SAPIENZA
UNIVERSITÀ DI ROMA

Non-malleable secret sharing against joint tampering attacks

Dipartimento di Informatica
Dottorato in INFORMATICA (XXXV cycle)

Gianluca Brian
ID number 1615294

Advisor
Prof. Daniele Venturi

Academic Year 2022/2023

Non-malleable secret sharing against joint tampering attacks
PhD thesis. Sapienza University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: brian@di.uniroma1.it

Abstract

Since thousands of years ago, the goal of cryptography has been to hide messages from prying eyes. In recent times, cryptography two important changes: first, cryptography itself evolved from just being about encryption to a broader class of situations coming from the digital era; second, the way of studying cryptography evolved from creating “seemingly hard” cryptographic schemes to constructing schemes which are *provably* secure.

However, once the mathematical abstraction of cryptographic primitives started to be too hard to break, attackers found another way to defeat security. Side channel attacks have been proved to be very effective in this task, breaking the security of otherwise provably secure schemes. Because of this, recent trends in cryptography aim to capture this situation and construct schemes that are secure even against such powerful attacks.

In this setting, this thesis specializes in the study of *secret sharing*, an important cryptographic primitive that allows to balance privacy and integrity of data and also has applications to multi-party protocols. Namely, continuing the trend which aims to protect against side channel attacks, this thesis brings some contributions to the state of the art of the so-called *leakage-resilient* and *non-malleable* secret sharing schemes, which have stronger guarantees against attackers that are able to learn information from possibly all the shares and even tamper with the shares and see the effects of the tampering.

The main contributions of this thesis are twofold. First, we construct secret sharing schemes that are secure against a very powerful class of attacks which, informally, allows the attacker to *jointly* leak some information and tamper with the shares in a *continuous* fashion. Second, we study the *capacity* of continuously non-malleable secret sharing schemes, that is, the maximum achievable information rate. Roughly speaking, we find some lower bounds to the size that the shares must have in order to achieve some forms of non-malleability.

Acknowledgments

First and foremost, I would like to thank my advisor Daniele Venturi, who inspired me with his engaging enthusiasm since the class on Cryptography. You showed me constant support through all the years of this PhD, encouraging me whenever necessary and allowing me to grow professionally and personally.

A big dziękuję *goes also to Stefan Dziembowski, for the wonderful year I spent in Warsaw. Thank you for being always so nice and helpful, I am very grateful for the opportunities you gave me.*

Speaking of Warsaw, I would also like to thank the people I met in UW. Thanks in particular to Mikołaj Leonarski, Chan Nam Ngo, Tomasz Lizurej, Małgorzata Galązka, Paweł Kędzior, Tomasz Michalak, Grzegorz Fabiański, and to the administration ladies, especially Anna Modzelewska and Ewa Puchalska-Farah.

During these three years I had the opportunity to meet many interesting people. A special thank goes to Antonio Faonio, who has been almost like a second advisor to me, and to João Lourenço Ribeiro, Sebastian Faust, Daniele Friolo and Elena Micheli, who have always been supportive to me. I would also like to thank other people I had occasion to work or discuss with, in particular Nico Döttling, Jesko Dujmović, Maciej Obremski, Maciej Skorski, Mark Simkin, Yiannis Tselekounis. I should also mention "La Sapienza" university of Rome, thank you for enabling my studies and making all of this possible.

Last, but not least, I would also like to thank my family, for their patience and unconditional support during these years, my friends and Silvia.

Contents

Chapter 1

Introduction

Since thousands of years ago, a fight goes on between who finds ever more sophisticated ways to transmit secret messages and who, instead, does whatever is possible to violate such secrecy. Cryptography, originally meaning “hidden writings”, is the art of constructing secure communication in the presence of adversarial behavior. Before the modern era, “cryptography” was synonymous with encryption, the act of transforming some readable information into some seemingly random text. This system has been used widely in the past by many civilizations, including the Roman Empire, the Ancient Greece and the Ancient Egypt.

Between the 9th and the 14th century, the simple ciphers developed so far started to fall apart. For instance, by using frequency analysis of the letters[1] it is possible to crack a sufficiently long text encrypted with the Caesar’s Cipher. The ciphers started to be more and more sophisticated: the Vigenère Cipher added a bit more complexity by using multiple alphabetic substitutions instead of just one. Unfortunately, this was not enough. During the Second World War, the German Army used the way more complicated Enigma cipher to communicate confidential information, involving spinning rotors, permutation of letters and an internal state which was constantly changing, thus strongly mitigating the frequency analysis attacks. However, the cipher was complex enough to need a machine, with moving parts and some simple electronics, in order to compute a ciphertext. After a while, the cryptanalysis techniques became sophisticated as well. The polish *bomba kryptologiczna* was a special machine designed with the purpose of breaking Enigma, and so it did until the Enigma code became stronger (more spinning rotors, more permutations, etc.). And then, again, Alan Turing came up with the british *bombe*, a net improvement over the *bomba kryptologiczna*, which helped the british to discover the daily configuration of Enigma.

With the beginning of the digital revolution, thanks to the semiconductors, computation became even faster and more automated. New challenges arose and the old cryptosystems were not suitable anymore. In fact, a personal computer could crack Enigma in just a few hours [SW05]. Cryptography was about to drastically change as well, and this change was twofold.

[1]Frequency analysis is the technique that compares the number of occurrences of the letters in an encrypted text with the frequency of such letters in the possible plaintexts. In english, the most frequent letter is `E`; this implies that, in a substitution cipher encrypting a text which is originally in english language, the most common letter is probably encrypting a letter `E`.

First, with the development of computers and Internet, communication needed protection against a broader class of attacks. Hiding secrets was not anymore the only necessity, and similar techniques could also be applied to achieve authentication, key exchange, data integrity, secure multi-party protocols and much more. This thesis focuses on *secret sharing*, an important primitive used both alone, to protect data secrecy and integrity at the same time, and inside other protocols, to make secure computation involving multiple parties possible. A more detailed introduction to secret sharing can be found in Section 1.3.

Second, a breakthrough change in how to study cryptography was proposed by Goldwasser and Micali [GM82], in what is called "provable security". This approach uses the formalism of mathematics to show the security of a cryptographic primitive: indeed, the security requirement is stated as a *formal definition*, and then a *mathematical proof* is shown for the cryptographic primitive to satisfy such definition. This makes the approach of provable security very strong, and allows to formalize the specifications which the cryptosystems need to satisfy; however, the formal definitions should be meaningful and should capture the real setting as close as possible. We will tell a bit more about this topic in Section 1.1.

Finding a formal definition which is both meaningful and able to capture the real setting can be quite challenging. Even if a primitive is *perfectly secure*[2], computation happens by means of physical phenomena – usually electrical signals – which are measurable in several ways (e.g. currents, time of execution, etc.). Because of this, it is usually more convenient to attack the actual implementation of a cryptographic primitive rather than its mathematical abstraction. This leads to the so called *side-channel attacks*, which are able to completely compromise the security of otherwise provably secure schemes. Some more details about side-channel attacks are in Section 1.2.

A modern research line in cryptography is aiming to fill this gap between the mathematical abstraction and the physical implementation, constructing primitives which are provably secure even against side-channel attacks. This thesis deals with some of these attacks in the context of secret sharing, improving the state of the art on this topic. In Section 1.4, we highlight the contributions of this thesis.

1.1 Provable security

We start with the following question: what does *secure* mean? First of all, it depends on what we are trying to achieve. Intuitively, an encryption scheme is secure when any ciphertext (i.e. the result of the encryption) completely "hides" the plaintext (i.e. the original message before the encryption). Conversely, the goal of a digital signature scheme is not to hide information[3], but to certify that such information comes from the right sender. In both cases, we need some formal specification which such schemes need to satisfy.

[2] *Perfect security* has a precise meaning which will be shown in Section 1.1 and formalized in Chapter 2; for now, it suffices to know that it is a very strong guarantee.

[3] Indeed, sometimes the signed information is even transmitted in clear along with its signature.

Semantic security. A first step towards defining security is stating what the adversary wants to achieve and what are their capabilities. In the case of encryption, one may think that the adversary wants to learn the secret in full. However, more often than not, the adversary just needs to learn *some* information.

> “ Encrypting messages in a way that ensures the secrecy of all partial information is an extremely important goal in Cryptography. The importance of this point of view is particularly apparent if we want to use encryption to play card games over the telephone. If the suit or color of a card could be compromised the whole game could be invalid.
>
> – *Shafi Goldwasser and Silvio Micali, “Probabilistic encryption and how to play mental poker keeping secret all partial information” [GM82]*

For this reason, we would like to hide the message in such a way that not even partial information is leaked.

Claude Shannon [Sha49] formally defined this kind of security in 1949 as *perfect secrecy*. His intuition was the following: a cryptographic scheme achieves *perfect secrecy* when any information on the plaintext obtained after learning the ciphertext can also be obtained without the ciphertext. A bit more formally, let $\boldsymbol{M}$ be the random variable of the plaintext and let $\boldsymbol{C}$ be the random variable of the ciphertext. Shannon's *perfect secrecy* states that, for all messages $\mu \in \mathcal{M}$ and all ciphertexts $\gamma \in \mathcal{C}$,

$$\mathbb{P}\left[\boldsymbol{M} = \mu | \boldsymbol{C} = \gamma\right] = \mathbb{P}\left[\boldsymbol{M} = \mu\right].$$

An equivalent formulation, obtained by applying Bayes' Theorem, is

$$\mathbb{P}\left[\boldsymbol{C} = \gamma | \boldsymbol{M} = \mu\right] = \mathbb{P}\left[\boldsymbol{C} = \gamma\right].$$

From the latter, we get that, given two different messages $\mu_0, \mu_1 \in \mathcal{M}$,

$$\mathbb{P}\left[\boldsymbol{C} = \gamma | \boldsymbol{M} = \mu_0\right] = \mathbb{P}\left[\boldsymbol{C} = \gamma | \boldsymbol{M} = \mu_1\right],$$

or, with a compact notation, $\mathsf{Encrypt}(\mu_0) \triangleq \mathsf{Encrypt}(\mu_1)$, meaning that a ciphertext coming from μ_0 and a ciphertext coming from μ_1 are identically distributed. Unfortunately, this kind of security has two drawbacks.

First, it is impossible to achieve such level of security with small cryptographic keys. More in detail, this result is only possible when the encryption key is as large as the plaintext. A solution is to introduce some error in the probabilities. Namely, instead of requiring that the quantities $\mathbb{P}\left[\boldsymbol{C} = \gamma | \boldsymbol{M} = \mu_0\right]$ and $\mathbb{P}\left[\boldsymbol{C} = \gamma | \boldsymbol{M} = \mu_1\right]$ are equal we could just require the difference between them to be small. In this case, we obtain what is called *statistical security*. However, this just reduces the length of the key by a small factor, but the keys are still quite long.

Second, even the softened condition is impossible to achieve for several cryptographic schemes. For instance, public key encryption[4] would not be possible without further assumptions. Luckily, there are some longstanding problems in mathematics which are believed to be hard to solve even for powerful machines like supercomputers.

[4]Public key encryption is a form of encryption consisting of a pair of keys: a message is encrypted using the *public key* and can only be decrypted using the *secret key*. A good analogy is to see the public key as a lock and the private key as the key for the lock. Anyone can close the lock, but once the lock has been closed, it can only be opened by whoever has the respective key.

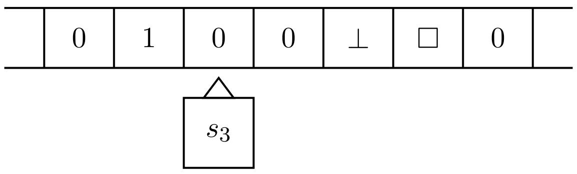

Figure 1.1. A Turing Machine in state s_3. The head of the turing machine is pointing to a cell with symbol 0. If the rule of the Turing Machine is $(s_3, 0) \mapsto (s_5, 1, \mathsf{right})$, in the next step s_3 will be replaced with s_5, the symbol 0 in the cell pointed by the machine head will be replaced with 1, and the head moves one step to the right.

This fact allowed Whitfield Diffie and Martin Hellman [DH76], in 1976, to construct a system to exchange secret keys without the need of a secure (encrypted) channel. Shortly after, Ronald Rivest, Adi Shamir and Leonard Adleman invented what is still currently one of the most used public key cryptosystems [RSA78]. However, there was not yet an equivalent of *perfect secrecy* for this kind of encryption schemes.

