there is no choice but to have access to all of the input shares that are involved in the remaining expressions, which is why shares(.) extracts the indices of these input shares.

Lemma 22. *Let G be an n-share gadget. Let $\boldsymbol{P} = (p_1, \ldots, p_d)$ be a tuple of probes on G. Let $S \in \mathbb{K}^{d\times\rho}$ be the matrix such that*

$$S^{\mathsf{T}} = (\boldsymbol{s}_{p_1} \mid \boldsymbol{s}_{p_2} \mid \cdots \mid \boldsymbol{s}_{p_d})$$

(i.e. each $\boldsymbol{s}_{p_i}$ is a row vector of S) and let S' be the row reduced form of the matrix S such that S' is of the form

$$S' = \begin{pmatrix} 0_{m,d-m} & 0_{m,R-d+m} \\ I_{d-m} & S'' \end{pmatrix}$$

up to some permutations on the rows with $S' = N \cdot S$ where N is an invertible matrix in $\mathbb{K}^{d\times d}$. Let $\boldsymbol{P'}$ be defined as

$$\boldsymbol{P'} = N \cdot \boldsymbol{P} = (p'_1, \ldots, p'_m, p'_{m+1}, \ldots, p'_d) \ .$$

Then, the sets of input shares necessary to simulate the probes in $\boldsymbol{P}$ are $\mathsf{shares}(p'_1, \ldots, p'_m)$.

Proof. The proof of the result follows the proof of Theorem 3.1 from [16] and Theorem 3.2 of [17]. It is shown in the latter that we can perfectly simulate the probes in $\boldsymbol{P}$ by perfectly simulating all probes in $\boldsymbol{P'}$ since the matrix N is invertible and we can obtain $\boldsymbol{P}$ from $N^{-1} \cdot \boldsymbol{P'}$. Then, to perfectly simulate probes in the tuple $\boldsymbol{P'}$, we observe from S' that each algebraic expression in the tuple $(p'_{m+1}, \ldots, p'_d)$ contains a random value that does not appear in any other algebraic expression in $\boldsymbol{P'}$. We can thus perfectly simulate $(p'_{m+1}, \ldots, p'_d)$ by generating $d - m$ uniform random values without the need for any input share. The remaining algebraic expressions $(p'_1, \ldots, p'_m)$ contain no random values and are all of the form $p'_i = f_{p'_i}(\widehat{\boldsymbol{x}})$. Hence, to perfectly simulate each of them, we need (and only need) the input shares which are involved in each $f_{p'_i}(\widehat{\boldsymbol{x}})$, namely the input shares indexed by $\boldsymbol{I} := \mathsf{shares}(p'_1, \ldots, p'_m)$. Using the input shares $\widehat{\boldsymbol{x}}|_{\boldsymbol{I}}$, we can perfectly simulate $(p'_1, \ldots, p'_m)$ and thus perfectly simulate all algebraic expressions in $\boldsymbol{P'}$, from which we get a perfect simulation of the probes in $\boldsymbol{P}$. □

Lemma 22 provides a way to evaluate the function SIS in the case of LR-gadgets. Note that the set of probes $\boldsymbol{P}$ in the lemma must be defined as the union of $\boldsymbol{P}$ and $\widehat{\boldsymbol{y}}|_O$ in an evaluation of $\mathsf{SIS}_G(\boldsymbol{P}, O)$ (while used to define security notions, SIS is based on two arguments to differentiate probes on internal wires and probes on output shares whereas this distinction is not used in the evaluation process of Lemma 22). According to the above lemma, an evaluation of SIS consists of a row reduction on the matrix of the random dependencies S, after which the function $\mathsf{shares}(.)$ is used on the obtained expressions without random values (*i.e.* $(p'_1, \ldots, p'_m)$ in the lemma). The output of SIS is then the output of $\mathsf{shares}(.)$, which is the set of input shares necessary for a perfect simulation of all the probes. Section 7.3 shows how this technique is efficiently implemented in our verification tool.

7.2.2 Characterization of Gadgets with Non-Linear Randomness

In this section, we extend the algebraic characterization for LR-gadgets of Section 7.2.1 to NLR-gadgets, *i.e.* gadgets performing non-linear operations on input shares mixed with randomness. An NLR-gadgets is an ℓ-to-m gadget $G : \widehat{\boldsymbol{x}} \mapsto \widehat{\boldsymbol{y}}$ with the output of the form:

$$\widehat{\boldsymbol{y}} := R_{\ell+1}\big(F(R_1(\widehat{x}_1, \boldsymbol{r}_1), \ldots, R_\ell(\widehat{x}_\ell, \boldsymbol{r}_\ell)), \boldsymbol{r}_{\ell+1}\big)$$

where F is any arithmetic circuit, the R_i are linear arithmetic circuits and the $\boldsymbol{r}_i$ are vectors of random values uniformly drawn from $\mathbb{K}^{\rho_i}$. We further assume that F computes a homogeneous multi-linear form, namely $F(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_\ell)$ is a sum of degree-ℓ monomials, each of which being a product containing exactly one coordinate from each $\boldsymbol{z}_i$.

For clarity, we describe the verification method for the case of 2-input gadgets; the extension to ℓ inputs is straightforward. We thus present NLR-gadgets as 2-to-m gadgets $G : (\widehat{x}_1, \widehat{x}_2) \mapsto (\widehat{y}_1, \ldots, \widehat{y}_m)$ with the output of the form:

$$(\widehat{y}_1, \ldots, \widehat{y}_m) := R_3\big(F(R_1(\widehat{x}_1, \boldsymbol{r}_1), R_2(\widehat{x}_2, \boldsymbol{r}_2)), \boldsymbol{r}_3\big)$$

This characterization notably covers most (if not the totality of) multiplication gadgets. It covers, in particular, multiplication gadgets which first start by refreshing one of (resp. each of) their inputs before performing share-wise products that are finally recombined into the output sharing (with additional randomness). Such multiplication gadgets have been recently described in [54, 18, 33] and Section 5.4.

Any probe on such an NLR-gadget is either a probe on the inner circuits $R_i(\widehat{x}_i, \boldsymbol{r}_i)$ and is of the form:

$$p = \widehat{x}_i^{\mathsf{T}} \cdot \boldsymbol{w}_p + \boldsymbol{r}_i^{\mathsf{T}} \cdot \boldsymbol{s}_p \tag{7.5}$$

for $i \in \{1, 2\}$ (since the R_i are linear arithmetic circuits) with $\boldsymbol{w}_p \in \mathbb{K}^n$, $\boldsymbol{s}_p \in \mathbb{K}^{\rho_i}$, or is a probe on the outer circuit and is of the form:

$$p = f_p(\boldsymbol{z_1}, \boldsymbol{z_2}) + \boldsymbol{r}_3^{\mathsf{T}} \cdot \boldsymbol{s}_p \tag{7.6}$$

where $\boldsymbol{z_i} := R_i(\widehat{x}_i, \boldsymbol{r}_i)$ for $i \in \{1, 2\}$ with $\boldsymbol{s}_p \in \mathbb{K}^{\rho_3}$, and for some arithmetic function $f_p : (\mathbb{K}^n)^2 \to \mathbb{K}$. We show hereafter that we can still obtain an efficient and complete evaluation of SIS for those gadgets, yielding an efficient verification of the considered security notions.

The verification technique for NLR-gadgets essentially consists of several iterations of the verification process for LR-gadgets used in Lemma 22. The steps of the technique are as follows. Suppose that we have a tuple of probes $\boldsymbol{P} = (p_1, \ldots, p_k, p_{k+1}, \ldots, p_d)$ where $(p_1, \ldots, p_k)$ are all of the form (7.6) while $(p_{k+1}, \ldots, p_d)$ are all of the form (7.5).

1. First, we apply the Gaussian elimination technique of Section 7.2.1 on the probes $(p_1, \ldots, p_k)$ with respect to the vector of randoms $\boldsymbol{r}_3$. This is possible since all of these probes respect the form (7.4) w.r.t. inputs $(\boldsymbol{z_1}, \boldsymbol{z_2})$ and randomness $\boldsymbol{r}_3$. Specifically, let $S_3^{\mathsf{T}} := (\boldsymbol{s}_{p_1} \mid \boldsymbol{s}_{p_2} \mid \cdots \mid \boldsymbol{s}_{p_k})$, with $\boldsymbol{s}_{p_i}$ defined from (7.6), and let N_3 be the permutation matrix such that $S'_3 = N_3 \cdot S_3$ is the row reduced form of S_3 (see Lemma 22). From this, we get a new derived tuple $\boldsymbol{P'} := N_3 \cdot \boldsymbol{P} = (p'_1, \ldots, p'_m, p'_{m+1}, \ldots, p'_k)$ and we know from Lemma 22 that each of the expression in $(p'_{m+1}, \ldots, p'_k)$ can be perfectly simulated by simply generating $k - m$ uniform random values. Thus, we end up with $(p'_1, \ldots, p'_m)$, which we need to simulate perfectly, and where each of the p'_i is of the form $f_{p'_i}(\boldsymbol{z_1}, \boldsymbol{z_2})$ with no random values from $\boldsymbol{r}_3$, along with the remaining probes $(p_{k+1}, \ldots, p_d)$. We then construct the new tuple to simulate $\boldsymbol{P''} = (p'_1, \ldots, p'_m, p_{k+1}, \ldots, p_d)$, which we rewrite as $\boldsymbol{P''} = (p''_1, \ldots, p''_{m+d-k})$. Thus, to perfectly simulate the tuple of probes $\boldsymbol{P}$, we need to perfectly simulate the probes in $\boldsymbol{P''}$.

   We stress at this stage that each algebraic expression p''_i in $\boldsymbol{P''}$ is either of the form $p''_i = f_{p''_i}(\boldsymbol{z_1}, \boldsymbol{z_2})$ with $f_{p''_i}$ of a homogeneous bilinear form (this is of the first m coordinates resulting from Gaussian elimination) or of the form (7.5) (*i.e.* the probes on R_1 or R_2 that are not affected by the previous Gaussian elimination since they do not contain any randoms from $\boldsymbol{r}_3$).

2. For each p''_i in $\boldsymbol{P''}$ of the form $p''_i = f_{p''_i}(\boldsymbol{z_1}, \boldsymbol{z_2})$, we factor its algebraic expression with respect to the vector of values $(\widehat{x}_1 || \boldsymbol{r}_1)$. In other terms, we rewrite each p''_i as

$$p''_i = (\widehat{x}_1 || \boldsymbol{r}_1)^{\mathsf{T}} \cdot \boldsymbol{h}_{p''_i} \tag{7.7}$$

where $\boldsymbol{h}_{p_i''}$ is a tuple of $n+\rho_1$ algebraic expressions of the form (7.5) w.r.t. $(\widehat{x}_2, \boldsymbol{r}_2)$. We then construct a new tuple $\boldsymbol{P}_2 := (\boldsymbol{h}_{p_1''} \parallel \cdots \parallel \boldsymbol{h}_{p_m''})$ to which we append all the expressions p_i'' of the form (7.5) w.r.t. $(\widehat{x}_2, \boldsymbol{r}_2)$ (*i.e.* probes from R_2).

3. We perform the same procedure as in the last step but this time factoring each p_i'' in $\boldsymbol{P''}$ of the form $p_i'' = f_{p_i''}(\boldsymbol{z_1}, \boldsymbol{z_2})$ with respect to $(\widehat{x}_2||\boldsymbol{r}_2)$, rewriting each p_i'' as

$$p_i'' = (\widehat{x}_2||\boldsymbol{r}_2)^T \cdot \boldsymbol{g}_{p_i''} \tag{7.8}$$

From those expressions we define a new tuple $\boldsymbol{P}_1 := (\boldsymbol{g}_{p_1''} \parallel \cdots \parallel \boldsymbol{g}_{p_m''})$ where the coordinates of the $\boldsymbol{g}_{p_i''}$'s are of the form (7.5) w.r.t. $(\widehat{x}_1, \boldsymbol{r}_1)$, to which we append all the expressions p_i'' of the form (7.5) w.r.t. $(\widehat{x}_1, \boldsymbol{r}_1)$ (*i.e.* probes from R_1).

4. Recall from the first step that perfectly simulating $\boldsymbol{P}$ amounts to perfectly simulating $\boldsymbol{P''}$. We will prove later in this section that the input shares from $\widehat{x}_1$ and $\widehat{x}_2$ that are necessary and sufficient to produce a perfect simulation of $\boldsymbol{P''}$ are the same as the ones for a perfect simulation of $\boldsymbol{P}_1, \boldsymbol{P}_2$ constructed in the last two steps. Observe that all probes in $\boldsymbol{P}_1, \boldsymbol{P}_2$ respect the form (7.5), which is a special case of (7.4). Hence, we separately apply the Gaussian elimination technique of Lemma 22 on $\boldsymbol{P}_1$ with respect to $(\widehat{x}_1, \boldsymbol{r}_1)$, and on $\boldsymbol{P}_2$ with respect to $(\widehat{x}_2, \boldsymbol{r}_2)$. This provides us with the sets of input shares I_1 on $\widehat{x}_1$ and I_2 on $\widehat{x}_2$ that are respectively necessary and sufficient to produce a perfect simulation of the expressions in $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$. These sets are, therefore, output as the necessary and sufficient sets of input shares for a perfect simulation of $\boldsymbol{P}$.

We state in the following lemma that the above verification method is complete.

Lemma 23. *Let G be a 2-input n-share NLR-gadget. Let $\boldsymbol{P} = (p_1, \ldots, p_d)$ be a tuple of probes on G. Let $\boldsymbol{P}_1, \boldsymbol{P}_2$ be the tuples of linear expressions w.r.t. $(\widehat{x}_1, \boldsymbol{r}_1)$ and $(\widehat{x}_2, \boldsymbol{r}_2)$ obtained by applying the above method. The sets I_1, I_2 obtained by applying the method of Lemma 22 on $\boldsymbol{P}_1$ with respect to $(\widehat{x}_1, \boldsymbol{r}_1)$ and separately on $\boldsymbol{P}_2$ with respect to $(\widehat{x}_2, \boldsymbol{r}_2)$ are the sets of input shares necessary and sufficient to simulate $\boldsymbol{P}$.*

Proof. The proof follows the different steps of the method described above. All the statements hold from Lemma 22 except that the sets I_1, I_2 are necessary and sufficient for a perfect simulation of $\boldsymbol{P''}$. We prove this statement hereafter.

For any random distributions $\mathcal{D}_{\widehat{x}_1}$ and $\mathcal{D}_{\widehat{x}_2}$ over $\mathbb{K}^n$, we denote $\mathcal{D}_{\boldsymbol{P}}$ the distribution induced on $\boldsymbol{P}$ by picking $\widehat{x}_1 \leftarrow \mathcal{D}_{\widehat{x}_1}$, $\widehat{x}_2 \leftarrow \mathcal{D}_{\widehat{x}_2}$, $\boldsymbol{r}_1 \leftarrow \mathbb{K}^{\rho_1}$, $\boldsymbol{r}_2 \leftarrow \mathbb{K}^{\rho_2}$. Then I_1 and I_2 are the minimal sets such that for any distributions $\mathcal{D}_{\widehat{x}_1}$ and $\mathcal{D}_{\widehat{x}_2}$, there exists a probabilistic algorithm $\mathcal{S}$ (the simulator) which given $\widehat{x}_1|_{I_1}$ and $\widehat{x}_1|_{I_2}$ outputs a tuple $\boldsymbol{P}$ which is i.i.d. as $\mathcal{D}_{\boldsymbol{P}}$ w.r.t. the random draw $\widehat{x}_1 \leftarrow \mathcal{D}_{\widehat{x}_1}$, $\widehat{x}_2 \leftarrow \mathcal{D}_{\widehat{x}_2}$ and the random coins of $\mathcal{S}$.

Direction 1: The sets (I_1, I_2) are necessary to simulate $\boldsymbol{P''}$. Here we need to perfectly simulate the distribution of $\boldsymbol{P''}$ given the random samplings $\widehat{x}_1 \leftarrow \mathcal{D}_{\widehat{x}_1}$, $\widehat{x}_2 \leftarrow \mathcal{D}_{\widehat{x}_2}$, $\boldsymbol{r}_1 \leftarrow \mathbb{K}^{\rho_1}$, $\boldsymbol{r}_2 \leftarrow \mathbb{K}^{\rho_2}$ for any distributions $\mathcal{D}_{\widehat{x}_1}$ and $\mathcal{D}_{\widehat{x}_2}$ over $\mathbb{K}^n$. Let us consider the uniform distribution for $\mathcal{D}_{\widehat{x}_1}$, then the m first coordinates of $\boldsymbol{P''}$ (*i.e.* the expressions of the form (7.7)) can be written as $\boldsymbol{P''}|_{[m]} = (\boldsymbol{u} \cdot \boldsymbol{h}_{p_1''}, \ldots, \boldsymbol{u} \cdot \boldsymbol{h}_{p_m''})$, where $\boldsymbol{u}$ is a vector uniformly sampled on $\mathbb{K}^{n+\rho_1}$. Recall that the $\boldsymbol{h}_{p_i''}$'s corrdinates are expressions of the form (7.5) w.r.t. $(\widehat{x}_2, \boldsymbol{r}_2)$.

Given the values taken by the $\boldsymbol{h}_{p_i''}$'s we can have different distributions for $\boldsymbol{P''}|_{[m]}$. A particular case is the distribution "$\boldsymbol{P''}|_{[m]} = \boldsymbol{0}$ with probability 1" which appears if and only if $\boldsymbol{h}_{p_1''} = \cdots = \boldsymbol{h}_{p_m''} = \boldsymbol{0}$. In order to evaluate the probability of outputting $\boldsymbol{P''}|_{[m]} = \boldsymbol{0}$ (which must be exact for a perfect simulation), the simulator must hence evaluate the probability that $\boldsymbol{h}_{p_1''} = \cdots = \boldsymbol{h}_{p_m''} = \boldsymbol{0}$ occurs, which must be further conditioned on the remaining expressions

p''_i of the form (7.5) w.r.t. $(\widehat{x}_2, \boldsymbol{r}_2)$ (*i.e.* the probes on R_2). This precisely means solving the linear system obtained from the expressions in $\boldsymbol{P}_2$ which can be done by Gaussian elimination w.r.t. the $\boldsymbol{r}_2$ variables (just as what is actually performed by step 4 of the verification method). The resulting equations without $\boldsymbol{r}_2$ variables imply some linear constraints on some of the shares from $\widehat{x}_2$. These shares must then be known by the simulator in order to decide if the system has a solution (and to evaluate the probability to get $\boldsymbol{h}_{p''_1} = \cdots = \boldsymbol{h}_{p''_m} = \boldsymbol{0}$). Moreover, these shares are by construction the shares of indices in I_2.

The exact same proof apply to I_1 by taking the uniform distribution for $\mathcal{D}_{\widehat{x}_2}$ and considering the expressions of the form (7.8) (together with the the probes on R_1).

Direction 2: The sets (I_1, I_2) are sufficient to simulate $\boldsymbol{P''}$. Suppose that we can perfectly simulate the tuples of algebraic expressions $\boldsymbol{P}_1, \boldsymbol{P}_2$ using sets of input shares I_1 on input sharing $\widehat{x}_1$ and I_2 on $\widehat{x}_2$ respectively as described in step 4 above. Let two new sets of input shares $\tilde{I}_1 = [n]$ on $\widehat{x_1}$ and $\tilde{I}_2 = [n]$ on $\widehat{x_2}$.

Observe first that we can perfectly simulate $\boldsymbol{P''}$ using the sets of input shares $\tilde{I}_1$ and I_2. In fact, in the algebraic expression of each probe p''_i in $\boldsymbol{P''}$ of the form (7.7), the coordinates of the $\boldsymbol{h}^{(p''_i)}$'s can all be perfectly simulated using I_2 since by hypothesis we can perfectly simulate $\boldsymbol{P_2}$. Also, the randoms in $\boldsymbol{r_1}$ are perfectly simulated by generating uniform random values, and all shares of input $\widehat{x_1}$ are simulated using the full input sharing in $\tilde{I}_1 = [n]$. Since we can perfectly simulate each term in the expression of p''_i, then we can perfectly simulate the expression p''_i and hence we can perfectly simulate $\boldsymbol{P''}$ using $\tilde{I}_1$ and I_2. Similarly, we can perfectly simulate $\boldsymbol{P''}$ using the sets of input shares I_1 and $\tilde{I}_2 = [n]$ by observing the expressions of p''_i of the form (7.8).

Thanks to [16, Lemma 7.5] (which demonstrates that if a set of probes can be simulated from different sets of inputs shares, then it can also be simulated by the intersection of these sets), we get that $\boldsymbol{P''}$ can be perfectly simulated using the sets of input shares $\tilde{I}_1 \cap I_1 = I_1$ and $\tilde{I}_2 \cap I_2 = I_2$, which proves that by perfectly simulating the tuples $\boldsymbol{P}_1, \boldsymbol{P}_2$ using I_1, I_2, we can perfectly simulate $\boldsymbol{P''}$ using I_1, I_2. This concludes the proof for this direction. We hence conclude the proof of the lemma. □

The verification method introduced above describes the procedure of the function SIS in the case of NLR-gadgets to determine the simulatability of a set of probes on such gadgets. Section 7.3 shows how this technique is implemented in IronMask. We now present a concrete example of SIS execution on a set of probes on an NLR-gadget.

Example. Let us consider the following 2-share multiplication gadget (with inputs a and b, and output e) while taking $\mathbb{K} = \mathbb{F}_2$:

$$\begin{aligned} c_1 &= a_1 + r_a, & c_2 &= a_2 + r_a \\ d_1 &= b_1 + r_b, & d_2 &= b_2 + r_b \\ e_1 &= (c_1 * d_1 + r) + c_1 * d_2 \\ e_2 &= (c_2 * d_1 + r) + c_2 * d_2 \end{aligned}$$

The above gadget is an example of NLR-gadgets, and uses 3 random values: r_a is used to refresh the input sharing $\widehat{a}$, r_b is used to refresh the input sharing $\widehat{b}$, and r is used during the compression of the products into the output sharing $\widehat{e}$. The non-linear random values are r_a and r_b with respect to $\widehat{e}$.

Suppose that we would like to verify one of the security properties defined in Section 7.1 using SIS. To do this, we need to determine for each set of probes (formed of intermediate

values or output shares) on the gadget the exact sets of input shares necessary and sufficient for a perfect simulation of all the probes in the set. Let us consider, for instance, the following set of 2 probes on the gadget:

$$P = (p_1 = c_1 * d_1 + r, \quad p_2 = c_2 * d_1 + r)$$

We need to determine the sets of input shares of $\widehat{a}$ and $\widehat{b}$ necessary to simulate probes in P perfectly. SIS will be executed in four steps, as described earlier.

1. Get rid of the random values that are additive in the compression step (which are not additive to the shares of $\widehat{a}$ and $\widehat{b}$). In this case, it is the unique random value r. Using the Gaussian elimination technique, we construct a new tuple :

   $$P' = (p_1 + p_2 = c_1 * d_1 + c_2 * d_1, \quad p_2 = c_2 * d_1 + r)$$

   Since r only appears in p_2, a uniform random value can perfectly simulate this probe. Next we need to consider the simulation of the new tuple

   $$P'' = (c_1 * d_1 + c_2 * d_1) = ((a_1 + r_a) * (b_1 + r_b) + (a_2 + r_a) * (b_1 + r_b)) \ .$$

2. Factor the expressions in P'' with respect to the elementary variables of shares of $\widehat{a}$ and random values which are additive to the shares of $\widehat{a}$ and the constant term 1, in this case the variables $(a_1, a_2, r_a, 1)$. Since there is a single expression in P'', we can rewrite it as:

   $$P'' = (a_1 * (b_1 + r_b) + a_2 * (b_1 + r_b) + r_a * (b_1 + r_b + b_1 + r_b) + 1 * (0))$$

   from which we construct the new set of the expressions multiplying $(a_1, a_2, r_a, 1)$

   $$P_2 = (b_1 + r_b, b_1 + r_b, 0, 0) \ .$$

3. Do the same thing with respect to $(b_1, b_2, r_b, 1)$:

   $$P'' = (b_1 * (a_1 + r_a + a_2 + r_a) + b_2 * (0) + r_b * (a_1 + r_a + a_2 + r_a) + 1 * (0))$$

   from which we construct

   $$P_1 = (a_1 + a_2, 0, a_1 + a_2, 0) \ .$$

4. Determine the input shares of $\widehat{a}$ necessary to simulate the expressions in P_1 and the shares of $\widehat{b}$ necessary to simulate the expressions in P_2.

   - for input sharing $\widehat{a}$, we trivially need both input shares (a_1, a_2) to perfectly simulate expressions in P_1.
   - for input sharing $\widehat{b}$, we apply one step of Gaussian elimination with respect to r_b, to obtain the new set $P_2' = (b_1 + r_b, 0, 0, 0)$. We can see that we can perfectly simulate the single non-zero expression with a uniform random value. Thus, in this case, no shares of $\widehat{b}$ are necessary to perfectly simulate P_2' and hence also P_2.

   Hence, to perfectly simulate P_1 and P_2, we need both input shares of $\widehat{a}$ and no shares of $\widehat{b}$. Thanks to Lemma 23, we can conclude that P can be perfectly simulated using both shares of $\widehat{a}$ and no shares of $\widehat{b}$.

7.3 Efficient Verification: IronMask

In this section, we present IronMask, a new tool that we developed to check probing and random probing security properties using the algorithms presented in Section 7.2. The implementation of IronMask currently considers a finite field $\mathbb{K}$ of characteristic 2. It can be extended to any finite field since the verification methods introduced in the previous sections work in any finite field $\mathbb{K}$. IronMask is written in C, and the only external libraries it depends on are the GNU Multiple Precision Arithmetic Library (GMP) and the POSIX Threads (pthreads) library.

7.3.1 Data Representation

```
#shares 2
#in a b
#randoms r0
#out c

m0 = a0 * b1
t0 = r0 + m0
m1 = a1 * b0
t1 = t0 + m1

m2 = a0 * b0
c0 = m2 + r0

m3 = a1 * b1
c1 = m3 + t1
```

(a) 2-share ISW multiplication

```
#shares 3
#in a
#randoms r0 r1 r2
#out d

d0 = a0 + r0
d0 = d0 + r1

d1 = a1 + r0
d1 = d1 + r2

d2 = a2 + r1
d2 = d2 + r2
```

(b) 3-share refresh

Figure 7.1: Masking gadgets written in IronMask's syntax

IronMask takes as input gadgets written in a simple syntax to describe circuits, borrowed from VRAPS [18]: a gadget is a list of assignments of additions and multiplications into variables, alongside directives to specify the number of shares, the inputs, the outputs, and the randoms. Figure 7.1 illustrates our input syntax on a 2-share ISW multiplication (Figure 7.1a) and a 3-share refresh gadget (Figure 7.1b). In Figure 7.1a, the variables `a0`/`b0` (resp. `c0`) and `a1`/`b1` (resp. `c1`) are the 1st and 2nd shares of the input `a`/`b` (resp. output `d`). Similarly to maskVerif [9, 10, 6], the syntax `![ expr ]` can be used to stop the propagation of glitches in the robust probing model. For instance, `tmp = a0*b0` could be replaced by `tmp = ![ a0*b0 ]`, in which case `tmp` would leak `a0*b0` instead of leaking `a0` and `b0` separately.

Internally, IronMask represents each wire of the gadget as an array of integers composed of three parts. The first ℓ parts correspond to linear dependencies on the inputs of the gadget: if the k^{th} bit of the i^{th} element is set to 1, then the wire depends linearly on the k^{th} share of the i^{th} input. The second part is a bit vector, where the k^{th} bit set to 1 indicates a linear dependency on the k^{th} random of the gadget. Finally, the third part is a bit vector as well, where the k^{th} bit set to 1 indicates a linear dependency on the k^{th} quadratic monomial appearing in the symbolic expressions of the gadget wires. For instance, the internal representation of the wires `a0`, `a1`, `r0`, `m3`, `t1` and `c1` of Figure 7.1a are as follows:

```
     inputs      randoms       mults
a0: [ 1, 0,         0,        0,0,0,0 ]
```

```
a1: [ 2, 0,          0,        0,0,0,0 ]
r0: [ 0, 0,          1,        0,0,0,0 ]
m3: [ 0, 0,          0,        0,0,0,1 ]
t1: [ 0, 0,          1,        1,1,0,0 ]
c0: [ 0, 0,          1,        0,0,1,0 ],
```

with an additional data structure storing the operands of each multiplication:

```
0: a0 * b1           1: a1 * b0
2: a0 * b0           3: a1 * b1
```

Using this internal representation enables efficient operations: the linear dependencies of a wire on the input shares are accessible with a single operation, and the number of such input shares is efficiently obtained by counting the number of bits to one in the first element (or first two elements for 2-input gadgets), and `xor`ing two wires, which is one of the basic operations of our Gaussian elimination can be done by `xor`ing pointwise the arrays representing them.

We use the same glitch model as in [6] to model glitches in the robust probing model. Namely, we consider that an expression $a + b$ (resp $a * b$) leaks a and b separately, instead of leaking $a + b$ (resp $a * b$). Registers (usually called *flip-flops*) can be used to stop the propagation of these glitches.

In IronMask, when taking glitches and transitions into consideration, each wire is represented by an array of arrays instead of a single array since the leakage of an assignment is the union of the leakages of its right-hand side operands. For instance, the wire `c0` in Figure 7.1a in the presence of glitches is represented as:

```
       inputs      randoms       mults
c0: [ [ 0, 0,         1,        0,0,0,0 ],
      [ 1, 0,         0,        0,0,0,0 ],
      [ 0, 1,         0,        0,0,0,0 ] ]
```

If a flip-flop was added to `m2` to stop the propagation of glitches by doing `m2 = ![a0*b0]`, then the robust leakage of `c0` would become:

```
       inputs      randoms       mults
c0: [ [ 0, 0,         1,        0,0,0,0 ],
      [ 0, 0,         0,        0,0,1,0 ] ]
```

7.3.2 Basic Verification

This section presents the procedures implemented in IronMask for verifying probing and random probing properties. Recall that in Section 7.1, we give definitions of all the security properties based on a single building block SIS: a primitive that, given a set of probes (internal probes and output probes), determines the input shares necessary for a perfect simulation of these probes. Thus, to verify any security property, IronMask uses a concrete implementation of the function SIS based on the algebraic characterization techniques discussed in Section 7.2.

Gadgets with linear randomness. For the verification of LR-gadgets introduced in Section 7.2.1 (*i.e.* gadgets in which all random values are additive), IronMask relies on the *SIS_LR* procedure (Algorithm 9). This procedure directly applies the result presented in Lemma 22. We recall that Algorithm 9 in IronMask currently considers a finite field of characteristic 2.

As in Lemma 22, Gaussian elimination is first performed on the tuple by the procedure *GaussElimination*, after which each probe of the input tuple is either "replaced" by a random

Algorithm 9: SIS_LR returns the input shares that are leaked by the tuple $\boldsymbol{P}$ with expressions of the form (7.4), assuming ℓ input sharings

```
1  procedure GaussElimination(P)
2      for each probe p_i of P do
3          if p_i contains at least one random variable then
4              r ← choose (any) one random variable in p_i;
5              for each probe p_j of P with i ≠ j do
6                  if p_j contains r then
7                      p_j ← p_i + p_j;
8                  end
9              end
10             p_i ← r;
11         end
12     end
13 procedure shares(P)
14     I_1 ← ∅, ..., I_ℓ ← ∅;
15     for each probe p_i of P do
16         Add all input shares in p_i of each input j to I_j;
17     end
18     return I_1, ..., I_ℓ;
19 procedure SIS_LR(P)
20     P' ← GaussElimination(P);
21     return shares(P');
```

r (as shown on line 10 of the procedure *GaussElimination*), or contains one or more input shares and no random values. What we mean by replacing the probe p_i by a random value r on line 10 is that after eliminating r from the expressions of all other expressions p_j in the same tuple (loop from line 5 to 7 where the instruction $p_j \leftarrow p_i + p_j$ aims to remove r from p_j in a finite field of characteristic 2), we end up with r only appearing in the expression of p_i and so as explained in the proof of Lemma 22, simulating p_i amounts to generating r uniformly at random without the need for any other variables. We represent this by replacing the expression of p_i with the single random value r. Then, the shares leaked by the input tuple can be found on the probes that do not contain any randoms using the procedure *shares*. The latter corresponds to an implementation of the function shares(.) used in Lemma 22.