This gap has been closed by Shafi Goldwasser and Silvio Micali in 1984 with the notion of *semantic security* [GM84]. Informally, a cryptographic scheme achieves *semantic security* when any information on the plaintext computed *after* learning the ciphertext can also be computed *without* the ciphertext. The actual definition is quite complex, but in [GM84] they show that it is possible to obtain a formulation which is very similar to the one of perfect secrecy. Namely, they prove that a cryptographic scheme is *semantically secure* if and only if $\mathsf{Encrypt}(\mu_0) \overset{c}{\approx} \mathsf{Encrypt}(\mu_1)$, where $\overset{c}{\approx}$ is a special symbol that, informally, states that no machine can efficiently distinguish between the two random variables. In other words, $\overset{c}{\approx}$ is a relaxation of $\triangleq$ in the setting of *computational security*. A formal definition of both can be found in Section 2.1.

Computational complexity. Now that we established what we want to achieve, we need to formalize the environment in which we want to work. Terms like "efficiently computable function" or "intractable problems" are great to convey the general idea, but they are far from being formal; the goal of complexity theory is to give these terms precise meaning.

Before talking about efficiency, we need to define what does "computing a function" mean. Intuitively, computation happens through a sequence of *steps*, or *instructions*, which are executed by a machine. This sequence of instructions is what is called an *algorithm*; however, formally defining what is an algorithm is a hard task [BG03] which is way beyond the scope of this thesis. Usually, the notion of algorithm is considered to be equivalent to the notion of *Turing Machines*.

In short, a Turing Machine, shown in Section 1.1 is a mathematical model which describes an abstract machine capable of computation. The way in which a Turing Machine works can be summarized as follows. The machine is composed by two parts, namely a head and a tape. The tape is of infinite length and divided in cells, while the head is placed over one cell of the tape and has the ability to read from or write onto such cell; moreover, the head has an internal state. In each step, the head reads the cell, updates its internal state, writes back onto the cell and finally either stays on the same tape cell or moves one tape cell on the left or on the right. A more detailed introduction to Turing Machines is out of the scope of this thesis, and we refer the interested reader to, e.g., [BC94]. In Section 1.2, instead, we give a more concrete idea of how computation happens in real life. For now, it suffices to know that Turing Machines allows to formalize computation in an useful way.

Now we are ready to talk about efficiency. Intuitively, the faster an algorithm outputs the correct result, the more efficient it is. The running time may depend on several factors, including (a) the input of the algorithm, (b) the number of steps performed by the algorithm, and (c) the execution time of each step. Items (b) and (c) can be easily addressed by measuring the running time in *number of steps*, thus leaving us to only deal with item (a). However, knowing the exact number of steps is, more often than not, unnecessary; instead, we just aim for a *good estimate*. Finally, the effort needed to perform some task usually depends on the size of the task. Take, for instance, the textbook multiplication between two 3-digit numbers: this operation always takes a person roughly the same time and the same number of steps, regardless of the two numbers (except for a few special cases, like 200×300); on the other side, multiplying two 6-digit numbers almost always takes longer than multiplying two 3-digit numbers. The same happens with algorithms: usually, the (good estimate of the) running time only depends on the size of the input.[5]

We can now classify these execution times, thus obtaining a concrete idea of the efficiency of an algorithm. An algorithm which runs in a number of steps proportional to the size of the input is usually called *linear-time*; an example of this is the textbook addition of two numbers. Similarly, an algorithm which runs in a number of steps proportional to the square of the size of its input is usually called *quadratic-time*; an example of this is the textbook multiplication. These kind of algorithms are quite fast, especially for a modern computer, which is able to perform billions of operation in one second. More in general, an algorithm running in a number of steps proportional to a polynomial of the size of its input is usually called *polynomial-time*. In complexity theory, the class of the problems which are *solvable* by a *polynomial-time* algorithm is called $\mathcal{P}$.

Sometimes, things do not go so well, and, for certain problems, polynomial-time algorithms do not (yet) exist. Nonetheless, it may be the case that there exists a polynomial-time algorithm which checks whether a *solution* to the problem is correct. In this case, we say that such problem is *verifiable* in *polynomial-time*, and we denote the class of such problems with $\mathcal{NP}$.[6] Clearly, every problem in $\mathcal{P}$ is also in $\mathcal{NP}$: to verify the correctness of the solution, it suffices to solve the problem again with the same initial parameters. On the opposite, it is not yet known whether $\mathcal{NP} = \mathcal{P}$ or $\mathcal{NP} \neq \mathcal{P}$. If $\mathcal{NP} = \mathcal{P}$, then any problem verifiable in polynomial-time is also solvable in polynomial-time. Since this fact is quite counter-intuitive, it is commonly assumed that $\mathcal{NP} \neq \mathcal{P}$.

Examples of problems that are in $\mathcal{NP}$ but are not known to be in $\mathcal{P}$ are *integer*

[5]Notice that this is not always the case. To stick with the informal language, there are certain problems which are "easy" to solve in the average case but "hard" to solve in the worst case.

[6]Actually, $\mathcal{NP}$ is defined as the class of the problems which are solvable by polynomial-time *non-deterministic* algorithms (or machine). Roughly speaking, a non-deterministic machine is a machine which is able to take multiple paths of execution at the same time. However, it has been proven that the two definitions of $\mathcal{NP}$ are equivalent.

factorization (used, for instance, in RSA) and *discrete logarithm* (used, for instance, in the Diffie-Hellman key exchange).

Reductions. The last question of this section is the following: what is the role of hard-to-solve but easy-to-verify problems?

The culprit is that some of these problems can be transformed into cryptosystems; after that, a *proof by reduction* proves that, under the assumption that the underlying problem is hard to solve, the cryptosystem is secure. Indeed, a reduction is a polynomial-time algorithm that converts an adversary attacking the cryptosystem into an adversary trying to solve the problem, hence showing that cracking the cryptosystem is at least as hard as solving the underlying problem.

Actually, the proof by reduction works in a more general setting. Assume that a cryptosystem A works by using, internally, a cryptosystem B. If B is already proven to be secure, a proof by reduction would prove that A is at least as secure as B.[7] Notice that, in this case, we did not make any computational assumption. Indeed, if B has statistical (or even perfect) security, it is possible to use *unbounded* algorithms to prove that the scheme A has a similar flavour of security.

1.2 Real life or just fantasy?

In the previous section, among the other things, we introduced the notion of Turing Machine to describe an algorithm, and then we used this notion to discuss about the efficiency of algorithms in terms of its number of steps.

On one side, this model is not too far from the real situation. A computer program is simply a list of instructions which are executed by the processor. These instructions are very basic, sometimes they are just simple arithmetics, logic, read or write instructions or instructions which modify the flow of the program (for instance, "go back to instruction 3 if flag `ZF` is 0"). The main advantage of computers over Turing Machines is that computers are able to access very distant memory cells in just one step, while Turing Machines need to scroll the full tape; otherwise, the computing capability of computers and Turing Machines are pretty similar.

On the other side, however, this is not the full story.

Computer architecture. Computers are very complex machines. For the next examples, we need to go back to the 80's.

The very first microprocessor of the x86 family was the Intel 8086. The 8086 is a 16-bit microprocessor, meaning that it is able to process data 16 bits at a time (or, numbers from 0 to 65535). Internally, the 8086 has a *microcode* which handles almost all the instructions; however, this means that every instruction is composed by *micro-instructions* and takes more than one clock cycle (think of them as "sub-steps") to be executed. The `ADD` instruction, for summing two numbers, may take from 3 to 33 clock cycles, depending on whether the two operands are in memory or already loaded into internal "registers". The same `ADD` instruction in the later Intel 80486 may take from 1 to 9 clock cycles, depending also on whether the

[7] More in detail, a proof by reduction would prove that breaking A implies breaking B, but this is in contrast with the starting assumption that B is secure.

Figure 1.2. The logos of the Spectre and Meltdown security vulnerabilities.

two operands have been used recently or not (i.e. it takes shorter if they are loaded in the cache). With a completely different architecture, the Atmel atmega2560 microcontroller, from the 90's, executes the instruction `ADD` in just 1 clock cycle.

Another difference between the above microprocessors and microcontroller is the maximum allowed clock speed: for the 8086, it's 10 MHz (i.e. 10 millions clock-cycles in one second); for the Atmel, it's 16 MHz; for the 80486, it's 33 MHz (or up to 100 MHz for subsequent versions of the same processor). After 30 to 40 years, we have now microprocessors able to run at 5 GHz, or 5000 MHz. This suggests that quantifying the efficiency of an algorithm in actual time (e.g. seconds) may lead to inconsistent results.

Let's dig a bit further into the 8086. The `MUL` instruction, for multiplying two numbers, may take from 70 to over 150 clock cycles depending on a number of factors. Assume that the operands are 8-bit long and that they are already loaded into registers. Then, the multiplication instruction takes from 70 to 77 clock cycles. By reverse-engineering the microcode [Jen] obtained by analyzing the internal parts of the 8086 [Shi] (the so-called *die*), it is possible to see that the number of clock cycles depends on one of the inputs and, in particular, from its *hamming weight*. This means that, by measuring the number of clock cycles necessary to execute the `MUL` instruction, one could leak the hamming weight of one of the operands

Leakage attacks. As we can see, the harsh reality may be pretty different than the theorized model. More sophisticated analysis than the one just performed falls within the category of *timing attack*. This kind of attack aims to measure the running time of an algorithm in order to get vital information about one of the inputs (for instance, in the case of a cryptosystem, the secret key). Such attacks, exploiting conditional statements, cache hits or misses, instructions duration and any other time-dependant operation, have been proven to be very harmful [Koc96], allowing, for instance, to completely recover the secret key of a cryptosystem. Very recently, two security vulnerabilities have been discovered which are based on timing attacks and affect most of the modern CPUs. The *Spectre* vulnerability [KHF$^+$19] is based on *speculative execution*, an optimization technique in which some tasks are performed before even knowing whether they are needed or not; this is done in order to prevent delays during execution. The *Meltdown* vulnerability [LSG+18] is able to break memory isolation between user and kernel space by exploiting side-effects of *out-of-order execution*, another optimization technique in which instructions are, when possible, executed in an order which is not the original; this optimization allows the CPU to be always busy so to not waste clock cycles.

We can go even deeper. CPUs operate on group of bits, digital signals which may only assume two values: either 0 or 1. On the physical level, this is conveyed by electrical signal: a tension of 5 V in a 5 V microprocessor usually corresponds to a bit with value 1, while a tension of 0 V usually corresponds to a bit with value 0.[8] Operations on these signals are possible thanks to *transistors*, tiny electrical switches which do not require manual interaction.[9] A modern CPU has billions of them and, being physical objects operating on electrical currents, transistors are constantly interacting with the external environment. For instance, a CPU computing a multiplication will consume more power than a CPU which is just idling or updating some register. This fact allows for another class of attacks based on *power analysis* [KJJ99]. Furthermore, currents create magnetic fields, and changing currents create electromagnetic radiation, which can be detected and exploited as well [GMO01]. In 2020, Mordechai Guri even managed to turn a computer's RAM into a Wi-Fi card, thus effectively allowing him to transmit data [Gur20].

Faults and tampering attacks. So far, we have only seen *passive* side-channel attacks, in which secrets or partial information is leaked; however, this is not yet the full picture. Other than influencing the environment, electrical components can be affected by it. For instance, *cosmic rays* (i.e. high-energy subatomic particles) are able to cause an unintended bit flip in memory. For this reason, usually servers and machines in critical roles use error-correcting RAM modules, which are able to correct one or two bit flips at the cost of some efficiency. Simulating cosmic rays is not much efficient, but this kind of faults can be induced as well by some attacker [GA03] in order to modify the internal state of the machine and gain access to protected resources.

On the other side, the same can be achieved in a relatively cheap way with lasers [BS03, CML+11, CLFT14, GGD17, CMD+18], causing serious threats to the security of cryptographic schemes. Further attacks could be performed exploiting other physical properties or phenomena like temperature [GA03, HS14], voltage [KJP14, BBB+10] or Eddy currents [BS03].

One way to break security of cryptographic schemes by tampering with them relies on observing the effect of the tampering on the output of the scheme. This has been applied successfully in *related key attacks* [Bih94], in which an attacker is able to produce encryptions under different but related keys. Related key attacks can be devastating, leading to a full key recovery by the attacker. Successful related-key attacks have been studied for several encryption schemes and cryptographic

[8]Actually, it is more complex than that. In USB communication, ones are represented by a changing signal (either 0 V to 5 V or 5 V to 0 V) while zeroes are represented by a steady signal. The very recent GDDR6X memory modules are able to double the transmission capacity by properly dividing the space between 0 V and maximum voltage in 4 parts instead of just 2.

[9]For the sake of completeness, transistors are a lot more than this. They are used both in digital and analog circuits for several different tasks, another large use case being signal amplification.

protocols, including DES and its variations [KSW96, KSW97], RC4 and the WEP protocol [FMS01, SIR02], and, although still an impractical attack, AES [BK09].