Gadgets with non-linear randomness. For NLR-gadgets (*i.e.* gadgets performing non-linear operations on input shares mixed with randomness), IronMask uses the *SIS_NLR* procedure (Algorithm 10), which implements the four steps described in Section 7.2.2. As mentioned in Section 7.2.2, *SIS_NLR* currently only supports gadgets with two input sharings but can be extended to ℓ input sharings.

First, *SIS_NLR* performs Gaussian elimination with respect to the vector of output randoms (*i.e.* $\boldsymbol{r_3}$), using a modified version of *GaussElimination* that takes as inputs the randoms to use for the elimination. This corresponds to step 1 of Section 7.2.2. Next, two new tuples of probes $\boldsymbol{P_1}, \boldsymbol{P_2}$ are constructed from the probes in $\boldsymbol{P'}$ that do not contain any randoms from $\boldsymbol{r_3}$, using the *FactAndExtract* procedure, which corresponds to the factoring technique discussed in steps $2-3$ of Section 7.2.2. We thus get two tuples $\boldsymbol{P_1}$ and $\boldsymbol{P_2}$ containing input shares, randoms, and refreshed input shares from each input. Since those variables are linear, we can use the initial *SIS_LR* procedure to extract the input shares that they leak.

Algorithm 10: SIS_NLR returns the input shares that are leaked by the tuple P in an NLR-gadget refreshing its output with the randoms $\boldsymbol{r_3}$ (*c.f.* sec. 7.2.2), assuming 2 input sharings

```
1  procedure SIS_NLR(P, r3)
2      P' ← GaussElimination(P, r3);
3      P1 ← ( ), P2 ← ( );
4      for each probe p_i in P' do
5          if p_i contains no randoms of r3 then
6              (P1', P2') ← FactAndExtract(p_i);
7              (P1, P2) ← (P1||P1', P2||P2');
8          end
9      end
10     I_1 ← SIS_LR(P1), I_2 ← SIS_LR(P2);
11     return I_1, I_2;
```

Verification of security properties. Checking any probing or random probing property (*e.g.* NI, SNI, RPC, RPE, ...) consists of enumerating tuples of probes, using SIS_LR or SIS_NLR to get the input shares that they leak (we abbreviate with SIS and suppose that we make a call to the correct algorithm for LR-gadgets and NLR-gadgets), and take some action in consequence (see Section 7.1). In the following, we shall call a *t-failure tuple* (or simply a *failure tuple* when t is not made explicit) any tuple of probes that leaks more than t input shares of one or more input sharings (*i.e.* for which SIS outputs a set or more of cardinality strictly greater than t).

For instance, to verify if an n-share gadget is t-NI, we enumerate all tuples of size t and make sure that none of them is a t-failure tuple (Algorithm 11). This corresponds to the first row of Table 7.1. Verifying random probing-like properties amounts to computing the failure probability ε as a function $f(p)$ of the leakage probability p as described in Section 3.3 and Section 4.3. For instance, to verify the (p, ε)-RPS* of an n-share gadget G in the random probing model (first row of Table 7.2), we need to compute the coefficients c_i from equation (7.2) of the failure probability function $f(p) = \varepsilon$ as explained in Section 7.1. This corresponds to enumerating all the tuples of probes (excluding the output wires) of size 1 to s (where s is the total number of wires in the circuit) and counting how many leak more than $n-1$ shares. When it is computationally infeasible to enumerate all tuples, we only enumerate tuples of size 1 to a certain threshold β. From these β computed coefficients, we can derive upper and lower bounds on $f(p)$. For example, an upper bound is obtained by replacing each coefficient c_i for $i > \beta$ by the binomial coefficient $\binom{s}{i}$ (*c.f.* Section 3.3). We will see in the experimental results that in most of the cases, the upper and lower bounds on $f(p)$ are tight and give a precise estimation of the actual function $f(p)$[1].

Enumerating all tuples becomes impractical as soon as gadgets start growing larger than a few hundred variables since the number of tuples of size k in a gadget containing s variables is $\binom{s}{k}$. For instance, checking that a 9-share masked ISW multiplication containing 279 variables is 8-NI requires enumerating $\binom{279}{8} \approx 8 \times 10^{14}$ tuples, which is not far from being out of reach for modern computers.

The rest of the Section is organized as follows. In Section 7.3.3, we address dimension reduction techniques proposed in [16, 28] to reduce the search space of the enumerated tuples. In Section 7.3.4, we present some optimizations of our implementation that make verification

[1]This technique has already been used in VRAPS [18]

Algorithm 11: Is_t_NI returns true if G is t-NI and false otherwise, assuming G has ℓ input sharings

```
1  procedure Is_t_NI(G, t)
2      for each tuple P of size t in G do
3          I_1, ..., I_ℓ ← SIS(P);
4          if |I_1| > t or ... or |I_ℓ| > t then
5              return false;
6          end
7      end
8      return true;
```

Algorithm 12: GetCoeffsRPS* returns an array of β cells where the k^{th} index contains the number of failure tuples of k probes on n-share gadget G with ℓ input sharings

```
1  procedure GetCoeffsRPS*(G, β)
2      coeffs ← [0, ..., 0] of size β;
3      for k = 1 to β do
4          for each tuple P of k probes on G do
5              I_1, ..., I_ℓ ← SIS(P);
6              if |I_1| > n − 1 or ... or |I_ℓ| > n − 1 then
7                  coeffs[k] += 1;
8              end
9          end
10     end
11     return coeffs;
```

faster by reducing the cost of SIS_LR (since the latter is also a building block for SIS_NLR) and parallelizing our procedures. Finally, in Section 7.3.5, we introduce a constructive algorithm to generate failures without enumerating all tuples in linear gadgets.

7.3.3 Dimension Reduction

Checking any probing or random probing property requires enumerating many tuples. For instance, for a gadget G made of s variables, $\binom{s}{t}$ tuples must be checked to assess whether G is t-NI. We remove some variables from the search to reduce the number of tuples that have to be considered. First, as proposed in [16] and further explained in [28], elementary deterministic probes can be removed when checking any probing or random probing property. Then, when checking for probing properties only, we use the "reduced sets" optimization proposed in [28], eliminating some "less powerful" variables from the search. We recall hereafter the principle of those two optimizations and show how to make the first one work in the random probing model and why the second one cannot be used in this model. Note that the dimension reduction technique is proved to be sound in [28], which means that our verification technique implementing the optimization remains sound.

Removing elementary probes. Elementary deterministic probes refer to input shares and products of input shares. The idea behind the removal of those probes is that if a tuple $\boldsymbol{P}$ functionally depends on k input shares, we can always make it depend on $k + k'$ input shares (with $k + k'$ less or equal to the number of shares of the gadget) by adding elementary

deterministic probes. For instance, if a tuple t does not depend on the input share a_0, then the tuple (t, a_0) does, and so do any of the tuples of the form $(t, a_0 b_i)$.

The goal of our search procedure, instead of finding tuples of size k_1 that depend on t input shares, now becomes to find tuples of size $k_2 \leq k_1$ that depend on $t - (k_1 - k_2)$ input shares. In the probing model, such a tuple is enough to know that the property being checked does not hold. However, we want to generate and count all failures in the random probing model. When we find such a tuple $\boldsymbol{P}$, we thus generate all expansions of $\boldsymbol{P}$ combined with elementary deterministic probes that leak t input shares, thus making sure that all failures of the gadget are generated.

Similarly, we can remove elementary *random* probes: if, after the Gaussian elimination on a tuple $\boldsymbol{P}$, some input shares are masked by random variables, we can make them appear by adding the corresponding randoms. For instance, consider the 1-element tuple `a0+r1+a1`. It is easy to see that adding `r1` to this tuple would make it leak `a0` and `a1`. For simplicity, we keep elementary random probes when checking random probing properties and only perform this optimization for probing properties.

Using reduced sets. This optimization is formally introduced and proven correct in [28]. For completeness, we informally recall its principle here.

Let $\boldsymbol{P}$ and $\boldsymbol{P'}$ be two sets of probes. $\boldsymbol{P'}$ is said to be a *reduced set* for $\boldsymbol{P}$ iff $|\boldsymbol{P'}| \leq |\boldsymbol{P}|$ and for every linear combination of probes of $\boldsymbol{P}$, there exists a linear combination of $\boldsymbol{P'}$ using an equal or lower number of probes, which contains the same random dependencies, and at least as many input share dependencies. [28] proved that if $\boldsymbol{P}$ is a set of all wires of a gadget G and $\boldsymbol{P'}$ is a reduced set for $\boldsymbol{P}$, then, to prove that G is NI or SNI, the set of probes $\boldsymbol{P'}$ can be used instead of $\boldsymbol{P}$ to enumerate all tuples. If no failure is found in $\boldsymbol{P'}$, then none can be found in $\boldsymbol{P}$ either, and, conversely, if a failure is found in $\boldsymbol{P'}$, the same failure exists in $\boldsymbol{P}$.

For instance, if we consider a set $\boldsymbol{P} = (r_0, a_0, a_1, a_0 + r_0, a_0 + r_0 + a_1)$ (where r_0 is a random value and a_0, a_1 are input shares), then the set $\boldsymbol{P'} = (r_0, a_0, a_1, a_0 + r_0 + a_1)$ is a reduced set for $\boldsymbol{P}$. Evaluating the probing security of a gadget using the latter would yield the same conclusion as with the former while being faster since the latter contains one less probe.

This optimization is especially potent on ISW-like multiplications, which contain many wires of the form $X + a_i \cdot b_j$: the wire X can often be omitted since (informally) $X + a_i \cdot b_j$ contains the same random dependencies as X. However, it contains some additional input shares.

This optimization cannot be used in the random probing model because a set $\boldsymbol{P}$ and a reduced set $\boldsymbol{P'}$ would yield different failures. For instance, consider the sets proposed earlier as an example: $\boldsymbol{P} = (r_0, a_0, a_1, a_0 + r_0, a_0 + r_0 + a_1)$, and $\boldsymbol{P'} = (r_0, a_0, a_1, a_0 + r_0 + a_1)$. The tuple $(a_0 + r_0, a_0 + r_0 + a_1, a_1)$, made of wires of $\boldsymbol{P}$, reveals two input shares and cannot be built from wires of $\boldsymbol{P'}$ since $a_0 + r_0$ is not in $\boldsymbol{P'}$.

7.3.4 Implementation Optimizations

On-the-fly Gaussian Elimination. In order to find all failures of a given size, we enumerate all the tuples of that size and apply the SIS procedure on each. This means a complete Gaussian elimination must be performed on each tuple. However, we generate the tuples in lexicographic order, meaning that two consecutive tuples only differ by their last elements and, in most cases, only by their last element. For two consecutive tuples, it is thus very likely that most of the Gaussian elimination will be identical. We take advantage of this by implementing our Gaussian elimination on the fly: we only recompute the elimination on the elements that differ from the previous tuple for each tuple.

The cost of the Gaussian elimination for a single tuple of k elements of a gadget containing d inputs and randoms is $\mathcal{O}(dk^2)$. Performing the elimination on-the-fly brings the amortized complexity down to $\mathcal{O}(dk)$.

The authors of [28] used a similar, slightly more efficient technique to speed up their implementation. They used a revolving-door algorithm to generate the tuples so that each consecutive tuple differs by exactly one element, which allows for amortizing the analysis's cost. However, we cannot use this revolving-door algorithm because when changing the i^{th} element of a tuple, the Gaussian elimination needs to be recomputed from this i^{th} element up to the end of the tuple.

Parallelization. Recall that we generate the tuples in lexicographic order, which admits an efficient *unranking* algorithm. This means that we can efficiently compute the j^{th} tuple of size k for any j and any k. Multi-threading the verification of n tuples is thus trivial: run l threads in parallel, the j^{th} thread starts with the $\lfloor j \times n/l^{th} \rfloor$ tuple, and verifies the next $\lfloor n/l \rfloor$ tuples.

Our implementation is multi-threaded in this fashion using POSIX threads provided by the `pthread` library. In order to be transparent from the properties' point of view (*e.g.*, from Algorithms 11 and Algorithm 12), the multi-threading is done inside SIS. To this end, we use a few mutexes, which incur an overhead in the random probing model: the more failures a gadget contains, the less of a speedup multi-threading offers. Although it would not be hard to implement multi-threading on the properties' side rather than in SIS, we opted for readability and maintainability of the code at the slight expense of performance.

7.3.5 Constructive Approach

The enumerative approach of Section 7.3.2 generates many tuples that are trivial non-failures because they do not contain enough shares to be failures or their shares are masked by random variables. We designed a constructive algorithm only to generate potential failures to overcome this issue. Our constructive algorithm aims at generating *incompressible failure tuples*, which we define as follows:

Definition 27 (Incompressible failure tuple)**.** *A tuple $\boldsymbol{P}$ is an incompressible failure tuple if it is a failure, and if no tuple $\boldsymbol{P'} \subset \boldsymbol{P}$ is a failure itself. ($\subset$ between two tuples means that all wires of $\boldsymbol{P'}$ are included in the tuple $\boldsymbol{P}$).*

We will describe our constructive algorithm for the well-adapted case of LR-gadgets and then explain how to extend it to NLR-gadgets. The idea is that given wires which are all of the form (7.4), a failure tuple of probes on these wires has a specific form: it contains some wires with input shares, and if those input shares are *masked by randoms*, it contains some additional wires to cancel out those randoms. The expression *masked by randoms* means that the perfect simulation of the considered probe amounts to generating a uniform random value. This is typically the case in a tuple of probes where a random value appears only once in one of the expressions, which then can be used to mask the expression of that probe. To cause a failure event and avoid masking the expressions, we add wires using the same randoms to cancel them out.

We start by giving the intuition of the algorithm on the 3-share refresh gadget presented in Figure 7.1b. We first build a map (Table 7.3), called Columns, whose keys are the input shares and randoms of the gadget considered and whose values are all the wires that depend on those inputs and randoms (a wire will be displayed in several columns if it depends on several shares or randoms). To build a failure tuple that leaks three shares, we can pick one wire from each of the `a0`, `a1` and `a2` buckets, say `(a0, a1+r0,a2+r1)` for instance. This tuple is not a

Algorithm 13: Our constructive algorithm to generate failures. G is the gadget we are considering, and n is the number of shares required for a tuple to be a failure.

```
1  procedure UnmaskTuple(G, S, P, unmask_index)
2      if P is a failure then
3          if P is incompressible then
4              S ← S ∪ {P};
5          end
6          return;
7      end
8      if unmask_index > length(P) then
9          return;
10     end
11     UnmaskTuple(G, S, P, unmask_index+1);
12     P_Gauss ← GaussElimination(P);
13     if P_Gauss[unmask_index] contains no randoms then
14         return;
15     end
16     r ← any random from P_Gauss[unmask_index];
17     for each wire w of G containing r and not in P do
18         P' ← P || (w);
19         UnmaskTuple(G, S, P', unmask_index+1);
20     end
21 procedure ConstructiveFailuresGenLR(G, n)
22     S ← ∅;
23     for each tuple P in L do
24         UnmaskTuple(G, S, P, 0);
25     end
26     return S;
```

failure because the shares `a1` and `a2` are masked by the randoms `r0` and `r1` (the two random values appear only once in the tuple and can be used to mask the corresponding expressions). We thus pick a wire from the `r0` bucket and add it to the tuple, say `a0 + r0 + r1` (which happens to cancel `r1` as well as `r0`). The resulting tuple is `(a0,a1+r0,a2+r1,a0+r0+r1)`, which is a failure. By doing this for every possible wire of each column, we can generate all failures of the gadget of Figure 7.1b.

Algorithm 13 introduces more formally this procedure for LR-gadgets. This algorithm lists all of the tuples composed of one element from each input share column (line 23); we note the resulting list $\mathcal{L}$. Note that those tuples might have some duplicates since some wires appear in several columns: these duplicates are removed while building the tuples (which implies that the tuples in $\mathcal{L}$ contain possibly less than n elements).

Then, for each tuple in $\mathcal{L}$, the recursive procedure *UnmaskTuple* adds wires to the tuple to cancel the randoms that mask its input shares. The procedure takes as argument the circuit G, the set of incompressible tuples already computed S, a tuple $\boldsymbol{P}$ that needs to be turned into a failure and an integer *unmask_index* that contains the next index of $\boldsymbol{P}$ that we should try to unmask. First, *UnmaskTuple* checks if $\boldsymbol{P}$ is a failure (line 2). To do so, we can use the procedure *SIS_LR* (Algorithm 9). If $\boldsymbol{P}$ is a failure, we then check if it is incompressible (line 3) by checking if any tuple $\boldsymbol{P'} \subset \boldsymbol{P}$ is already in S. Ignoring line 11 for now, a Gaussian elimination is then performed on $\boldsymbol{P}$ (line 12). If the *unmask_index*$^{\text{th}}$ element of the resulting

Table 7.3: COLUMNS map for the constructive generation of incompressible tuples of Figure 7.1b

input shares			randoms		
a0	a1	a2	r0	r1	r2
a0	a1	a2	r0	r1	r2
a0 + r0	a1 + r0	a2 + r1	a0 + r0	a2 + r1	a1 + r0 + r2
a1 + r0 + r1	a1 + r0 + r2	a2 + r1 + r2	a1 + r0	a2 + r1 + r2	a2 + r1 + r2
			a0 + r0 + r1	a0 + r0 + r1	

tuple $\boldsymbol{P}_{\text{Gauss}}$ contains no random, then there is nothing to unmask, and we can move on to the next index (which was already done by line 11). Otherwise, we select any random r of $\boldsymbol{P}_{\text{Gauss}}$ and try to add to $\boldsymbol{P}$ each wire that contains r (*i.e.*, each wire of the r column of the COLUMNS map, and move to the next *unmask_ index* (lines 16 to 19).

Unmasking each element of $\boldsymbol{P}$ one by one misses some failures. For this reason, line 11 skips the unmasking of the element of $\boldsymbol{P}$ at index *unmask_ index* to move directly to index *unmask_ index*+1. Consider, for instance, a 2-share gadget and the tuple $\boldsymbol{P} = (a_0 + r_0, a_1 + r_1 + r_0)$. After the Gaussian elimination, the 1st element of $\boldsymbol{P}$ is r_0, and lines 16 to 19 of UNMASKTUPLE will thus try to add a wire containing r_1 to the tuple. However, this would be missing the fact that the 2nd element of $\boldsymbol{P}$, $a_1 + r_1 + r_0$, already contains r_1 and thus *somewhat* unmasks $a_0 + r_1$. By skipping the first element of $\boldsymbol{P}$, we will try to unmask its second element by adding the wire r_1 to the tuple. This will produce the tuple $(a_0 + r_0, a_1 + r_1 + r_0, r_1)$, which is a failure, and would have been missed without the recursive call of line 11.

This constructive method is exhaustive since any incompressible failure tuple $\boldsymbol{P}$ can be built by taking one element in each column (and possibly removing duplicates) and adding necessary elements to remove the masks. Consider a minimal sub-tuple $\boldsymbol{P'}$ of $\boldsymbol{P}$, which contains one element of each column. This sub-tuple $\boldsymbol{P'}$ will be listed in line 11. The other coordinates of $\boldsymbol{P}$ are necessary to remove the masks remaining after an application of the Gaussian elimination to $\boldsymbol{P'}$ (otherwise $\boldsymbol{P}$ would not be incompressible). Since Algorithm 13 is exhaustive in removing those masks, it will necessarily build $\boldsymbol{P}$ at some point.

Implementation. Implementing *UnmaskTuple* in Algorithm 13 does not perform a full Gaussian elimination at every recursive call. Instead, the elimination is performed on the fly, similar to what we do for the enumerative algorithm (see Section 7.3.4). Likewise, we keep a variable *input_ shares* containing the input shares already revealed by the current tuple $\boldsymbol{P}$, which enables to check if $\boldsymbol{P}$ is a failure in constant time, without having to call *SIS_ LR*: we can check if *input_ shares* contains n input shares.

Extension to gadgets with non-linear randomness. The procedure *UnmaskTuple* of Algorithm 13 only considers gadgets with linear randomness. To adapt it for gadgets with non-linear randomness, we proceed similarly as in *SIS_ NLR* (Algorithm 10): a first step unmasks randoms that are used to refresh outputs, while a second step unmasks randoms that are used to refresh inputs. We call *ConstructiveFailuresGenNLR* this version of our constructive algorithm and define *ConstructiveFailuresGen* as the function that chooses between *ConstructiveFailuresGenLR* and *ConstructiveFailuresGenNLR* depending on whether its input gadget contains linear or non-linear randomness.

Application. Algorithm 14 shows how to use *ConstructiveFailuresGen* to compute all gadget failures. While the latter returns all incompressible failures to evaluate the failure function coefficients for random probing notions (RPC, RPE1, RPE2, RPS*), we need to count the number of all failures, regardless of their incompressibility. To do so, we expand all incompressible

failures into regular failures by adding wires one by one (using the procedure *ExpandTuple*, whose pseudo-code is trivial and hence omitted). However, doing so will lead to the same tuple being generated multiple times: for instance, if the tuples (x_1, x_2) and (x_1, x_3) are both incompressible failures, expanding them will generate (x_1, x_2, x_3) and (x_1, x_3, x_2), which are the same tuple. We thus use a hash table (called S_{failure} in Algorithm 14, and abstracted as a set for simplicity) to store the tuples that we generate and prevent counting multiple times the same tuple. In practice, our hash function returns the sum of the hashes of the indices of wires in the tuple, which results in a fairly low number of collisions.

Algorithm 14: GetCoeffsRPconstr returns an array of c_{max} cells where the k^{th} index contains the number of failure tuples of size k in G

```
1  procedure GetCoeffsRPconstr(G, c_max)
2      coeffs ← empty array;
3      t ← number of shares in G;
4      S_incompr ← ConstructiveFailuresGen(G, t);
5      S_failure ← ∅;
6      for k = 1 to c_max do
7          S'_failure ← all tuples of S_incompr of size k;
8          for each tuple P of S_failure do
9              S'_failure ← S'_failure ∪ ExpandTuple(P);
10         end
11         S_failure ← S'_failure;
12         coeffs[k] ← number of tuples in S_failure;
13     end
14     return coeffs;
```

Remark 7. *We initially tried to count the failures from the incompressible failures without generating all of them. This problem can be formulated as follows:* Let W be a set of integers (the wires). Let S be a set of subsets of W of arbitrary sizes (the set of incompressible failures). How many subsets of W of size k are super-sets of elements of S (those subsets are non-incompressible failures)? *This is a problem of inclusion-exclusion, and solving it requires computing the intersections of all pairs of sets in the powerset of S. Since there are $2^{|S|}$ such sets, this approach would be prohibitively expensive for any gadget with more than a few incompressible failures.*

Performance & limitations. This constructive algorithm is faster than the traditional enumerative algorithm of Section 7.3.2 when checking $(n-1)$-NI and RPS* properties for linear gadgets. Table 7.4 shows the exact performance improvements when checking the RPS* property on some common linear gadgets (ISW refresh [59], $\mathcal{O}(n \log n)$ refresh [13], and an "ISW addition" made of a share-wise addition preceded by an ISW refresh of each input), and on an ISW multiplication with a circular refresh on one of the inputs. On linear gadgets, the constructive algorithm can go about two coefficients further than the enumerative one simultaneously, thus producing much more precise results.

Furthermore, the constructive algorithm verifies larger, previously out-of-reach gadgets. For instance, a 9-share ISW addition contains 243 variables and thus contains $\binom{243}{9} \approx 7 * 10^{15}$ tuples of size 9 (for 8-RPS*), which is clearly beyond the capabilities of the enumerative algorithm. Nevertheless, our constructive algorithm can generate all of its failures of size 9 in 7 minutes.

Table 7.4: RPS* performance of our new constructive algorithm against our traditional enumerative one

Gadget	Shares	#wires	Enumerative		Constructive	
			β	Verif time	β	Verif time
ISW refresh	8	140	8	5min	8	3sec
					10	6min
ISW add	7	224	7	18min	8	2sec
					9	2min
$\mathcal{O}(n \log n)$ refresh	8	100	9	1min	9	2sec
					11	2min
ISW mult refreshed	6	297	6	2min	6	12min
			7	38min		

However, the constructive algorithm is slower on multiplication gadgets than the enumerative one. Table 7.4 illustrates this on an ISW multiplication with an ISW refresh on one of the inputs. This can be explained by the fact that many tuples are generated multiple times by this constructive algorithms, each time through a different path in the recursion. For instance, on the refresh gadget of Figure 7.1b, the tuple $(a_0, a_1+r_0, a_2+r_1, a_1+r_0+r_1)$ can be generated by selecting (a_0, a_1+r_0, a_2+r_1) as the initial tuple (line 23 of Algorithm 13), and then adding $a_1+r_0+r_1$ at line 18. However, the same tuple can also be generated by selecting $(a_0, a_1+r_0+r_1, a_2+r_1)$ at line 23 and then adding a_1+r_0 at line 18. This phenomenon is even more impactful when dealing with multiplication gadgets because multiple shares of the same input will appear on the same wire, resulting in larger columns (particularly in the "inputs" part of the Columns map), which can lead to a worst complexity than simply enumerating all tuples.

Additionally, the constructive algorithm does not perform well when checking t-NI or t-RPS* with $t < n-1$, SNI, RPC, and RPE. The reason is that the traditional enumerative algorithm enumerates all tuples regardless of the property being checked and the failure condition. However, with the constructive algorithm, for an n-share gadget, to generate tuples of that leak k shares, all $\binom{n}{k}$ possible combinations of shares must be tested. For instance, in our example of Figure 7.1b, to generate all failures leaking two shares, we would generate all failures leaking the 1st and the 2nd shares, then the ones leaking the 1st and the 2nd share, and, finally, the ones leaking the 2nd and the 3rd share. Testing these combinations quickly becomes less efficient than just enumerating all tuples, as in the basic enumerative approach.

7.4 Evaluation and Performance

To showcase IronMask, we start in Section 7.4.1 by providing new bounds for the maximum RPE leakage probability tolerated by some common gadgets (in the random probing model). Then, we compare the scope and performance (Section 7.4.2) of IronMask and existing verification tools: VRAPS and STRAPS in the random probing model, and maskVerif, matverif and SILVER in the probing model. The description files of the gadgets tested in the following sections are publicly available on IronMask's GitHub repository.

7.4.1 New Random Probing Expandability Results

So far, VRAPS [18] was the only tool verifying the (t, p, ε)-RPE property. IronMask is several orders of magnitude faster than VRAPS, in addition to being complete (IronMask avoids failure

Table 7.5: Maximum t-RPE leakage probabilities tolerated by some common masking gadgets

Gadget		Shares	t	Ampl. order	β	#wires	$\log_2$ maximum tolerated proba.	Verif. time
Linear Randomness								
ISW [59]	mult	5	2	3/2	6	180	-10.54	24min
		6	2	3/2	5	267	-12.00	13min
		7	3	2	5	371	[-10.45, -8.73]	28min
	refresh	5	2	3	10	50	-4.28	2min
		6	2	3	8	75	[-4.81, -4.61]	5min
		7	3	4	7	105	[-5.50,-4.01]	21min
	add	5	2	3	7	110	[-6.48,-4.70]	11min
		6	2	3	6	162	[-7.81,-5.03]	17min
		7	3	4	6	224	[-8.47,-4.15]	3h
	copy	5	2	3	6	105	[-5.92,-5.54]	12min
		6	2	3	5	156	[-6.92,-5.93]	24min
		7	3	4	4	217	[-8.02,-3.87]	33min
$n \log n$ [13]	refresh	4	1	2	30	30	-5.27	1sec
		8	3	4	7	100	[-5.42,-4.36]	18min
	add	4	1	2	8	68	-5.40	4min
		8	3	4	6	216	[-8.40,-4.40]	4h
	copy	4	1	2	6	64	-6.96	27sec
		8	3	4	4	208	[-7.94,-4.25]	55min
circular refresh [11]		5	2	3	25	25	-4.84	1sec
		10	4	3	8	50	-5.21	1min
Sec. 5.5	add	5	2	3	9	55	[4.67,-4.42]	10min
	copy	5	2	3	6	60	-6.17	41sec
Non-linear Randomness								
Double-SNI ISW		4	1	2	5	190	-9.85	5min
		5	2	5/2	5	305	[-10.01,-8.09]	31min
mult Sec. 5.5		5	2	3	6	405	[-9.67,-7.66]	31h

false positives *i.e.* detected failure tuples which are not failures, unlike VRAPS), allowing us to compute more precise bounds for the coefficient of the failure function $f(p) = \varepsilon$ and hence more precise bounds on the tolerated leakage probability. In particular, we consider the ISW multiplication and refresh [59], the $\mathcal{O}(n \log n)$ refresh [13], the circular refresh [11], as well as the addition, copy, and multiplication from Section 5.5. Additionally, we consider addition (resp. copy) gadgets obtained by doing an ISW or $\mathcal{O}(n \log n)$ refresh on one of the inputs followed by a simple addition (resp. copy). Finally, we also evaluate a double-SNI multiplication [54] made of an ISW multiplication where one of the inputs is refreshed using circular refresh [44] (with n shares).

The results are shown in Table 7.5. For the t parameters, we used $t = \lfloor (n-1)/2 \rfloor$ (with n shares) to maximize the amplification order (*c.f.* Section 4.4.1). We cannot precisely compute the maximum leakage probability tolerated in a reasonable time for large gadgets. Instead, we compute all failures up to a given c_β, which allows us to obtain upper and lower bounds for the leakage probability.

For ISW multiplication, our results improve previous results obtained with VRAPS in two ways: by increasing the value β of the verification, we obtain tighter bounds on the failure event function $f(p)$ and thus tighter intervals (and even exact values in some cases) for the tolerated leakage probability. Plus, thanks to the completeness of the verification of IronMask by avoiding failure false positives (unlike VRAPS), we obtain better values for the estimated tolerated leakage probability (by better, we mean higher probability values). For example, our results show that the (exact) tolerated leakage probability of the 6-share ISW multiplication is 2^{-12} instead of the 2^{-13} lower bound of Table 4.1 (Section 4.6, Chapter 4). For all the other

gadgets in the table, we report the first verification results of the RPE property.

7.4.2 Comparison with State-Of-The-Art Tools

We compare IronMask to six carefully chosen state-of-the-art tools: maskVerif [9, 10, 6] (and its extension scVerif [12]), matverif [28], SILVER [62], VRAPS [18], and STRAPS [32], with which our new tool IronMask share the following features:

- do not rely on any gadget's structure (unlike *e.g.*, `maskComp` [10], `tightPROVE` [20], `Tornado` [19]),
- verify probing or random probing-like security notions.

We discuss the concretely verified properties and provide benchmarks highlighting the main differences with IronMask.

Scope. Introduced in 2015 [9] and then extended multiple times ([10, 6]), maskVerif is the very first tool able to verify reasonable higher-order masking schemes. Based on a symbolic representation of leakage, it integrates the language-based verification of (robust) probing security and (S)NI notions with or without leakage on register transitions. One step further, the latest extension of maskVerif, referred to as scVerif [32], captures even more hardware side effects, potentially configurable by the user [12]. Compared to our proposal, maskVerif includes tricks to verify bigger circuits (*e.g.*, s-boxes, block encryption scheme) but fails to provide a complete verification as soon as the randomness is not linear (*i.e.*, failure false positives may be produced).

Similarly, matverif [28] targets the same properties as maskVerif. It features a new method to obtain a complete verification (*i.e.*, without any failure false positive) for specific circuits (*e.g.*, ISW multiplications) and significantly improve its performance thanks to dimension-reduction strategies. Regarding supported gadgets, matverif is more limited than maskVerif and IronMask, as it does not support gadgets with non-linear randomness. Unlike our proposal and similarly to maskVerif, matverif focuses only on verifying probing-like properties.