1.3 Secret Sharing

In [Liu68], Chung Laung Liu considers the following problem.

> “Eleven scientists are working on a secret project. They wish to lock up the documents in a cabinet so that the cabinet can be opened if and only if six or more of the scientists are present. What is the smallest number of locks needed? What is the smallest number of keys to the locks each scientist must carry?
>
> – *Chung Laung Liu, “Introduction to Combinatorial Mathematics” [Liu68]*

If the documents are physical objects, the solution can be very impractical, using 462 locks and 252 keys per scientist. On the other side, if the documents are just *data*, which can be manipulated in different ways, these numbers can be made way smaller. Indeed, Adi Shamir proved in [Sha79] that, with the use of polynomials over finite fields, giving each scientist just one piece of information the same size of the original data suffices to accomplish the same goal. A concurrent work of George Blakley [Bla79] achieved a similar result with the use of hyperplanes.

The original problem. Intuitively, the goal of *secret sharing* is to share a piece of secret information among a certain number of parties, in such a way that at least t parties are able to reconstruct the message; additionally, less than t parties should not be able to learn any information on the original message.

In the setting in which all the parties are required to reconstruct the secret (i.e. $t = n$), the first requirement can be simply achieved by splitting the original message into n pieces, where n is the number of involved parties; however, this does not satisfy the second requirement: each party alone would know $\frac{1}{n}$ of the original message. Even worse, $k < n$ colluding parties would know $\frac{k}{n}$ of the original message, even though they do not constitue an authorized subset. This is still in net contrast with the goal of *semantic security*, which aims to protect the information in full, i.e. without partial leaks.

In addition, this method does not work when $t < n$.

Blakley's solution. The solution proposed by Blakley in [Bla79] uses geometry and, in particular, intersecting hyperplanes in an n-dimensional space.

As an example, assume that $n = 3$. We can consider the following situations in the tridimensional space $\mathbb{R}^3$. Let $m \in \mathbb{R}^3$.

- Given 3 pairwise non-parallel planes $\pi_1, \pi_2, \pi_3 \subseteq \mathbb{R}^3$ such that $m \in \pi_1 \cap \pi_2 \cap \pi_3$, it is impossible to identify m with just two of the three planes. Indeed, the intersection between two planes is a straight line, and, without the third plane, m could be anywhere on such line. Giving to each party one plane, we can obtain a scheme in which all the three parties are required to reconstruct m, and less than three parties reveal (almost) no information about m. In this case, $t = 3$ and $n = 3$.

- Given 3 pairwise non-parallel straight lines $r_1, r_2, r_3 \subseteq \mathbb{R}^3$ such that $m \in r_1 \cap r_2 \cap r_3$, it is impossible to identify m with just one of them; however, two of them are enough to locate m in $\mathbb{R}^3$. Giving to each party one line, we can obtain a scheme in which any two of the three parties are able to reconstruct m, and less than three parties reveal (almost) no information about m. In this case, $t = 2$ and $n = 3$.

- The situation $t = 1$ and $n = 3$ is trivial: it suffices to give the point m to each party.

Blakley generalizes this example to n parties: the secret message is a point $m \in \mathbb{R}^n$, an each party is given a t-dimensional hyperplane $\pi \subseteq \mathbb{R}^n$. If the hyperplanes are randomly generated so that they are pairwise non-parallel and intersecting in m, this (almost) gives a secret sharing scheme.

The careful reader probably noticed that, to describe this setting, we used everywhere the word “almost”. Indeed, this method has two problems.

- An $(n-1)$-dimensional hyperplane in $\mathbb{R}^n$ can be described with one n-variate linear equation, while a 1-dimensional hyperplane (i.e. a straight line) requires a system of $n-1$ n-variate linear equations to be described. Space efficiency can be made better for low-dimensional hyperplanes by switching to generating vectors instead of systems of equations; however, the problem is still there in the case of, e.g., $t = \frac{n}{2}$.

- From the point of view of security, this method reveals partial information on the secret $m \in \mathbb{R}^n$. Indeed, any hyperplanes gives a hint about *which hyperplane* contains m; furthermore, any subset of hyperplanes intersect in an hyperplane of lower dimension, which further restricts the possibilities for the position of m.

However, with further restrictions on the planes and tweaks on the encoding system, this scheme can be made equivalent to the following solution.

Shamir’s solution. The solution proposed by Shamir in [Sha79] uses polynomial interpolation and evaluation. For this solution, we directly use finite fields. Let $\mathbb{F}$ be a finite field with characteristic greater than n, let $m \in \mathbb{F}$ be the secret message and let $p(x) = a_0 + a_1 x + \ldots + a_{t-1} x^{t-1}$ be a $(t-1)$-degree polynomial. By letting $a_0 = m$ and, for $i \in \{1, \ldots, n\} \subseteq \mathbb{F}$, $y_i := p(i)$, we get $n+1$ distinct points on a plane. The first point is $(0, m)$, which identifies the secret message. The subsequent n points are of the form $(i, p(i))$, which we give to each party.

Given t such points, we obtain a solvable system with t linear equations in the t variables $a_0, \ldots, a_{t-1}$; solving this system gives back the polynomial p, with which is possible to get back the message by computing $m = p(0)$.

Given $t-1$ such points, we obtain a system with $t-1$ linear equations in t variables. By adding to the system $a_0 = m'$ for $m' \in \mathbb{F}$, the system becomes solvable, and solving it gives some polynomial p'. Since this could be done for any $m' \in \mathbb{F}$, it means that any message in $\mathbb{F}$ is equally likely, hence $t-1$ points still do not reveal anything about the secret.

Motivation. The first motivation behind secret sharing should be quite clear at this point: it allows to spread trust among several parties without the need to trust single parties or too small groups of parties. From the point of view of data storage, secret sharing allows to save data in multiple places with a good compromise between privacy and data integrity. Indeed, secret sharing allows a client using n servers to withstand up to $t-1$ data breaches before losing the privacy guarantee and $n-t$ server failures before losing the ability to reconstruct.

This technique could be used in the opposite sense as well: secret sharing could be used to store, in a distributed fashion, encrypted information with a limited lifetime and, once the lifetime expires, deleting $n-t+1$ shares suffices to make the information not available anymore. This is the idea behind the *Vanish* system [Van09]. Unfortunately, Vanish has been proven susceptible to Sybil[10] attacks in [WHH+10], and therefore insecure.

With additional properties, secret sharing is used in threshold cryptosystems [FP01, KY02, LY11, BGG+18]. The goal of threshold cryptography is to allow several parties to do cryptography (e.g. encryption, signatures) in a distributed way, without the need of a trusted third party and without the need of trusting the single parties.

More in general, secret sharing has applications in secure multi-party computation and distributed protocols in general e.g. in [BGW88, CDM00, BCG+15, Pat16].

Additional properties. Secret sharing has several flavours of security. Here, we list some of them.

Proactive secret sharing [HJKY95] allows to refresh the shares, thus actively protecting the secret even in case an attacker manages to steal some of the shares.

Verifiable secret sharing [CGMA85] allows the parties to check whether their shares are consistent with the shared secret. This is often used when the dealer is not trusted, e.g. in the context of multi-party computation.

Robust secret sharing [TW88] prevents the corruption of the shared secret caused by a certain number of cheating parties.

A *linear secret sharing* scheme allows to perform linear operations on the shared secret by just performing such operations on the shares. Using the linear properties of polynomials, it is easy to show that Shamir's secret sharing is a linear secret sharing scheme.

Homomorphic secret sharing [BGI16] is an upgrade of linear secret sharing in that it allows to perform *any* operation (instead of only linear operations) on the secret through the shares.

Finally, *leakage resilient* [KMS18] and *non-malleable* [GK18a] secret sharing schemes have the additional guarantees that they are secure against side-channel attacks on the shares.

1.4 Thesis contributions

In the setting described in Section 1.2, it is necessary to improve the security of the cryptographic primitives so that they are able to withstand side-channel attacks, and

[10]A Sybil attack is an attack in the context of distributed services in which the attacker gains more power by creating multiple fake identities.

secret sharing is not an exception. In the last few years, a lot of work has been done in this direction, and the focus of this thesis is on *leakage-resilient non-malleable secret sharing.*

Background. In the context of secret sharing, a tampering attack is formalized as a function $f : \mathcal{S} \to \mathcal{S}$ which takes as input the original shares and produces a new set of shares. Informally, we say that the tampering attack is successful if the new shares reconstruct to a secret which is related to (but different from) the original one. This relation is measured by means of a "security game" in which the attacker tries to distinguish between the secret sharing of two arbitrary messages by only observing the result of the tampering attack on the reconstructed message. More in general, the attacker may be able to attack the same scheme multiple times and see all the reconstructed values. In this case, the multiple tampering attacks are formalized by a sequence of (adaptively chosen) functions $(f_1, \ldots, f_p) : \mathcal{S} \to \mathcal{S}$. The details and the formal definition can be found in Section 2.3, but, intuitively, the scheme is *non-malleable* if no attacker is able to distinguish between any two messages (except with negligible advantage), meaning that the tampering attack(s) cannot produce a related message.

Similarly, a leakage attack is formalized as a function $g : \mathcal{S} \to \Sigma$, where Σ is an arbitrary set (usually, a set of binary strings: $\Sigma = \{0, 1\}^*$), and, in this case, the goal of the attacker is to distinguish between the secret sharing the two messages by only learning the output of g. Here, multiple attacks are again formalized as a sequence of (adaptively chosen) functions $g_1, \ldots, g_q : \mathcal{S} \to \Sigma$.

Clearly, it is impossible to protect against arbitrary attacks: for the case of tampering, the adversary could simply choose $f(\sigma) = \mathsf{SS}.\mathsf{Share}(0^{n-1}1 \oplus \mathsf{SS}.\mathsf{Reconstruct}(\sigma))$ as a tampering function: f reconstructs the secret, flips the last bit and then shares it again, thus creating a secret sharing of a message related to the original one. However, by restricting the class $\mathcal{F}$ of the admissible tampering functions, it is possible to achieve such security guarantee. A common restriction is that the tampering function f is only allowed to modify each share independently (the so-called *independent tampering*). A weaker restriction (thus leading to a stronger tampering model) allows the tampering function f to partition the set of shares in non-overlapping unauthorized subsets and then tamper jointly with all the shares in each partition set (the so-called *joint tampering*). Similarly, for the case of leakage, the adversary could simply choose the function $f(\sigma) = \mathsf{SS}.\mathsf{Reconstruct}(\sigma)$ which outputs the message in clear. The same restrictions of before hold for the leakage function family as well, which lead to, respectively, *independent leakage* and *joint leakage.*

First contribution. The main question we look at is how far it is possible to push the security of secret sharing schemes. More in detail, we ask ourselves if it is possible to withstand attacks in which the attacker is able to *continuously* (i.e. without bounding the number of queries in advance) leak from and tamper with the shares in a joint fashion.

In [BFO+20] we make a step forward towards answering this question by building a secret sharing scheme which is able to withstand multiple tampering attempts against disjoint groups of shares. Our result achieves a very strong flavour of security which we call *semi-adaptive partitioning*. Namely, the attacker is able to *adaptively* change the partition within each tampering query, with the further restriction that, once some of the shares have been tampered together, they are always either considered jointly or completely ignored. Unfortunately, the number of tampering attempts allowed to the adversary is still bounded a-priori.

Theorem (informal)**.** *Assuming the existence of one-to-one one-way functions, there exists a secret sharing scheme satisfying k-joint p-time non-malleability under semi-adaptive partitioning, for $k \in O(\sqrt{\log(n)})$ and $p > 0$. Furthermore, the secret sharing scheme is allowed to have any access structure tht can be described by a polynomial-size monotone span program for which authorized sets have size greater than k.*

In [BFV21] we are finally able to answer positively to this question, proposing the first secret sharing scheme which is leakage-resilient and continuously non-malleable against adversaries which are able to jointly tamper with the shares.

Theorem (informal)**.** *Assuming the existence of one-to-one one-way functions, there exists a secret sharing scheme satisfying k-joint leakage-resilient continuous non-malleability under selective partitioning. The secret sharing scheme supports the threshold t-out-of-n access structure for $t > 2n/3$ and has the optimal parameter of $k = t - 1$.*

Finally, we would also like to stress that our results hold in the plain model (i.e. without any CRS or random oracle) and under standard assumptions (i.e. the existence of one-to-one one-way functions).

Second contribution. Another important question is about the rate of secret sharing schemes, that is, the proportion between the message and the size of the resulting shares. Interestingly, our first result comes from a simple but powerful theorem: in [BFO$^+$20], we prove that any one-time non-malleable code has a certain degree of leakage-resilience. A corollary of this fact is the possibility to apply a lower bound from [NS20] which, in turns, gives a lower bound for the size of the shares of any (statistically) non-malleable secret sharing scheme.

In [BFV21], we discover an upper bound for the rate of continuously non-malleable secret sharing scheme with a certain degree of security.

Theorem (informal)**.** *Any continuously non-malleable secret sharing scheme for n parties realizing a threshold access structure t-out-of-n and security against jointly tampering with $k > t/2$ shares cannot achieve better asymptotic rate than $\rho \leq t - k$.*

In the same work, we show a compiler to get better rates. Using this compiler, we achieve a continuously non-malleable secret sharing scheme with optimal rate. Furthermore, we achieve the first continuously non-malleable secret sharing scheme against independent tampering breaking the rate-one barrier. In particular, our construction achieves rate $t/2$.

For comparison, when not considering leakage or tampering, Krawczyk proved in [Kra94] that the best possible rate for secret sharing is t.

Previous work on non-malleable secret sharing. Non-malleable secret sharing has been introduced by Goyal and Kumar in [GK18a] against both independent and joint tampering attacks. In a follow-up paper [GK18b], they focused on n-out-of-n secret sharing, achieving a stronger flavour of non-malleability. These two papers opened the way for the study of non-malleable secret sharing schemes [CL18, ADN+19, KMS19, BS19, FV19, SV19, BFV19, GSZ20, CKOS21, CKOS22], both in the setting of independent and joint tampering.

In the computational setting, Faonio and Venturi [FV19] obtained the first construction which is able to resist against any (polynomial) number of tampering attempts. They also show that such kind of security is impossible to achieve in the information-theoretic model. Furthermore, another necessary condition is the self-destruct feature, stating that, once a faulty reconstruction occurs, all the shares must be erased.

Finally, Goyal, Srinivasan and Zhu [GSZ21] obtained a construction in an even stronger model in which partitions are allowed to overlap.

Related work. The notion of non-malleable secret sharing is closely related to the one of *non-malleable codes*, introduced by Dziembowski, Pietrzak and Wichs in [DPW10]. Intuitively, a non-malleable code allows to encode a message in a way that makes it resist tampering attempts. The result is that, after the encoding has been modified, either the encoded message is the same or it is a completely unrelated one. Again, it is impossible to protect against arbitrary attacks, therefore some tampering models are necessary.

A long line of research [DPW10, LL12, DKO13, CG14a, CG14b, ADL14, CZ14, ADKO15a, ADKO15b, CGL16, CKR16, Li17, KOS17, ADN+19] focuses on the so-called *n-split-state* tampering, in which the encoding is split into n blocks and then the attacker is only allowed to tamper with each block independently. This notion is closely related to the one of non-malleable secret sharing. In [ADKO15b], Aggarwal *et al.* show that every 2-split state non-malleable code is also a 2-out-of-2 non-malleable secret sharing. Unfortunately, this does not hold for $n \geq 3$.

Similarly, any (leakage-resilient) 2-split-state *continuously* non-malleable code [FMNV14, FNSV18, OPVV18, CFV19] is a (leakage-resilient) 2-out-of-2 *continuously* non-malleable secret sharing.

However, non-malleable codes do not have "shares" which need to be sent to different parties. This makes non-malleable codes more suitable for other classes of tampering attacks such as decision-tree tampering [BDKM18, BGW19, BFMV22], tampering in AC^0 [BDKM18, BGW19], tampering via bounded-depth circuits [BDKM16, CL17, BDG+18, BFMV22], tampering via partial functions [KLT18], space-bounded tampering [FHMV17, CCHM19], and many others.

Structure. The structure of the thesis is as follows.

- In Chapter 2 we establish the basic notation and we define the cyrptographic primitives that we are using in our results.

- Chapter 3 is focused on how to achieve the stronger flavours of non-malleable secret sharing described above, pointing out the challenges and highlighting the main ideas.

- In Chapter 4, we start by showing that a non-malleable secret sharing scheme has a certain degree of leakage resilience. Then, we prove the aforementioned upper and lower bounds. Finally, we show how to achieve optimal rate.

Chapter 2

Preliminaries

In this chapter, we establish the foundations for the next chapters, introducing the basic tools that we are going to use in the remainder of the thesis.

In Section 2.1, we establish the basic notation and the necessary mathematical notions. Then, we introduce the notation for algorithms and cryptographic scheme and we give an overview on security games.

In Section 2.2, we state the formal definitions of all the cryptographic primitives that we are going to use in the rest of the thesis.

Finally, in Section 2.3 we give an overview on leakage resilience and non-malleability applied to secret sharing schemes and we establish the specific definitions and notations to talk about this topic.

2.1 Notation

Basic notation. Let $\mathbb{N}, \mathbb{Z}, \mathbb{R}$ be respectively the set of natural numbers, the set of integers and the set of real numbers. If $n \in \mathbb{N}$ and $n \geq 1$, we write $[n]$ for the set $\{1, \ldots, n\}$. If $a, b \in \mathbb{R}$, with $a \leq b$, we write $[a, b]$ for the interval $\{x : a \leq x \leq b\}$. If $p \in \mathbb{N}$ is a prime number, we denote by $\mathbb{F}_p$ the finite field of size p, or we simply write $\mathbb{F}$ when p is clear from the context. For any other set, we use the calligraphic uppercase letters $\mathcal{A}, \mathcal{B}, \mathcal{I}, \ldots$. The cardinality of a set $\mathcal{S}$ is the number of its elements, which we denote as $\#\mathcal{S}$.

For $n \in \mathbb{N}$, if $x = (x_1, \ldots, x_n) \in \mathcal{X}_1 \times \ldots \times \mathcal{X}_n$ is a tuple of n elements and $\mathcal{I} \subseteq [n]$ is a set of indices, we write $x_\mathcal{I}$ as a shorthand for $(x_i)_{i \in \mathcal{I}}$, that is, the tuple of length $\#\mathcal{I}$ formed by the elements in position $i \in \mathcal{I}$ of the original tuple x.

If $\mathcal{S}$ is a finite and non-empty set, we can consider $\mathcal{S}$ to be an *alphabet*. A string of length $\ell \in \mathbb{N}$ over the alphabet $\mathcal{S}$ is an element $x \in \mathcal{S}^\ell$. Sometimes, it's simpler to just consider the set of all the strings over $\mathcal{S}$: we denote this set by $\mathcal{S}^* := \cup_{n \in \mathbb{N}} \mathcal{S}^n$. In any case, it is always defined the *length* of a string $x \in \mathcal{S}^*$, which is the value $|x| \in \mathbb{N}$ such that $x \in \mathcal{S}^{|x|}$. If $x = (x_1, \ldots, x_m) \in \mathcal{S}^m$ and $y = (y_1, \ldots, y_n) \in \mathcal{S}^n$ are two strings, we define their concatenation as $x \| y = (x_1, \ldots, x_m, y_1, \ldots, y_n) \in \mathcal{S}^{m+n}$.

For a function f with domain in $\mathcal{X}$ and codomain in $\mathcal{Y}$, we write $f : \mathcal{X} \to \mathcal{Y}$, or $f : \mathcal{X} \to \mathcal{Y} : x \mapsto f(x)$ if we want to explicit the name of the variable or the function map. As an example for the latter case, the function which maps $x \in \mathbb{R}$ to its square can be written as $f : \mathbb{R} \to \mathbb{R} : x \mapsto x^2$. Finally, in some cases we may need to use some function without giving it an explicit name, domain or codomain; in that case, we simply write the last part, e.g. $x \mapsto x^2$, or simply the expression x^2 if the variable is clear from the context.

Asymptotic notation. Some times we are not interested in the exact value of a function, but only in how "big" or "small" its values are compared to other functions. Let $f, g : \mathbb{R} \to \mathbb{R}$ be two arbitrary functions. We say that:

- $f \in O(g)$ if there exist constants $c, x_0 \in \mathbb{R}$, with $c > 0$, such that
$$\forall x > x_0, f(x) \leq c \cdot g(x).$$
Intuitively, $f \in O(g)$ if f is *bounded* by g from above, up to a constant factor.
- $f \in o(g)$ if, for all $c \in \mathbb{R}, c > 0$, there exists a constant x_0 such that
$$\forall x > x_0, |f(x)| \leq c \cdot |g(x)|\,.$$
Intuitively, $f \in o(g)$ if f is *asymptotically dominated* by g.
- $f \in \Omega(g)$ if $g \in O(f)$, that is, if there exist constants $c, x_0 \in \mathbb{R}$, with $c > 0$, such that
$$\forall x > x_0, f(x) \geq c \cdot g(x).$$
Intuitively, $f \in \Omega(g)$ if f is *bounded* by g from below, up to a constant factor.
- $f \in \omega(g)$ if $g \in o(f)$, that is, for all $c \in \mathbb{R}, c > 0$, there exists a constant x_0 such that
$$\forall x > x_0, |f(x)| \geq c \cdot |g(x)|\,.$$
Intuitively, $f \in \omega(g)$ if f *asymptotically dominates* g.
- Finally, we say that $f \in \Theta(g)$ if $f \in O(g) \cap \Omega(g)$, that is, if there exist constants $c_1, c_2, x_0 \in \mathbb{R}$, with $c_1, c_2 > 0$, such that
$$\forall x > x_0, c_1 \cdot g(x) \leq f(x) \leq c_2 \cdot g(x).$$
Intuitively, $f \in \Theta(g)$ if f and g have the same asymptotic behavior, up to constant factors.

A common abuse of notation is to replace the symbol $\in$ with $=$. This is indeed very useful, in that it allows to compose large statements without adding too much notation.

In cryptography it is also common to use two other classes of functions:

- A function $f : \mathbb{N} \to \mathbb{R} : n \mapsto f(n)$ is *polynomially bounded* if there exists $c \in \mathbb{N}$ such that $f \in O(n^c)$. In this case, we write $f \in \mathsf{Poly}(n)$, or $f \in \mathsf{Poly}$ if the variable is clear from the context.
- A function $f : \mathbb{N} \to \mathbb{R} : n \mapsto f(n)$ is *negligible* if, for all $c \in \mathbb{N}$, $f \in o(n^{-c})$. In this case, we write $f \in \mathsf{Negl}(n)$, or $f \in \mathsf{Negl}$ if the variable is clear from the context.

These two classes of functions are closed under sum and product, meaning that, if f and g are in one of these classes, then both $f + g$ and $f \cdot g$ are in the same class. Moreover, it is easy to show that, for all $f \in \mathsf{Poly}$ and $g \in \mathsf{Negl}$, $f \cdot g \in \mathsf{Negl}$.

Probabilities and random variables. We denote random variables with uppercase bold face letters. For a random variable $\boldsymbol{X} \in \mathcal{X}$, we denote by $\mathbb{P}[\boldsymbol{X} = x]$ the probability that $\boldsymbol{X}$ assumes the value x. The corresponding distribution will be denoted as $p_{\boldsymbol{X}}$; in particular, for all $x \in \mathcal{X}$, $p_{\boldsymbol{X}}(x) = \mathbb{P}[\boldsymbol{X} = x]$. If $\mathcal{Y} \subseteq \mathcal{X}$, we use a slight abuse of notation for the probability that $\boldsymbol{X} \in \mathcal{Y}$ by writing $p_{\boldsymbol{X}}(\mathcal{Y}) := \mathbb{P}[\boldsymbol{X} \in \mathcal{Y}]$. Unless stated otherwise, we reserve the bold face letter $\boldsymbol{U}_{\mathcal{X}}$ for the *uniform* random variable over $\mathcal{X}$, that is, the random variable with the constant distribution $p_{\boldsymbol{U}_{\mathcal{X}}}(x) = \frac{1}{\#\mathcal{X}}$ for all $x \in \mathcal{X}$. When it is clear from the context, we omit the set $\mathcal{X}$ and we simply write $\boldsymbol{U}$. Sometimes, we need to sample multiple times from the same distribution $p_{\boldsymbol{X}}$. In this case, we mark the different random variables with a superscript number in parenthesis: $\boldsymbol{X}^{(1)}, \boldsymbol{X}^{(2)}, \boldsymbol{X}^{(3)}, \ldots$.

The (total variation) statistical distance between two random variables $\boldsymbol{X}, \boldsymbol{Y}$ over the same set $\mathcal{X}$ is the quantity

$$\Delta(\boldsymbol{X}, \boldsymbol{Y}) := \frac{1}{2} \sum_{x \in \mathcal{X}} \left| \mathbb{P}[\boldsymbol{X} = x] - \mathbb{P}[\boldsymbol{Y} = y] \right|.$$

We say that the two random variables are *identically distributed* if $\Delta(\boldsymbol{X}, \boldsymbol{Y}) = 0$ or, equivalently, $p_{\boldsymbol{X}} = p_{\boldsymbol{Y}}$, and we denote this fact by writing $\boldsymbol{X} \triangleq \boldsymbol{Y}$.