Following a different strategy, SILVER [62] was built by Knichel, Sasdrich, and Moradi to verify the physical security of hardware designs. It takes as input either a Verilog implementation or an instruction list and checks the probing (S/PI)NI notions in the standard and robust models and the uniformity of some output sharing. On the one hand, it outperforms the capacities of maskVerif by offering a complete verification based on a symbolic and exhaustive analysis of probability distributions and statistical independence of joint distributions. On the other hand, its verification is significantly slower than that of maskVerif.

Introduced in 2020, VRAPS is the first tool to verify random-probing-like properties [18]. Written in Python and SageMath, it was built to evaluate the RPE security of some base gadgets to assess the global security of the expanding compiler of [18]. Specifically, VRAPS detects all the leaking tuples within an implementation with respect to the RPS*, RPE1, RPE2, and RPC security properties as described in Section 7.1. Nevertheless, it suffers from low performance and, unlike IronMask, can generate failure false positives for both gadgets with linear and non-linear randomness. VRAPS supports more gadgets than IronMask which is limited to LR-gadgets and NLR-gadgets. Nevertheless, to our knowledge, all the masking gadgets in the literature fit the latter representations. While VRAPS can additionally (directly) verify bigger gadgets (*i.e.*, composition of atomic gadgets), the performance and completeness would be deficient in practice. In addition, the verification of atomic gadgets using IronMask already makes it possible to obtain secure global circuits since once individually verified (for probing or random probing properties), they can be safely composed (*c.f.* [10] or Section 3.2).

Finally, STRAPS is a recent tool designed to verify random probing-like properties [32]. In particular, it was built to compute the distribution of a gadget's input sets of shares with respect to the output observations and the leakage probability of each internal wire. In its deterministic mode, it relies on maskVerif as a basic primitive. It integrates a probabilistic mode based on Monte-Carlo methods, significantly improving performance by avoiding a complete exploration and limiting the analysis to selected tuples. While the probabilistic mode can allow increased performance and thus more accurate results for random probing properties, it uses a set of rules from maskVerif as a building block. These rules by construction do not provide complete verification, which implies that the verification method of STRAPS is incomplete too.

Table 7.6: Verified security properties on higher-order masked implementations for carefully chosen state-of-art automatic tools.

Tools	probing-like		RP-like	
	soft	robust	soft	robust
maskVerif	✓	✓	✗	✗
scVerif	✓	✓	✗	✗
matverif	✓	✓	✗	✗
SILVER	✓	✓	✗	✗
VRAPS	✓	✗	✓	✗
STRAPS	✗	✗	✓	✗
IronMask	✓	✓	✓	✓

Table 7.6 recalls the categories of properties (as in Section 7.1) that are verified by the tools mentioned above on higher-order masked implementations. It additionally specifies the consideration of hardware effects, *i.e.* glitches (captured in the robust probing model). A green check (✓) means the row tool verifies the column property. On the contrary, a red cross (✗) means that the row tool does not handle the column property. We can see that IronMask is the first tool to verify probing-like properties and random probing-like properties in the standard model and in the presence of glitches (robust model).

Additionally, IronMask offers a complete verification method for gadgets with linear randomness and for most deployed gadgets with non-linear randomness (mainly all known multiplication gadgets). The only other tool providing complete verification for such gadgets is SILVER but this is achieved by an exhaustive approach making its running time quickly prohibitive (see comparison hereafter).

Performance. We evaluate the performance of IronMask compared to other state-of-the-art verification tools in both the probing and random probing models.

Probing Model. We compared the time required by IronMask, maskVerif and matverif to check that some commonly-used gadgets are $(n-1)$-NI and $(n-1)$-SNI (abstracted NI and SNI hereafter for conciseness). In particular, the gadgets we considered are the ISW multiplication [59], the double-SNI multiplication [54, 33] using an ISW multiplication and a circular refresh [44] on one of the inputs, the new NI and SNI multiplications from [28], and the $\mathcal{O}(n \log n)$ refresh [13]. The results are presented in Table 7.7.

We used the multi-threaded version of each tool, setting the maximal number of cores to use to 4, to give a fair chance to maskVerif: while IronMask and matverif can use an arbitrary number of cores, maskVerif is limited to 4 cores. We evaluated several masking orders for each gadget to highlight each tool's scaling. To save time, we did not run the verification of gadgets

Table 7.7: Comparison of the performance of IronMask, maskVerif and matverif on higher-order masked gadgets. The multithreaded versions of each tool were used, with the maximum number of threads set to 4. `N/A` means that a tool cannot check a gadget, whereas - means that a tool was not evaluated on a gadget because we deemed it too slow.

Gadget	**Type**	**Shares**	**Property**	**Verification time**		
				IronMask	maskVerif	matverif
ISW mult	LR	7	NI	7sec	1min30sec	24sec
			SNI	8sec	3min56sec	25sec
		8	NI	4min 6sec	2h 10min	5min 19sec
			SNI	5min 15sec	6h 30min	5min 15sec
		9	NI	2h 22min	-	2h 3min
			SNI	3h 7min	-	1h 58min
ISW mult refreshed	NLR	6	NI	2sec	3sec	`N/A`
			SNI	3sec	10sec	`N/A`
		7	NI	3min 41sec	1min 50sec	`N/A`
			SNI	6min	8min 16sec	`N/A`
		8	NI	8h 52min	2h 2min	`N/A`
			SNI	14h 46min	10h 4min	`N/A`
NI mult [28]	LR	7	NI	1sec	1min50sec	5sec
		8		5sec	2h 10min	9sec
		9		2min50sec	-	40sec
		10		6h 28min	-	1h 40min
SNI mult [28]	LR	7	SNI	1sec	6min	8sec
		8		46sec	6h 26min	17sec
		9		24min	-	4min 37sec
		10		24h	-	1h 54min
refresh $\mathcal{O}(n \log n)$	LR	9	NI	1sec	$<$ 1sec	1sec
			SNI	24sec	2sec	1sec
		10	NI	1sec	<1sec	10sec
			SNI	16min	9sec	10sec
		11	NI	1sec	<1sec	3min
			SNI	7h 50min	1min	3min
		12	NI	1sec	<1sec	3h 35min
			SNI	-	5min	1h 52min

when it would have taken more than a few hours. Finally, to analyze a gadget, matverif needs probes description files to be generated using a SageMath script and the main program to be recompiled. While the incurred additional time was ignored in [28], we take it into account so that the time we report reflects the actual time that a user would need to check those gadgets.

Before discussing the results, we recall that the three tools are not functionally equivalent. maskVerif can handle any circuit, while IronMask is limited to LR-gadgets and NLR-gadgets as characterized in Section 7.2, and matverif only handles LR-gadgets. On the other hand, the verification of IronMask is complete for both types of gadgets, while maskVerif's is not. In addition, while IronMask is limited to LR-gadgets and NLR-gadgets, to the best of our knowledge, all the masking gadgets in the literature fit the latter representations, and the verification of atomic gadgets already makes it possible to obtain secure global circuits through composition.

Overall, the three tools have similar performance and allow the analysis of gadgets up to similar masking orders, although each tool has strengths and weaknesses. For instance, maskVerif performs better on LR-gadgets, while matverif shines on LR multiplications. In the following, we investigate the relative speed of each tool in more detail.

Five main factors need to be considered to analyze the performance of IronMask compared to matverif. First, matverif's method to verify a tuple of probes on an LR-gadget has linear complexity in the tuple's length, while our SIS_LR method has the complexity of a Gaussian elimination which in our implementation is quadratic in the tuple's length. Second, the dimension reduction performed by IronMask is faster than that of matverif in most cases, probably because ours is written in C and matverif's in SageMath. Third, our dimension reduction often removes more wires, resulting in fewer tuples. Fourth, matverif needs to be recompiled for each gadget. Fifth and last, on linear gadgets (add and copy) and when checking $(n-1)$-NI, IronMask uses the constructive algorithm (see Section 7.3.5), which is much faster than any enumerative algorithm.

It can be observed through Table 7.7 that the scale of the speedups offered by IronMask compared to matverif and maskVerif depends mainly on the structure and nature of the gadgets. We will explain the reason for this scaling depending on the tested gadgets.

matverif tends to be slower than IronMask at smaller masking orders and at any order when checking $(n-1)$-NI on linear gadgets. However, at the highest masking orders, the cost of the dimension reduction and the recompilation of matverif becomes negligible compared to the main enumerative algorithm. matverif thus becomes faster than IronMask, thanks to its linear complexity in the tuples' length (compared to the quadratic complexity of IronMask's SIS_LR primitive).

On the standard ISW multiplication and the multiplication from [28], both matverif and IronMask outperform maskVerif thanks to the dimension reduction they use (see Section 7.3.3), which is not implemented in maskVerif. For instance, the 7-order ISW multiplication initially contains 220 variables, but only 77 remain after the dimension reduction.

The dimension reduction is much less potent on the double-SNI ISW multiplication with a circular refresh on the inputs. For the 6^{th}-order double-SNI multiplication, 175 probes must thus be considered against 57 for the standard 6^{th}-order ISW multiplication. As a result, maskVerif and IronMask have similar performance on this gadget.

The $\mathcal{O}(n \log n)$ refresh is NI at any order since at no point multiple secret shares are part of the same probe. maskVerif has a special rule to detect that, resulting in the verification of NI that is almost instantaneous. We did not add this special rule in IronMask, but our

constructive algorithm (presented in Section 7.3.5) can detect that the gadget is indeed NI very quickly. However, to check that this gadget is SNI, both matverif and IronMask enumerate all tuples, which becomes very expensive as the masking order grows. As a result, checking that the 11^{th}-order gadget is SNI with IronMask would require at least a few days. On the other hand, maskVerif does not need to enumerate all tuples and can quickly determine that this gadget is SNI, taking just 5 minutes to do so at order 11.

Table 7.8: Verification time of NI and SNI verification of the ISW multiplication and $\mathcal{O}(n \log n)$ refresh by IronMask and SILVER

Gadget	**Shares**	**Property**	**Verification time**	
			IronMask	SILVER
ISW mult	4	NI	<1sec	1sec
		SNI	<1sec	2sec
	5	NI	<1sec	9min
		SNI	<1sec	14min
	6	NI	<1sec	>10h
		SNI	<1sec	>10h
refresh $n \log n$	6	NI	<1sec	8sec
		SNI	<1sec	20sec
	7	NI	<1sec	7min
		SNI	<1sec	14min
	8	NI	<1sec	>10h
		SNI	2sec	>10h

As mentioned earlier, SILVER, while being the only tool mentioned here that is complete on any gadget suffers from severe performance limitations. This is illustrated in Table 7.8, which shows that SILVER is several orders of magnitude slower than IronMask on the ISW multiplication and the $\mathcal{O}(n \log n)$ refresh.

Random Probing Model. Table 7.9 shows the time needed by VRAPS and IronMask to compute the maximum tolerated leakage probability of the ISW multiplications when setting β to 4 (which is the maximum that is computable by VRAPS in reasonable time). IronMask was not multi-threaded in this benchmark, and we recall that VRAPS does not support multi-threading. IronMask is several orders of magnitude faster than VRAPS in addition to being more precise (since VRAPS can incorrectly classify tuples as failures). Several factors explain the performance gains. First, IronMask is written in C, whereas VRAPS is written in SageMath. Second, IronMask uses a complete technique based on Gaussian elimination to determine if tuples are failures, whereas VRAPS uses SageMath's symbolic calculus to apply simplification rules inspired by maskVerif iteratively. Third, IronMask allocates less memory and performs its Gaussian elimination on the fly (see Section 7.3.4), whereas VRAPS allocates chunks of memory to store batches of tuples and restarts the simplifications from scratch for each tuple.

Table 7.9: Performance of (t, p, ε)-RPE verification of IronMask and VRAPS on ISW multiplication gadgets at orders 4 to 6 (*i.e.* 5 to 7 shares)

Shares	**t**	β	**#wires**	$\log_2$ **maximum tolerated proba**	**Verification time**	
					IronMask	VRAPS
5	2	4	180	[-11.00,-10.67]	3sec	1h 15min
6	2	4	267	-13	17sec	24h
7	3	4	371	[-12.00,-7.83]	24sec	24h

The performance gains of IronMask over VRAPS have two main benefits. First, IronMask can be more beneficial for prototyping since it can provide approximate results for small β within a few seconds. Second, IronMask can compute exact and more precise results by increasing β, as shown in Table 7.5.

7.5 Conclusion

In this chapter, we introduced our verification tool IronMask, which offers an exact verification for most gadgets in the state of the art, using an algebraic characterization of the probes inspired from [16, 17]. We discussed how to extend this algebraic characterization to gadgets with linear and non-linear randomness and how to use it to implement SIS, our building block used to verify any (random) probing-like property. We showed that IronMask is competitive with other tools in the literature for probing-like properties and is more efficient than existing tools for checking properties in the random probing model. Namely, IronMask can verify the RPC and RPE properties discussed in the previous chapters. It gives tighter results than the original tool VRAPS and can verify gadgets for bigger numbers of shares.

Future works include generalizing the algebraic characterization of probes to gadgets with more than quadratic randomness. In other words, non-linear randomness is restricted to quadratic, where we refresh some inputs and then perform multiplication between those inputs. This characterization could be generalized to any level of non-linearity on the randoms in the gadget. A possible approach would be to extend this characterization to a leveled approach where we partition randoms into different layers and process them iteratively from bottom to top. One challenge is when the same random is used in different layers, in which case, special processing is needed. Nevertheless, the benefit of this generalization would be to verify broader types of circuits not restricted to atomic gadgets anymore.

Another direction for future works is to combine the fast verification of IronMask with recent automatic verification tools combining fault and probing attacks on circuits, such as VERICA [71]. The latter tool verifies combined security properties in the fault probing model but relies on the slow verification technique of SILVER. The goal would be to extend the algebraic characterization and verification techniques of IronMask to take faults into account (*i.e.* in the fault probing and fault random probing models), to achieve faster verification in these combined models, inspired by the verification techniques against fault attacks in VERICA.

Chapter 8

Conclusion

During this thesis, we studied the security of masked constructions in the random probing model. We introduced the first composition notion in this model in [18], analogous to the ones in the probing model. Our *random probing composability* notion allows safely composing gadgets while preserving global security in the resulting circuit.

In [18], we also studied a powerful approach, the expansion strategy, which allows for achieving arbitrary levels of security in the random probing model. Inspired by [3], we provided a practical instantiation of the approach using expanding compilers equipped with gadgets instead of MPC techniques as in [3]. We presented the *random probing expandability* notion and showed that any gadget satisfying this notion could be used with an expanding compiler. Indeed, the gadgets' choice affects the expansion's complexity to achieve the desired security level. Namely, we showed that achieving κ bits of security using expansion on a circuit C induces a complexity of $\mathcal{O}(|C| \cdot \kappa^e)$, where $|C|$ is the number of gates in the circuit and e is a parameter which depends on the size of the gadgets in the expanding compiler and the parameter that we referred to as the *amplification order*. The latter is the size of the smallest set of probes that reveal information about the secret inputs and relates to the simulation failure function of the random probing security of the gadgets. The bigger this amplification order and the smaller the size of the gadgets, the lower the complexity exponent e becomes. We provided bounds on the best amplification order achievable. Then, we constructed a 3-share expanding compiler, which outperforms the MPC construction from [3] without achieving the optimal amplification order. Our construction tolerates a leakage rate of $2^{-7.09}$ with a complexity of $\mathcal{O}(|C| \cdot \kappa^{7.5})$, while we showed that the one from [3] tolerates a rate of 2^{-26} with a complexity of $\approx \mathcal{O}(|C| \cdot \kappa^8)$. We finally provided an open-source implementation of our expanding compiler and a proof-of-concept masked AES[1].

After that, we pushed the limits of the expansion strategy and showed that optimal complexities could be achieved in [22]. We tightened the random probing expandability notion, showed its relation to the SNI notion in the probing model, and used this analysis to exhibit generic constructions of addition and copy gadgets achieving the maximal amplification order, using a refresh gadget as a building block. Using this generic construction instantiated with the well-known ISW scheme and a new multiplication gadget we introduced, we could construct 3-share and 5-share expanding compilers, achieving the maximal amplification orders and thus giving much better complexities. Our 3-share compiler tolerates a leakage rate of $2^{-7.5}$ with a complexity of $\mathcal{O}(|C| \cdot \kappa^{3.9})$, while our 5-share compiler tolerates a slightly lower leakage rate $2^{-7.66} \leq p \leq 2^{-9.67}$ with a better complexity of $\mathcal{O}(|C| \cdot \kappa^{3.23})$.

Next, we discussed that the complexity of the expanding compiler decreases as the number of shares n grows, but also the tolerated leakage rate decreases. This motivated us to introduce

[1] `https://github.com/CryptoExperts/poc-expanding-compiler`

the *dynamic random probing expansion strategy* in [23] , which benefits from the best of both worlds: the best asymptotic complexities of constructions for large numbers of shares and the best tolerated leakage rates of constructions for a small and fixed number of shares. We tackled the problem of optimal asymptotic complexity, providing new constructions of linear gadgets with quasi-linear global complexity using the $\mathcal{O}(n \log n)$ refresh gadget, as well as a multiplication gadget that performs a linear number of multiplications. These new constructions gave a new expanding compiler with sub-quadratic values for the complexity exponent e.

We saw that many future works are possible in the random probing model. For instance, we can look for constructions of small gadgets that tolerate the best leakage probability for a fixed number of shares (*e.g.* $n = 3$). Proving bounds on the tolerated leakage rate depending on the structure of the gadgets can also help perform an exhaustive search for such optimal constructions. Also, as stated in Remark 2, we can integrate the tight composition technique from [32] based on probe distribution tables into our expansion strategy to benefit from tighter results but also from the security levels that can be achieved using the expanding compilers. Another example for future works is to look for efficient constructions of the expanding compilers while considering the underlying parameters, such as the used field or the physical defaults of the underlying device.

In a parallel work [21], we introduced a powerful verification tool, IronMask, offering a complete verification technique for most gadgets in the state-of-the-art and covering all security properties in the probing and random probing models. Our technique relies on an algebraic characterization of probes on gadgets with a Gaussian elimination-based approach. We showed that our tool is the fastest verification tool for the random probing model and is competitive with verification tools in the probing model. Future works include extending the algebraic characterization used in IronMask to cover more gadgets and circuits than the current ones. Another direction would be to integrate the combined probing and fault verification techniques from VERICA [71] into IronMask to extend the verification to more extended models. Indeed, VERICA only considers the probing model and relies on techniques from SILVER, which are relatively slow, while a possible approach is to also consider faults in the random probing model and extend the verification technique with the efficient method of IronMask.

The end of this thesis is only the beginning of research in this field. Having proven security levels against side-channel attacks is essential. While current theoretical constructions are still far from being practical to achieve high security levels (such as 128 bits of security), as shown in Section 4.7 on the AES implementation, it still offers a proof-of-concept achieving such levels. However, further research is necessary to ultimately bridge the gap between theory and practice. Verification tools help us achieve this goal by offering a reliable method to validate our constructions. One can even look for constructions directly in the noisy leakage model based on the obtained results on the random probing model.

8.1 On the Practical Usability of Leakage Models

The noisy leakage, random probing, and t-probing models have proven helpful for the community to model side-channel attacks theoretically and provide formal security proofs on masked implementation. Meanwhile, applying these security proofs to real-world implementations is still challenging.

First, the theoretical literature lacks a proper methodology to implement proven secure constructions in the leakage models on a physical device while preserving the proven security levels. Second, these leakage models rely on two assumptions about the physical device for

which a systematic investigation is lacking: the leakage of an elementary operation only depends on its inputs (*i.e.* the *data isolation* assumption), and the leakage's noise of an operation is independent of the previous and following noises (*i.e.* the *noise independence* assumption).

The first assumption (data isolation) can be easily broken, for instance, due to physical effects on a device. In particular, transitions occurring on memory buses or CPU registers between a previously processed value x_{i-1} and the current one x_i usually leak some information correlated to $x_{i-1} \oplus x_i$, which violates the data isolation principle [39, 5]. On the hardware level, glitches further make the successive gates' leakages mutually dependent on their respective inputs [64, 65, 66]. On the software level, CPU synchronization limits, but does not eliminate, the issue of glitches. These issues can be avoided by adding registers and controlling transitions [49, 34] in hardware, and by trying to avoid transitions using assembly programming tricks in software [50, 29, 26]. However, these techniques still rely on abstract models for the leakage, and current techniques in the literature test the data isolation assumption only indirectly, by estimating the statistical security order of an implementation [11, 73].

As for the second assumption (noise independence), the noise in the side channel leakage of a device is multivariate, and the noises occurring during successive operations likely include some dependency. This assumption is only currently studied at a high level in a few works [57, 36].

In [24], we tackle the above issues by proposing a complete methodology to transform abstract circuits into physical implementations secure against side-channel attacks. For this purpose, we rely on a random probing compiler and discuss the concrete steps to use the reduction from the noisy leakage model to the random probing model on physical implementations. While this reduction is well-studied in theory, our methodology summarizes all hypotheses that must be met in practice and identifies technical challenges that must be overcome to achieve formally secure circuits on physical devices.

Then, we propose new tools to solve these technical challenges. Namely,

- we explain how to enforce the data isolation assumption and introduce a novel practical test for its validation on a physical implementation. Our test offers a direct approach, contrasting with existing methods in the literature. We conduct experiments on a real target, an `STM32F3` MCU, using NewAE's ChipWhisperer-Lite CW1173 board. While our test does not provide a formal proof for the assumption, it stands as the first literature instance of directly addressing and validating this hypothesis with a practical, dedicated procedure,

- we also offer a method to integrate the noise independence assumption into the analysis, making it possible to quantify the loss of security implied by a lack of independence. We specifically discuss a relaxation of the assumption aiming to split the noise occurring during the execution of the algorithm into independent noises on each of the operations. We first show a trivial way of doing the split and then express it as a constrained optimization problem that better scales with the size of the circuit. We propose a direct non-optimal solution to the problem and leave the question of optimally and efficiently solving it as an open problem.

Finally, we highlight the design goals that this security reduction involves. We also exhibit the remaining limitations and open problems of the practical usability of the leakage models. Our goal is to show that it is possible to bridge theory and practice and to motivate further research on remaining issues to fully close the gap, that is to get practical implementations

proven secure against side-channel attacks on a physical device without any ideal assumption about the leakage.

Bibliography

[1] Miklós Ajtai. Secure computation with information leaking to an adversary. In Lance Fortnow and Salil P. Vadhan, editors, *43rd Annual ACM Symposium on Theory of Computing*, pages 715–724, San Jose, CA, USA, June 6–8, 2011. ACM Press.

[2] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 191–219, Hanoi, Vietnam, December 4–8, 2016. Springer, Heidelberg, Germany.

[3] Prabhanjan Ananth, Yuval Ishai, and Amit Sahai. Private circuits: A modular approach. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 427–455, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

[4] Marcin Andrychowicz, Stefan Dziembowski, and Sebastian Faust. Circuit compilers with $O(1/\log(n))$ leakage rate. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 586–615, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.

[5] Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the cost of lazy engineering for masked software implementations. In Marc Joye and Amir Moradi, editors, *Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers*, volume 8968 of *Lecture Notes in Computer Science*, pages 64–81. Springer, 2014.

[6] Gilles Barthe, Sonia Belaïd, Gaëtan Cassiers, Pierre-Alain Fouque, Benjamin Grégoire, and François-Xavier Standaert. maskVerif: Automated verification of higher-order masking in presence of physical defaults. In Kazue Sako, Steve Schneider, and Peter Y. A. Ryan, editors, *ESORICS 2019: 24th European Symposium on Research in Computer Security, Part I*, volume 11735 of *Lecture Notes in Computer Science*, pages 300–318, Luxembourg, September 23–27, 2019. Springer, Heidelberg, Germany.

[7] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Improved parallel mask refreshing algorithms: generic solutions with parametrized non-interference and automated optimizations. *Journal of Cryptographic Engineering*, 10(1):17–26, April 2020.

[8] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Improved parallel mask refreshing algorithms: generic solutions with parametrized non-interference and automated optimizations. *J. Cryptogr. Eng.*, 10(1):17–26, 2020.

[9] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified proofs of higher-order masking. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 457–485, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

[10] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 116–129, Vienna, Austria, October 24–28, 2016. ACM Press.

[11] Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Parallel implementations of masking schemes and the bounded moment leakage model. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 535–566, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany.

[12] Gilles Barthe, Marc Gourjon, Benjamin Grégoire, Maximilian Orlt, Clara Paglialonga, and Lars Porth. Masking in fine-grained leakage models: Construction, implementation and verification. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(2):189–228, 2021. `https://tches.iacr.org/index.php/TCHES/article/view/8792`.

[13] Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems – CHES 2016*, volume 9813 of *Lecture Notes in Computer Science*, pages 23–39, Santa Barbara, CA, USA, August 17–19, 2016. Springer, Heidelberg, Germany.

[14] Alberto Battistello, Jean-Sebastien Coron, Emmanuel Prouff, and Rina 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`.

[15] Ali Galip Bayrak, Francesco Regazzoni, David Novo, and Paolo Ienne. Sleuth: Automated verification of software power analysis countermeasures. In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems – CHES 2013*, volume 8086 of *Lecture Notes in Computer Science*, pages 293–310, Santa Barbara, CA, USA, August 20–23, 2013. Springer, Heidelberg, Germany.

[16] Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Randomness complexity of private circuits for multiplication. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 616–648, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.

[17] Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Private multiplication over finite fields. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 397–426, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[18] Sonia Belaïd, Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Abdul Rahman Taleb. Random probing security: Verification, composition, expansion and new constructions. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part I*, volume 12170 of *Lecture Notes in Computer Science*, pages 339–368, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany.

[19] Sonia Belaïd, Pierre-Évariste Dagand, Darius Mercadier, Matthieu Rivain, and Raphaël Wintersdorff. Tornado: Automatic generation of probing-secure masked bitsliced implementations. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 311–341, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany.

[20] Sonia Belaïd, Dahmun Goudarzi, and Matthieu Rivain. Tight private circuits: Achieving probing security with the least refreshing. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 343–372, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany.

[21] Sonia Belaïd, Darius Mercadier, Matthieu Rivain, and Abdul Rahman Taleb. Ironmask: Versatile verification of masking security. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 142–160. IEEE, 2022.

[22] Sonia Belaïd, Matthieu Rivain, and Abdul Rahman Taleb. On the power of expansion: More efficient constructions in the random probing model. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 313–343, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany.

[23] Sonia Belaïd, Matthieu Rivain, Abdul Rahman Taleb, and Damien Vergnaud. Dynamic random probing expansion with quasi linear asymptotic complexity. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021, Part II*, volume 13091 of *Lecture Notes in Computer Science*, pages 157–188, Singapore, December 6–10, 2021. Springer, Heidelberg, Germany.

[24] Sonia Belaïd, Gaëtan Cassiers, Camille Mutschler, Matthieu Rivain, Thomas Roche, François-Xavier Standaert, and Abdul Rahman Taleb. A methodology to achieve provable side-channel security in real-world implementations. Cryptology ePrint Archive, Paper 2023/1198, 2023. `https://eprint.iacr.org/2023/1198`.

[25] Sonia Belaïd, Gaëtan Cassiers, Matthieu Rivain, and Abdul Rahman Taleb. Unifying freedom and separation for tight probing-secure composition. Cryptology ePrint Archive, Paper 2023/835, 2023. `https://eprint.iacr.org/2023/835`.

[26] Roderick Bloem, Barbara Gigerl, Marc Gourjon, Vedad Hadzic, Stefan Mangard, and Robert Primas. Power contracts: Provably complete power leakage models for processors. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on Computer and Communications Security*, pages 381–395, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.

[27] Roderick Bloem, Hannes Groß, Rinat Iusupov, Bettina Könighofer, Stefan Mangard, and Johannes Winter. Formal verification of masked hardware implementations in the presence of glitches. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 321–353, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

[28] Nicolas Bordes and Pierre Karpman. Fast verification of masking schemes in characteristic two. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 283–312, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany.

[29] Olivier Bronchain and Gaëtan Cassiers. Bitslicing arithmetic/boolean masking conversions for fun and profit with application to lattice-based KEMs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):553–588, 2022.

[30] Olivier Bronchain and François-Xavier Standaert. Breaking masked implementations with many shares on 32-bit software platforms: Or when the security order does not matter. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 202–234, 2021.

[31] Claude Carlet, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Algebraic decomposition for probing security. Cryptology ePrint Archive, Report 2016/321, 2016. `https://eprint.iacr.org/2016/321`.

[32] Gaëtan Cassiers, Sebastian Faust, Maximilian Orlt, and François-Xavier Standaert. Towards tight random probing security. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part III*, volume 12827 of *Lecture Notes in Computer Science*, pages 185–214, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany.

[33] Gaëtan Cassiers and François-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IEEE Trans. Inf. Forensics Secur.*, 15:2542–2555, 2020.

[34] Gaëtan Cassiers and François-Xavier Standaert. Provably secure hardware masking in the transition- and glitch-robust probing model: Better safe than sorry. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(2):136–158, 2021. `https://tches.iacr.org/index.php/TCHES/article/view/8790`.

[35] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Heidelberg, Germany.

[36] Marios O Choudary. Efficient multivariate statistical techniques for extracting secrets from electronic devices. Technical report, University of Cambridge, Computer Laboratory, 2015.

[37] Gil Cohen, Ivan Bjerre Damgård, Yuval Ishai, Jonas Kölker, Peter Bro Miltersen, Ran Raz, and Ron D. Rothblum. Efficient multiparty protocols via log-depth threshold formulae - (extended abstract). In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 185–202, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

[38] Jean-Sébastien Coron. Formal verification of side-channel countermeasures via elementary circuit transformations. In Bart Preneel and Frederik Vercauteren, editors, *ACNS 18: 16th International Conference on Applied Cryptography and Network Security*, volume 10892 of *Lecture Notes in Computer Science*, pages 65–82, Leuven, Belgium, July 2–4, 2018. Springer, Heidelberg, Germany.

[39] Jean-Sébastien Coron, Christophe Giraud, Emmanuel Prouff, Soline Renner, Matthieu Rivain, and Praveen Kumar Vadnala. Conversion of security proofs from one leakage model to another: A new issue. In Werner Schindler and Sorin A. Huss, editors, *Constructive Side-Channel Analysis and Secure Design - Third International Workshop, COSADE 2012, Darmstadt, Germany, May 3-4, 2012. Proceedings*, volume 7275 of *Lecture Notes in Computer Science*, pages 69–81. Springer, 2012.

[40] Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-order side channel security and mask refreshing. In Shiho Moriai, editor, *Fast Software Encryption – FSE 2013*, volume 8424 of *Lecture Notes in Computer Science*, pages 410–424, Singapore, March 11–13, 2014. Springer, Heidelberg, Germany.

[41] Jean-Sebastien Coron, Franck Rondepierre, and Rina Zeitoun. High order masking of look-up tables with common shares. Cryptology ePrint Archive, Report 2017/271, 2017. `https://eprint.iacr.org/2017/271`.

[42] Jean-Sébastien Coron and Lorenzo Spignoli. Secure wire shuffling in the probing model. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part III*, volume 12827 of *Lecture Notes in Computer Science*, pages 215–244, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany.

[43] Ivan Damgård and Marcel Keller. Secure multiparty AES. In Radu Sion, editor, *FC 2010: 14th International Conference on Financial Cryptography and Data Security*, volume 6052 of *Lecture Notes in Computer Science*, pages 367–374, Tenerife, Canary Islands, Spain, January 25–28, 2010. Springer, Heidelberg, Germany.

[44] Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Masking AES with d+1 shares in hardware. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems – CHES 2016*, volume 9813 of *Lecture Notes in Computer Science*, pages 194–212, Santa Barbara, CA, USA, August 17–19, 2016. Springer, Heidelberg, Germany.

[45] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 423–440, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.

[46] Alexandre Duc, Sebastian Faust, and François-Xavier Standaert. Making masking security proofs concrete - or how to evaluate the security of any leaking device. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 401–429, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

[47] Stefan Dziembowski, Sebastian Faust, and Karol Zebrowski. Simple refreshing in the noisy leakage model. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019, Part III*, volume 11923 of *Lecture Notes in Computer Science*, pages 315–344, Kobe, Japan, December 8–12, 2019. Springer, Heidelberg, Germany.