Min-entropy and conditional average min-entropy. The *min-entropy* of a random variable $\boldsymbol{X}$ is, intuitively, a measure of its unpredictability. More formally, it is defined as

$$\mathbb{H}_\infty(\boldsymbol{X}) := -\log\left(\max_{x \in \mathcal{X}} \mathbb{P}[\boldsymbol{X} = x]\right).$$

For conditional distributions and random variables, we use the notion of *conditional average min-entropy* from [DORS03].

$$\begin{aligned}\widetilde{\mathbb{H}}_\infty(\boldsymbol{X}|\boldsymbol{Y}) &:= -\log \mathbb{E}_{y \leftarrow\$ \boldsymbol{Y}}\left[2^{-\mathbb{H}_\infty(\boldsymbol{X}|\boldsymbol{Y}=y)}\right] \\ &= -\log \mathbb{E}_{y \leftarrow\$ \boldsymbol{Y}}\left[\max_{x \in \mathcal{X}} \mathbb{P}[\boldsymbol{X} = x | \boldsymbol{Y} = y]\right].\end{aligned}$$

The lemma below is sometimes known as the "chain rule" for conditional average min-entropy.

Lemma 2.1 ([DORS03], Lemma 2.2)**.** *Let $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$ be random variables. If $\boldsymbol{Y}$ has at most 2^ℓ possible values, then*

$$\widetilde{\mathbb{H}}_\infty(\boldsymbol{X}|\boldsymbol{Y}, \boldsymbol{Z}) \geq \widetilde{\mathbb{H}}_\infty(\boldsymbol{X}, \boldsymbol{Y}|\boldsymbol{Z}) - \ell \geq \widetilde{\mathbb{H}}_\infty(\boldsymbol{X}|\boldsymbol{Z}) - \ell.$$

In particular,

$$\widetilde{\mathbb{H}}_\infty(\boldsymbol{X}|\boldsymbol{Y}) \geq \widetilde{\mathbb{H}}_\infty(\boldsymbol{X}, \boldsymbol{Y}) - \ell \geq \widetilde{\mathbb{H}}_\infty(\boldsymbol{X}) - \ell.$$

Algorithms. We denote algorithms in sans-serif $\mathsf{CamelCase}$. If an algorithm $\mathsf{Algorithm}$ computes a function $f : \mathcal{X} \to \mathcal{Y}$, we identify the algorithm with the function and we just write $y = \mathsf{Algorithm}(x)$ for the computation of $\mathsf{Algorithm}$ of $f(x)$. However, we write $y \leftarrow \mathsf{Algorithm}(x)$ if we want to make explicit the fact that y is obtained by running the algorithm $\mathsf{Algorithm}$.

So far, we only talked about *deterministic* functions and algorithms. A *probabilistic* or *randomized* function $f : \mathcal{X} \to \mathcal{Y}$ is a function whose output depends on internal randomness; this is equivalent to writing $f : \mathcal{X} \times \mathcal{R} \to \mathcal{Y}$, where the values from $\mathcal{R}$ are sampled in an uniformly random fashion. Unless specified otherwise, we assume all algorithms to be randomized (notice that this is without loss of generality, since a deterministic algorithm can be seen as a randomized algorithm with $\#\mathcal{R} = 1$). When sampling an output from a randomized algorithm, we write $y \leftarrow\!\!\$\ \mathsf{Algorithm}(x)$. Similarly, we write $x \leftarrow\!\!\$\ \mathcal{X}$ for a uniformly sampled value from $\mathcal{X}$ and $x \leftarrow\!\!\$\ \boldsymbol{X}$ for sampling a value $x \in \mathcal{X}$ distributed as the random variable $\boldsymbol{X}$. When we need to make explicit the randomness of the algorithm, we pass it as the last argument of the algorithm, preceeded by a semicolon: $y \leftarrow \mathsf{Algorithm}(x; \rho)$ for some $\rho \leftarrow\!\!\$\ \mathcal{R}$.

An algorithm computes a function by executing a certain number of steps. Moreover, an algorithm operates on encodings, and the input set $\mathcal{X}$ is usually encoded in a suitable way (e.g. binary); however, in order to simplify notation, we commit a slight abuse of notation and, unless specified, identify $\mathcal{X}$ with its (binary) encoding. Let $\mathcal{X}$ be a set and let $\lambda := \lceil \log(\#\mathcal{X}) \rceil$; sometimes, we refer to λ as the *size* or *length* of (the encoding of) the set $\mathcal{X}$. We say that an algorithm is *efficient* or *PPT* (probabilistic polynomial-time) if it terminates in a number of steps bounded by $\mathsf{Poly}(\lambda)$; if this requirement is not needed, we usually say that the algorithm is *unbounded.*

For an algorithm $\mathsf{D} : \mathcal{X} \to \{0, 1\}$ and two random variables $\boldsymbol{X}, \boldsymbol{Y}$ over the same set $\mathcal{X}$ we call *advantage* of D the quantity

$$\mathsf{Adv}[\mathsf{D}](\boldsymbol{X}, \boldsymbol{Y}) := \left| \mathbb{P}\left[\mathsf{D}(\boldsymbol{X}) = 1\right] - \mathbb{P}\left[\mathsf{D}(\boldsymbol{Y}) = 1\right] \right| .$$

Let $\lambda \in \mathbb{N}$, let $\mathcal{X}_\lambda$ be of size λ and let $\boldsymbol{X}_\lambda, \boldsymbol{X}_\lambda$ random variables over $\mathcal{X}_\lambda$. If, for all *efficient* algorithms $\mathsf{D}_\lambda : \mathcal{X}_\lambda \to \{0, 1\}$, $\mathsf{Adv}[\mathsf{D}_\lambda](\boldsymbol{X}_\lambda, \boldsymbol{Y}_\lambda) \in \mathsf{Negl}(\lambda)$, we say that $\boldsymbol{X}_\lambda$ and $\boldsymbol{Y}_\lambda$ are *computationally close*, and we denote this fact by writing $\boldsymbol{X}_\lambda \overset{\mathcal{C}}{\approx}_\lambda \boldsymbol{Y}_\lambda$; we can simply use $\overset{\mathcal{C}}{\approx}$ instead of $\overset{\mathcal{C}}{\approx}_\lambda$ if λ is clear from the context.

It is easy to show that if, for a parameter $\varepsilon \in [0, 1]$ and for all (even unbounded) algorithms $\mathsf{D} : \mathcal{X} \to \{0, 1\}$, $\mathsf{Adv}[\mathsf{D}](\boldsymbol{X}, \boldsymbol{Y}) \leq \varepsilon$, then $\boldsymbol{X}$ and $\boldsymbol{Y}$ are ε-statistically close. By introducing again the parameter λ referred to the size of the set $\mathcal{X}_\lambda$, we denote by $\boldsymbol{X}_\lambda \overset{\triangle}{\approx}_\lambda \boldsymbol{Y}_\lambda$ (again, replacing $\overset{\triangle}{\approx}_\lambda$ with $\overset{\triangle}{\approx}$ if λ is clear from the context) the fact that $\varepsilon_\lambda \in \mathsf{Negl}(\lambda)$. Furthermore, if $\varepsilon_\lambda = 0$, we write $\boldsymbol{X}_\lambda \triangleq_\lambda \boldsymbol{Y}_\lambda$ (or $\boldsymbol{X}_\lambda \triangleq \boldsymbol{Y}_\lambda$); notice that this is not in contrast with the previous definition of $\triangleq$.

Oracles. Sometimes, we need to grant some algorithm A the ability to compute a particular function f. However, such function cannot be computed by the algorithm A because either A does not have enough information (the function may embed a private key from some encryption scheme) or A does not have enough computational power, or f may be some kind of an *ideal* function (for instance, a *truly random function*). In these cases, we construct what is called an *oracle*, that is, an algorithm $\mathcal{O}_f$ which is either initialized with the inaccessible data or has the capabilities to compute f. Then, A can interact with $\mathcal{O}_f$ in a *black-box* way, meaning that A runs $\mathcal{O}_f$ with some input and gets the corresponding output, but is unable to see how the computation happens inside the oracle. We denote the fact that A has oracle access to $\mathcal{O}_f$ by writing $\mathsf{A}^{\mathcal{O}_f}$, or, e.g., $\mathsf{A}^{\mathcal{O}_f(x,\cdot)}$ if we need to be more explicit, for instance when f is the function $f : (x, y) \mapsto z$, x needs to stay secret and y is chosen by A. Sometimes, when querying on some value y an oracle with pre-initialized values like $\mathcal{O}_f(x, \cdot)$, we write $\mathcal{O}_f(y)$ as a shorthand for $\mathcal{O}_f(x, y)$.

Schemes. A cryptographic scheme is a tuple of algorithms meeting some specifications. For instance, a public key encryption scheme needs three algorithms: an algorithm KGen which generates a pair of keys, an algorithm Encrypt which uses a key to encrypt a plaintext, and an algorithm Decrypt which uses another key to decrypt a ciphertext. If we denote the public key encryption scheme as PKE, the respective algorithms are PKE.KGen, PKE.Encrypt and PKE.Decrypt. In this way, if we have more than one scheme of the same type, say PKE_1 and PKE_2, we are able to identify the respective algorithms by writing PKE_1.KGen and PKE_2.KGen.

Usually, cryptographic schemes have a few additional parameters, which we can either state in the definition (es. PKE has security ε) or carry in the name of the scheme (es. $\mathsf{PKE}[n]$ for some parameter n). Sometimes, there are many parameters, and keeping them implicit or anonymous may be confusing. In these cases, we summarize them in a table:

Scheme PKE	
KeySize :	λ
SecurityClass :	$\mathbf{Ind}^{\mathsf{CPA}}$
SecurityError :	$\mathcal{N}egl(\lambda)$

If we then need to address some parameters of the scheme PKE or add new parameters, we use the same notation used for the algorithms: PKE.KeySize $= n$.

Security games. We define security by means of an interactive protocol with the adversary A. Namely, we define an *interactive experiment* in which the adversary A is allowed to make certain kinds of queries and, at the end of the interaction, outputs some value. More formally, a security experiment is formalized as a random variable $\mathbf{Experiment}_{\mathsf{A}}[\lambda, \dots]$ which is parametrized by the adversary A and takes as input the *security parameter* λ and, if necessary, any other parameter needed to run the experiment. Internally, $\mathbf{Experiment}_{\mathsf{A}}[\lambda, \dots]$ runs the adversary A (that is formalized as one or more algorithms), answers to its queries and finally outputs some value. Security is then defined by establishing two different experiments and requiring that these are closely distributed (either computationally or statistically, depending on the kind of security).

In *simulation based* security, there is usually one experiment $\mathbf{Real}_{\mathsf{A}}[\lambda, \dots]$ in which the adversary interacts with the real scheme and one experiment $\mathbf{Sim}_{\mathsf{A}}[\lambda, \dots]$, in which the adversary interacts with some dummy values and the real protocol is computed by a (simulated) trusted third party. In this case, the security requirement is that, for instance, $\mathbf{Real}_{\mathsf{A}}[\lambda, \dots] \overset{\mathcal{C}}{\approx} \mathbf{Sim}_{\mathsf{A}}[\lambda, \dots]$.

The notion of *game based* security is further divided into two categories. In *game based indistinguishability*, the goal of the adversary is to *distinguish* between two values, say m_0, m_1. The experiment is then parametrized by a bit b which denotes what value has been used in the experiment. In this case, the security requirement is that, for instance, $\mathbf{Experiment}_{\mathsf{A}}[\lambda, m_0, m_1, 0] \overset{\mathcal{C}}{\approx} \mathbf{Experiment}_{\mathsf{A}}[\lambda, m_0, m_1, 1]$. In *game based unpredictability*, the goal of the adversary is to produce a value satisfying some condition (e.g. a valid signature for a message). The output of the experiment, in this case, will be 1 if the adversary manages to output a valid value and 0 if instead the value does not meet the condition. The security requirement will then be that $\mathbb{P}\left[\mathbf{Experiment}_{\mathsf{A}}[\lambda, \ldots] = 1\right]$ is small enough or, with a slight abuse of notation, $\mathbf{Experiment}_{\mathsf{A}}[\lambda, \ldots] \overset{\mathcal{C}}{\approx} 0$.

Finally, sometimes security cannot be proven directly, and it is necessary to introduce some *hybrid* experiment representing an intermediate step between the starting experiment and the final one. Then, security is shown by showing the relations

$$\mathbf{Start}_{\mathsf{A}}[\lambda, \ldots] \overset{\mathcal{C}}{\approx} \mathbf{Hyb}^{1}_{\mathsf{A}}[\lambda, \ldots] \overset{\mathcal{C}}{\approx} \mathbf{Hyb}^{2}_{\mathsf{A}}[\lambda, \ldots] \overset{\mathcal{C}}{\approx} \mathbf{Finish}_{\mathsf{A}}[\lambda, \ldots] \,.$$

2.2 Standard cryptographic primitives

In this chapter, we define the standard cryptographic primitives which we are going to use in the rest of the thesis.

Symmetric Encryption. A secret-key encryption scheme SKE allows to encrypt and decrypt a message with the same key, which is thus needed to kept secret. As introduced in Section 1.1, the desiderable property for an encryption scheme is that no information about the message can be obtained through its encryption.

More formally, let $\lambda \in \mathbb{N}$ be a parameter and let $\mathcal{K}, \mathcal{M}, \mathcal{C}$ be sets, where $\mathcal{K} = \mathcal{K}_\lambda$ is implicitly parametrized by λ. Let SKE be a scheme consisting of the following two algorithms.

- Randomized algorithm $\mathsf{SKE.Encrypt}$: upon input a key $\kappa \in \mathcal{K}$ and a message $\mu \in \mathcal{M}$, output a ciphertext $\gamma \in \mathcal{C}$.
- Deterministic algorithm $\mathsf{SKE.Decrypt}$: upon input a key $\kappa \in \mathcal{K}$ and a ciphertext $\gamma \in \mathcal{C}$, output a message $\mu \in \mathcal{M} \cup \{\perp\}$.

Consider the following oracles.

Oracle:	$\mathcal{O}_{\mathsf{Enc}}$
Input:	$(\kappa, \mu) \in \mathcal{K} \times \mathcal{M}$
	$\gamma \leftarrow\$ \mathsf{SKE.Encrypt}(\kappa, \mu)$
Output:	γ

Oracle:	$\mathcal{O}_{\mathsf{Dec}}$
Input:	$(\kappa, \gamma) \in \mathcal{K} \times \mathcal{C}$
	$\mu \leftarrow\$ \mathsf{SKE.Decrypt}(\kappa, \gamma)$
Output:	μ

Let $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$ be an adversary where

- A_1 takes as input the parameter λ, has oracle access to $\mathcal{O}_{\mathsf{Enc}}, \mathcal{O}_{\mathsf{Dec}}$ and outputs two messages $\mu_0, \mu_1 \in \mathcal{M}$ and an auxiliary state α;

- A_2 takes as input the auxiliary state α and a ciphertext γ, has oracle access to $\mathcal{O}_{\mathsf{Enc}}$ and outputs a bit b'.

Consider the following experiment parametrized by the adversary A, the parameter λ and a bit $b \in \{0, 1\}$.

Experiment:	$\mathbf{Ind}_{\mathsf{A}}^{\mathsf{CCA2}}[\lambda, b]$
	$\kappa \leftarrow\$ \mathcal{K}$
	$(\mu_0, \mu_1, \alpha) \leftarrow\$ \mathsf{A}_1^{\mathcal{O}_{\mathsf{Enc}}(\kappa,\cdot),\mathcal{O}_{\mathsf{Dec}}(\kappa,\cdot)}(1^\lambda)$
	$\gamma \leftarrow\$ \mathsf{SKE}.\mathsf{Encrypt}(\kappa, \mu_b)$
	$b' \leftarrow\$ \mathsf{A}_2^{\mathcal{O}_{\mathsf{Enc}}(\kappa,\cdot),\mathcal{O}_{\mathsf{Dec}}(\kappa,\cdot)}(\alpha, \gamma)$
Output:	b'

Definition 2.2 (Security against chosen-ciphertext attacks)**.** *The scheme* SKE *defined above is a* symmetric encryption scheme *with* indistinguishability against (adaptively) chosen ciphertext attacks *if, for all PPT adversaries* A*,*

$$\mathbf{Ind}_{\mathsf{A}}^{\mathsf{CCA2}}[\lambda, 0] \overset{c}{\approx} \mathbf{Ind}_{\mathsf{A}}^{\mathsf{CCA2}}[\lambda, 1] .$$

Remark 2.3. *Further notions of security exist for symmetric key encryption schemes.*

- *By removing oracle $\mathcal{O}_{\mathsf{Dec}}$ access from A_2, we obtain a weaker notion of chosen ciphertext attacks; the corresponding experiment is* $\mathbf{Ind}^{\mathsf{CCA}}$.

- *By entirely removing the oracle $\mathcal{O}_{\mathsf{Dec}}$, A only has access to an encryption oracle. This is an even weaker security requirement called* indistinguishability against chosen plaintext attacks*, denoted by experiment* $\mathbf{Ind}^{\mathsf{CPA}}$.

Non-Interactive Commitment. A commitment scheme [Blu81] is a two-party protocol that allows one party to commit to a value x and reveal x in a later time. Intuitively, the two requirements satisfied by a commitment scheme are the following.

- A commitment scheme should be *binding*, meaning that the committed value x cannot be changed later for the same commitment.

- A commitment scheme should be *hiding*, meaning that the committed value x should remain secret until it is revealed at a later point.

Non-interactive commitment schemes have the additional properties that no interaction between the two parties is required.

More formally, let $\mathcal{M}, \mathcal{C}$ be two sets and let $\lambda \in \mathbb{N}$ be a parameter. Let Com be a scheme consisting of the following algorithm.

- Randomized algorithm Com.Commit: upon input a message $\mu \in \mathcal{M}$, output a commitment value $\zeta \in \mathcal{C}$.

We start by formalizing the binding property.

Let A be an adversary that takes as input the parameter λ, a message $\mu \in \mathcal{M}$ and random coins $\rho \in \mathcal{R} = \mathcal{R}_\lambda$ and outputs another message $\mu' \in \mathcal{M}$ and random coins $\rho' \in \mathcal{R}$. Consider the following experiment parametrized by the adversary A, the parameter λ, a message $\mu \in \mathcal{M}$ and random coins $\rho \in \mathcal{R}$.

```
Experiment:  Binding_A[λ, μ, ρ]
             $(\mu', \rho') \leftarrow_{\$} \mathsf{A}(1^\lambda, \mu, \rho)$
             $b^* \leftarrow 1$
             IF $\mathsf{Com.Commit}(\mu; \rho) \neq \mathsf{Com.Commit}(\mu'; \rho')$ :
               $b^* \leftarrow 0$
             IF $\mu' = \mu$ : $b^* \leftarrow 0$
    Output:  $b^*$
```

Definition 2.4 (Binding property)**.** *A scheme* Com *has the* binding *property if, for all adversaries* A*, all messages* $\mu \in \mathcal{M}$ *and all random coins* $\rho \in \mathcal{R}$*,*

$$\mathbf{Binding}_{\mathsf{A}}[\lambda, \mu, \rho] \approx 0,$$

where $\approx$ *is one of the following.*

- *If* $\approx$ *is* $\overset{\mathcal{C}}{\approx}$ *and* A *is PPT, we say that* Com *is* computationally binding.
- *If* $\approx$ *is* $\overset{\triangle}{\approx}$*, we say that* Com *is* statistically binding.
- *If* $\approx$ *is* $\triangleq$*, we say that* Com *is* perfectly binding*, meaning that each commitment can only be opened in a single way.*

Formalizing the hiding property should be easier.

Definition 2.5 (Hiding property)**.** *A scheme* Com *has the* hiding *property if, for all pairs of messages* $\mu_0, \mu_1 \in \mathcal{M}$*,*

$$\mathsf{Com.Commit}\left(1^\lambda; \mu_0\right) \approx \mathsf{Com.Commit}\left(1^\lambda; \mu_1\right)$$

where $\approx$ *is one of the following.*

- *If* $\approx$ *is* $\overset{\mathcal{C}}{\approx}$*, we say that* Com *is* computationally hiding.
- *If* $\approx$ *is* $\overset{\triangle}{\approx}$*, we say that* Com *is* statistically hiding.
- *If* $\approx$ *is* $\triangleq$*, we say that* Com *is* perfectly hiding.

A scheme Com is a *commitment scheme* if it has both the *binding* and the *hiding property.*

Remark 2.6. *It is easy to show that a commitment scheme cannot be both perfectly (or statistically) binding and perfectly (or statistically) hiding. Indeed, if a commitment scheme is perfectly binding, there exists only one valid opening* (μ, ρ) *for a given commitment* ζ*; therefore,* ζ *uniquely identifies* μ*.*

For this reason, the commitment schemes considered are usually either computationally binding/statistically hiding *or* statistically binding/computationally hiding.

Secret Sharing. We introduced secret sharing informally in Section 1.3; here, we give the formal definition.

The information on the authorized parties is conveyed by the so-called *access structure*.

Definition 2.7 (Access structure)**.** *Let $n \in \mathbb{N}$. An* access structure $\mathcal{A}$ *for n parties is a monotone class of subsets of $[n]$. In other words, if $\mathcal{I}_1 \in \mathcal{A}$ and $\mathcal{I}_1 \subseteq \mathcal{I}_2$, then $\mathcal{I}_2 \in \mathcal{A}$. We call* authorized *or* qualified *all the sets $\mathcal{I} \in \mathcal{A}$, and we call* unauthorized *any other subset $\mathcal{U} \notin \mathcal{A}$.*

The most common example of access structure is the threshold access structure. In this case, it is defined a parameter $t \in [n]$, which is called *reconstruction threshold*. The *t-out-of-n* access structure is then defined as the class of all the subsets of $\mathcal{A}$ with at least t elements:

$$\mathsf{Threshold}(t, n) := \{\mathcal{I} \subseteq [n] : \#\mathcal{I} \geq t\}.$$

Now we are ready to state the requirements for a secret sharing scheme. Let $n, \lambda \in \mathbb{N}$ be parameters and let $\mathcal{M}, \mathcal{S}_1, \dots, \mathcal{S}_n$ be sets. Let SS be a scheme consisting of the following algorithm.

- Randomized algorithm $\mathsf{SS}.\mathsf{Share}$: upon input a message $\mu \in \mathcal{M}$, output shares $(\sigma_1, \dots, \sigma_n) \in \mathcal{S}_1 \times \cdots \times \mathcal{S}_n$.
- Deterministic algorithm $\mathsf{SS}.\mathsf{Reconstruct}$: upon input $(\mathcal{I}, (\sigma_i)_{i \in \mathcal{I}})$, where $\mathcal{I} \in \mathcal{A}$ and, for all $i \in \mathcal{I}$, $\sigma_i \in \mathcal{S}_i$, output a message $\mu \in \mathcal{M} \cup \{\perp\}$, where $\perp$ is a special symbol denoting that the reconstruction failed due to invalid shares.

Definition 2.8 (Correctness)**.** *A scheme* SS *has the* correctness *property if, for all messages $\mu \in \mathcal{M}$ and all authorized subsets $\mathcal{I} \in \mathcal{A}$,*

$$\mathsf{SS}.\mathsf{Reconstruct}(\mathcal{I}, (\mathsf{SS}.\mathsf{Share}(\mu))_{\mathcal{I}}) = \mu$$

with probability 1 *over the randomness of* $\mathsf{SS}.\mathsf{Share}$.

Definition 2.9 (Privacy)**.** *A scheme* SS *has the* privacy *property if, for all pairs of messages $\mu_0, \mu_1 \in \mathcal{M}$ and all unauthorized subsets $\mathcal{U} \subseteq [n], \mathcal{U} \notin \mathcal{A}$,*

$$(\mathsf{SS}.\mathsf{Share}(\mu_0))_{\mathcal{U}} \approx (\mathsf{SS}.\mathsf{Share}(\mu_1))_{\mathcal{U}},$$

where $\approx$ is one of the following.

- *If $\approx$ is $\overset{c}{\approx}$, we say that* SS *has* computational privacy*.*
- *If $\approx$ is $\overset{\Delta}{\approx}$, we say that* SS *has* statistical privacy*.*
- *If $\approx$ is $\triangleq$, we say that* SS *has* perfect privacy*.*

A scheme SS is a *secret sharing scheme* if it has both the *correctness* and the *privacy*.

Finally, we can argue about the space efficiency of a secret sharing scheme.

Definition 2.10 (Information rate)**.** *For a scheme* SS*, let* $s := \max_{i\in[n]} \log \#\mathcal{S}_i$*. Notice that* $s = s(|\mu|, \lambda)$*, since* $\#\mathcal{S}_i$ *depends on both the size* $|\mu|$ *of the message and the security parameter* λ*. The* rate *of the secret sharing scheme* SS *is the quantity*

$$\mathsf{SS}.\mathsf{Rate} := \inf_{\lambda\in\mathbb{N}} \frac{|\mu|}{s(|\mu|, \lambda)}.$$

The asymptotic rate *of the secret sharing scheme* SS *is the quantity*

$$\mathsf{SS}.\mathsf{AsymptoticRate} := \inf_{\lambda\in\mathbb{N}} \lim_{|\mu|\to+\infty} \frac{|\mu|}{s(|\mu|, \lambda)}.$$

Intuitively, the *rate* of the secret sharing scheme captures the proportion between the size of the message and the size of a share, while the *asymptotic rate* captures how this rate behaves when messages are very large.