[48] Hassan Eldib, Chao Wang, and Patrick Schaumont. Formal verification of software countermeasures against side-channel attacks. *ACM Trans. Softw. Eng. Methodol.*, 24(2):11:1–11:24, 2014.

[49] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable masking schemes in the presence of physical defaults & the robust probing model. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3):89–120, 2018. `https://tches.iacr.org/index.php/TCHES/article/view/7270`.

[50] Si Gao, Ben Marshall, Dan Page, and Elisabeth Oswald. Share-slicing: Friend or foe? *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(1):152–174, 2019. `https://tches.iacr.org/index.php/TCHES/article/view/8396`.

[51] Barbara Gigerl, Vedad Hadzic, Robert Primas, Stefan Mangard, and Roderick Bloem. Coco: Co-design and co-verification of masked software implementations on CPUs. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021: 30th USENIX Security Symposium*, pages 1469–1468. USENIX Association, August 11–13, 2021.

[52] Louis Goubin and Jacques Patarin. DES and differential power analysis (the "duplication" method). In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES'99*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172, Worcester, Massachusetts, USA, August 12–13, 1999. Springer, Heidelberg, Germany.

[53] Dahmun Goudarzi, Thomas Prest, Matthieu Rivain, and Damien Vergnaud. Probing security through input-output separation and revisited quasilinear masking. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(3):599–640, 2021. `https://tches.iacr.org/index.php/TCHES/article/view/8987`.

[54] Dahmun Goudarzi and Matthieu Rivain. How fast can higher-order masking be in software? In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 567–597, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany.

[55] Benjamin Grégoire, Kostas Papagiannopoulos, Peter Schwabe, and Ko Stoffelen. Vectorizing higher-order masking. In Junfeng Fan and Benedikt Gierlichs, editors, *COSADE 2018: 9th International Workshop on Constructive Side-Channel Analysis and Secure Design*, volume 10815 of *Lecture Notes in Computer Science*, pages 23–43, Singapore, April 23–24, 2018. Springer, Heidelberg, Germany.

[56] Hannes Groß, Ko Stoffelen, Lauren De Meyer, Martin Krenn, and Stefan Mangard. First-order masking with only two random bits. In Begül Bilgin, Svetla Petkova-Nikova, and Vincent Rijmen, editors, *Proceedings of ACM Workshop on Theory of Implementation Security Workshop, TIS@CCS 2019, London, UK, November 11, 2019*, pages 10–23. ACM, 2019.

[57] Vincent Grosso and François-Xavier Standaert. Masking proofs are tight and how to exploit it in security evaluations. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 385–412, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

[58] Martin Hirt and Ueli M. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13(1):31–60, January 2000.

[59] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume

2729 of *Lecture Notes in Computer Science*, pages 463–481, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany.

[60] Anthony Journault and François-Xavier Standaert. Very high order masking: Efficient implementation and security evaluation. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, volume 10529 of *Lecture Notes in Computer Science*, pages 623–643, Taipei, Taiwan, September 25–28, 2017. Springer, Heidelberg, Germany.

[61] Pierre Karpman and Daniel S. Roche. New instantiations of the CRYPTO 2017 masking schemes. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 285–314, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany.

[62] David Knichel, Pascal Sasdrich, and Amir Moradi. SILVER - statistical independence and leakage verification. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part I*, volume 12491 of *Lecture Notes in Computer Science*, pages 787–816, Daejeon, South Korea, December 7–11, 2020. Springer, Heidelberg, Germany.

[63] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113, Santa Barbara, CA, USA, August 18–22, 1996. Springer, Heidelberg, Germany.

[64] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-channel leakage of masked CMOS gates. In Alfred Menezes, editor, *Topics in Cryptology – CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365, San Francisco, CA, USA, February 14–18, 2005. Springer, Heidelberg, Germany.

[65] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-channel leakage of masked CMOS gates. In Alfred Menezes, editor, *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2005.

[66] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully attacking masked AES hardware implementations. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2005.

[67] Ueli M. Maurer. Secure multi-party computation made simple (invited talk). In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN 02: 3rd International Conference on Security in Communication Networks*, volume 2576 of *Lecture Notes in Computer Science*, pages 14–28, Amalfi, Italy, September 12–13, 2003. Springer, Heidelberg, Germany.

[68] Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 278–296, Cambridge, MA, USA, February 19–21, 2004. Springer, Heidelberg, Germany.

[69] Andrew Moss, Elisabeth Oswald, Dan Page, and Michael Tunstall. Compiler assisted masking. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems – CHES 2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 58–75, Leuven, Belgium, September 9–12, 2012. Springer, Heidelberg, Germany.

[70] Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 142–159, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany.

[71] Jan Richter-Brockmann, Jakob Feldtkeller, Pascal Sasdrich, and Tim Güneysu. VERICA - verification of combined attacks automated formal verification of security against simultaneous information leakage and tampering. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):255–284, 2022.

[72] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems – CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427, Santa Barbara, CA, USA, August 17–20, 2010. Springer, Heidelberg, Germany.

[73] Tobias Schneider and Amir Moradi. Leakage assessment methodology - A clear roadmap for side-channel evaluations. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware and Embedded Systems – CHES 2015*, volume 9293 of *Lecture Notes in Computer Science*, pages 495–513, Saint-Malo, France, September 13–16, 2015. Springer, Heidelberg, Germany.

[74] Kai Schramm and Christof Paar. Higher order masking of the AES. In David Pointcheval, editor, *Topics in Cryptology – CT-RSA 2006*, volume 3860 of *Lecture Notes in Computer Science*, pages 208–225, San Jose, CA, USA, February 13–17, 2006. Springer, Heidelberg, Germany.

[75] Jun Zhang, Pengfei Gao, Fu Song, and Chao Wang. Scinfer: Refinement-based verification of software countermeasures against side-channel attacks. In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II*, volume 10982 of *Lecture Notes in Computer Science*, pages 157–177. Springer, 2018.

Appendices

A.1 Proof of Lemma 13

Recall the procedure of the gadget. We consider that we have an n-share (t, f')-TRPE refresh gadget G_{refresh} achieving the amplification order $d \geq \min(t+1, n-t)$. First, the gadget G_{mult} performs n executions of the gadget G_{refresh} on the input sharing $(b_1, \ldots, b_n)$ to produce:

$$\begin{array}{rcl} (b_1^{(1)}, \ldots, b_n^{(1)}) & \leftarrow & G_{\text{refresh}}(b_1, \ldots, b_n) \\ & \cdots & \\ (b_1^{(n)}, \ldots, b_n^{(n)}) & \leftarrow & G_{\text{refresh}}(b_1, \ldots, b_n) \end{array}$$

then, the gadget constructs the matrix of the cross product of input shares using the refreshed input shares of b:

$$M = \begin{pmatrix} a_1 \cdot b_1^{(1)} & a_1 \cdot b_2^{(1)} & \cdots & a_1 \cdot b_n^{(1)} \\ a_2 \cdot b_1^{(2)} & a_2 \cdot b_2^{(2)} & \cdots & a_2 \cdot b_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ a_n \cdot b_1^{(n)} & a_n \cdot b_2^{(n)} & \cdots & a_n \cdot b_n^{(n)} \end{pmatrix}.$$

Then, it picks n^2 random values which define the following matrix:

$$R = \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,n} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{n,1} & r_{n,2} & \cdots & r_{n,n} \end{pmatrix}.$$

It then performs an element-wise addition between the matrices M and R:

$$P = M + R = \begin{pmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,n} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n,1} & p_{n,2} & \cdots & p_{n,n} \end{pmatrix}.$$

At this point, the gadget randomized each product of input shares from the matrix M with a single random value from R. In order to generate the correct output, the gadget adds all the columns of P into a single column V of n elements, and adds all the columns of the transpose matrix R^{T} into a single column X of n elements:

$$V = \begin{pmatrix} p_{1,1} + \cdots + p_{1,n} \\ p_{2,1} + \cdots + p_{2,n} \\ \vdots \\ p_{n,1} + \cdots + p_{n,n} \end{pmatrix}, \qquad X = \begin{pmatrix} r_{1,1} + \cdots + r_{n,1} \\ r_{1,2} + \cdots + r_{n,2} \\ \vdots \\ r_{1,n} + \cdots + r_{n,n} \end{pmatrix}$$

The n-share output is finally defined as $(c_1, \ldots, c_n)^T = V + X$ such that

$$\begin{aligned} c_1 &= V_1 + X_1 \\ &\ldots \\ c_n &= V_n + X_n. \end{aligned}$$

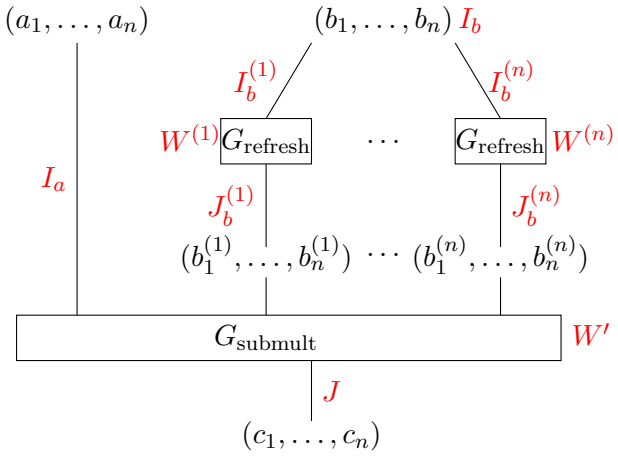

Figure A.1: G_{mult} gadget from Section 5.4.

Figure A.1 represents the G_{mult} gadget from a high-level, composed of several blocks. First, a refresh gadget G_{refresh} is executed n independent times on the input sharing of b to produce n fresh copies $b^{(1)}$, ..., $b^{(n)}$. Then, the gadget G_{submult} takes as input $(a_1, \ldots, a_n)$ and the outputs of the refreshing gadgets $b^{(1)}$, ..., $b^{(n)}$ to produce the output of G_{mult}.

In the following proofs, we will denote W to be any set of probes on the global gadget G_{mult}, then W can be split as $W = W' \cup W^{(1)} \cup \ldots \cup W^{(n)}$ where $W^{(i)}$ is the set of probes on the internal wires of the execution of G_{refresh} for the fresh sharing $b^{(i)}$ of b, and W' is the set of probes on the internal wires of G_{submult}. We will also denote J to be any set of output wires of G_{mult} (which are the output wires of G_{submult}), and $J_b^{(i)}$ (resp. $I_b^{(i)}$) any set of output wires (resp. input wires) of the execution of G_{refresh} for the fresh sharing $b^{(i)}$ of b. Observe that any probe on the output wires of G_{refresh} for any sharing $b^{(i)}$ can be obtained through internal probes in W' on G_{submult}, so in the beginning we always consider that $J_b^{(i)} = \emptyset$ for all $i \in [n]$.

Observe that any probe in the set W' on the internal wires of G_{submult} is of one of the following forms:

(a) a_i, $b_j^{(i)}$, $a_i \cdot b_j^{(i)}$, $r_{i,j}$, $p_{i,j} = a_i \cdot b_j^{(i)} + r_{i,j}$,

(b) $V_{i,j}$ partial sum of the first j terms of V_i. Observe that $V_{i,n} = V_i$,

(c) $X_{i,j}$ partial sum of the first j terms of X_i. Observe that $X_{i,n} = X_i$.

Also observe that each random value $r_{i,j}$ only appears in the expression of the wires $r_{i,j}$, $p_{i,j}$, $V_{i,j}$, or $X_{j,i}$ (so also $c_i = V_i + X_i$ and $c_j = V_j + X_j$), and does not appear anywhere else in the wires.

We will first start by proving some simple claim.

Claim 1. *Let J be a set of output shares of G_{mult} and $W = W' \cup W^{(1)} \cup \ldots \cup W^{(n)}$ be a set of leaking wires as described above such that $|J| + |W'| \leq n - 1$ (we only consider the set W' of probes on the internal wires to $G_{submult}$). Then, for any $i \in J$ such that $V_{i,j} \notin W$ for any $j \in [n]$, the output wire c_i can be perfectly simulated by generating a uniform random value without knowing any of the input shares.*

Proof. Let $i \in J$ such that $V_{i,j} \notin W'$ for any $j \in [n]$. Then we know that the expression of V_i in $c_i = V_i + X_i$ contains $n-1$ random values since $V_i = p_{i,1} + \ldots + p_{i,n}$ and each $p_{i,j} = a_i \cdot b_j^{(i)} + r_{i,j}$ (without counting the random $r_{i,i}$ because it is cancelled out in c_i as it appears in V_i and X_i and $c_i = V_i + X_i$). Observe that each random value $r_{i,k}$ in V_i appears in exactly one other output share $c_k = V_k + X_k$ that comes from the expression of $X_k = r_{1,k} + \ldots + r_{i,k} + \ldots + r_{n,k}$. In other terms, each output share c_k has exactly one random value in common with V_i in c_i. Then, by probing $|J|$ output shares in J including c_i, there are at least $n - |J|$ remaining random values in V_i that do not appear in any other expression of the output shares. In addition, observe that any probed variable in W' can have in its expression at most one random value in common with V_i (because each random value $r_{i,j}$ appears exactly once in each of the wires $p_{i,j}$, $r_{i,j}$ or X_j). Then, since $|W'| \leq n - |J| - 1$ (because $|J| + |W'| \leq n - 1$), there is at least $n - |J| - (n - |J| - 1) = 1$ remaining random value $r_{i,\ell}$ where $\ell \in [n]$ in V_i, that does not appear in any other expression of the probed values in W' or J. So $c_i = V_i + X_i$ can be perfectly simulated by generating the uniform random value $r_{i,\ell}$, which concludes the proof. □

In the following, we will separately prove the TRPE1 then the TRPE2 property on G_{mult} via Lemmas 24 and 27 to demonstrate Lemma 13.

A.1.1 Proof for TRPE1 property

Lemma 24. *The multiplication gadget G_{mult} is (t, f_1)-TRPE1 of amplification order $d = \min(t+1, n-t)$*

Proof. We proceed in two steps through the following two lemmas 25 and 26, considering the leaking wires in two distinct ranges.

Lemma 25. *Let J be a set of at most t output shares of G_{mult}. Let W be a set of leaking wires as described above such that $|W| \leq d - 1 \leq t$. Then W and J can be perfectly simulated from at most $\min(t, |W|) = |W|$ shares of each of the inputs a and b.*

Proof. Let J be the set of t output shares of G_{mult} (i.e of G_{submult}), and let $W = W' \cup W^{(1)} \cup \ldots \cup W^{(n)}$ with $|W| \leq d-1 \leq t$ be the set of probes on the global gadget G_{mult} and decomposed as explained earlier. We organize the proof in two steps:

1. We first identify the set of input shares I_a and the sets $J_b^{(i)}$ for $i \in [n]$ which are necessary to perfectly simulate J and W' in G_{submult}.

2. Then, we show that we can perfectly simulate the sets $J_b^{(i)}$ and $W^{(i)}$ for $i \in [n]$ using the simulator of the gadget G_{refresh}. This will determine the sets $I_b^{(i)}$ necessary for each of the n simulations of G_{refresh}, and thus determine the set I_b of input shares on b as $I_b = I_b^{(1)} \cup \ldots \cup I_b^{(n)}$.

Using I_b, we will be able to perfectly simulate $J_b^{(i)}$ and $W^{(i)}$ for $i \in [n]$. Then using I_a and $J_b^{(i)}$ for $i \in [n]$, we will be able to perfectly simulate W' and J. This will lead to a perfect simulation of all probes W and output shares in J on the global gadget G_{mult}.

We first start by constructing the set of input shares indices I_a and the sets $J_b^{(k)}$ for $k \in [n]$ depending on the probes in the set W' as follows[1]:

(a) For probes of form (a), we add index i to I_a, and index j to $J_b^{(k)}$ for $k \in [n]$.

[1]We consider that all $J_b^{(k)}$ are empty at first since all the output shares of G_{refresh} can be probed directly in W'.

(b) For probes of form (b), we add index i to I_a and to $J_b^{(k)}$ for $k \in [n]$.

(c) For probes of form (c), we add index i to $J_b^{(k)}$ for $k \in [n]$.

Observe that since $|W| \leq d-1$, then in particular $|W'| \leq d-1 \leq t$, then $|I_a| \leq |W'| \leq |W| \leq \min(t, |W|)$ so we have no failure on the input a. We also have $|J_b^{(k)}| \leq |W'| \leq t$.

Simulation of W': probes of the form (a) can be perfectly simulated from the corresponding input shares in I_a and $J_b^{(k)}$, and by generating uniformly random values $r_{i,j}$ when necessary. Probes of the form (c) are also perfectly simulated by simply generating uniformly random values, since $X_{i,j} = r_{1,i} + \ldots + r_{j,i}$. As for probes of the form (b), we know that $i \in I_a$ and $i \in J_b^{(i)}$, then we look at each of the terms $p_{i,j'}$ for $j' \in [j]$ in $V_{i,j} = p_{i,1} + \ldots p_{i,j}$. In particular, if $j \geq i$, the term $p_{i,i}$ is in the partial sum $V_{i,j}$ and is perfectly simulated using the input shares a_i and $b_i^{(i)}$ and by generating the random value $r_{i,i}$. Next, for each $p_{i,j'}$ such that $j' \neq i$, if $j' \in J_b^{(i)}$, then $p_{i,j'}$ can be perfectly simulated from the corresponding input shares and by generating uniformly at random $r_{i,j'}$. Otherwise, if $j' \notin J_b^{(i)}$, then that means that the wires $p_{i,j'}, r_{i,j'}$ and $X_{j'}$ are not probed in W' because otherwise j' would have been added to all $J_b^{(k)}$ for $k \in [n]$. Since the random value $r_{i,j'}$ only appears in the expression of the wires $p_{i,j'}, r_{i,j'}$ and $X_{j'}$ (besides $V_{i,j}$ which is already probed), and of the output wire $c_{j'} = V_{j'} + X_{j'}$, we need to consider two cases:

- $j' \notin J$: in this case, the random value $r_{i,j'}$ can be used to mask the expression of $p_{i,j'}$ in the partial sum $V_{i,j}$, perfectly simulating it without the need to the share $b_{j'}^{(i)}$.
- $j' \in J$: $c_{j'} = V_{j'} + X_{j'}$, and $r_{i,j'}$ is the one of the summed terms in the expression of $X_{j'}$. We know that $V_{j',k} \notin W'$ for any $k \in [n]$ since otherwise j' would have been added to $J_b^{(i)}$. Since in addition we have $|J| + |W| \leq t + d - 1 \leq t + n - t - 1 \leq n - 1$, by claim 1, the output share c'_j can be masked by some random value $r_{j',\ell}$. Thus, $X_{j'}$ is masked and $r_{i,j'}$ does not appear anymore in $c_{j'}$. So $r_{i,j'}$ can be used to mask the expression of $p_{i,j'}$ in the partial sum $V_{i,j}$. This brings us to a perfect simulation of $p_{i,j'}$ simply by generating at random $r_{i,j'}$.

By perfectly simulating each of the terms $p_{i,j'}$ for $j' \in [j]$ in the probed wire $V_{i,j}$ independently, we can perfectly simulate their sum and thus perfectly simulate $V_{i,j}$. This brings us to a perfect simulation of the set W'.

Simulation of J: Let $i \in J$.

- if $V_{i,j} \notin W'$ for any $j \in [n]$, then by claim 1, c_i is perfectly simulated by simply generating a uniform random value $r_{i,\ell}$ for some $\ell \in [n]$.
- if $V_{i,j} \in W'$ for at least one $j \in [n]$, then let $V_{i,j'}$ be the largest of the probed partial sums. All of the partial sums including $V_{i,j'}$ are perfectly simulated as described earlier. Then, let us consider $c_i + V_{i,j'} = p_{i,j'+1} + \ldots + p_{i,n} + X_i$. The wire X_i can be perfectly simulated by generating uniform random values. As for each of the terms $p_{i,j'+1}, \ldots, p_{i,n}$, they can each be perfectly simulated in the exact same way each of the terms in $V_{i,j'}$ are simulated independently.

  In the particular case where $j' \leq i$ then the term $p_{i,i} = a_i \cdot b_i^{(i)} + r_{i,i}$ appears in the expression of $c_i + V_{i,j'}$, and in this case, the random value $r_{i,i}$ is cancelled out in the expression of $c_i + V_{i,j'}$ since it appears in both $p_{i,i}$ and X_i, and $c_i + V_{i,j'} = p_{i,j'+1} + \ldots + p_{i,i} + \ldots + p_{i,n} + X_i$. So to simulate the term $p_{i,i}$ in $c_i + V_{i,j'}$ we need both input shares a_i and $b_i^{(i)}$. This is already the case by construction because we assume that $V_{i,j'} \in W'$.

Thus, by perfectly simulating $V_{i,j'}$ and $c_i + V_{i,j'}$, the output share c_i is also perfectly simulated.

Also, since $|J_b^{(k)}| \leq |W'| \leq t$ and $|W^{(k)}| \leq d-1$, and since G_{refresh} is (t, f')-TRPE achieving the amplification order d, then we can perfectly simulate sets $J_b^{(k)}$ and $W^{(k)}$ from the set of input shares $I_b^{(k)}$ such that $|I_b^{(k)}| \leq |W^{(k)}| \leq t$ for $k \in [n]$. Thus, we can let $I_b = I_b^{(1)} \cup \ldots \cup I_b^{(n)}$ and we have $|I_b| \leq |W^{(1)}| + \ldots + |W^{(n)}| \leq |W| \leq \min(|W|, t)$, so we have no failure on the input b either. Until now, we have shown that we can simulate all sets $W^{(k)}$ and $J_b^{(k)}$ from I_b of size at most $\min(|W|, t)$. It remains to show that we can also perfectly simulate the sets W' and J from I_a and $J_b^{(k)}$ for $k \in [n]$.

We have shown that we can perfectly simulate any set of t output shares J and any set of probes W of size at most $d-1$, with at most $\min(|W|, t)$ shares of each of the inputs a and b. This concludes the proof of Lemma 25. □

Remark 8. *We can observe that for this lemma to apply on G_{mult}, we do not need the pre-processing phase of the refresh on input b. In fact, we can see that during the construction of the sets $J_b^{(k)}$, we add each index to all of the sets for all $k \in [n]$. However, executing n refreshings on the input b is necessary to prove the next result, specifically when we consider W such that $d \leq |W| \leq 2d-1$.*

To get back to the proof of Lemma 24, we also need the following result.

Lemma 26. *Let J be a set of at most t output shares of G_{mult}. Let W be a set of leaking wires as described above such that $d \leq |W| \leq 2d-1$. Then W and J can be perfectly simulated from the sets of input shares I_a and I_b such that $|I_a| \leq \min(|W|, t)$ **or** $|I_b| \leq \min(|W|, t)$. In other terms, we have a simulation failure on at most one of the inputs a or b.*

Proof. Recall that the set W can be split into subsets $W = W' \cup W^{(1)} \cup \ldots \cup W^{(n)}$ as described above. We can consider two cases.

Case 1: $|\mathbf{W'}| \leq \mathbf{d-1}$. This case is similar to the case of Lemma 25, so we can construct the set I_a in the same way as in the proof of Lemma 25, and we can eventually consider $I_b = [n]$. We know that $|I_a| \leq |W'| \leq d-1 \leq t$, so there is no failure on the input a. And all probes in W' can be simulated like in the proof of Lemma 25 with I_a and trivially with $I_b = [n]$. Also, all probes in $W^{(1)} \cup \ldots \cup W^{(n)}$ can be trivially simulated since we have access to the full input b. As for output shares in J, whenever $i \in J \cap I_a$, then $c_i = V_i + X_i$ is easily simulated using $I_b = [n]$. If $i \in J$ but $i \notin I_a$, then $V_{i,j} \notin W'$ for any $j \in [n]$ and since $|J| + |W'| \leq t + d - 1 \leq t + n - t - 1 \leq n - 1$, c_i is perfectly simulated by a single random value thanks to claim 1. Thus, W and J are perfectly simulated with at most $|W'| \leq \min(|W|, t)$ shares of a and eventually n shares of b.

Case 2: $|\mathbf{W'}| \geq \mathbf{d}$ (and thus $|W^{(1)} \cup \ldots \cup W^{(n)}| \leq d-1$). In this case, we will construct the sets I_a and $J_b^{(k)}$ from empty sets, in a way that we will have a simulation failure on at most one of the inputs a or b, and we will be able to perfectly simulate W' and output shares in J using I_a and $J_b^{(k)}$. We will also show how to perfectly simulate all $J_b^{(k)}$ and $W^{(k)}$ using a set of input shares I_b.

First, we construct the sets I_a and $J_b^{(k)}$ depending on the probes in W' as follows:

(a) For probes of form (a), we add index i to I_a, and index j only to $J_b^{(i)}$.

(b) For probes of form (b), we add index i to I_a and only to $J_b^{(i)}$.

(c) For probes of form (c), we add index i to $J_b^{(k)}$ for all $k \in [n]$.

In the rest of the proof, we will show that if we have a failure on one of the inputs, we can still perfectly simulate W and J without a failure on the other input. In this purpose, we will consider two cases: in the first case (2.1), we will have a failure on input a (*i.e.*, more than $\min(t, |W|)$ shares of a are added to I_a) and in the second case (2.2), we won't have a failure on input a, and so we will eventually have a failure on input b.

Case 2.1: Simulation failure on input a. Notice that by construction we always have $|I_a| \leq |W'| \leq |W|$. Thus, a simulation failure on input a for TRPE1 means that the set I_a is of size $|I_a| \geq t+1 \geq d$. We will first start by showing that the sets $W^{(k)}$ and $J_b^{(k)}$ can be perfectly simulated using the simulator of G_{refresh} without a failure on the input b. Next, we will show that W' and output shares in J can be perfectly simulated using I_a and $J_b^{(k)}$.

Since we only add shares indices to I_a when we have probes of the form (a) or (b), this means that we have at least $t+1$ probes of these two forms with $t+1$ different values for the index i. In addition, since we have at least $t+1$ probes (a) or (b) with distinct values for the index i, then this also means that each of the sets $J_b^{(i)}$ built from these probes has at most one share of $b^{(i)}$ added to it by construction. In other terms, when we only consider probes of the form (a) and (b) with distinct i, we have $|J_b^{(k)}| \leq 1$ for each $k \in [n]$.

Now let us consider the remaining probes in W which are either in W' of the form (c), in W' of the form $(a)/(b)$ for which $i \in I_a$ or in $W^{(1)} \cup \ldots \cup W^{(n)}$. Since $|I_a| \geq t+1 \geq d$, then there are at most $d-1$ of these remaining probes. Without loss of generality, we consider that there are exactly $d-1$ instead of at most $d-1$ probes. Let m be the number of probes in $W^{(1)} \cup \ldots \cup W^{(n)}$ and $d-1-m$ the remaining in W' of the form (c) or of the form $(a)/(b)$ for which $i \in I_a$.

Since each wire in W' of the form (c) or of the form $(a)/(b)$ for which $i \in I_a$ results in adding at most one more share index to each $J_b^{(k)}$ for $k \in [n]$, then we have $|J_b^{(k)}| \leq 1 + (d-1-m) = d-m$. And $|W^{(1)} \cup \ldots \cup W^{(n)}| \leq m$, in particular $|W^{(k)}| \leq m$ for any $k \in [n]$.

- if $m = 0$, then $W^{(k)} = \emptyset$ for any $k \in [n]$, and $|J_b^{(k)}| \leq d \leq \min(t+1, n-t) \leq \lfloor \frac{n+1}{2} \rfloor \leq n-1$, so by the TRPE property of G_{refresh} for any $t \leq n-1$, all of the $J_b^{(k)}$ sets can be perfectly simulated with no knowledge of the input shares of b since $W^{(k)} = \emptyset$, so $I_b^{(k)} = \emptyset$. Hence, $I_b = I_b^{(1)} \cup \ldots \cup I_b^{(n)} = \emptyset$ and we have no simulation failure on the input b.

- if $m > 0$, then $|J_b^{(k)}| \leq d-1 \leq t$ and $|W^{(1)} \cup \ldots \cup W^{(n)}| \leq d-1$, in particular $|W^{(k)}| \leq d-1$ for each $k \in [n]$. Thus, by the (t,f)-TRPE property of the refresh gadget G_{refresh} achieving the amplification order d for any $t \leq n-1$, we can perfectly simulate both sets for each $k \in [n]$ with $I_b^{(k)}$ such that $|I_b^{(k)}| \leq |W^{(k)}|$. Thus, we can let $I_b = I_b^{(1)} \cup \ldots \cup I_b^{(n)}$ so we can have $|I_b| \leq |W^{(1)} \cup \ldots \cup W^{(n)}| \leq d-1 \leq t$, and we can perfectly simulate $W^{(1)} \cup \ldots \cup W^{(n)}$ along with $J_b^{(1)} \cup \ldots \cup J_b^{(n)}$ from the set I_b without a simulation failure on input b.

So far we proved that if we have $|I_a| \geq t+1$, then we must have $|I_b| \leq |W^{(1)} \cup \ldots \cup W^{(n)}| \leq d-1 \leq t$, and $W^{(1)} \cup \ldots \cup W^{(n)}$ can be perfectly simulated along with $J_b^{(1)} \cup \ldots \cup J_b^{(n)}$ from the set I_b. Next we need to prove that we can perfectly simulate W' and J from these sets I_a and $J_b^{(1)} \cup \ldots \cup J_b^{(n)}$.

Case 2.1.1: $I_a = [n]$. This only occurs by construction in the case where $|W| = 2d-1 = n$ so when $d = d_{\max} = \lfloor \frac{n+1}{2} \rfloor$ for $t = \lceil \frac{n-1}{2} \rceil$. In this case, since $|W| \leq 2d-1 \leq (n+1)-1 \leq n$,

then all probes in W are all in W' of the form (a) or (b) with n distinct values for the index i and so $|J_b^{(i)}| \leq 1$ for all $i \in [n]$. In other words, for each $i \in [n]$ there is exactly one probe in W' of the form (a) or (b) and no probe of the form (c) i.e $X_{i,j}$ nor probes in $W^{(1)} \cup \ldots \cup W^{(n)}$. We will prove that all the probes in W and in J can be perfectly simulated from these constructed sets I_a and $J_b^{(i)}$ for $i \in [n]$. For this, for each $i \in [n]$ we consider three cases:

- $V_{i,j} \notin W'$ for any $j \in [n]$, then we know that there exists a probe of the form (a) in W' with index i, in other terms, $a_i \in W'$, or $\exists!\ j \in [n]$ such that $b_j^{(i)}$ or $a_i \cdot b_j^{(i)}$ or $r_{i,j}$ or $p_{i,j} = a_i \cdot b_j^{(i)} + r_{i,j}$ is probed in W'. The corresponding probe is perfectly simulated by construction of the sets I_a and $J_b^{(i)}$.

  If we also have $i \in J$, then we know that we only have one probe of the form (a) for the considered index i in W' and no probe of the form (b) or any probe of the form (c). And since there are t output shares probed in J, then there are at least $n - t - 1 > 1$ (since $t = \lceil \frac{n-1}{2} \rceil$) remaining random values which only appear in the expression of c_i, and any of them can be used to perfectly simulate c_i without the knowledge of the input shares (*i.e.*, to mask c_i).

- $V_{i,n} \in W'$ then $V_{i,n}$ contains in its expression n random values $r_{i,1}, \ldots, r_{i,n}$. Since there are no probes of the form (a) for the index i, and no probes of the form (c), then each of these random values appears at most once in each of the expressions of the probed outputs c_j in J. With t probed output shares, there are $n - t > 1$ (since $t = \lceil \frac{n-1}{2} \rceil$) remaining random values which only appear in the expression of $V_{i,n}$ and any of them can be used to perfectly simulate $V_{i,n}$, *i.e.*, mask $V_{i,n}$.

  If in addition we have $i \in J$, then the output share c_i is perfectly simulated by simulating $V_{i,n}$ and simulating $c_i + V_{i,n} = X_i$ which is perfectly simulated by generating uniform random values.

- $V_{i,j} \in W'$ for some $j \in [n]$ such that $1 < j < n$ ($j > 1$ because otherwise it would be the wire $p_{i,1}$ which is probed). Thus, $V_{i,j}$ is the sum of at least two wires p_{i,j_1} and p_{i,j_2}.

  - If $i \notin J$, then c_i is not probed and $V_{i,j}$ is the sum of at most $n-1$ terms of the form $p_{i,1} = a_i \cdot b_1^{(i)} + r_{i,1}, \ldots p_{i,j} = a_i \cdot b_j^{(i)} + r_{i,j}$. We have that $i \in I_a$ by construction and $j \in J_b^{(i)}$. In fact we can reconstruct $J_b^{(i)}$ into $J_b^{(i)} = \{1, \ldots, j\}$ such that $|J_b^{(i)}| \leq n-1$ and since $W^{(i)} = \emptyset$, then by the (t, f)-TRPE1 property of G_{refresh} for any $t \leq n-1$, we still have no failure on the input b and we still have $|I_b^{(i)}| \leq |W^{(i)}| = 0$. In addition, we can perfectly simulate this way all of the summed terms in $V_{i,j}$ by using the corresponding input shares and thus we can perfectly simulate $V_{i,j}$. Since we have no probes of the form (a) for this same index i, then reconstructing $J_b^{(i)}$ does not affect the simulation of the probes.
  - If $i \in J$, then we consider $V_{i,j}$ and $c_i + V_{i,j}$. Since we have no probes of the form (a) for the index i, then as proven before, with t probed output shares, there are at least $n - t > 1$ remaining random values which only appear in the expression of $V_{i,j}$ or $c_i + V_{i,j}$. Any of these random values can be used to mask the expression of $V_{i,j}$ or $c_i + V_{i,j}$. In the case where the expression of $V_{i,j}$ is masked, then we can reconstruct as before the set $J_b^{(i)}$ with at most $n-1$ output shares of $b^{(i)}$ in order to perfectly simulate all the terms $p_{i,k}$ in $c_i + V_{i,j}$ including the shares of $b^{(i)}$ and thus perfectly simulate $c_i + V_{i,j}$ (the rest of the terms are just random values to be generated uniformly at random). In the other case where the expression of $c_i + V_{i,j}$

is masked, we can also reconstruct the set $J_b^{(i)}$ with at most $n-1$ output shares of $b^{(i)}$ in order to perfectly simulate all the summed terms in $V_{i,j}$. In either case, by perfectly simulating one term ($V_{i,j}$ or $c_i + V_{i,j}$) masked by a random value, and perfectly simulating the remaining one with $i \in I_a$ and the reconstructed set $J_b^{(i)}$, we can perfectly simulate both $V_{i,j}$ and $c_i + V_{i,j}$ and hence also perfectly simulate the output share c_i.

So we proved that we can perfectly simulate the sets W' and J from the constructed set I_a and from sets $J_b^{(i)}$ such that $|J_b^{(i)}| \leq n-1$ for all $i \in [n]$. Furthermore, from the TRPE property of G_{refresh} for any $t \leq n-1$ and the fact that $W^{(i)} = \emptyset$ for all $i \in [n]$, we have no simulation failure on the input b. This concludes the simulation of W and output shares in J for the case where $I_a = [n]$.

Case 2.1.2: $I_a \subset [n]$ with $|I_a| \leq n-1$. In this case, we have at least one index $k \in [n] \setminus I_a$ for which there are no probes in W' of the form (a) or (b). In other terms, no partial sum of V_k is probed, no product of shares $a_k \cdot b_j^{(k)}$ or $p_{k,j}$ is probed, and no random value $r_{k,j}$ is probed since otherwise we would have $k \in I_a$ by construction.

On another hand, since $|I_a| \geq t+1$, there are at most $d-1 \leq n-t-1$ remaining probes of the form (c) in W', and since we have t output shares in the set J, there exists at least one wire X_ℓ such that $\ell \notin J$ and for which there is no partial sum $X_{\ell,j}$ probed in W'.

These two wires X_ℓ and V_k for $\ell, k \in [n]$ will be very important for the simulation of the sets W' and J. In particular, we need the two following claims.

Claim 2. *Let $i \in J$. Suppose that $i \notin I_a$. Then the expression of $c_i = V_i + X_i$ can be masked by the random value $r_{i,\ell}$, in other terms $c_i \leftarrow r_{i,\ell}$.*

Proof. This claim can be proved easily, since we suppose that $i \notin I_a$ so the random value $r_{i,\ell}$ and $p_{i,\ell}$ are not probed in W'. In addition, since $\ell \notin J$ and $X_{\ell,j} \notin W'$ for all $j \in [n]$, then the random value $r_{i,\ell}$ does not appear in any other probed wire expression except in c_i, then c_i can be masked by the random value $r_{i,\ell}$. □

Claim 3. *Let $i \in J$. Suppose that $X_{i,j} \notin W'$ for any $j \in [n]$. Suppose that $i \in I_a$. Then the expression of $c_i = V_i + X_i$ can be masked by the random value $r_{k,i}$, in other terms $c_i \leftarrow r_{k,i}$.*

Proof. Since we suppose that $k \notin I_a$, then the random value $r_{k,i}$ or $p_{k,i}$ or $V_{k,j}$ for all $j \in [n]$ are not probed in W'. Then, if $k \notin J$, then the random value $r_{k,i}$ does not appear in the expression of any other probed wire in W' or J and c_i can be masked by the random value $r_{k,i}$. Otherwise, if $k \in J$, then by Claim 2, $c_k = V_k + X_k$ can be masked by $r_{k,\ell}$ and so c_i can also be masked by $r_{k,i}$ since $i \neq \ell$ (because $i \in J$ and $\ell \notin J$). □

From these two claims, we are now ready to show that W' and J can be perfectly simulated with the sets I_a and $J_b^{(1)} \cup \ldots \cup J_b^{(n)}$ as constructed earlier with respect to the probes in the set W'. Recall that all probes in $W^{(1)} \cup \ldots \cup W^{(n)}$ and $J_b^{(1)} \cup \ldots \cup J_b(n)$ are perfectly simulated using I_b and the simulator of G_{refresh}.

Simulation of W'. Probes of the form (a) and (c) are trivially simulated by construction of the sets of input shares and by generating uniformly at random the necessary random values. Let us now check the probes of the form (b). Let $V_{i,j} = p_{i,1} + \ldots + p_{i,j}$ be such a probe. Let us consider each of the terms $p_{i,j'}$ for $j' \in [j]$. if $j' = i$, then by construction $p_{i,i}$ is perfectly simulated using a_i and $b_i^{(i)}$ and by generating the random value $r_{i,i}$ if needed. Otherwise, let $j' \neq i$. If $j' \in J_b^{(i)}$ then the simulation of $p_{i,j'}$ is straightforward. Otherwise if $j' \notin J_b^{(i)}$, then we know that none of the wires $r_{i,j'}$ or $p_{i,j'}$ or $X_{j',s}$ for all $s \in [n]$ are probed in W'. Thus, $r_{i,j'}$

can be eventually used to mask the expression of $p_{i,j'}$ without the need of the share $b^{(i)}_{j'}$ for the simulation. Meanwhile, we still need to check if $j' \in J$, since $X_{j'}$ appears in the expression of $c_{j'} = V_{j'} + X_{j'}$. Then we consider two cases:

- If $j' \notin I_a$, then by claim 2, $c_{j'}$ can be masked by the random value $r_{j',\ell}$ and so $r_{i,j'}$ does not appear in the expression of $X_{j'}$ in $c_{j'}$ anymore, and $r_{i,j'}$ can be used to mask $p_{i,j'}$.
- Otherwise, if $j' \in I_a$, then by claim 3, $c_{j'}$ can be masked by the random value $r_{k,j'}$ and so $r_{i,j'}$ does not appear in the expression of $X_{j'}$ in $c_{j'}$ anymore, and $r_{i,j'}$ can be used to mask $p_{i,j'}$ (since $i \notin k$).

Thus, each term $p_{i,j'}$ in $V_{i,j}$ can be perfectly simulated and thus $V_{i,j} = p_{i,1} + \ldots + p_{i,j}$ can be perfectly simulated. This concludes the simulation of the set W'.

Simulation of J. Let $i \in J$. If $i \notin I_a$, then by claim 2, c_i is perfectly simulated by generating the random value $r_{i,\ell}$. Otherwise, let $i \in I_a$. If $X_{i,j} \notin W$ for any $j \in [n]$, then by claim 3, c_i is perfectly simulated by generating the random value $r_{k,i}$. Otherwise, we can show that we can perfectly simulate each term in $c_i = V_i + X_i$. In particular, each term in X_i can be simulated by generating the underlying random value uniformly. For each term in the sum V_i, we know in particular that $a_i \cdot b^{(i)}_i$ is perfectly simulated since $X_{i,j} \in W'$ for at least one $j \in [n]$ so $i \in J^{(i)}_b$ by construction. For the other terms in V_i, they can be perfectly simulated in the exact same way as we simulated the probes $V_{i,j}$ of the form (b) in the set W'. So c_i is perfectly simulated by summing all the perfectly simulated terms. This concludes the simulation proof for the set J.

Up until now, we have concluded that if we have a constructed set I_a of size at least $t+1$, then we can perfectly simulate the sets W and J without having a simulation failure on the input b. In the rest of the proof, we will consider that $|I_a| \leq t$ (along with $|I_a| \leq |W|$ by construction meaning that we have no failure on input a), and we will prove that we can perfectly simulate W and J with at most a simulation failure on b. Recall that we are also considering that $|W'| \geq d$ and $|W^{(1)} \cup \ldots \cup W^{(n)}| \leq d-1$.

Case 2.2: $|I_a| \leq t$. This means that the number of probes of the form (a) or (b) in W' with distinct values for the index i is at most t.

First, let us consider that $|I_a| \geq d$ (this is the case where $d = n - t \leq t+1$). Then, as proved earlier, and with t additional output shares in J of the form $c_i = V_i + X_i$, there are at least one X_ℓ remaining such that $\ell \notin J$ and $X_{\ell,j} \notin W$ for all $j \in [n]$. In this case, we can set $I_b = [n]$ and I_a as constructed with respect to the probes in W'. It is clear that all probes in $W^{(1)} \cup \ldots \cup W^{(n)}$ and $J^{(1)}_b \cup \ldots \cup J^{(n)}_b$ are trivially simulated using $I_b = [n]$. In addition, all probes in W' are also perfectly simulated by construction of the set I_a and using $I_b = [n]$ and generating the necessary random values. This means that we can perfectly simulate all of the set of probes $W = W' \cup W^{(1)} \cup \ldots \cup W^{(n)}$. As for the set of output shares indexed in J. Let $i \in J$. If $i \in I_a$, then c_i is perfectly simulated using the share a_i and $I_b = [n]$, and by generating the necessary random values. Otherwise, if $i \notin I_a$, then in the same way as in claim 2, c_i can be masked by the random value $r_{i,\ell}$ (because $\ell \notin J$ and $X_{\ell,j} \notin W$ for all $j \in [n]$), so a_i is not needed for the simulation of c_i. This proves that we can perfectly simulate the output shares in J with I_a and $I_b = [n]$.

In the rest, we suppose that $|I_a| \leq d-1 \leq n-t-1$, *i.e.*, the number of probes of the form (a) or (b) in W' with distinct values for the index i is at most $d-1 \leq n-t-1$. In this case, and with t additional output shares, we have at least one index k such that $k \notin J$ and for which there are no probes in W' of the form (a) or (b). In other terms, no partial sum of V_k

is probed, no product of shares $a_k \cdot b_j^{(k)}$ or $p_{k,j}$ is probed, and no random value $r_{k,j}$ is probed. Now we reason on the number of probes of the form (c) in W':

- We first consider the special case where the number of $X_{i,j}$ probed (of form (c) in W') for distinct values of i is equal to n. In other terms, we have probes $X_{1,j_1}, \ldots, X_{n,j_n}$ for certain values $j_1, \ldots, j_n$. Since the set of probes W satisfies $|W| \leq n$ (because $2d-1 \leq n$), then this means that there are no remaining probes in the set W except for the n probes of the form (c) in W'. This is an easy case since we can let $I_b = [n]$ and $I_a = J$ (always without a failure on a since in the case where $|W| = n$, we have $d = t + 1 = n - t$ so $|I_a| = |J| \leq \min(t, |W|)$ where $t \leq |W|$). This allows us to trivially simulate all output wires indexed in J, and since the remaining wires in W are just sums of random values, we can simulate them by generating the corresponding random values.

- Next, we consider that there is at least one index ℓ such that $X_{\ell,j} \notin W$ for all $j \in [n]$ (in other terms, the number of probes of the form $X_{i,j}$ for distinct values of i is at most $n-1$). Notice that this case is slightly different than the case of claims 2 and 3, since ℓ can be in the set J. In this case, we can let $I_b = [n]$ so that we can perfectly simulate all wires in $W^{(1)} \cup \ldots \cup W^{(n)}$ and $J_b^{(1)} \cup \ldots \cup J_b^{(n)}$ using $I_b = [n]$, and we can perfectly simulate all wires in W' using I_a by construction and $I_b = [n]$ and generating the necessary random values. Next, we need to prove that we can perfectly simulate all output shares in J. Let $i \in J$. If $i \in I_a$, then c_i is perfectly simulated using a_i and $I_b = [n]$ and generating the necessary random values. Next, if $i \notin I_a$, then if $\ell \notin J$, we can use claim 2 to prove that we can replace the expression of $c_i = V_i + X_i$ by the random value $r_{i,l}$ and so the share a_i is not needed for the simulation of c_i (even if $X_{i,j}$ for a certain j is probed, the expression of V_i is still masked by $r_{i,l}$ and a_i is not needed to simulate X_i which is a sum of random values). Meanwhile, if $\ell \in J$, then we cannot directly use the random value $r_{i,\ell}$ to mask the expression of c_i. But since $X_{\ell,j} \notin W$ for all $j \in [n]$, and since $r_{k,\ell} \notin W$ because $k \notin I_a$ by assumption, then c_ℓ can be masked by the random value $r_{k,\ell}$, i.e $c_\ell = V_\ell + X_\ell \leftarrow r_{k,\ell}$. Since $i \in J$ and $k \notin J$, then $i \neq k$ and the random value $r_{i,\ell}$ does not appear anymore in X_ℓ in the expression of c_ℓ. Since $i \notin I_a$ then $r_{i,\ell}$ can be used to mask the expression of the output share c_i indexed in J and so the share a_i is not needed for the simulation of c_i. This proves that we can perfectly simulate all shares in J with the constructed sets I_a and $I_b = [n]$.

We managed to show that whenever the construction of the set I_a gives $|I_a| \leq t$, then we can perfectly simulate the sets W and J with at most a failure on input b and while still having $|I_a| \leq t$ and $|I_a| \leq |W|$.

By considering both cases $|I_a| \geq t + 1$ and $|I_a| \leq t$, we covered all the cases for the simulation, and we proved that we can always perfectly simulate the set of probes W along with the set of output shares J while having a failure on at most one of the inputs. This concludes the proof of Lemma 26. □

□

A.1.2 Proof for TRPE2 property

Lemma 27. *The above multiplication gadget is (t, f_2)-TRPE2 of amplification order $d \geq \min(t+1, n-t)$*

Proof. To prove the lemma, we proceed in two steps through the following two lemmas 28 and 29.

Lemma 28. *Let W be a set of leaking wires as described above such that $|W| < \min(t+1, n-t)$. Then there exists a set J of $n-1$ output shares, such that W and J can be perfectly simulated from at most $\min(|W|, t) = |W|$ shares of each of the inputs a and b.*

Proof. We will construct the set of input shares indices I_a and the sets of output shares $J_b^{(k)}$ for $k \in [n]$ depending on the probes in the set W' (recall that $W = W' \cup W^{(1)} \cup \ldots \cup W^{(n)}$) as follows (we consider that all $J_b^{(k)}$ are empty at first since all the output shares of G_{refresh} can be probed directly in W'):

(a) For probes of form (a), we add index i to I_a, and index j to $J_b^{(k)}$ for $k \in [n]$.

(b) For probes of form (b), we add index i to I_a and to $J_b^{(k)}$ for $k \in [n]$.

(c) For probes of form (c), we add index i to $J_b^{(k)}$ for $k \in [n]$.

Observe that since $|W| < \min(t+1, n-t)$, then in particular $|W'| \leq \min(t+1, n-t) - 1 \leq t$, then $|I_a| \leq |W'| \leq |W| \leq t$ so we have no failure on the input a. Also, since $|J_b^{(k)}| \leq |W'| \leq t$ and $|W^{(k)}| < \min(t+1, n-t)$, then by the (t, f')-TRPE1 property of G_{refresh}, we will be able to simulate sets $J_b^{(k)}$ and $W^{(k)}$ from the set of input shares $I_b^{(k)}$ such that $|I_b^{(k)}| \leq |W^{(k)}| \leq t$ for $k \in [n]$. Thus, we can let $I_b = I_b^{(1)} \cup \ldots \cup I_b^{(n)}$ and we have $|I_b| \leq |W^{(1)} \cup \ldots \cup |W^{(n)}| \leq |W| \leq t$, so we have no failure on the input b either. Until now, we have shown that we can simulate all sets $W^{(k)}$ and $J_b^{(k)}$ from I_b of size at most $\min(|W|, t) = |W|$. It remains to show that we can also perfectly simulate the set W' and a well chosen set J of $n-1$ output shares, from I_a and $J_b^{(k)}$ for $k \in [n]$. We will choose the set J from two subsets $J = J_1 \cup J_2$, where $J_1 = \{i \mid i \in J_b^{(k)} \text{ for any } k \in [n]\}$, and $J_2 \subset [n]$ is any set such that $J_1 \cap J_2 = \emptyset$ and $|J_1 \cup J_2| = n-1$. Let $\ell \in [n]$ be the index such that $\ell \notin J$. Since $|W| \leq \min(t+1, n-t) - 1 \leq n-1$, then by construction of the sets $J_b^{(k)}$, we have that $|J_b^{(1)} \cup \ldots \cup J_b^{(n)}| \leq n-1$, then for the index ℓ, we have that $\ell \notin J_b^{(k)}$ for all $k \in [n]$, then $X_{\ell,j} \notin W$ for any $j \in [n]$ by construction of the sets $J_b^{(k)}$. The value of X_ℓ will be useful to use the following claim.

Claim 4. *Let $i \in J$. Suppose that $V_{i,j} \notin W$ for all $j \in [n]$. Then the expression of $c_i = V_i + X_i$ can be masked by the random value $r_{i,\ell}$, in other terms $c_i \leftarrow r_{i,\ell}$.*

Proof. The proof of this claim is quite straightforward since we suppose that $V_{i,j} \notin W$ for all $j \in [n]$, so none of the partial sums $V_{i,j}$ has been probed. Then V_i in c_i contains $n-1$ random values. In particular, we know that $r_{i,\ell}$ and $p_{i,\ell}$ only appear in the expression of the probed output c_i, because if they were probed in W then we would have $\ell \in J_b^{(i)}$ by construction, but we suppose that $\ell \notin J_b^{(k)}$ for all $k \in [n]$. In addition, since $X_{\ell,j} \notin W$ for all $j \in [n]$ (because otherwise then by construction $\ell \in J_b^{(k)}$ which does not hold), then $r_{i,\ell}$ does not appear in any other expression of the probed wires in W, so we can simply use it to perfectly simulate c_i. □

We can now show that the sets W' and J can be perfectly simulated from the constructed sets I_a and $J_b^{(k)}$.

Simulation of W'. Probes of the form (a) can be perfectly simulated from the corresponding input shares in I_a and $J_b^{(k)}$, and by generating uniformly random values $r_{i,j}$ when necessary. Probes of the form (c) are also perfectly simulated by simply generating uniformly random values, since $X_{i,j} = r_{1,i} + \ldots + r_{j,i}$. As for probes of the form (b), we know that $i \in I_a$, then we look at each of the terms $p_{i,j'}$ for $j' \in [j]$ in $V_{i,j} = p_{i,1} + \ldots p_{i,j}$. For each $p_{i,j'}$, if $j' \in J_b^{(i)}$,

then $p_{i,j'}$ can be perfectly simulated from the corresponding input shares and by generating uniformly at random $r_{i,j'}$. Otherwise, if $j' \notin J_b^{(i)}$, then that means that the wires $p_{i,j'}, r_{i,j'}$ and $X_{j'}$ are not probed in W'. That means that we can potentially replace $p_{i,j'}$ by a random value $r_{i,j'}$ since $r_{i,j'}$ does not appear in any other expression of the variables probed in W'. Meanwhile, we also need to check the case where $j' \in J$, since $c_{j'} = V_{j'} + X_{j'}$, and $r_{i,j'}$ is the one of the summed terms in the expression of $X_{j'}$:

- If $j' \notin J$, then we can replace $p_{i,j'}$ by a random value $r_{i,j'}$ since $r_{i,j'}$ does not appear in any other expression of the variables probed in W' and is not probed either through $c_{j'}$.
- If $j' \in J$, then we also know that $V_{j'} \notin W'$ (because otherwise we would have by construction $j' \in J_b^{(k)}$ for $k \in [n]$ which does not hold), then we know from claim 4 that $c_{j'}$ can be masked by the random value $r_{j',\ell}$, which masks $V_{j'} + X_{j'}$. Since $\ell \neq j'$ (because $\ell \notin J$ while $j' \in J$), then $r_{i,j'}$ does not appear anymore in any other wire expression of the probed variables in W or J except in the term $p_{i,j'}$ of $V_{i,j}$, so $r_{i,j'}$ can be used to mask the expression of $p_{i,j'}$.

By perfectly simulating each term $p_{i,j'}$ in $V_{i,j}$, we can perfectly simulate $V_{i,j}$. Thus, we can perfectly simulate all wires in W'.

Simulation of J. Let $i \in J$. Let us first consider the case where $V_{i,j} \notin W'$ for any $j \in [n]$, then by claim 4, the output share c_i can be masked by the random variable $r_{i,\ell}$, so c_i is perfectly simulated by generating a fresh random value. Otherwise, if $V_{i,j} \in W'$ for a certain $j \in [n]$, then we know that the value of $V_{i,j}$ is perfectly simulated as proven above. Now, let us check each term $p_{i,j'}$ for $j' \in [j+1, n]$. Actually, we can also perfectly simulate each of these terms like the terms $p_{i,j'}$ for $j' \in [j]$. Plus, the term $p_{i,i}$ is perfectly simulated by construction of the sets I_a and $J_b^{(i)}$ (because $V_{i,j} \in W'$). In addition, all terms in X_i in $c_i = V_i + X_i$ can be perfectly simulated by generating a fresh random value. Thus, c_i can be perfectly simulated by summing all of the perfectly simulated terms in it. This brings us to a perfect simulation of all output shares in J. We have shown that we can perfectly simulate any set of probes W of size at most $\min(t+1, n-t) - 1$ with a chosen set J of $n-1$ output shares, with at most $\min(|W|, t) = |W|$ shares of each of the inputs a and b. This concludes the proof of Lemma 28. □

Remark 9. *We can observe that for this lemma to apply on G_{mult}, we don't need the pre-processing phase of the refresh on input b. In fact, you can see that during the construction of the sets $J_b^{(k)}$, we add each index to all of the sets for all $k \in [n]$. However, executing n refreshings on the input b will be necessary for the proof of the next result, specifically when we consider W such that* $\min(t+1, n-t) \leq |W| < 2 \cdot \min(t+1, n-t)$.

To get back to the proof of Lemma 27, we also need the following result.

Lemma 29. *Let W be a set of leaking wires as described above such that* $\min(t+1, n-t) \leq |W| < 2 \cdot \min(t+1, n-t)$. *Then there exists a set J of $n-1$ output shares such that W and J can be perfectly simulated from sets of input shares I_a and I_b such that $|I_a| \leq \min(|W|, t)$* ***or*** *$|I_b| \leq \min(|W|, t)$. In other terms, we have a simulation failure on at most one of the inputs a or b.*

Proof. Recall that the set W can be split into subsets $W = W' \cup W^{(1)} \cup \ldots \cup W^{(n)}$ as described above. We consider two cases.

Case 1: $|W'| < \min(t+1, n-t)$**.** This case is similar to the case of Lemma 28, so we can construct the set I_a in the same way as in the proof of Lemma 28, and we can eventually consider $I_b = [n]$. We know that $|I_a| \leq |W'| \leq \min(t+1, n-t) - 1 \leq t$, so there is no failure on the input a. And all probes in W' can be simulated like in the proof of Lemma 28 with I_a and trivially with $I_b = [n]$. Also, all probes in $W^{(1)} \cup \ldots \cup W^{(n)}$ can be trivially simulated since we have access to the full input b. In addition, we choose the set J of size $n-1$ in the same way as in Lemma 28. Whenever $i \in J$ and $V_{i,j} \in W'$ for some $j \in [n]$, then $c_i = V_i + X_i$ is easily simulated using $I_b = [n]$ and the share a_i. If $i \in J$ but $V_{i,j} \notin W'$ for all $j \in [n]$, then as in the proof of Lemma 28, c_i in this case can be masked by the random value $r_{i,\ell}$ (because $|W'| < \min(t+1, n-t)$) and so simulating c_i amounts to generating uniformly at random the corresponding random value. Thus, W and J are perfectly simulated with at most $\min(|W|, t)$ shares of a and eventually the full input b.

Case 2: $|W'| \geq \min(t+1, n-t)$ **(and thus** $|W^{(1)} \cup \ldots \cup W^{(n)}| < \min(t+1, n-t)$**).** In this case, we will construct the sets I_a and $J_b^{(k)}$ from empty sets, in a way that we will have a simulation failure on at most one of the inputs a or b. We construct the mentioned sets depending on the probes in W' as follows:

(a) For probes of form (a), we add index i to I_a, and index j only to $J_b^{(i)}$.

(b) For probes of form (b), we add index i to I_a and only to $J_b^{(i)}$.

(c) For probes of form (c), we add index i to $J_b^{(k)}$ for all $k \in [n]$.

In the rest of the lemma, we will prove that if we have a failure on one of the inputs, we can still perfectly simulate W and a chosen set J of $n-1$ output shares without a failure on the other input. For this, we will consider two cases, the first where we have a failure on input a, the second where we don't have a failure on input a, and so we can eventually have a failure on input b.

Case 2.1: simulation failure on input a**,** ***i.e.*** $I_a > t$**.** This means that the set I_a is of size $|I_a| \geq t+1 \geq \min(t+1, n-t)$ (this is because by construction $|I_a| \leq |W|$, so to have $|I_a| > \min(|W|, t)$, we must have $|I_a| > t$). We will first start by showing that the sets $W^{(k)}$ and $J_b^{(k)}$ can be perfectly simulated using the simulator of G_{refresh} without a failure on the input b. Next, we will show that W' and a well chosen set of $n-1$ output shares in J can be perfectly simulated using I_a and $J_b^{(k)}$.

Since we only add shares indices to I_a, when we have probes of the form (a) or (b), this means that we have at least $t+1$ probes of these two forms with $t+1$ different values for the index i. In addition, since we have at least $t+1$ probes (a) or (b) with distinct values for the index i, then this also means that each of the sets $J_b^{(i)}$ has at most one share of $b^{(i)}$ added to it. In other terms, $|J_b^{(k)}| \leq 1$ for each $k \in [n]$ (from the probes (a) and (b) with distinct indices i).

Now let us consider the remaining probes in W which are either in W' of the form (c) or in $W^{(1)} \cup \ldots \cup W^{(n)}$ or of the forms (a) or (b) with $i \in I_a$. Since $|I_a| \geq t+1 \geq \min(t+1, n-t)$, then there are at most $\min(t+1, n-t) - 1$ of these remaining probes. Without loss of generality, we consider that there are exactly $\min(t+1, n-t) - 1$ instead of at most $\min(t+1, n-t) - 1$ probes. Let m be the number of probes in $W^{(1)} \cup \ldots \cup W^{(n)}$ and $d-1-m$ the remaining probes in W' of the form (c) or (a)/(b) with $i \in I_a$. Since each wire in W' of the form (c) or (a)/(b) with $i \in I_a$ results in adding at most one more share index to each $J_b^{(k)}$ for $k \in [n]$, then we have $|J_b^{(k)}| \leq 1 + (\min(t+1, n-t) - 1 - m) = d - \min(t+1, n-t)$. And $|W^{(1)} \cup \ldots \cup W^{(n)}| \leq m$, in particular $|W^{(k)}| \leq m$ for any $k \in [n]$.

- if $m = 0$, then $W^{(k)} = \emptyset$ for any $k \in [n]$, and $|J_b^{(k)}| \leq \min(t+1, n-t) \leq \lfloor \frac{n+1}{2} \rfloor \leq n-1$, so by the TRPE property of G_{refresh} for any $t \leq n-1$, all of the $J_b^{(k)}$ sets can be perfectly simulated with no knowledge of the input shares of b since $W^{(k)} = \emptyset$, so $I_b^{(k)} = \emptyset$. Hence, $I_b = I_b^{(1)} \cup \ldots \cup I_b^{(n)} = \emptyset$ and we have no simulation failure on the input b.
- if $m > 0$, then $|J_b^{(k)}| \leq \min(t+1, n-t) - 1 \leq t$ and $|W^{(1)} \cup \ldots \cup W^{(n)}| < \min(t+1, n-t)$, in particular $|W^{(k)}| \leq \min(t+1, n-t) - 1$ for each $k \in [n]$. Thus, by the (t, f)-TRPE property of the refresh gadget G_{refresh} achieving the amplification order d, we can perfectly simulate both sets for each $k \in [n]$ with $I_b^{(k)}$ such that $|I_b^{(k)}| \leq |W^{(k)}|$. Thus, we can let $I_b = I_b^{(1)} \cup \ldots \cup I_b^{(n)}$ so we can have $|I_b| \leq |W^{(1)} \cup \ldots \cup W^{(n)}| \leq \min(t+1, n-t) - 1 \leq t$, and we can perfectly simulate $W^{(1)} \cup \ldots \cup W^{(n)}$ along with $J_b^{(1)} \cup \ldots \cup J_b^{(n)}$ from the set I_b without a simulation failure on input b.

So far we proved that if we have $|I_a| > t$, then we must have $|I_b| \leq t$, and $W^{(1)} \cup \ldots \cup W^{(n)}$ can be perfectly simulated along with $J_b^{(1)} \cup \ldots \cup J_b^{(n)}$ from the set I_b. Next we need to prove that we can perfectly simulate W' and a chosen set J of $n-1$ output shares, from these sets I_a and $J_b^{(1)} \cup \ldots \cup J_b^{(n)}$. We consider two sub-cases.