Information Dispersal. When the privacy property is not needed anymore, things can be more space-efficient.

An *information dispersal* scheme ID consists of the following algorithms.

- Randomized algorithm $\mathsf{ID}.\mathsf{Share}$: upon input a message $\mu \in \mathcal{M}$, output shares $(\sigma_1, \dots, \sigma_n) \in \mathcal{S}_1 \times \dots \times \mathcal{S}_n$.
- Deterministic algorithm $\mathsf{ID}.\mathsf{Reconstruct}$: upon input $(\mathcal{I}, (\sigma_i)_{i\in\mathcal{I}})$, where $\mathcal{I} \in \mathcal{A}$ and, for all $i \in \mathcal{I}$, $\sigma_i \in \mathcal{S}_i$, output a message $\mu \in \mathcal{M} \cup \{\perp\}$, where $\perp$ is a special symbol denoting that the reconstruction failed due to invalid shares.

Furthermore, ID satisfies the same *correctness* property of a secret sharing scheme.

2.3 Leakage and tampering attacks

The main take of Section 1.2 is that side-channel attacks can be devastating, both via hardware and via software. Nonetheless, due to their nature, formalizing such attacks can be quite challenging. In this section, we give an overview of the main leakage and tampering models in the context of secret sharing.

Partitioning. Leakage resilience and non-malleability in the context of secret sharing usually come in two flavours. In the case of independent leakage or tampering, each share is considered independently, and no partition of the shares is needed. In the stronger case of *joint* leakage or tampering, the adversary partitions the set of the shares into non-overlapping subsets and operates jointly on the shares within each subset.

Definition 2.11 (partition)**.** *Let* $k, m, n \in \mathbb{N}$ *be parameters and let* $\mathcal{B} = (\mathcal{B}_1, \dots, \mathcal{B}_m)$ *be a tuple of subsets of* $[n]$*. We say that* $\mathcal{B}$ *is a* k-sized partition *if*

- $\bigcup_{i\in[m]} \mathcal{B}_i = [n]$*, i.e. the partition covers all* $[n]$*;*
- $\forall i_1, i_2 \in [m]$ *such that* $i_1 \neq i_2$*,* $\mathcal{B}_{i_1} \cap \mathcal{B}_{i_2} = \emptyset$*, i.e. the subsets are pairwise disjoint;*

- *$\forall i \in [m]$, $\#\mathcal{B}_i \leq k$, i.e. each subset has at most k elements.*

We denote the set of the k-sized partitions of $[n]$ as $\mathfrak{B}_n(k)$.

The case of joint tampering can be further divided depending on *when* this partitioning happens. The simplest model is the one in which the partition is fixed at the beginning of the security experiment and the adversary only uses the established partition. Another model allows the adversary to choose a possibly different partition for each query, in an adaptive fashion.

Finally, we introduce the notion of *semi-adaptive* partitioning, which is between the two notions of selective and adaptive partitioning. In this model, we distinguish between two types of partitions, namely L and T, and two tables $\mathtt{Part}_\mathsf{L}$ and $\mathtt{Part}_\mathsf{T}$ are kept. The tables are initially empty; however, every time the adversary leaks from or tampers with the shares, the tables are updated. More in detail, every time the adversary leaks from the shares using partition $\mathcal{B} = (\mathcal{B}_1, \ldots, \mathcal{B}_m)$, the table $\mathtt{Part}_\mathsf{L}$ is updated by adding in it the sets $\mathcal{B}_i$ that are missing. Similarly, every time the adversary tampers with the shares, the table $\mathtt{Part}_\mathsf{T}$ is updated instead. Then, the adversary is constrained to maintain the following invariant.

- For all $\mathcal{B}_1, \mathcal{B}_2 \in \mathtt{Part}_\mathsf{T}$, either $\mathcal{B}_1 = \mathcal{B}_2$ or $\mathcal{B}_1 \cap \mathcal{B}_2 = \emptyset$.
- For all $\mathcal{B}_\mathsf{T} \in \mathtt{Part}_\mathsf{T}, \mathcal{B}_\mathsf{L} \in \mathtt{Part}_\mathsf{L}$, either $\mathcal{B}_\mathsf{L} \subseteq \mathcal{B}_\mathsf{T}$ or $\mathcal{B}_\mathsf{L} \cap \mathcal{B}_\mathsf{T} = \emptyset$.

Intuitively, this allows the adversary to "construct", step-by-step, a selective partition for tampering queries, while retaining the ability to perform adaptive leakage from any share that is not tampered.

An adversary $\mathcal{A}_\mathcal{P}$ following a partitioning rule $\mathcal{P}$ is said to be *$\mathcal{P}$-partition admissible*, or *admissible* if $\mathcal{P}$ is given from the context.

Bounded leakage resilience. As we said, a secret sharing scheme is leakage-resilient when, even after learning some information from possibly all the shares, still the adversary is not able to learn anything about the shared secret. However, how do we measure such information?

The simplest way to do it is to model leakage as a binary string $\Lambda \in \{0,1\}^*$, and allowing the adversary to only get strings up to a certain length ℓ. This is the idea behind the *bounded leakage* model, which we formalize with the following oracle.

Let $n \in \mathbb{N}$ and let $\mathcal{S} := \mathcal{S}_1 \times \ldots \times \mathcal{S}_n$ be a share space for n shares. For a partition $\mathcal{B} = (\mathcal{B}_1, \ldots, \mathcal{B}_m)$ of $[n]$, we define the following set.

$$\mathfrak{L}_\mathcal{B} := \left\{ g = (g_1, \ldots, g_m) \,\middle|\, \forall i \in [m], \exists \ell_i \in \mathbb{N} : g_i : \mathcal{S}_{\mathcal{B}_i} \to \{0,1\}^{\ell_i} \right\}.$$

Informally, this is the set of all the bounded leakage functions for n parties using the partition $\mathcal{B}$.

```
Oracle: $\mathcal{O}_{\mathsf{BL}}$
Input:  $(\sigma, \mathcal{B}, g) \in \mathcal{S} \times \mathfrak{B}_n(k) \times \mathfrak{L}_{\mathcal{B}}$
        $(\sigma_1, \dots, \sigma_n) := \sigma$          // Parse $\sigma$ into the tuple of shares.
        $(\mathcal{B}_1, \dots, \mathcal{B}_m) := \mathcal{B}$     // Parse the partition $\mathcal{B}$.
        $(g_1, \dots, g_m) := g$          // Parse the function tuple $g$.
        $\forall\ i \in [m]$ :
            $\Lambda_i \leftarrow g_i(\sigma_{\mathcal{B}_i})$        // Compute the leakage.
        $\Lambda = (\Lambda_1, \dots, \Lambda_m)$     // Compose the leakage string.
Output: $\Lambda$
```

Any algorithm making queries to $\mathcal{O}_{\mathsf{BL}}$ and obtaining an overall string of length up to ℓ is called *ℓ-leakage admissible* (in the bounded leakage model), or *admissible* if ℓ is given from the context.

Now we just need to define the leakage experiment. Let $\mu_0, \mu_1 \in \mathcal{M}$ be two messages, let $\mathcal{P} \in \{\mathtt{Selective}, \mathtt{SemiAdaptive}, \mathtt{Adaptive}\}$ be a partitioning and let A be a $\mathcal{P}$-partitioning ℓ-leakage admissible adversary with oracle access to $\mathcal{O}_{\mathsf{BL}}(\sigma, \cdot, \cdot)$, where σ is given by the following experiment.

```
Experiment: $\mathbf{Leak}_{\mathsf{A}}^{\mathsf{Bounded}}[\lambda, \mu_0, \mu_1, b]$
            $\sigma \leftarrow\!\!\$\ \mathsf{SS}.\mathsf{Share}(\mu_b)$
            $b^* \leftarrow\!\!\$\ \mathsf{A}^{\mathcal{O}_{\mathsf{BL}}(\sigma,\cdot,\cdot)}(1^\lambda)$
Output:     $b^*$
```

We are finally ready for the formal definition.

Definition 2.12. *Let $\lambda, n, \ell \in \mathbb{N}$ be parameters. Let* SS *be a secret sharing scheme for n parties. We say that* SS *is* ℓ-bounded leakage-resilient *under partitioning rule $\mathcal{P}$ if, for all admissible adversaries* A*,*

$$\mathbf{Leak}_{\mathsf{A}}^{\mathsf{Bounded}}[\lambda, \mu_0, \mu_1, 0] \approx \mathbf{Leak}_{\mathsf{A}}^{\mathsf{Bounded}}[\lambda, \mu_0, \mu_1, 1],$$

where $\approx$ is one of the following.

- *If $\approx$ is $\overset{c}{\approx}$ and* A *is PPT, we say that* SS *has* computational leakage-resilience.
- *If $\approx$ is $\overset{\Delta}{\approx}$, we say that* SS *has* statistical leakage-resilience.

Furthermore, when needed, we denote by $\mathsf{SS}.\mathsf{Partitioning} = \mathcal{P}$ *the fact that* A *is $\mathcal{P}$-partitioning admissible. Finally, we denote by* $\mathsf{SS}.\mathsf{LeakageModel} = \mathtt{Bounded}(\ell)$ *the fact that* $\mathsf{A}^{\mathcal{O}_{\mathsf{BL}}}$ *is ℓ-leakage admissible or, equivalently,* SS *supports up to ℓ bits of* bounded *leakage.*

Noisy leakage resilience. A more complex, but more realistic and powerful, way to measure leakage is through the notion of conditional average min-entropy. Let $\boldsymbol{X}$ be a random variable and let $\boldsymbol{\Lambda}$ be some information on $\boldsymbol{X}$. We say that $\boldsymbol{\Lambda}$ is *ℓ-noisy-leakage* of $\boldsymbol{X}$ if

$$\widetilde{\mathbb{H}}_\infty(\boldsymbol{X}|\boldsymbol{\Lambda}) \geq \mathbb{H}_\infty(\boldsymbol{X}) - \ell.$$

Intuitively, this notion captures the fact that the leakage could be a large chunk of information which is "noisy" and, therefore, contains small effective information about the leaked value.

Let $n \in \mathbb{N}$ and let $\mathcal{S} := \mathcal{S}_1 \times \ldots \times \mathcal{S}_n$ be a share space for n shares. For a partition $\mathcal{B} = (\mathcal{B}_1, \ldots, \mathcal{B}_m)$ of $[n]$, we define the following set.

$$\mathfrak{G}_{\mathcal{B}} := \{g = (g_1, \ldots, g_m) \,|\, \forall i \in [m], g_i : \mathcal{S}_{\mathcal{B}_i} \to \{0,1\}^*\}\,.$$

Informally, this is the set of all the binary functions for n parties using the partition $\mathcal{B}$. We now define the oracle for the noisy leakage. In this case, we stick with selective partitioning since (1) our results achieving or requiring noisy leakage are against selective partitioning and (2) defining noisy leakage against adaptive partitioning poses a new set of challenges which is beyond the scope of this thesis.

```
Oracle: $\mathcal{O}_{\mathsf{NL}}$
Input:  $(\sigma, \mathcal{B}, g) \in \mathcal{S} \times \mathfrak{B}_n(k) \times \mathfrak{G}_{\mathcal{B}}$
        $(\sigma_1, \ldots, \sigma_n) := \sigma$              // Parse $\sigma$ into the tuple of shares.
        $(\mathcal{B}_1, \ldots, \mathcal{B}_m) := \mathcal{B}$   // Parse the partition $\mathcal{B}$.
        $(g_1, \ldots, g_m) := g$            // Parse the function tuple $g$.
        $\forall\ i \in [m]$ :
            $\Lambda_i \leftarrow g_i(\sigma_{\mathcal{B}_i})$          // Compute the leakage.
        $\Lambda = (\Lambda_1, \ldots, \Lambda_m)$         // Compose the leakage string.
Output: $\Lambda$
```

Let A be an algorithm with oracle access to $\mathcal{O}_{\mathsf{NL}}$ and let, for $i \in [m]$, $\mathbf{\Lambda}_i^*$ be the random variable of the total leakage obtained by A from subset $\mathcal{B}_i$. If A is such that

$$\forall i \in [m], \widetilde{\mathbb{H}}_\infty\left(\mathbf{\Sigma}_{\mathcal{B}_i} | \mathbf{\Lambda}_i^*\right) \geq \mathbb{H}_\infty\left(\mathbf{\Sigma}_{\mathcal{B}_i}\right) - \ell,$$

we say that A is *ℓ-leakage admissible* (in the noisy leakage model) or *admissible* if ℓ is given from the context.