Case 2.1.1: $I_a = [n]$**.** In this case, since $|W| \geq n$ (from I_a) and $|W| < 2\min(t+1, n-t) \leq n+1$, then $|W| = n$ and all probes in W are all in W' of the form (a) or (b) with n distinct values for the index i. We neither have probes in $W^{(1)} \cup \ldots \cup W^{(n)}$ nor in W' of the form (c). Thus, we can reconstruct each $|J_b^{(k)}|$ of size at most $n-1$ without having a failure on the input b (since G_{refresh} is (t', f')-TRPE for any $t' \leq n-1$ achieving $d' = \min(t'+1, n-t')$ and all $W^{(k)}$ are empty). We consider two cases:

- Suppose that for each $i \in [n]$, we have at least one probe in W' of the form $r_{k,i}$ or $p_{k,i}$ for some $k \in [n]$, note this probe $q_{k,i} \in \{r_{k,i}, p_{k,i}\}$. Since also $I_a = [n]$, this means that we have probes $q_{k_1,1}, \ldots, q_{k_n,n}$, such that $k_1 \neq \ldots \neq k_n$. Because $|W'| = n$, then all probes in W' are of the form (a) (specifically $q_{k,i}$), and we have no probes of the form $V_{i,j}$ for any $i, j \in [n]$. In this case, the simulation of the probes in W' is straightforward by construction of the sets I_a and $J_b^{(k)}$. As for the set J, we let $J \subset [n]$ such that $|J| = n-1$ (any set of $n-1$ shares works), and let $\ell \in [n]$ such that $\ell \notin J$. Observe that out of all the random values $r_{i,\ell}$ in X_ℓ in the expression of $c_\ell = V_\ell + X_\ell$, only the random value $r_{k_\ell,\ell}$ appears in the expression of the probe $q_{k_\ell,\ell}$ in the set W', and all other random values $r_{i,\ell}$ for $i \neq k_\ell$ do not appear in any other probed variable in W' (since $W' = \{q_{k_1,1}, \ldots, q_{k_\ell,\ell}, \ldots, q_{k_n,n}\}$, such that $k_1 \neq \ldots \neq k_n$). Then, for each $i \in J$ such that $i \neq k_\ell$, the expression of $c_i = V_i + X_i$ can be masked by the random value $r_{i,\ell}$, so simulating c_i amounts to generating a fresh random value $r_{i,\ell}$. Now let's check $i = k_\ell \in J$. Since $q_{k_\ell,\ell}$ is probed, then we cannot mask the expression of c_{k_ℓ} using $r_{k_\ell,\ell}$. However, for each $i \in J$ with $i \neq k_\ell$, we have that c_i is masked by $r_{i,\ell}$. Since $c_i = V_i + X_i$, and the random value $r_{k_\ell,i}$ is one of the terms in X_i, then $r_{k_\ell,i}$ does not appear anymore in the expression of c_i. And since $W' = \{q_{k_1,1}, \ldots, q_{k_\ell,\ell}, \ldots, q_{k_n,n}\}$, such that $k_1 \neq \ldots \neq k_n$ and $q_{k_\ell,\ell} \in W'$, then $q_{k_\ell,i} \notin W'$ and $r_{k_\ell,i}$ only appears in the expression of c_{k_ℓ}, so c_{k_ℓ} can be masked by the random value $r_{k_\ell,i}$. Thus, we proved that we can perfectly simulate the sets W' and J using the sets I_a and $J_b^{(k)}$.
- Next, we suppose that there exists $\ell \in [n]$ such that we have no probes in W' of the form $q_{k,\ell} \in \{r_{k,\ell}, p_{k,\ell}\}$. In this case, we choose $J = [n] \setminus \{\ell\}$. Next, we show that we can perfectly simulate all probes in W' and output shares in J for each $i \in I_a = [n]$. For this, first let $i \in [n] \setminus \{\ell\}$ (notice that we automatically have $i \in J$):

– if for the considered i, the probe in W' is of the form (a) i.e $a_i, b_j^{(i)}, a_i \cdot b_j^{(i)}, p_{i,j} = a_i \cdot b_j^{(i)} + r_{i,j}$, then the simulation of this probe is trivial by construction of the sets I_a and $J_b^{(i)}$. In addition, we know that $r_{i,\ell}$ and $p_{i,\ell}$ are not probed in W' by assumption, and since $X_{\ell,j} \notin W'$ for all $j \in [n]$, then the random value $r_{i,\ell}$ only appears in the expression of $c_i = V_i + X_i$ (specifically in V_i), and so can be used to mask c_i. So simulating c_i amounts to generating a fresh random value.
– if for the considered i, the probe in W' is of the form (b), i.e $V_{i,j} \in W'$ for a certain $j \in [n]$ (there is a unique probe of this form), then:
  * either $j < \ell$, and so the random value $r_{i,\ell}$ can be used as before to mask the expression of $c_i + V_{i,j}$, and since in this case $V_{i,j}$ contains less than $n-1$ terms $p_{i,j'}$, then we can add all the necessary shares of $b^{(i)}$ to $J_b^{(i)}$without having a failure on b (recall that $W^{(i)} = \emptyset$). So we can perfectly simulate $V_{i,j}$ and $c_i + V_{i,j}$, and hence also simulate c_i.
  * or $j \geq \ell$, and so the random value $r_{i,\ell}$ can be used in this case to mask the expression of $V_{i,j}$ so simulating $V_{i,j}$ amounts to generating a fresh random value, and since $V_{i,j}$ is the sum of at least two terms of the form $p_{i,j'}$, then $c_i + V_{i,j}$ can be simulated with at most $n-1$ shares of $b^{(i)}$, so there is no simulation failure on input $b^{(i)}$. So we can perfectly simulate $V_{i,j}$ and $c_i + V_{i,j}$, and hence also simulate c_i.

Next we consider the case of the probe $V_{\ell,j}$:

– either $j < \ell$, and so in this case $V_{\ell,j}$ contains less than $n-1$ terms $p_{\ell,j'}$, then we can add all the necessary shares of $b^{(\ell)}$ to $J_b^{(\ell)}$without having a failure on b (recall that $W^{(\ell)} = \emptyset$). So we can perfectly simulate $V_{\ell,j}$ using the input share a_ℓ, the input shares of $b^{(\ell)}$ and by generating necessary random values.
– or $j \geq \ell$, and so the random value $r_{\ell,\ell}$ can be used in this case to mask the expression of $V_{\ell,j}$ so simulating $V_{\ell,j}$ amounts to generating a fresh random value.

Thus, also in this case, we can perfectly simulate W' and a chosen set of $n-1$ output shares without a failure on input b, using $I_a = [n]$.

This concludes the simulation for the special case where $I_a = [n]$.

Case 2.1.2: $I_a \subset [n]$ **such that** $|I_a| \leq n-1$**.** Let k such that $k \notin I_a$. Recall that $|I_a| \geq t+1 \geq \min(t+1, n-t)$ and $|W| < 2 \cdot \min(t+1, n-t)$, then there are at most $\min(t+1, n-t) - 1 \leq t \leq n-1$ probes remaining either in $W^{(1)} \cup \ldots \cup W^{(n)}$, of the form (c) in W', or of the form (a)/(b) with $i \in I_a$. Thus, there exists at least one index $\ell \in [n]$ such that $X_{\ell,j} \notin W'$ for all $j \in [n]$. In this case, we choose $J = [n] \setminus \{\ell\}$. Next, we will prove that we can perfectly simulate the sets W' and J from the constructed sets I_a and $J_b^{(k)}$, using the following claims.

Claim 5. *Let $i \in J$. Suppose that $i \notin I_a$. Then the expression of $c_i = V_i + X_i$ can be masked by the random value $r_{i,\ell}$, in other terms $c_i \leftarrow r_{i,\ell}$.*

Proof. This claim can be proved easily, since we suppose that $i \notin I_a$ so the random value $r_{i,\ell}$ and $p_{i,\ell}$ are not probed in W'. In addition, since $\ell \notin J$ and $X_{\ell,j} \notin W$ for all $j \in [n]$, then the random value $r_{i,\ell}$ does not appear in any other probed wire expression except in c_i, then c_i can be masked by the random value $r_{i,\ell}$. □

Claim 6. *Let $i \in J$. Suppose that $X_{i,j} \notin W$ for any $j \in [n]$. Suppose that $i \in I_a$. Then the expression of $c_i = V_i + X_i$ can be masked by the random value $r_{k,i}$, in other terms $c_i \leftarrow r_{k,i}$.*

Proof. Since we suppose that $k \notin I_a$, then the random value $r_{k,i}$ or $p_{k,i}$ or $V_{k,j}$ for all $j \in [n]$ are not probed in W. Then, if $k \notin J$, then the random value $r_{k,i}$ does not appear in the expression of any other probed wire in W or J and c_i can be masked by the random value $r_{k,i}$ (Recall that $c_i = V_i + X_i$ and $X_i = r_{1,i} + \ldots + r_{n,i}$). Otherwise, if $k \in J$, then by Claim 5, $c_k = V_k + X_k$ can be masked by $r_{k,\ell}$ and so c_i can also be masked by $r_{k,i}$ since $i \neq \ell$ (because $i \in J$ and $\ell \notin J$) and $i \neq k$ (because $i \in I_a$ and $k \notin I_a$). □

Probes of the forms (a) or (c) in W' are trivially simulated using the constructed sets of input shares, and generating the necessary random values. Let us now check the probes of the form (b). Let $V_{i,j} = p_{i,1} + \ldots + p_{i,j}$ be such a probe. Let us consider each of the terms $p_{i,j'}$ for $j' \in [j]$. if $j' = i$, then by construction $p_{i,i}$ is perfectly simulated using a_i and $b_i^{(i)}$ and by generating the random value $r_{i,i}$ if needed. Otherwise, let $j' \neq i$. If $j' \in J_b^{(i)}$ then the simulation of $p_{i,j'}$ is straightforward. Otherwise if $j' \notin J_b^{(i)}$, then we know that none of the wires $r_{i,j'}$ or $p_{i,j'}$ or $X_{j',s}$ for all $s \in [n]$ are probed in W'. Thus, $r_{i,j'}$ can be eventually used to mask the expression of $p_{i,j'}$ without the need of the share $b_{j'}^{(i)}$ for the simulation. Meanwhile, we still need to check if $j' \in J$, since $r_{i,j'}$ appears in $X_{j'}$ in the expression of $c_{j'} = V_{j'} + X_{j'}$.

- If $j' \in J$ and $j' \notin I_a$, then by claim 5, $c_{j'}$ can be masked by the random value $r_{j',\ell}$ and so $r_{i,j'}$ does not appear in the expression of $X_{j'}$ in $c_{j'}$ anymore, and $r_{i,j'}$ can be used to mask $p_{i,j'}$.
- Otherwise, if $j' \in J \cap I_a$, then by claim 6 $c_{j'}$ can be masked by the random value $r_{k,j'}$ and so $r_{i,j'}$ does not appear in the expression of $X_{j'}$ in $c_{j'}$ anymore, and $r_{i,j'}$ can be used to mask $p_{i,j'}$ (since $i \notin k$).

Thus, each term $p_{i,j'}$ in $V_{i,j}$ can be perfectly simulated and thus $V_{i,j} = p_{i,1} + \ldots + p_{i,j}$ can be perfectly simulated. This concludes the simulation of the set W'.

We now focus on the simulation of J. Let $i \in J$. If $i \notin I_a$, then by claim 5, c_i is perfectly simulated by generating the random value $r_{i,\ell}$. Otherwise, let $i \in I_a$. If $X_{i,j} \notin W$ for any $j \in [n]$, then by claim 6, c_i is perfectly simulated by generating the random value $r_{k,i}$. Otherwise, we can show that we can perfectly simulate each term in $c_i = V_i + X_i$. In particular, each term in X_i can be simulated by generating the underlying random value uniformly. For V_i, we know in particular that $a_i \cdot b_i^{(i)}$ is perfectly simulated since $X_{i,j} \in W'$ for at least one $j \in [n]$ so $i \in J_b^{(i)}$ by construction. For the other terms in V_i, they can be perfectly simulated in the exact same way as we simulated the probes $V_{i,j}$ of the form (b) in the set W'. So c_i is perfectly simulated by summing all the perfectly simulated terms. This concludes the simulation proof for the set J.

Up until now, we have concluded that if we have a constructed set I_a of size at least $t+1$, then we can perfectly simulate the sets W and a chosen set J of $n-1$ output shares, without having a simulation failure on the input b. In the rest of the proof, we will consider that $|I_a| \leq t$, and we will prove that we can perfectly simulate W and J with at most a simulation failure on b. Recall that we are also considering that $|W'| \geq d$ and $|\mathbf{W^{(1)}} \cup \ldots \cup \mathbf{W^{(n)}}| \leq \mathbf{d-1}$.

Case 2.2: $|I_a| \leq t$**.** This means that the number of probes of the form (a) or (b) in W' with distinct values for the index i is at most t.

First, let us check the special case where the number of probes of the form (c) in W' with different values for the index i is equal to n (notice that this cannot occur when we have $|I_a| \geq t+1$). Since $|W| \leq 2 \cdot \min(t+1, n-t) - 1 \leq n$, then we have $W = \{X_{1,j_1}, \ldots, X_{n,j_n}\}$ for certain $j_1, \ldots, j_n \in [n]$. So we can let $J_b^{(k)} = [n]$ for all $k \in [n]$ and $I_b = [n]$, and by construction $I_a = \emptyset$. In this case, we choose $J = [n-1]$. The simulation of the set W is

straightforward since all wires of the form (c) are just sums of random values. Then, let us consider the output shares in J.

- If for at least one $\ell \in J$, we have $X_{\ell,n} \in W$, we can mask the expression of $X_{\ell,n}$ by the random value $r_{n,\ell}$ (because there are no probes of the form (a) or (b) in W and $n \notin J$, so $r_{n,\ell}$ only appears in the expression of $X_{\ell,n}$). Recall that $X_{\ell,n} = r_{1,\ell} + \ldots + r_{n,\ell}$, so $r_{n,\ell}$ masks all random values $r_{j,\ell}$ for $j \in [n-1]$. Each of the random values $r_{j,\ell}$ for $j \in [n-1] \setminus \{\ell\}$ can be used to mask the corresponding output share c_j for $j \in J$ because there are no probes of the form (a) or (b) in W and $X_{\ell,n}$ is already masked by $r_{n,\ell}$, so $r_{j,\ell}$ only appears in the expression of $c_j = V_j + X_j$, so $c_j \leftarrow r_{j,\ell}$. As for the output c_ℓ, we can let $I_a = \{\ell\}$ and we can perfectly simulate c_ℓ using a_ℓ and $I_b = [n]$. Since $|W| = n$, so $\min(t+1, n-t) > \dfrac{n}{2} \geq 1$, so we have no failure on the input a, and we can perfectly simulate the chosen set J and the set of probes W.

- Now we consider that for all $W = \{X_{1,j_1}, \ldots, X_{n,j_n}\}$, we have $j_1 < n, \ldots, j_n < n$. In this case, the set W is also trivially simulated by generating random values, and we let $J = [n-1]$. Since, $n \notin J$ and there are no probes of the form (a) or (b) in W, then the random values $r_{n,i}$ for $i \in [n-1]$ only appear in the expression of the output share $c_i = V_i + X_i$ each. And since all probes of the form $X_{i,j}$ are such that $j < n$, then we can let $r_{n,i}$ be used to mask the expression of $c_i + X_{i,j}$ because $r_{n,i}$ does not appear in $X_{i,j}$ for $j < n$, i.e $c_i + X_{i,j} \leftarrow r_{n,i}$. By perfectly simulating the masked expression of $c_i + X_{i,j}$ and the sum of random values $X_{i,j}$, we can perfectly simulate c_i. Thus, simulating all output shares in J amounts to generating random values uniformly. So we can perfectly simulate sets W and J from $I_a = \emptyset$ and $I_b = [n]$.

Next, we suppose that the number of probes of the form (c) in W' with different values for the index i is strictly smaller than n. So, there is at least one index ℓ such that $X_{\ell,j} \notin W'$ for all $j \in [n]$. We let $J = [n] \setminus \{\ell\}$. We also let $I_b = [n]$ and we keep the set I_a as constructed according to the probes in the set W'. Observe that all probes in W' are perfectly simulated by easily using the set I_a and $I_b = [n]$. As for the output shares in J, observe that for each $i \in J$ such that $i \notin I_a$, we can use claim 5 to mask the expression of c_i by $r_{i,\ell}$, and so the share a_i is not needed for the simulation of c_i. Otherwise, if $i \in J \cap I_a$, then c_i is perfectly simulated using a_i and $I_b = [n]$.

This proves that whenever $i \notin I_a$, the output share c_i can be simulated without the need of the share a_i. Since we suppose that $|I_a| \leq t$, then we conclude that we can perfectly simulate W and a chosen set of $n-1$ output shares J with at most a simulation failure on input b.

By considering both cases $|I_a| \geq t+1$ and $|I_a| \leq t$, we covered all the cases for the simulation, and we proved that we can always perfectly simulate the set of probes W along with a chosen set of $n-1$ output shares J while having a failure on at most one of the inputs. This concludes the proof of Lemma 29.

□

From Lemmas 28 and 29, we conclude that G_{mult} is (t, f_2)-TRPE2 of amplification order $d \geq \min(t+1, n-t)$. This concludes the proof of Lemma 27. □

A.2 Proof of Theorem 3

We consider that we have ℓ compilers $CC_1, \ldots, CC_\ell$, and we want to prove the following result:

Lemma 30. *Let $CC_1, \ldots, CC_\ell$ RPE compilers with expanding functions $f_1, \ldots, f_\ell$. The dynamic expanding compiler for $CC_1, \ldots, CC_\ell$,which on input circuit C outputs the compiled circuit $CC_\ell \circ \cdots \circ CC_1(C)$, is an RPE compiler with expanding function f such that*

$$f = f_\ell \circ \cdots \circ f_1.$$

It can be seen that proving Lemma 30 implies proving the result of Theorem 3. Indeed, we can replace ℓ in the lemma by $k_1 + \ldots + k_\mu$ from Theorem 3 and consider the corresponding compilers with their expansion levels. Thus, we will prove in this appendix Lemma 30 and the proof of the Theorem will follow directly.

To prove the lemma, we first start by introducing some definitions from [18] for random probing expandability of level-ℓ with different sharing orders $n_1, \ldots, n_\ell$ gadgets. First, we introduce a generalized definition of *adequate subsets of* $[n_1 \times \ldots \times n_\ell]$ as in [18]. For this, we define recursively a family $S_k \in \mathcal{P}([n_1 \times \ldots \times n_k])$ for $k \leq \ell$, where $\mathcal{P}([n_1 \times \ldots \times n_k])$ denotes the set of all subsets of $[n_1 \times \ldots \times n_k]$, as follows:

$$\begin{aligned} S_1(n,t) =& \{I \in [n],\ |I| \leq t\} \\ S_k(\{n_i\}_{i\in[k]}, \{t_i\}_{i\in[k]}) =& \{(I_1, \ldots, I_{n_k}) \in (S_{k-1}(\{n_i\}_{i\in[k-1]}, \{t_i\}_{i\in[k-1]}) \cup [n_1 \times \ldots n_{k-1}])^{n_k}, \\ & I_j \in S_{k-1}\ \forall\ j \in [1, n_k] \text{ except at most } t_k\} \end{aligned}$$

In other words, a subset I belongs to S_k if among the n_k subset parts of I, at most t_k of them are full, while the other ones recursively belong to S_{k-1}. For simplicity, we will sometimes denote S_k without the parameters $(\{n_i\}_{i\in[k]}, \{t_i\}_{i\in[k]})$ which will be implicit in the notation. We will also denote for simplicity $N_i = n_1 \cdot \ldots \cdot n_i$ for $i \in \mathbb{N}$.

Then we recall the generalized definition of RPE with S_k for level-k gadgets.

Definition 28 (Random Probing Expandability with $\{S_k\}_{k\in\mathbb{N}}$)**.** *Let $f : \mathbb{R} \to \mathbb{R}$ and $k \in \mathbb{N}$. An N_k-share gadget $G : \mathbb{K}^{N_k} \times \mathbb{K}^{N_k} \to \mathbb{K}^{N_k}$ is (S_k, f)*-random probing expandable (RPE) *if there exists a deterministic algorithm Sim_1^G and a probabilistic algorithm Sim_2^G such that for every input $(\widehat{x}, \widehat{y}) \in \mathbb{K}^{N_k} \times \mathbb{K}^{N_k}$, for every set $J \in S_k \cup [N_k]$ and for every $p \in [0,1]$, the random experiment*

$$\begin{aligned} & W \leftarrow \mathsf{LeakingWires}(G, p) \\ & (I_1, I_2, J') \leftarrow \mathsf{Sim}_1^G(W, J) \\ & out \leftarrow \mathsf{Sim}_2^G(W, J', \widehat{x}|_{I_1}, \widehat{y}|_{I_2}) \end{aligned}$$

ensures that

1. *the failure events $\mathbb{F}_1 \equiv \big(I_1 \notin S_k\big)$ and $\mathbb{F}_2 \equiv \big(I_2 \notin S_k\big)$ verify*

$$\Pr(\mathbb{F}_1) = \Pr(\mathbb{F}_2) = \varepsilon \quad \text{and} \quad \Pr(\mathbb{F}_1 \wedge \mathbb{F}_2) = \varepsilon^2 \tag{A.1}$$

   with $\varepsilon = f(p)$ (in particular $\mathbb{F}_1$ and $\mathbb{F}_2$ are mutually independent),

2. *the set J' is such that $J' = J$ if $J \in S_k$, and $J' = [N_k] \setminus \{j^\star\}$ for some $j^\star \in [N_k]$ otherwise,*

3. *the output distribution satisfies*

$$out \overset{id}{=} \left(\mathsf{AssignWires}(G, W, (\widehat{x}, \widehat{y}))\ ,\ \widehat{z}|_{J'}\right) \tag{A.2}$$

where $\widehat{z} = G(\widehat{x}, \widehat{y})$.

We are now ready to prove Lemma 30.

Lemma 30. We will prove the Lemma recursively. In other words, we will suppose that we have RPE compilers $CC_1, \ldots, CC_k$ with expanding functions $f_1, \ldots, f_k$ and (t_i, f_i)-RPE gadgets for each $i \leq k$, and we will prove that the gadgets of the expanding compiler $CC_k \circ \cdots \circ CC_1$ are (S_k, f)-RPE with $f = f_k \circ \cdots \circ f_1$. This will imply that the expanding compiler $CC_k \circ \cdots \circ CC_1$ is RPE with expanding function f.

The base case is one of the theorem hypotheses, namely for $k = 1$, the level-1 gadgets are (t_1, f_1)-RPE, which is equivalent to $(\mathcal{S}_1, f_1)$-RPE. We must then show the induction step: assuming that the level-k gadgets are $(S_k, f_k \circ \cdots \circ f_1)$-RPE, show that the level-$(k+1)$ gadgets are $(S_{k+1}, f_{k+1} \circ \cdots \circ f_1)$-RPE. For the sake of simplicity, we depict our proof by assuming that all the gadgets are 2-to-1 gadget (which is actually not the case for copy gadgets). The proof mechanism for the general case (with 2-to-1 and 1-to-2 gadgets) is strictly similar but heavier on the form. We also denote in the following

- $\varepsilon_k = f_k \circ \cdots f_1(p)$,
- G^{CC_k} to be a gadget of the expanding compiler CC_k,
- $G^{(k)}$ to be the gadget resulting from applying $CC_{k-1} \circ \ldots \circ CC_1(G^{CC_k})$, *i.e.* obtained by replacing each gate of the base gadget G^{CC_k} by the corresponding level-$(k-1)$ gadget $G^{(k-1)}$ and by replacing each wire of the base gadget by N_{k-1} wires carrying a N_{k-1}-linear sharing of the original wire.

In order to show that a gadget $G^{(k+1)}$ is $(S_{k+1}, \varepsilon_{k+1})$-RPE we must construct two simulators $\mathsf{Sim}_1^{G^{(k+1)}}$ and $\mathsf{Sim}_2^{G^{(k+1)}}$ that satisfy the conditions of Definition 28 for the set of subsets S_{k+1}. More precisely, we must construct two simulators $\mathsf{Sim}_1^{G^{(k+1)}}$ and $\mathsf{Sim}_2^{G^{(k+1)}}$ such that for every $(\widehat{x}^*, \widehat{y}^*) \in \mathbb{K}^{N_{k+1}} \times \mathbb{K}^{N_{k+1}}$, and for every set $J^* \in S_{k+1} \cup [N_{k+1}]$, the random experiment

$$\begin{aligned} &W^* \leftarrow \mathsf{LeakingWires}(G^{(k+1)}, p) \\ &(I_1^*, I_2^*, J^{*\prime}) \leftarrow \mathsf{Sim}_1^{G^{(k+1)}}(W^*, J^*) \\ &out \leftarrow \mathsf{Sim}_2^{G^{(k+1)}}(W^*, J^*, \widehat{x}^*|_{I_1^*}, \widehat{y}^*|_{I_2^*}) \end{aligned}$$

ensures that

1. the failure events $\mathbb{F}_1^* \equiv \left(I_1^* \notin S_{k+1}\right)$ and $\mathbb{F}_2^* \equiv \left(I_2^* \notin S_{k+1}\right)$ verify
$$\Pr(\mathbb{F}_1^*) = \Pr(\mathbb{F}_2^*) = \varepsilon_{k+1} \quad \text{and} \quad \Pr(\mathbb{F}_1^* \wedge \mathbb{F}_2^*) = \varepsilon_{k+1}^2 \tag{A.3}$$
2. the set $J^{*\prime}$ is such that $J^{*\prime} = J^*$ if $J^* \in S_{k+1}$ and $J^{*\prime} = [N_{k+1}] \setminus \{j^\star\}$ otherwise,
3. the output distribution satisfies
$$out \overset{\text{id}}{=} \left(\mathsf{AssignWires}(G^{(k+1)}, W, (\widehat{x}, \widehat{y}))\ ,\ \widehat{z}|_{J^{*\prime}}\right) \tag{A.4}$$

where $\widehat{z} = G^{(k+1)}(\widehat{x}, \widehat{y})$.

We distinguish two cases: either $J^* \in S_{k+1}$ (normal case), or $J^* = [N_{k+1}]$ (saturated case).

Normal case: $J^* \in S_{k+1}$**.**

By definition of the expanding compiler, we have that a level-$(k+1)$ gadget $G^{(k+1)}$ is obtained by replacing each gate of the base gadget $G^{CC_{k+1}}$ of the compiler CC_{k+1} by the corresponding level-k gadget $G^{(k)}$ and by replacing each wire of the base gadget by N_k wires carrying a N_k-linear sharing of the original wire. In particular $G^{(k+1)}$ has N_{k+1} output wires which can be split in n_{k+1} groups of N_k wires, each group being the output of a different $G^{(k)}$ gadget. We split the set J^* accordingly so that $J^* = J_1^* \cup \cdots \cup J_{n_{k+1}}^*$, where each set J_i^* pertains to the ith group of output wires. By definition of S_k, since $J^* \in S_{k+1}$, we must have $J_i^* \in S_k$ for all $1 \leq i \leq n_{k+1}$, except at most t_{k+1} of them for which $J_i^* = [N_k]$. We define J_{base} as the set of indexes i such that $J_i^* \notin S_k$. Therefore we must have $|J_{\text{base}}| \leq t_{k+1}$.

We first describe the simulator $\mathsf{Sim}_1^{G^{(k+1)}}$ that takes the leaking wires W^* and the output wires $J^* \in S_{k+1}$ to be simulated and produce the sets $I_1^* \subseteq [N_{k+1}]$ and $I_2^* \subseteq [N_{k+1}]$ of required inputs. The simulator $\mathsf{Sim}_1^{G^{(k+1)}}$ starts by defining a set W_{base} which is initialized to $\emptyset$; this will correspond to the set of leaking wires for the base gadget $G^{CC_{k+1}}$. Then the simulation goes through all the level-k gadgets composing $G^{(k+1)}$ from bottom to top *i.e.* starting with the level-k gadgets producing the output sharing up to the level-k gadgets processing the input sharings. Let us denote by $\{G_j^{(k)}\}_j$ these level-k gadgets. For each $G_j^{(k)}$, one runs the simulator Sim_1 from the $(S_k, f_k \circ \ldots \circ f_1)$-RPE property on input W_j and J_j defined as follows. The set of leaking wires W_j is defined as the subset of W^* corresponding to the wires of $G_j^{(k)}$. For the gadgets $G_j^{(k)}$ on the bottom layer, the set J_j is set to one of the J_i^* (with indices scaled to range in $[N_k]$). For all the other gadgets $G_j^{(k)}$ (which are not on the bottom layer), the set J is defined as the set I_1 or I_2 output from Sim_1 for the child gadget $G_{j'}^{(k)}$ (for which Sim_1 has already been run).

Whenever a failure event occurs for a $G_j^{(k)}$ gadget, namely when the set I (either I_1 or I_2) output from Sim_1 is such that $I \notin S_k$, we add the index of the wire corresponding to this input in the base gadget $G^{CC_{k+1}}$ to the set W_{base}. Once the Sim_1 simulations have been run for all the $G_j^{(k)}$ gadgets, ending with the top layers, we get the final sets I corresponding to the input shares. Each of these sets corresponds to an N_k-sharing as input of a $G_j^{(k)}$ gadget, which corresponds to a wire as input of the base gadget among the $2 \cdot n_{k+1}$ wires carrying the two input n_{k+1}-sharings of the base gadget. We denote by $I_{1,1}^*, \ldots, I_{1,n_{k+1}}^*$ and $I_{2,1}^*, \ldots, I_{2,n_{k+1}}^*$ the corresponding sets so that defining

$$I_1^* = I_{1,1}^* \cup \ldots \cup I_{1,n_{k+1}}^* \quad \text{and} \quad I_2^* = I_{2,1}^* \cup \ldots \cup I_{2,n_{k+1}}^* \ , \tag{A.5}$$

the tuple $\widehat{x}^*|_{I_1^*}$ and $\widehat{y}^*|_{I_2^*}$ contains the shares designated by the final I sets.

At the end of the $\mathsf{Sim}_1^{G^{(k+1)}}$ simulation, the set W_{base} contains all the labels of wires in the base gadget $G^{CC_{k+1}}$ for which a failure event has occurred in the simulation of the corresponding $G_j^{(k)}$ gadget. Thanks to the (S_k, ε_k)-RPE property of these gadgets, the failure events happen (mutually independently) with probability ε_k which implies

$$W_{\text{base}} \overset{\text{id}}{=} \mathsf{LeakingWires}(G^{CC_{k+1}}, \varepsilon_k) \tag{A.6}$$

Recall that $|J_{\text{base}}| \leq t_{k+1}$. We can then run $\mathsf{Sim}_1^{G^{CC_{k+1}}}$ to obtain:

$$(I_{1,\text{base}}, I_{2,\text{base}}) = \mathsf{Sim}_1^{G^{CC_{k+1}}}(W_{\text{base}}, J_{\text{base}}) \ . \tag{A.7}$$

For all $1 \leq i \leq n_{k+1}$, if $i \in I_{1,\text{base}}$, we force $I_{1,i}^* \leftarrow [N_k]$, so that the corresponding i-th input wire of the base gadget can be computed from the corresponding input wires in $I_{1,i}^*$. The simulator $\mathsf{Sim}_1^{G^{(k+1)}}$ then returns (I_1^*, I_2^*) as output.

The (t_{k+1}, f_{k+1})-RPE property of the base gadget $G^{CC_{k+1}}$ implies that the *base failure events* $|I_{1,\text{base}}| = n_{k+1}$ and $|I_{2,\text{base}}| = n_{k+1}$ are ε_{k+1}-mutually unlikely, where $\varepsilon_{k+1} = f_{k+1}(\varepsilon_k)$. We argue that for all $1 \leq i \leq n_{k+1}$, $I^*_{1,i} \notin S_k \iff i \in I_{1,\text{base}}$. Namely if a failure event has occurred for a set $I^*_{1,i}$ (*i.e.* $I^*_{1,i} \notin S_k$) then we must have $i \in I_{1,\text{base}}$. Indeed, if a failure event has occurred for a set $I^*_{1,i}$ then the label of the ith input wire (for the first sharing) of the base gadget $G^{CC_{k+1}}$ has been added to W_{base} and $\mathsf{Sim}_1^{G^{CC_{k+1}}}$ has no choice but to include this index to the set $I_{1,\text{base}}$ so that $\mathsf{Sim}_2^{G^{CC_{k+1}}}$ can achieve a perfect simulation of the wire assignment (as required by the RPE property of $G^{CC_{k+1}}$). Moreover if $i \in I_{1,\text{base}}$ then by construction we have set $I^*_{1,i} = [N_k]$ and therefore $I^*_{1,i} \notin S_k$. This implies that if $|I_{1,\text{base}}| \leq t_{k+1}$ then $I^*_1 \in S_{k+1}$ (and the same happens for I^*_2 w.r.t. $I_{2,\text{base}}$). We deduce that the failure events $\mathbb{F}^*_1$ and $\mathbb{F}^*_2$ are also ε_{k+1}-mutually unlikely, as required by the $(S_{k+1}, \varepsilon_{k+1})$-RPE property of $G^{(k+1)}$.

We now describe the simulator $\mathsf{Sim}_2^{G^{(k+1)}}$ that takes as input $\widehat{x}^*|_{I^*_1}$ and $\widehat{y}^*|_{I^*_2}$ and produces a perfect simulation of $\big(\mathsf{AssignWires}(G^{(k+1)}, W^*, (\widehat{x}^*, \widehat{y}^*)), \widehat{z}|_{J^*}\big)$ where $\widehat{z} = G^{(k+1)}(\widehat{x}, \widehat{y})$. Let $\widehat{x}^b$ and $\widehat{y}^b$ denote the n_{k+1}-linear sharings obtained by applying the linear decoding to each group of N_k shares in $\widehat{x}^*$ and $\widehat{y}^*$, so that the elements of $\widehat{x}^b$ and $\widehat{y}^b$ correspond to the input wires in the base gadget $G^{CC_{k+1}}$. The assignment expansion property implies that a perfect assignment of the wires of $G^{(k+1)}$ on input $\widehat{x}^*$ and $\widehat{y}^*$ can be derived from an assignement of the wires of the base gadget $G^{CC_{k+1}}$ on input $\widehat{x}^b$ and $\widehat{y}^b$. The simulator makes use of this property by first running

$$out_{\text{base}} \leftarrow \mathsf{Sim}_2^{G^{CC_{k+1}}}(W_{\text{base}}, J_{\text{base}}, \widehat{x}^b|_{I_{1,\text{base}}}, \widehat{y}^b|_{I_{2,\text{base}}}) \ , \tag{A.8}$$

Note that the input values $\widehat{x}^b|_{I_{1,\text{base}}}$ and $\widehat{y}^b|_{I_{2,\text{base}}}$ can be obtained from the corresponding shares in I^*_1 and I^*_2. Thanks to the (t_{k+1}, f_{k+1})-RPE property of $G^{CC_{k+1}}$ and by construction of $I_{1,\text{base}}$ and $I_{2,\text{base}}$, this outputs a distribution satisfying

$$out_{\text{base}} \overset{\text{id}}{=} \Big(\mathsf{AssignWires}(G^{CC_{k+1}}, W_{\text{base}}, (\widehat{x}^b, \widehat{y}^b)),\ \widehat{z}^b|_{J_{\text{base}}}\Big) \tag{A.9}$$

The simulator then goes through all the $G^{(k)}_j$ gadgets from input to output and for each of them runs the simulator Sim_2 of the RPE property on inputs W_j, J_j, $\widehat{x}|_{I_1}$ and $\widehat{y}|_{I_2}$ where W_j and J_j are the sets from the first phase of the simulation for the gadget $G^{(k)}_j$, I_1 and I_2 are the corresponding sets produced by the Sim_1 simulator for $G^{(k)}_j$, and $\widehat{x}$ and $\widehat{y}$ are the inputs of $G^{(k)}_j$ in the evaluation of $G^{(k+1)}(\widehat{x}^*, \widehat{y}^*)$. Provided that the partial inputs $\widehat{x}|_{I_1}$ and $\widehat{y}|_{I_2}$ are perfectly simulated, this call to Sim_2 produces a perfect simulation of $\big(\mathsf{AssignWires}(G^{(k)}_j, W_j, (\widehat{x}, \widehat{y}), \widehat{z}|_{J_j}\big)$ where $\widehat{z} = G^{(k)}_j(\widehat{x}, \widehat{y})$. In order to get perfect simulations of the partial inputs $\widehat{x}|_{I_1}$ and $\widehat{y}|_{I_2}$, the simulator proceeds as follows. For the top layer of $G^{(k)}$ gadgets (the ones processing the input shares) the shares $\widehat{x}|_{I_1}$ and $\widehat{y}|_{I_2}$ can directly be taken from the inputs $\widehat{x}^*|_{I^*_1}$ and $\widehat{y}^*|_{I^*_2}$. For the next gadgets the shares $\widehat{x}|_{I_1}$ and $\widehat{y}|_{I_2}$ match the shares $\widehat{z}|_J$ output from the call to Sim_2 for a parent gadget. The only exception occurs in case of a failure event.

In that case the simulation needs the full input $\widehat{x} = (x_1, \ldots, x_{N_k})$ (and/or $\widehat{y} = (y_1, \ldots, y_{N_k})$), while we have set $|I_1| = N_k - 1$ (and/or $|I_2| = (N_k - 1)$ to satisfy the RPE requirements of the parent gadget in the first simulation phase. Nevertheless, for such cases a perfect simulation of the plain value $x = \mathsf{LinDec}(\widehat{x})$ (and/or $y = \mathsf{LinDec}(\widehat{y})$) is included to out_{base} by construction of W_{base}. We can therefore perfectly simulate the missing share from the $N_k - 1$ other shares and the plain value x (or y). We thus get a perfect simulation of $\big(\mathsf{AssignWires}(G^{(k)}_j, W_j, (\widehat{x}, \widehat{y}), \widehat{z}|_{J_j}\big)$ for all the level-k gadgets $G^{(k)}_j$ which gives us a perfect simulation of $\big(\mathsf{AssignWires}(G^{(k+1)}, W^*, (\widehat{x}^*, \widehat{y}^*)), \widehat{z}|_{J^*}\big)$.

Saturated case: $J^* = [N_{k+1}]$**.**

The saturated case proceeds similarly. The difference is that we must simulate all N_{k+1} output shares of the level-$(k+1)$ gadget, except for one share index j^* that can be chosen by the simulator.

The simulator $\mathsf{Sim}_1^{G^{(k+1)}}$ is defined as previously. Since $J^* = [N_{k+1}]$, we must define $J_{\text{base}} = [1, n_{k+1}]$. Moreover we have $J_i^* = [N_k]$ for all $1 \leq i \leq n_{k+1}$. This implies that for the gadgets $G_j^{(k)}$ on the output layer, the sets J_j are all equal to $[N_k]$ as well. The set W_{base} is defined as previously, and the simulator $\mathsf{Sim}_1^{G^{(k+1)}}$ returns (I_1^*, I_2^*) as previously. The failure events $\mathbb{F}_1^*$ and $\mathbb{F}_2^*$ are still ε_{k+1}-mutually unlikely, as required by the $(S_{k+1}, \varepsilon_{k+1})$-RPE property of $G^{(k+1)}$.

The simulator $\mathsf{Sim}_2^{G^{(k+1)}}$ is defined as previously. In particular, from the running of the base gadget simulator $\mathsf{Sim}_2^{G^{CC_{k+1}}}$, we obtain a perfect simulation of the output wires $\widehat{z}^b|_{J'_{\text{base}}}$ for some J'_{base} with $|J'_{\text{base}}| = n_{k+1} - 1$. Combined with the perfect simulation of the output wires corresponding to the output sets J'_j from the gadgets $G_j^{(k)}$ on the output layer, with $|J'_j| = N_k - 1$, we obtain a subset J' of output wires for our level-$(k+1)$ gadget with $|J'| = N_{k+1} - 1$ as required. Eventually this gives us a perfect simulation of $\big(\mathsf{AssignWires}(G^{(k+1)}, W^*, (\widehat{x}^*, \widehat{y}^*)), \widehat{z}|_{J'}\big)$. This terminates the proof of Lemma 30. As stated earlier, proving Lemma 30 implies proving Theorem 3. Thus, this also terminates the proof for the theorem.

□

A.3 Proof of Proposition 4

Proof. We consider the following matrices

$$\mathbf{L} = \left[\begin{array}{c|c|c|c|c|c|c|c|c} \mathbf{I}_n & \mathbf{0}_{n\times n} & \mathbf{I}_n & \mathbf{I}_n & \dots & \mathbf{I}_n & \mathbf{T}_n & \mathbf{T}_n & \dots & \mathbf{T}_n \end{array} \right]$$

$$\mathbf{M} = \left[\begin{array}{c|c|c|c|c|c|c|c|c} \mathbf{0}_{n\times n} & \mathbf{I}_n & \mathbf{I}_n & \mathbf{D}_{\gamma,1} & \dots & \mathbf{D}_{\gamma,n} & \mathbf{T}_n & \mathbf{T}_{\gamma,1} & \dots & \mathbf{T}_{\gamma,n} \end{array} \right]$$

The matrices $\mathbf{L}$ and $\mathbf{M}$ have $z = (2n+4)\cdot n$ columns. We want to lower-bound the probability, for $\boldsymbol{\gamma}$ picked uniformly at random in $\mathbb{F}_q^{n\times n}$, that for any vector $v \in \mathbb{F}_q^z$ of Hamming weight $hw(v) \leq n$, and for any $i_1, \dots, i_K \in [z]$ such that $v_{i_1} \neq 0, \dots, v_{i_K} \neq 0$ and the corresponding columns $i_1, \dots, i_K$ in $\mathbf{L}$ and in $\mathbf{M}$ have no zero coefficient (*i.e* there are K probes of the form $\sum_{i=1}^{n}(r_i + a_i)$ or $\sum_{j=1}^{n}(\gamma_{i,j} r_j + a_j)$ for any $i = 1, \dots, n$), if $\mathbf{M}.v = 0$, then we have $hw(\mathbf{L}.v) \leq hw(v) - K$.

For any set $I \subseteq \{1, \dots, z\}$, we denote by $\mathbf{L}_I$ the $n \times |I|$ submatrix of $\mathbf{L}$ obtained by only keeping the columns in $\mathbf{L}$ whose indices are in I and $\mathbf{M}_I$ is the $n \times |I|$ submatrix of $\mathbf{M}$ obtained by only keeping the columns in $\mathbf{M}$ whose indices are in I. We will lower-bound the probability that for any set $I \subseteq \{1, \dots, z\}$ of cardinal n and any vector $\boldsymbol{v} \in \mathbb{F}_q^n$, if $hw(\mathbf{L}_I \cdot \boldsymbol{v}) \geq hw(v) - K + 1$ then $\mathbf{M}_I \cdot \boldsymbol{v} \neq \mathbf{0}_n$.

We consider different cases (in order of increasing generality) which depend on the columns selected with the set I:

1. $I \subseteq \{(n+4)\cdot n + 1, \dots, z\}$, i.e., all columns in $\mathbf{M}_I$ are taken from the matrices $\mathbf{T}_{\boldsymbol{\gamma},i}$ for $i \in \{1, \dots, n\}$;
2. $I \subseteq \{(n+3)\cdot n + 1, \dots, z\}$, i.e., all columns in $\mathbf{M}_I$ are taken from the matrix $\mathbf{T}_n$ or the matrices $\mathbf{T}_{\boldsymbol{\gamma},i}$ for $i \in \{1, \dots, n\}$;
3. $I \subseteq \{1, \dots, n+1\} \cup \{(n+3)\cdot n + 1, \dots, z\}$, i.e., all columns in $\mathbf{M}_I$ are taken from the null vectors, from the matrix $\mathbf{T}_n$ or the matrices $\mathbf{T}_{\boldsymbol{\gamma},i}$ for $i \in \{1, \dots, n\}$;
4. $I \subseteq \{1, \dots, z\}$, i.e., the columns in $\mathbf{M}_I$ can be taken arbitrarily.

Case 1.

In order to analyze the probability in the first case, we recall the definition of a probability distribution on structured matrices introduced in [17]. In this distribution of structured matrices, a number of elements with known location are identically zero, and remaining elements are chosen uniformly at random independently of each other.

Definition 29. *Let n and m be two positive integers. Let $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_m)$ be a non-decreasing finite sequence with $1 \leq \alpha_1 \leq \alpha_2 \leq \cdots \leq \alpha_m \leq n$.*

- *A matrix $\boldsymbol{\Theta} = (\theta_{i,j}) \in \mathbb{F}_q^{n\times m}$ is called a* progressive patterned matrix *with* pattern $\boldsymbol{\alpha}$ *if $\theta_{i,j} = 0$ for all $j \in \{1, \dots, m\}$ and all $i \notin \{\alpha_{j-1} + 1, \dots, \alpha_j\}$ (where $\alpha_0 = 0$).*
- *The* unitary progressive patterned matrix $\boldsymbol{\Upsilon}_{\boldsymbol{\alpha}} = (u_{i,j}) \in \mathbb{F}_q^{n\times m}$ *with* pattern $\boldsymbol{\alpha}$ *is defined by $u_{i,j} = 0$ for all $j \in \{1, \dots, m\}$ and all $i \notin \{\alpha_{j-1} + 1, \dots, \alpha_j\}$ and $u_{i,j} = 1$ for all $j \in \{1, \dots, m\}$ and all $i \in \{\alpha_{j-1} + 1, \dots, \alpha_j\}$.*
- *The distribution $\mathcal{D}_{\boldsymbol{\alpha}}$ is the probability distribution on* random progressive patterned matrix $\mathbf{S}_{\boldsymbol{\alpha}} = (s_{i,j}) \in \mathbb{F}_q^{n\times m}$ *whose elements $s_{i,j}$ for $(i,j) \in \{1, \dots, n\} \times \{1, \dots, m\}$ are sampled uniformly at random and independently according to:*

$$\Pr[s_{i,j} = s] = \begin{cases} 1 & \text{if } s = 0 \text{ and } u_{i,j} = 0 \\ 0 & \text{if } s \neq 0 \text{ and } u_{i,j} = 0 \\ q^{-1} & \text{for all } s \in \mathbb{F}_q \text{ if } u_{i,j} = 1 \end{cases}$$

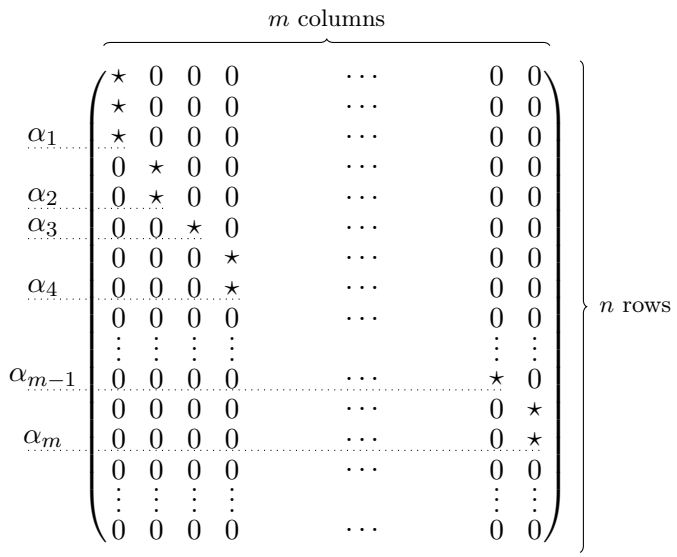

Figure A.2: Form of a progressive patterned matrix with *pattern* $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)$

where $\boldsymbol{\Upsilon}_{\boldsymbol{\alpha}} = (u_{i,j}) \in \mathbb{F}_q^{n \times m}$ *is the unitary progressive patterned matrix with pattern* α*.*

A matrix $\boldsymbol{\Theta}$ is thus a *progressive patterned matrix* with *pattern* $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)$ if it is of the form described in Figure A.2 where the symbol $\star$ denotes an arbitrary value in $\mathbb{F}_q$. For the unitary progressive patterned matrix $\boldsymbol{\Upsilon}_{\boldsymbol{\alpha}}$, this symbol $\star$ is replaced by a 1 and for a random progressive patterned matrix $\mathbf{S}_{\boldsymbol{\alpha}}$ each symbol $\star$ is replaced by a value picked uniformly and independently at random in $\mathbb{F}_q$. Note that such a matrix can contain a null column (when $\alpha_i = \alpha_{i+1}$ for some $i \in \{1, \ldots, m-1\}$).

Belaïd *et al.* [17] also defined more generally block column matrices formed of progressive patterned matrices.

Definition 30. *Let* n, m, t *be three positive integers. Let* $m_1, \ldots, m_t$ *be positive integers such that* $m_1 + \cdots + m_t = m$ *and let* $\boldsymbol{\alpha}^{(i)} = (\alpha_1^{(i)}, \ldots, \alpha_{m_i}^{(i)})$ *be a non-decreasing finite sequence with* $1 \leq \alpha_1^{(i)} \leq \alpha_2^{(i)} \leq \cdots \leq \alpha_{m_i}^{(i)} \leq n$ *for all* $i \in \{1, \ldots, t\}$*. We suppose that there exists at least one* $j \in \{1, \ldots, t\}$ *such that* $\alpha_{m_j}^{(j)} = n$*.*

- *A matrix* $\boldsymbol{\Theta} \in \mathbb{F}_q^{n \times m}$ *is called a* block progressive patterned matrix *with* pattern $(\boldsymbol{\alpha}^{(1)}, \ldots, \boldsymbol{\alpha}^{(t)})$ *if there exist progressive patterned matrices* $\boldsymbol{\Theta}^{(i)} \in \mathbb{F}_q^{n \times m_i}$ *with pattern* $\boldsymbol{\alpha}^{(i)}$ *for all* $i \in \{1, \ldots, t\}$ *such that* $\boldsymbol{\Theta} = (\boldsymbol{\Theta}^{(1)} | \ldots | \boldsymbol{\Theta}^{(t)})$*.*
- *The* block unitary progressive patterned matrix $\boldsymbol{\Upsilon}_{\boldsymbol{\alpha}^{(1)}, \ldots, \boldsymbol{\alpha}^{(t)}} \in \mathbb{F}_q^{n \times m}$ *with* pattern $(\boldsymbol{\alpha}^{(1)}, \ldots, \boldsymbol{\alpha}^{(t)})$ *is* $\boldsymbol{\Upsilon}_{\boldsymbol{\alpha}^{(1)}, \ldots, \boldsymbol{\alpha}^{(t)}} = (\boldsymbol{\Upsilon}_{\boldsymbol{\alpha}^{(1)}} | \ldots | \boldsymbol{\Upsilon}_{\boldsymbol{\alpha}^{(t)}})$*.*
- *The distribution* $\mathcal{D}_{\boldsymbol{\alpha}^{(1)}, \ldots, \boldsymbol{\alpha}^{(t)}}$ *is the probability distribution on* block random progressive patterned matrix *in* $\mathbb{F}_q^{n \times m}$ *defined by*

$$\mathcal{D}_{\boldsymbol{\alpha}^{(1)}, \ldots, \boldsymbol{\alpha}^{(t)}} = (\mathcal{D}_{\boldsymbol{\alpha}^{(1)}} | \ldots | \mathcal{D}_{\boldsymbol{\alpha}^{(t)}}).$$

The main ingredient of the proof of Proposition 4 is the following technical lemma:

Lemma 31. *Let* n, m, t *be three positive integers with* $m \geq n$ *and let* $\boldsymbol{\alpha}^{(i)}$ *for* $i \in \{1, \ldots, t\}$ *be patterns for block progressive patterned matrix as in Definition 30. For a block random progressive patterned matrix* $\mathbf{S}$ *drawn following the distribution* $\mathcal{D}_{\boldsymbol{\alpha}^{(1)}, \ldots, \boldsymbol{\alpha}^{(t)}}$*, there exists a linear subspace of* $\mathbb{F}_q^m$ *of dimension* $m - n$ *that contains* $\{\boldsymbol{v} \in \mathbb{F}_q^m \text{ s.t. } hw(\boldsymbol{v}) = m \text{ and } \mathbf{S}\boldsymbol{v} = \mathbf{0}\}$*, with probability at least* $1 - mq^{-1}$*.*

Lemma 31. We will prove this lemma by induction on m.

For $m = 1$, since $m \geq n \geq 1$, Definition 30 implies that the matrix $\mathbf{S}$ consists simply in a single entry $s_{1,1}$ which is picked uniformly at random in $\mathbb{F}_q$ and this entry is null with probability q^{-1}. The set $\{\boldsymbol{v} \in \mathbb{F}_q \text{ s.t. } hw(\boldsymbol{v}) = 1 \text{ and } \mathbf{S} \cdot \boldsymbol{v} = \mathbf{0}\}$ is therefore the empty set with probability at least $1 - q^{-1}$ and it is thus included in the subspace of dimension 0 with probability at least $1 - q^{-1}$.

We now consider $m \geq 2$ and we suppose Lemma 31 proven for all block random progressive patterned matrix with strictly less than m columns.

We first assume that the matrix $\boldsymbol{\Upsilon}_{\boldsymbol{\alpha}^{(1)},\ldots,\boldsymbol{\alpha}^{(t)}} \in \mathbb{F}_q^{n\times m}$ is the matrix of ones $\mathbf{U}_{n\times m}$ (i.e., does not contain any zero). Then $\mathbf{S}$ is simply a matrix drawn from $\mathbb{F}_q^{n\times m}$ with the uniform distribution.

It is well known that the number of full-rank $n \times m$ matrices over $\mathbb{F}_q$ (with $m \geq n$) is:

$$(q^m - 1)(q^m - q) \cdots (q^m - q^{n-1})$$

and the probability that $\mathbf{S}$ is of full rank is thus equal to:

$$(1 - q^{-m})(1 - q^{-m+1}) \ldots (1 - q^{-m+n-1})$$

which is greater than

$$1 - \sum_{i=m-n+1}^{m} q^{-i} \geq 1 - \sum_{i=m-n+1}^{\infty} q^{-i} = 1 - \frac{1}{q^{-m+n-1}(1 - 1/q)} \geq 1 - 2q^{n-m-1}.$$

The subspace $\{\boldsymbol{v} \in \mathbb{F}_q^m \text{ s.t. } \mathbf{S} \cdot \boldsymbol{v} = 0\}$ is therefore included in a linear subspace of dimension $m - n$ with probability at least $1 - 2q^{n-m-1}$ and the result follows (since $m \geq 2$).

We now assume that the matrix $\boldsymbol{\Upsilon}_{\boldsymbol{\alpha}^{(1)},\ldots,\boldsymbol{\alpha}^{(t)}} \in \mathbb{F}_q^{n\times m}$ contains some 0. By assumption, there exists some $j \in \{1, \ldots, t\}$ such that $\alpha_{m_j}^{(j)} = n$.

1. We first assume that $m_j > 1$ (*i.e.* that the column of index $m_1 + \cdots + m_j$ consists in $\alpha_{m_j-1}^{(j)} \geq 1$ zeroes followed by $\alpha_{m_j}^{(j)} - \alpha_{m_j-1}^{(j)} = n - \alpha_{m_j-1}^{(j)} \geq 1$ ones, see Figure A.3). We consider the submatrix of $\boldsymbol{\Upsilon}_{\boldsymbol{\alpha}^{(1)},\ldots,\boldsymbol{\alpha}^{(t)}} \in \mathbb{F}_q^{n\times m}$ obtained by deleting the column of index $m_1 + \cdots + m_j$ and the rows of indices in the set $\{\alpha_{m_j-1}^{(j)} + 1, \ldots, \alpha_{m_j}^{(j)}\}$.

   It is easy to see that this submatrix is a block unitary progressive patterned matrix with $n' \leq n - 1$ rows and $m - 1$ columns, where some columns may possibly contain only zeroes (see Figure A.3). We can thus apply the induction hypothesis to the submatrix $\mathbf{S}'$ of $\mathbf{S}$ obtained by deleting the same column and the same rows.

   By induction hypothesis, we know that with probability at least $1 - (m-1)q^{-1}$, there exists a linear subspace $V' \subseteq \mathbb{F}_q^{m-1}$ of dimension $m - 1 - n'$ that contains the set $\{\boldsymbol{v} \in \mathbb{F}_q^{m-1} \text{ s.t. } hw(\boldsymbol{v}) = m - 1 \text{ and } \mathbf{S}' \cdot \boldsymbol{v} = 0\}$.

   If V' is of dimension 0, then $\{\boldsymbol{v} \in \mathbb{F}_q^{m-1} \text{ s.t. } hw(\boldsymbol{v}) = m - 1 \text{ and } \mathbf{S}' \cdot \boldsymbol{v} = 0\} \subseteq \{\mathbf{0}_{m-1}\}$ and this set is thus the empty set. We then have $\{\boldsymbol{v} \in \mathbb{F}_q^m, hw(\boldsymbol{v}) = m \text{ and } \mathbf{S} \cdot \boldsymbol{v} = 0\} = \emptyset$ with probability at least $1 - (m-1)q^{-1} \geq 1 - mq^{-1}$, and so there exists a linear subspace V of dimension $m - n$ that contains this set.

   If V' is of dimension $m - 1 - n' > 0$, we can assume without loss of generality that the column of $\mathbf{S}$ deleted to obtain $\mathbf{S}'$ was the last one (by permuting the blocks of the matrix). We have the following block-decomposition of $\mathbf{S}$

$$\mathbf{S} = \begin{pmatrix} \mathbf{S}' & \mathbf{0}_{\mathbf{n}'\times\mathbf{1}} \\ \mathbf{S}'' & \mathbf{u} \end{pmatrix}$$

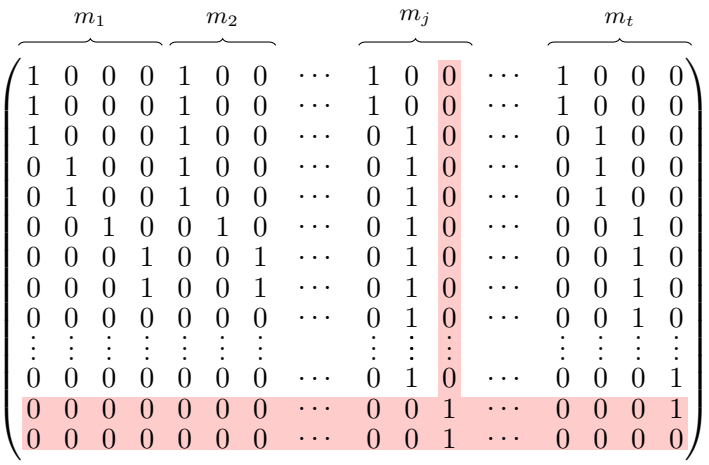

Figure A.3: Example of a matrix $\boldsymbol{\Upsilon}_{\boldsymbol{\alpha}^{(1)},\ldots,\boldsymbol{\alpha}^{(t)}} \in \mathbb{F}_q^{n\times m}$. The column and the rows highlighted in red are deleted in order to apply the induction hypothesis.

where $\mathbf{S}''$ is a $(n-n')\times(m-1)$ matrix and $\mathbf{u}$ a column vector of dimension $(n-n')$. Note that $\boldsymbol{u}$ is a random vector in $\mathbb{F}_q^{n-n'}$ independent from $\mathbf{S}'$ and $\mathbf{S}''$. Let $\boldsymbol{v}\in\mathbb{F}_q^m$ such that $hw(\boldsymbol{v})=m$ and $\mathbf{S}\boldsymbol{v}=0$.

We write $\boldsymbol{v}=\begin{pmatrix}\boldsymbol{w}\\ \tau\end{pmatrix}$ where $\boldsymbol{w}\in\mathbb{F}_q^{m-1}$ and $\tau\in\mathbb{F}_q$ is a scalar. We have $hw(\boldsymbol{w})=m-1$ and $\mathbf{S}'\boldsymbol{w}=0$, and therefore $\boldsymbol{w}\in V'$. Since $\tau\neq 0$ by assumption, the vector $\boldsymbol{u}$ thus belongs to the image W of V' by $\mathbf{S}''$ (with probability at least $1-(m-1)q^{-1}$). Moreover, W has dimension at most $\max(m-1-n', n-n')$.

- If W is of dimension at most $n-n'-1$, since $\boldsymbol{u}$ is independent of $\mathbf{S}'$ and $\mathbf{S}''$ (and thus of W), $\boldsymbol{u}$ belongs to W with probability at most q^{-1}. Therefore, with probability at least $(1-q^{-1})\cdot(1-(m-1)q^{-1})\geq 1-mq^{-1}$, $\{\boldsymbol{v}\in\mathbb{F}_q^m \text{ s.t. } hw(\boldsymbol{v})=m \text{ and } \mathbf{S}\boldsymbol{v}=0\}=\emptyset$.
- If W is of dimension $n-n'$, with probability $1-q^{-(n-n')}\geq 1-q^{-1}$, we have $\boldsymbol{u}\neq\mathbf{0}_{(\boldsymbol{n-n'})\times\mathbf{1}}$ and we can construct a basis $\boldsymbol{u_1}=\boldsymbol{u}, \ldots, \boldsymbol{u_{n-n'}}$ of W.
  All subspaces $V'\cap{\mathbf{S}''}^{-1}(\langle\boldsymbol{u_i}\rangle)$ are of dimension at least one and we have

$$V'=\bigoplus_{i=1}^{n-n'} V'\cap{\mathbf{S}''}^{-1}(\langle\boldsymbol{u_i}\rangle).$$

  Therefore the linear subspace V defined as $V=V'\cap{\mathbf{S}''}^{-1}(\langle\boldsymbol{u_1}\rangle)$ satisfies

$$\begin{aligned}\dim(V)&=\dim(V')-\sum_{i=2}^{n-n'}\dim\left(V'\cap{\mathbf{S}''}^{-1}(\langle\boldsymbol{u_i}\rangle)\right)\\&\leq m-1-n'-(n-n'-1)\\&=m-n.\end{aligned}$$

  Moreover, we have $\{\boldsymbol{v}\in\mathbb{F}_q^m \text{ s.t. } hw(\boldsymbol{v})=m \text{ and } \mathbf{S}\boldsymbol{v}=0\}\subseteq V$ and since this occurs with probability at least $(1-q^{-1})(1-(m-1)q^{-1})\geq 1-mq^{-1}$, the result follows.

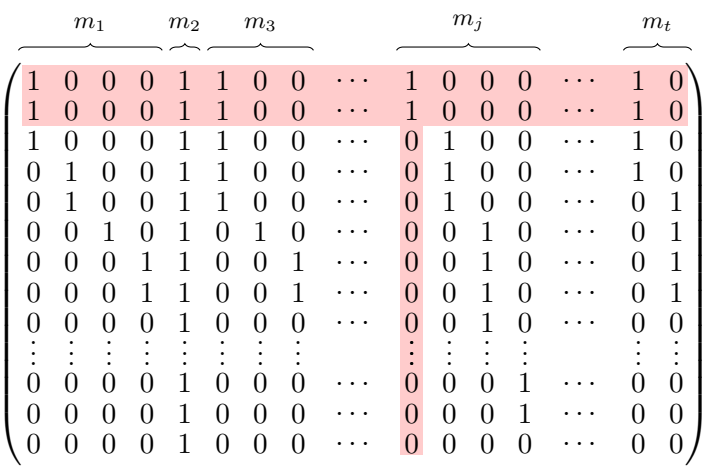

Figure A.4: Example of a matrix $\boldsymbol{\Upsilon}_{\boldsymbol{\alpha}^{(1)},\dots,\boldsymbol{\alpha}^{(t)}} \in \mathbb{F}_q^{n\times m}$. The column and the rows highlighted in red are deleted in order to apply the induction hypothesis.

2. We now assume that $m_i = 1$ for all i such that $\alpha^{(i)}_{m_i} = n$ (*i.e.* that all the columns with a one in the last row consists only of ones, see Figure A.4). Since the matrix $\boldsymbol{\Upsilon}_{\boldsymbol{\alpha}^{(1)},\dots,\boldsymbol{\alpha}^{(t)}} \in \mathbb{F}_q^{n\times m}$ contains some 0, there exists some $j \in \{1,\dots,t\}$ such that $m_j > 1$ and we consider such a $j \in \{1,\dots,t\}$ for which $\alpha^{(j)}_1$ is minimal (see Figure A.4).

   We consider the submatrix of $\boldsymbol{\Upsilon}_{\boldsymbol{\alpha}^{(1)},\dots,\boldsymbol{\alpha}^{(t)}} \in \mathbb{F}_q^{n\times m}$ obtained by deleting the column of index $m_1+\dots+m_{j-1}+1$ and the rows of indices in the set $\{1,\dots,\alpha^{(j)}_1-1\}$. It is easy to see that this submatrix is a block unitary progressive patterned matrix with $n' \leq n-1$ rows and $m-1$ columns (see Figure A.4). We can thus apply the induction hypothesis to the submatrix $\mathbf{S}'$ of $\mathbf{S}$ obtained by deleting the same column and the same rows.

   We know that with probability at least $1-(m-1)q^{-1}$, there exists a linear subspace $V' \subseteq \mathbb{F}_q^{m-1}$ of dimension $m-1-n'$ that contains the set $\{\boldsymbol{v} \in \mathbb{F}_q^{m-1}$ s.t. $hw(\boldsymbol{v}) = m-1$ and $\mathbf{S}'\boldsymbol{v} = 0\}$.

   If V' is of dimension 0, then $\{\boldsymbol{v} \in \mathbb{F}_q^{m-1}$ s.t. $hw(\boldsymbol{v}) = m-1$ and $\mathbf{S}'\boldsymbol{v} = 0\} \subseteq \{0\}$ and this set is thus the empty set. We then have $\{\boldsymbol{v} \in \mathbb{F}_q^m, hw(\boldsymbol{v}) = m$ and $\mathbf{S}\boldsymbol{v} = 0\} = \emptyset$ with probability at least $1-(m-1)q^{-1} \geq 1-mq^{-1}$, and so there exists a linear subspace V of dimension $m-n$ that contains this set.

   If V' is of dimension $m-1-n' > 0$, we can assume without loss of generality that the column of $\mathbf{S}$ deleted to obtain $\mathbf{S}'$ was the last one (by permuting the blocks of the matrix). We have the following block-decomposition of $\mathbf{S}$

$$\mathbf{S} = \begin{pmatrix} \mathbf{S}'' & \mathbf{u} \\ \mathbf{S}' & \mathbf{0}_{\mathbf{n}'\times\mathbf{1}} \end{pmatrix}$$

   where $\mathbf{S}''$ is a $(n-n')\times(m-1)$ matrix and $\mathbf{u}$ a column vector of dimension $(n-n')$. Note that $\boldsymbol{u}$ is a random vector in $\mathbb{F}_q^{n-n'}$ independent from $\mathbf{S}'$ and $\mathbf{S}''$. Let $\boldsymbol{v} \in \mathbb{F}_q^m$ such that $hw(\boldsymbol{v}) = m$ and $\mathbf{S}\boldsymbol{v} = 0$.

   We write $\boldsymbol{v} = \begin{pmatrix} \tau \\ \boldsymbol{w} \end{pmatrix}$ where $\boldsymbol{w} \in \mathbb{F}_q^{m-1}$ and $\tau \in \mathbb{F}_q$ is a scalar. We have $hw(\boldsymbol{w}) = m-1$ and $\mathbf{S}'\boldsymbol{w} = 0$, and therefore $\boldsymbol{w} \in V'$. Since $\tau \neq 0$ by assumption, the vector $\boldsymbol{u}$ thus belongs to the image W of V' by $\mathbf{S}''$ (with probability at least $1-(m-1)q^{-1}$). Moreover, W has dimension at most $\max(m-1-n', n-n')$.

- If W is of dimension at most $n-n'-1$, since $\boldsymbol{u}$ is independent of $\mathbf{S}'$ and $\mathbf{S}''$ (and thus of W), $\boldsymbol{u}$ belongs to W with probability at most q^{-1}. Therefore, with probability at least $(1-q^{-1})\cdot(1-(m-1)q^{-1}) \geq 1-mq^{-1}$, $\{\boldsymbol{v} \in \mathbb{F}_q^m \text{ s.t. } hw(\boldsymbol{v}) = m \text{ and } \mathbf{S}\boldsymbol{v} = 0\} = \emptyset$.
- If W is of dimension $n-n'$ then $\mathbf{S}''$ is invertible. With probability $1-q^{-(n-n')} \geq 1-q^{-1}$, we have $\boldsymbol{u} \neq \mathbf{0}_{(\boldsymbol{n-n'})\times\mathbf{1}}$ and we can construct a basis $\boldsymbol{u_1} = \boldsymbol{u}, \ldots, \boldsymbol{u_{n-n'}}$ of W.
  All subspaces $V' \cap \mathbf{S}''^{-1}(\langle \boldsymbol{u_i} \rangle)$ are of dimension at least one and we have
  $$V' = \bigoplus_{i=1}^{n-n'} V' \cap \mathbf{S}''^{-1}(\langle \boldsymbol{u_i} \rangle).$$
  Therefore the linear subspace V defined as $V = V' \cap \mathbf{S}''^{-1}(\langle \boldsymbol{u_1} \rangle)$ satisfies
  $$\begin{aligned}\dim(V) &= \dim(V') - \sum_{i=2}^{n-n'} \dim\left(V' \cap \mathbf{S}''^{-1}(\langle \boldsymbol{u_i} \rangle)\right) \\ &\leq m - 1 - n' - (n - n' - 1) \\ &= m - n.\end{aligned}$$
  Moreover, we have $\{\boldsymbol{v} \in \mathbb{F}_q^m \text{ s.t. } hw(\boldsymbol{v}) = m \text{ and } \mathbf{S}\boldsymbol{v} = 0\} \subseteq V$ and since this occurs with probability at least $(1-q^{-1})(1-(m-1)q^{-1}) \geq 1-mq^{-1}$, the result follows.

This concludes the proof of Lemma 31. □

Recall that we want to lower-bound the probability over the $\gamma \in \mathbb{F}_q^{n\times n}$, that for a given set $I \subseteq \{(n+4)\cdot n+1, \ldots, z\}$ of cardinal n, if $hw(\mathbf{L}_I \cdot \boldsymbol{v}) \geq n-K$ then $\mathbf{M}_I \cdot \boldsymbol{v} \neq \mathbf{0}_n$ for any vector $\boldsymbol{v} \in \mathbb{F}_q^n$. where K denotes the number of coordinates $i_1, \ldots, i_K \in [z]$ such that $v_{i_1} \neq 0, \ldots, v_{i_K} \neq 0$ and the corresponding columns $i_1, \ldots, i_K$ in $\mathbf{L}$ and in $\mathbf{M}$ have no zero coefficient.

Remark that the non-zero coefficients in the lower block of $\mathbf{L}_I$ and in $\mathbf{M}_I$ are at the same positions. If $K = 0$, then the matrices $\mathbf{M}_I$ and $\mathbf{L}_I$ have a null row. In this case, we have readily $hw(\mathbf{L}_I \cdot \boldsymbol{v}) \leq n-1 = n-K-1 < n-K$).

If $K \geq 1$, then the matrices $\mathbf{M}_I$ and $\mathbf{L}_I$ does not have a null row. The matrix $\mathbf{M}_I$ (up to some permutation of its columns) can be written as a block matrix where each block is of the form described in Figure A.5 (on the left).

From this matrix, one can construct another matrix $\tilde{\mathbf{M}}_I$ such that in each block, one substract each column to the following columns (i.e., one substract iteratively the i-th column to the columns of index in $\{i+1, \ldots, m\}$ for $i \in \{1, \ldots, m\}$). The blocks appearing in the matrix $\tilde{\mathbf{M}}_I$ are given in Figure A.5 (on the right). Since we apply only elementary operations on the columns, if there exists a vector $\boldsymbol{v} \in \mathbb{F}_q^n$ such that $\mathbf{M}_I\boldsymbol{v} = 0$ then, there exists a vector $\boldsymbol{v'} \in \mathbb{F}_q^n$ such that $\tilde{\mathbf{M}}_I\boldsymbol{v'} = 0$.

Since $\mathbf{M}_I$ has no null row, we have $\alpha_m = n$ in one of this block (with the notation from Figure A.5) and the matrix $\tilde{\mathbf{M}}_I$ is thus a block random progressive patterned matrix as defined in Definition 30. By Lemma 31, for each non-empty subset J of the n columns of $\tilde{\mathbf{M}}_I$, the probability over γ that there exists a vector $\boldsymbol{v'} \in \mathbb{F}_q^n$ with support J (i.e., set of non-zero coordinates) such that $\tilde{\mathbf{M}}_I\boldsymbol{v'} = 0$ is upper bounded by $n \cdot q^{-1}$. By the union bound over all supports, the probability over γ that there exists a vector $\boldsymbol{v'} \in \mathbb{F}_q^n$ such that $\tilde{\mathbf{M}}_I\boldsymbol{v'} = 0$ is thus upper-bounded by $2^n \cdot n \cdot q^{-1}$.

For the sets $I \subseteq \{(n+4)\cdot n+1, \ldots, z\}$ of cardinal n, we have proved that with probability at least $1-2^n\cdot n\cdot q^{-1}$ (over the choice of $\gamma \in \mathbb{F}_q^{n\times n}$), we have $hw(\mathbf{L}_I\cdot\boldsymbol{v}) < n-K$ or $\mathbf{M}_I\cdot\boldsymbol{v} \neq \mathbf{0}_n$ for any vector $\boldsymbol{v} \in \mathbb{F}_q^n$.

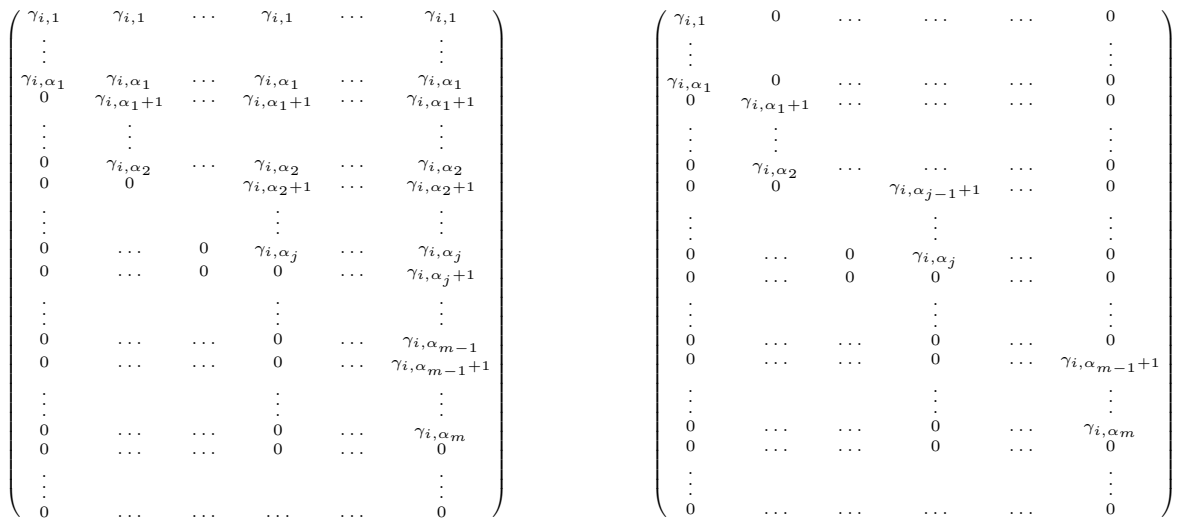

Figure A.5: Blocks appearing in matrices $\mathbf{M}_I$ and $\tilde{\mathbf{M}}_I$

Case 2.

We now consider matrices $\mathbf{M}_I$ were all columns are taken from the matrix $\mathbf{T}_n$ or the matrices $\mathbf{T}_{\boldsymbol{\gamma},i}$ for $i \in \{1,\dots,n\}$ (i.e., $I \subseteq \{(n+3)\cdot n+1,\dots,z\}$). With the notation from Definition 30, we consider the modified distribution $\tilde{\mathcal{D}}_{\boldsymbol{\alpha}^{(1)},\dots,\boldsymbol{\alpha}^{(t)}}$ defined as the following probability distribution in $\mathbb{F}_q^{n\times m}$:

$$\tilde{\mathcal{D}}_{\boldsymbol{\alpha}^{(1)},\dots,\boldsymbol{\alpha}^{(t)}} = (\boldsymbol{\Upsilon}_{\boldsymbol{\alpha}^{(1)}}|\mathcal{D}_{\boldsymbol{\alpha}^{(2)},\dots,\boldsymbol{\alpha}^{(t)}}) = (\boldsymbol{\Upsilon}_{\boldsymbol{\alpha}^{(1)}}|\mathcal{D}_{\boldsymbol{\alpha}^{(2)}}|\dots|\mathcal{D}_{\boldsymbol{\alpha}^{(t)}})$$

(i.e., in which the first block is a fixed unitary progressive patterned matrix instead of being a random progressive patterned matrix). We can easily extend Lemma 31 to this distribution:

Lemma 32. *Let n,m,t be three positive integers with $m \geq n$ and let $\boldsymbol{\alpha}^{(i)}$ for $i \in \{1,\dots,t\}$ be patterns for block progressive patterned matrix as in Definition 30. For a block random progressive patterned matrix matrix $\mathbf{S}$ drawn following the distribution $\tilde{\mathcal{D}}_{\boldsymbol{\alpha}^{(1)},\dots,\boldsymbol{\alpha}^{(t)}}$, there exists a linear subspace of $\mathbb{F}_q^m$ of dimension $m-n$ that contains $\{\boldsymbol{v} \in \mathbb{F}_q^m$ s.t. $hw(\boldsymbol{v}) = m$ and $\mathbf{S}\boldsymbol{v} = 0\}$, with probability at least $1 - mq^{-1}$.*

Lemma 32. We will prove Lemma 32 by induction on m.

For $m = 1$, since $m \geq n \geq 1$, Definition 30 implies that the the matrix $\mathbf{S}$ either (1) consists simply in a single entry $s_{1,1}$ which is picked uniformly at random in $\mathbb{F}_q$ or (2) a constant non-null vector. In the first case, this vector is null with probability q^{-1} and in all cases the set $\{\boldsymbol{v} \in \mathbb{F}_q$ s.t. $hw(\boldsymbol{v}) = 1$ and $\mathbf{S}\boldsymbol{v} = 0\}$ is therefore the empty set with probability at least $1 - q^{-1}$. It is thus included in the subspace of dimension 0 with probability at least $1 - q^{-1}$.

We now consider $m \geq 2$ and we assume Lemma 32 proven for all block random progressive patterned matrix matrix drawn from a distribution $\tilde{\mathcal{D}}_{\boldsymbol{\alpha}^{(1)},\dots,\boldsymbol{\alpha}^{(t)}}$ with strictly less than m columns.

We first assume that the matrix $\boldsymbol{\Upsilon}_{\boldsymbol{\alpha}^{(1)},\dots,\boldsymbol{\alpha}^{(t)}} \in \mathbb{F}_q^{n\times m}$ is the unitary matrix $\mathbf{U}_{n\times m}$ (i.e., does not contain any zero). Then, by assumption, we have $m_i = 1$ and $\alpha^{(i)} = n$ for $i \in \{1,\dots,t\}$. The matrix $\mathbf{S}$ is thus the concatenation of the vector $\mathbf{1}_{n\times 1}$ and a matrix picked from $\mathbb{F}_q^{n\times m-1}$ with the uniform distribution. Using elementary operations on the columns of $\mathbf{S}$, one can obtain a matrix of the form

$$\begin{pmatrix} 1 & \mathbf{0_{1\times m-1}} \\ \boldsymbol{u_{n-1}} & \mathbf{S'} \end{pmatrix}$$

where $\boldsymbol{u_{n-1}} \in \mathbb{F}_q^{n-1}$ is the all-one vector and $\mathbf{S}'$ is a matrix drawn from $\mathbb{F}_q^{n-1\times m-1}$ with the uniform distribution. As in the proof of Lemma 31, the matrix $\mathbf{S}'$ is of full rank $n-1$ with probability at least $1-2q^{n-m-2}$. The matrix $\mathbf{S}$ is thus of full rank n with probability at least $1-2q^{n-m-2}$ and thus with probability at least $1-mq^{-1}$.

We now assume that the matrix $\boldsymbol{\Upsilon}_{\boldsymbol{\alpha}^{(1)},\ldots,\boldsymbol{\alpha}^{(t)}} \in \mathbb{F}_q^{n\times m}$ contains some 0. By assumption, there exists $j \in \{1,\ldots,t\}$ such that $\alpha_{m_j}^{(j)} = n$ and in the following, it there exist two indices $j \in \{1,\ldots,t\}$ such that $\alpha_{m_j}^{(j)} = n$, we select one such index different from 1.

If $j = 1$, by assumption we have $\alpha_{m_i}^{(i)} < n$ for all $i \in \{2,\ldots,t\}$ and the last row of the matrix $\mathbf{S}$ has one coordinate equal to 1 and all other coordinates equal to 0. If $\boldsymbol{v} \in \mathbb{F}_q$ is of full Hamming weight $hw(\boldsymbol{v}) = m$, the last coordinate of the vector $\mathbf{S}\boldsymbol{v}$ is always non-null and the set $\{\boldsymbol{v} \in \mathbb{F}_q \text{ s.t. } hw(\boldsymbol{v}) = m \text{ and } \mathbf{S}\boldsymbol{v} = 0\}$ is therefore the empty set. It is thus included in the subspace of dimension 0 with probability at least $1 \geq 1 - mq^{-1}$. We therefore now assume that $j > 1$.

1. We first assume that $m_j > 1$ (*i.e.* that the column of index $m_1 + \cdots + m_j$ consists in $\alpha_{m_j-1}^{(j)} \geq 1$ zeroes followed by $\alpha_{m_j}^{(j)} - \alpha_{m_j-1}^{(j)} = n - \alpha_{m_j-1}^{(j)} \geq 1$ ones).

   We consider the submatrix of $\boldsymbol{\Upsilon}_{\boldsymbol{\alpha}^{(1)},\ldots,\boldsymbol{\alpha}^{(t)}} \in \mathbb{F}_q^{n\times m}$ obtained by deleting the column of index $m_1 + \cdots + m_j$ and the rows of indices i in $\{\alpha_{m_{j-1}+1}^{(j)},\ldots,\alpha_{m_j}^{(j)}\}$. This submatrix is a block unitary progressive patterned matrix with $n' \leq n$ rows and $m-1$ columns. We can thus apply the induction hypothesis to the submatrix $\mathbf{S}'$ of $\mathbf{S}$ obtained by deleting the same column and the same rows. We know that with probability $1-(m-1)q^{-1}$, there exist a linear subspace V' of dimension $m-1-n'$ that contains the set $\{\boldsymbol{v} \in \mathbb{F}_q^{m-1} \text{ s.t. } hw(\boldsymbol{v}) = m-1 \text{ and } \mathbf{S}'\boldsymbol{v} = 0\}$.

   If V' is of dimension 0, then $\{\boldsymbol{v} \in \mathbb{F}_q^{m-1} \text{ s.t. } hw(\boldsymbol{v}) = m-1 \text{ and } \mathbf{S}'\boldsymbol{v} = 0\} \subseteq \{0\}$ and the set is the empty set. We thus have $\{\boldsymbol{v} \in \mathbb{F}_q^m, hw(\boldsymbol{v}) = m \text{ and } \mathbf{S}\boldsymbol{v} = 0\} = \emptyset$ and with probability $1-(m-1)q^{-1} \geq 1-mq^{-1}$, there exist a linear subspace V of dimension $m-n$ that contains this set.

   If V' is of dimension $m-1-n' > 0$, we can assume without loss of generality that the deleted column of $\mathbf{S}$ to obtain $\mathbf{S}'$ was the last one in the last block (i.e., in a block where $\mathbf{S}$ is a random progressive patterned matrix since $j > 1$).

   By permuting some rows and columns, we can write

$$\mathbf{S} = \begin{pmatrix} \mathbf{S}' & \mathbf{0_{n'\times 1}} \\ \mathbf{S}'' & \mathbf{u} \end{pmatrix}$$

   where $\mathbf{S}'$ is a $(n-n')\times m-1$ matrix on which we can apply the induction hypothesis (since $m_j > 1$). Let $\boldsymbol{v} \in \mathbb{F}_q^m$ such that $hw(\boldsymbol{v}) = m$ and $\mathbf{S}\boldsymbol{v} = 0$.

   We write $\boldsymbol{v} = \begin{pmatrix} \boldsymbol{w} \\ \tau \end{pmatrix}$ where $\boldsymbol{w} \in \mathbb{F}_q^{m-1}$ and $\tau \in \mathbb{F}_q$ is a scalar. We have $hw(\boldsymbol{w}) = m-1$ and $\mathbf{S}'\boldsymbol{w} = 0$, and therefore $\boldsymbol{w} \in V'$. Since $\tau \neq 0$ by assumption, the vector $\boldsymbol{u}$ thus belongs to the image W of V' by $\mathbf{S}''$ (with probability at least $1-(m-1)q^{-1}$). Since $j > 1$, note that $\boldsymbol{u}$ is a random vector in $\mathbb{F}_q^{n-n'}$ independent from $\mathbf{S}'$. We can then conclude as in the proof of Lemma 31.

2. We now assume that $m_i = 1$ for all i such that $\alpha_{m_i}^{(i)} = n$ for $i \in \{1,\ldots,t\}$ (*i.e.* that all the columns with a one in the last row consists only of ones).

   Since the matrix $\boldsymbol{\Upsilon}_{\boldsymbol{\alpha}^{(1)},\ldots,\boldsymbol{\alpha}^{(t)}} \in \mathbb{F}_q^{n\times m}$ contains some 0, there exists some $j \in \{2,\ldots,t\}$ such that $m_j > 1$ and we consider such a $j \in \{2,\ldots,t\}$ for which $\alpha_1^{(j)}$ is minimal.

We consider the submatrix of $\boldsymbol{\Upsilon}_{\boldsymbol{\alpha}^{(1)},\ldots,\boldsymbol{\alpha}^{(t)}} \in \mathbb{F}_q^{n\times m}$ obtained by deleting the column of index $m_1+\cdots+m_{j-1}+1$ and the rows of indices in the set $\{1,\ldots,\alpha_1^{(j)}-1\}$. It is easy to see that this submatrix is a block unitary progressive patterned matrix with $n' \leq n-1$ rows and $m-1$ columns. We can thus apply the induction hypothesis to the submatrix $\mathbf{S}'$ of $\mathbf{S}$ obtained by deleting the same column and the same rows.

We write $\boldsymbol{v} = \begin{pmatrix} \boldsymbol{w} \\ \tau \end{pmatrix}$ where $\boldsymbol{w} \in \mathbb{F}_q^{m-1}$ and $\tau \in \mathbb{F}_q$ is a scalar. We have $hw(\boldsymbol{w}) = m-1$ and $\mathbf{S}'\boldsymbol{w} = 0$, and therefore $\boldsymbol{w} \in V'$. Since $\tau \neq 0$ by assumption, the vector $\boldsymbol{u}$ thus belongs to the image W of V' by $\mathbf{S}''$ (with probability at least $1-(m-1)q^{-1}$). Since $j > 1$, note that $\boldsymbol{u}$ is a random vector in $\mathbb{F}_q^{n-n'}$ independent from $\mathbf{S}'$. We can then conclude as in the proof of Lemma 31.

We know that with probability at least $1-(m-1)q^{-1}$, there exists a linear subspace $V' \subseteq \mathbb{F}_q^{m-1}$ of dimension $m-1-n'$ that contains the set $\{\boldsymbol{v} \in \mathbb{F}_q^{m-1}$ s.t. $hw(\boldsymbol{v}) = m-1$ and $\mathbf{S}'\boldsymbol{v} = 0\}$.

If V' is of dimension 0, then $\{\boldsymbol{v} \in \mathbb{F}_q^{m-1}$ s.t. $hw(\boldsymbol{v}) = m-1$ and $\mathbf{S}'\boldsymbol{v} = 0\} \subseteq \{0\}$ and this set is thus the empty set. We then have $\{\boldsymbol{v} \in \mathbb{F}_q^m, hw(\boldsymbol{v}) = m$ and $\mathbf{S}\boldsymbol{v} = 0\} = \emptyset$ with probability at least $1-(m-1)q^{-1} \geq 1-mq^{-1}$, and so there exists a linear subspace V of dimension $m-n$ that contains this set.

If V' is of dimension $m-1-n' > 0$, we can assume without loss of generality that the column of $\mathbf{S}$ deleted to obtain $\mathbf{S}'$ was the last one (by permuting the blocks of the matrix). We have the following block-decomposition of $\mathbf{S}$

$$\mathbf{S} = \begin{pmatrix} \mathbf{S}'' & \mathbf{u} \\ \mathbf{S}' & \mathbf{0}_{\mathbf{n'\times 1}} \end{pmatrix}$$

where $\mathbf{S}''$ is a $(n-n')\times(m-1)$ matrix and $\mathbf{u}$ a column vector of dimension $(n-n')$. Note that $\boldsymbol{u}$ is a random vector in $\mathbb{F}_q^{n-n'}$ independent from $\mathbf{S}'$ and $\mathbf{S}''$. Let $\boldsymbol{v} \in \mathbb{F}_q^m$ such that $hw(\boldsymbol{v}) = m$ and $\mathbf{S}\boldsymbol{v} = 0$. Since $j > 1$, note that $\boldsymbol{u}$ is a random vector in $\mathbb{F}_q^{n-n'}$ independent from $\mathbf{S}'$. We can then conclude as in the proof of Lemma 31.

This concludes the proof of the lemma. □ □

Using the same arguments as above for Case 1 (but replacing Lemma 31 by Lemma 32), we obtain that for any set $I \subseteq \{1,\ldots,z\}$ of cardinal n such that $\mathbf{M}_I$ has no identically zero column vectors, with probability at least $1-2^n\cdot n\cdot q^{-1}$ over the choice of γ, we have $hw(\mathbf{L}_I\cdot\boldsymbol{v}) < n-K$ or $\mathbf{M}_I\cdot\boldsymbol{v} \neq \mathbf{0}_n$ for any vector $\boldsymbol{v} \in \mathbb{F}_q^n$ (where K denotes the number of coordinates $i_1,\ldots,i_K \in [z]$ such that $v_{i_1} \neq 0,\ldots,v_{i_K} \neq 0$ and the corresponding columns $i_1,\ldots,i_K$ in $\mathbf{L}$ and in $\mathbf{M}$ have no zero coefficient). .

Case 3.

We now consider the sets $I \subseteq \{1,\ldots,n\}\cup\{(n+3)\cdot n+1,\ldots,z\}$ of cardinal n for which $\mathbf{M}_I$ has some identically zero column vectors (i.e., $I\cap\{1,\ldots,n\} \neq \emptyset$). For each $i \in I\cap\{1,\ldots,n\} \neq \emptyset$, the i-th column in $\mathbf{L}$ is the i-th vector in the canonical basis of $\mathbb{F}_q^n$ (i.e., it corresponds to a probe of a value a_i). We can consider the submatrix of $\mathbf{M}_I$ and $\mathbf{L}_I$ in which we delete for each $i \in I\cap\{1,\ldots,n\} \neq \emptyset$, the i-th column and the i-th row. We denote $\rho = \#I\cap\{1,\ldots,n\} \neq \emptyset$.

Let us denote $\mathbf{M}'_I$ and $\mathbf{L}'_I$ the corresponding matrices (with $m' = m-\rho$ columns). These matrices are of the form handled in the previous *Case 2* (with $m' < m$). The previous argument shows therefore that with probability at least $1-2^n\cdot n\cdot q^{-1}$, we have $hw(\mathbf{L}'_I\cdot\boldsymbol{v}) < n-\rho-K$ or $\mathbf{M}'_I\cdot\boldsymbol{v} \neq \mathbf{0}_n$ for any vector $\boldsymbol{v} \in \mathbb{F}_q^{m'}$ (where K denotes the number of coordinates $i_1,\ldots,i_K \in [z]$

such that $v_{i_1} \neq 0, \ldots, v_{i_K} \neq 0$ and the corresponding columns $i_1, \ldots, i_K$ in $\mathbf{L}$ and in $\mathbf{M}$ have no zero coefficient).

Going back to the original matrices $\mathbf{L}_I$ and $\mathbf{M}_I$ we have shown for any set $I \subseteq \{1, \ldots, n\} \cup \{(n+3) \cdot n + 1, \ldots, z\}$ of cardinal n, with probability at least $1 - 2^n \cdot n \cdot q^{-1}$ over the choice of γ, we have $hw(\mathbf{L}_I \cdot \boldsymbol{v}) < n - K$ or $\mathbf{M}_I \cdot \boldsymbol{v} \neq \mathbf{0}_n$ for any vector $\boldsymbol{v} \in \mathbb{F}_q^n$ (indeed a vector $\boldsymbol{v}$ satisfies $\mathbf{M}_I \cdot \boldsymbol{v} = \mathbf{0}_n$ if an only if $\mathbf{M}'_I \cdot \boldsymbol{v'} = \mathbf{0}_n$ where $\boldsymbol{v'}$ denotes the restriction of $\boldsymbol{v}$ to the support $I \cap \{1, \ldots, n\}$ and the Hamming weight of $hw(\mathbf{L}_I \cdot \boldsymbol{v})$ is at smaller than $hw(\mathbf{L}_I \cdot \boldsymbol{v'}) + \rho$ since at most ρ positions can be set arbitrarily..

Case 4.

We now consider all sets $I \subseteq \{1, \ldots, z\}$ (with no restrictions). Without loss of generality, we can assume that all not identically zero column vectors in $\mathbf{M}_I$ are pairwise distinct. Indeed, if two columns are equal, they come either from the two submatrices I_n of $\mathbf{M}$, or from the first column vectors of a submatrix I_n and the submatrix $\mathbf{T}_n$, or from the first column vectors of a submatrix $\mathbf{D}_{\boldsymbol{\gamma},i}$ for some $i \in \{1, \ldots, n\}$ and the corresponding submatrix $\mathbf{T}_{\boldsymbol{\gamma},i}$. In all these cases, one can replace the index of the second vector in I by an index in $\{1, \ldots, n-1\}$ (and modify the vector accordingly) in such a way that $\mathbf{M}_{I'}$ for the new set I' has a new null column vector for each duplicate in the original matrix $\mathbf{M}_I$.

We can now delete the columns corresponding to the null vectors as in Case 3 (i.e., for each $i \in I \cap \{1, \ldots, n+1\} \neq \emptyset$, the i-th column and the i-th row in $\mathbf{M}_I$ and $\mathbf{L}_I$). The only difference occurs if a column in $\mathbf{M}_I$ is equal to the i-th vector in the canonical basis (for $i \geq 2$) or to the scalar multiplication of this vector by some element of the matrix $\gamma \in \mathbb{F}_q$ (corresponding to the cases $I \cap \{n+1, \ldots, 2n\} \neq \emptyset$ and $I \cap \{2n+1, \ldots, (n+3) \cdot n + 1\} \neq \emptyset$ respectively). As in Case 3, we can delete the corresponding column and row in $\mathbf{M}_I$ and $\mathbf{L}_I$ (i.e., it corresponds to a probe of a value r_i, a value $a_i + r_i$ or a value $a_i + \gamma_{j,i} r_i$).

As above, if we denote $\mathbf{M}'_I$ and $\mathbf{L}'_I$ the corresponding matrices (with m' columns and $n' < n$ and $n' + 1$ rows, respectively), the previous argument shows that with probability at least $1 - 2^n \cdot n \cdot q^{-1}$, we have $hw(\mathbf{L}'_I \cdot \boldsymbol{v}) < n' - K$ or $\mathbf{M}'_I \cdot \boldsymbol{v} \neq \mathbf{0}_n$ for any vector $\boldsymbol{v} \in \mathbb{F}_q^{m'}$ (where K denotes the number of coordinates $i_1, \ldots, i_K \in [z]$ such that $v_{i_1} \neq 0, \ldots, v_{i_K} \neq 0$ and the corresponding columns $i_1, \ldots, i_K$ in $\mathbf{L}$ and in $\mathbf{M}$ have no zero coefficient).

Going back to the original matrices $\mathbf{L}_I$ and $\mathbf{M}_I$ we have shown for any set $I \subseteq \{1, \ldots, z\}$ of cardinal n, with probability at least $1 - 2^n \cdot n \cdot q^{-1}$ over the choice of γ, we have $hw(\mathbf{L}_I \cdot \boldsymbol{v}) < n - K$ or $\mathbf{M}_I \cdot \boldsymbol{v} \neq \mathbf{0}_n$ for any vector $\boldsymbol{v} \in \mathbb{F}_q^n$

A.3.1 Conclusion

. By the union on all such sets, we obtain that the probability that, for $\boldsymbol{\gamma}$ picked uniformly at random in $\mathbb{F}_q^{n \times n}$, the matrix $\mathbf{M}$ satisfies Condition 3, i.e., for any vector $\boldsymbol{v} \in \mathbb{F}_q^z$ of Hamming weight $hw(\boldsymbol{v}) \leq n$ we have $hw(\mathbf{L} \cdot \boldsymbol{v}) < n - K$ or $\mathbf{M} \cdot \boldsymbol{v} \neq \mathbf{0}_n$ is at least

$$1 - \binom{z}{n} 2^n \cdot n \cdot q^{-1} = 1 - \binom{(2n+4) \cdot n + 1}{n} 2^n \cdot n \cdot q^{-1}.$$

The binomial coefficient in this lower-bound is always less than $(6n)^n$ (this can be checked by hand for small values of n and it follows for large values using the classical upper-bound $\binom{r}{s} \leq ((r \cdot \exp(1))/s)^s$). We thus obtain the claimed bounds and this concludes the proof.

□