Since an algorithm is never given access to $\mathcal{O}_{\mathsf{BL}}$ and $\mathcal{O}_{\mathsf{NL}}$ at the same time, the notion of admissible adversary should not cause any ambiguity.

By replacing $\mathbf{Leak}^{\mathsf{Bounded}}$ with $\mathbf{Leak}^{\mathsf{Noisy}}$, $\mathsf{SS.LeakageModel} = \mathtt{Bounded}(\ell)$ with $\mathsf{SS.LeakageModel} = \mathtt{Noisy}(\ell)$ and $\mathcal{O}_{\mathsf{BL}}$ with $\mathcal{O}_{\mathsf{NL}}$ in Definition 2.12, we get automatically the definition of *ℓ-noisy leakage-resilience*.

Non-malleability. At this point, modelling tampering should be a bit more straightforward. First of all, we establish the function family. This time, since the tampering functions need to produce shares, they are of the form $f_i : \mathcal{S}_{\mathcal{B}_i} \to \mathcal{S}_{\mathcal{B}_i}$.

Let $n \in \mathbb{N}$, let $\mathcal{M}$ be the message space and let $\mathcal{S} := \mathcal{S}_1 \times \ldots \times \mathcal{S}_n$ be a share space for n shares. For a partition $\mathcal{B} = (\mathcal{B}_1, \ldots, \mathcal{B}_m)$ of $[n]$, we define the following set.

$$\mathfrak{T}_{\mathcal{B}} := \{f = (f_1, \ldots, f_m) \,|\, \forall i \in [m], f_i : \mathcal{S}_{\mathcal{B}_i} \to \mathcal{S}_{\mathcal{B}_i}\}\,.$$

All the schemes in this thesis have a self-destruct feature and can resist against a number $p \in \mathbb{N} \cup \{\infty\}$ of tampering queries, where ∞ means that no bound is set except for the one implicitly posed by the computational limitations of the attacker. Therefore, our tampering oracle is defined as follows.

Oracle: $\mathcal{O}^p_{\mathsf{T}}$
Input: $((\mu_0,\mu_1),\sigma,\mathcal{T},\mathcal{B},f) \in \mathcal{M}^2 \times \mathcal{S} \times \mathcal{A} \times \mathfrak{B}_n(k) \times \mathfrak{T}_{\mathcal{B}}$
Initial state: $p^* := 0, \mathsf{SD} := 0$

```
IF $\mathsf{SD} = 1$ :                                  // If a self-destruction
    RETURN $\perp$                                    // occurred, return $\perp$.
IF $p^* = p$ :                                        // If query limit has been
    RETURN $\perp$                                    // reached, return $\perp$.
$p^* \leftarrow p^* + 1$                              // Count the queries.
$(\sigma_1,\ldots,\sigma_n) := \sigma$                // Parse $\sigma$.
$(\mathcal{B}_1,\ldots,\mathcal{B}_m) := \mathcal{B}$ // Parse the partition $\mathcal{B}$.
$(f_1,\ldots,f_m) := f$                               // Parse the function tuple $f$.
$\forall\ i \in [m]$ :
    $\tilde{\sigma}_{\mathcal{B}_i} \leftarrow f_i(\sigma_{\mathcal{B}_i})$   // Compute the tampering.
$\tilde{\mu} \leftarrow \mathsf{SS}.\mathsf{Reconstruct}(\mathcal{T},\tilde{\sigma}_{\mathcal{T}})$   // Reconstruct.
IF $\tilde{\mu} = \perp$ :                            // If the reconstruction is
    $\mathsf{SD} \leftarrow 1$                        // invalid, self-destruct.
IF $\tilde{\mu} \in \{\mu_0,\mu_1\}$ :                // If the original message is
    $\tilde{\mu} \leftarrow \heartsuit$               // reconstructed, hide it.
```

Output: $\tilde{\mu}$

Let $p \in \mathbb{N} \cup \{\infty\}$ be a parameter, $\mu_0, \mu_1 \in \mathcal{M}$ be two messages, let $\mathcal{P} \in \{\mathtt{Selective}, \mathtt{SemiAdaptive}, \mathtt{Adaptive}\}$ be a partitioning and let A be an admissible adversary with oracle access to $\mathcal{O}^p_{\mathsf{T}}((\mu_0,\mu_1),\sigma,\cdot,\cdot,\cdot)$, where σ is given by the following experiment.

Experiment: $\mathbf{Tamper}_{\mathsf{A}}[\lambda,p,\mu_0,\mu_1,b]$

```
$\sigma \leftarrow\!\!\$\ \mathsf{SS}.\mathsf{Share}(\mu_b)$
$b^* \leftarrow\!\!\$\ \mathsf{A}^{\mathcal{O}^p_{\mathsf{T}}((\mu_0,\mu_1),\sigma,\cdot,\cdot,\cdot)}(1^\lambda)$
```

Output: b^*

Definition 2.13. *Let $\lambda, n \in \mathbb{N}, p \in \mathbb{N} \cup \{\infty\}$ be parameters. Let SS be a secret sharing scheme for n parties. We say that SS is p-*time non-malleable*, or* continuously non-malleable *if $p = \infty$, under partitioning rule $\mathcal{P}$ if, for all admissible adversaries A,*

$$\mathbf{Tamper}_{\mathsf{A}}[\lambda,p,\mu_0,\mu_1,0] \approx \mathbf{Tamper}_{\mathsf{A}}[\lambda,p,\mu_0,\mu_1,1]\,,$$

where $\approx$ is one of the following.

- *If $\approx$ is $\overset{c}{\approx}$ and A is PPT, we say that SS has* computational non-malleability*.*
- *If $\approx$ is $\overset{\Delta}{\approx}$, we say that SS has* statistical non-malleability*.*

Finally, several schemes achieve both leakage-resilience and non-malleability. By granting either $\mathcal{O}_{\mathsf{BL}}$ or $\mathcal{O}_{\mathsf{NL}}$ oracle access to A in Definition 2.13, we get the notion of *leakage-resilient non-malleability*, with all the different flavours given by mixing the various parameters. More formally, we get the experiments

```
**Experiment:** $\mathbf{Tamper}_{\mathsf{A}}^{\mathsf{BL}}[\lambda, p, \mu_0, \mu_1, b]$
    $\sigma \leftarrow\!\!\$\; \mathsf{SS}.\mathsf{Share}(\mu_b)$
    $b^* \leftarrow\!\!\$\; \mathsf{A}^{\mathcal{O}_{\mathsf{T}}^{p}((\mu_0,\mu_1),\sigma,\cdot,\cdot,\cdot),\mathcal{O}_{\mathsf{BL}}(\sigma,\cdot,\cdot)}(1^\lambda)$
**Output:** $b^*$
```

```
**Experiment:** $\mathbf{Tamper}_{\mathsf{A}}^{\mathsf{NL}}[\lambda, p, \mu_0, \mu_1, b]$
    $\sigma \leftarrow\!\!\$\; \mathsf{SS}.\mathsf{Share}(\mu_b)$
    $b^* \leftarrow\!\!\$\; \mathsf{A}^{\mathcal{O}_{\mathsf{T}}^{p}((\mu_0,\mu_1),\sigma,\cdot,\cdot,\cdot),\mathcal{O}_{\mathsf{NL}}(\sigma,\cdot,\cdot)}(1^\lambda)$
**Output:** $b^*$
```

In both cases, the tampering oracle $\mathcal{O}_{\mathsf{T}}^{p}$ and the leakage oracle $\mathcal{O}_{\mathsf{BL}}, \mathcal{O}_{\mathsf{NL}}$ share the internal state p^*, SD. Whenever the tampering oracle cannot answer anymore (for instance, because a self-destruction occurred or the maximum number of queries has been reached), the leakage oracle stops answering as well. This also means that, if $p \neq \infty$ and the adversary makes p tampering queries, no leakage can happen after the last tampering query.

For $p = 1$, it actually makes sense to split the adversary into two parts, namely, the *leakage* phase and the *tampering* phase. Therefore, for $p = 1$ and $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$ and for all $\mathsf{L} \in \{\mathsf{BL}, \mathsf{NL}\}$, the above experiments are equivalent to the following.

```
**Experiment:** $\mathbf{Tamper}_{\mathsf{A}}^{\mathsf{L}}[\lambda, 1, \mu_0, \mu_1, b]$
    $\sigma \leftarrow\!\!\$\; \mathsf{SS}.\mathsf{Share}(\mu_b)$
    $(\alpha; \mathcal{T}, \mathcal{B}, f) \leftarrow\!\!\$\; \mathsf{A}_1^{\mathcal{O}_{\mathsf{L}}(\sigma,\cdot,\cdot)}(1^\lambda)$
    $\tilde{\mu} \leftarrow\!\!\$\; \mathcal{O}_{\mathsf{T}}^{1}((\mu_0, \mu_1), \sigma, \mathcal{T}, \mathcal{B}, f)$
    $b^* \leftarrow\!\!\$\; \mathsf{A}_2(\alpha, \tilde{\mu})$
**Output:** $b^*$
```

Augmented leakage resilience. Finally, we also define a seemingly stronger variant of leakage-resilient secret sharing, in which the adversary A is allowed to obtain the shares within a subset of an admissible partition $\mathcal{B}$ at the end of the experiment. In particular, in the case of selective partitioning, an *augmented* admissible adversary is an attacker $\mathsf{A}^+ := (\mathsf{A}_1^+, \mathsf{A}_2^+)$ such that:

- A_1^+ is an admissible adversary in the sense of Definition 2.12 (or its variant in the noisy leakage model), the only difference being that A_1^+ outputs a tuple $(\alpha; \mathcal{B}, i^*)$ where α is an auxiliary state, $\mathcal{B}$ is an admissible partition and i^* is the index of a subset $\mathcal{B}_{i^*}$ of $\mathcal{B}$;

- A_2^+ takes as input the state α and all the shares $\sigma_{\mathcal{B}_{i^*}}$ and outputs a decision bit.

This flavour of security is called *augmented leakage resilience*. The theorem below, established by [BFV19, KMS19] for the case of independent leakage, shows that any leakage-resilient secret sharing scheme secure against joint leakage is also an augmented leakage-resilient secret sharing scheme at the cost of an extra bit of leakage.

Theorem 2.14. *Let $n, k, \ell, \lambda \in \mathbb{N}, \varepsilon \in [0,1]$ be parameters, let $\mathcal{A}$ be an access structure over n parties and let $\mathcal{P}$ be a partitioning strategy. Let* SS *be a k-joint $(\ell+1)$-leakage resilient secret sharing scheme realizing access structure $\mathcal{A}$ under partitioning $\mathcal{P}$ with security $\varepsilon = \varepsilon(\lambda)$. Then* SS *is an augmented k-joint ℓ-leakage resilient secret sharing scheme realizing the same access structure $\mathcal{A}$, under the same partitioning $\mathcal{P}$ and with the same security ε.*

Proof. By reduction to the non-augmented leakage. Suppose towards contradiction that there exist two distinct messages $\mu_0, \mu_1 \in \mathcal{M}$ and an adversary $\mathsf{A}^+ = (\mathsf{A}_1^+, \mathsf{A}_2^+)$ which is able to cause a statistical difference of more than ε between the experiments $\mathbf{Leak}^+_{\mathsf{A}^+}[\lambda, \mu_0, \mu_1, 0]$ and $\mathbf{Leak}^+_{\mathsf{A}^+}[\lambda, \mu_0, \mu_1, 1]$, where the experiment $\mathbf{Leak}^+$ is the same as **Leak** except that it additionally supports the augmented property.

Consider the following reduction R.

1. Run $(\alpha; \mathcal{B}, i^*) \leftarrow \mathsf{A}_1^{+,\mathsf{R}}$, forwarding each leakage query to the actual oracle and returning back the answer.
2. Parse $\mathcal{B} = (\mathcal{B}_1, \dots, \mathcal{B}_m)$.
3. Let $g_i(\cdot) = \mathsf{A}_2^+(\alpha; \cdot)$ and, for $i \in [m]$ such that $i \neq i^*$, let g_i be the function that returns the empty string.
4. Query $b \leftarrow \mathcal{O}_{\mathsf{L}}(\mathcal{B}, g)$, where $g = (g_1, \dots, g_m)$.
5. Output b.

Intuitively, what R does is to use A_2^+ as a leakage function returning one bit and then outputting the same bit. For the analysis, the reduction is perfect and R outputs the same distinguishing bit of A^+, thus retaining the same advantage of A^+. Furthermore, R just uses an additional leakage query leaking one bit, hence is $(\ell+1)$-admissible. Finally, R uses the same computational resources of A^+. In particular, if A^+ is efficient, the leakage query $(\mathcal{B}, g)$ is efficiently computable and, therefore, R is efficient. This concludes the proof. $\square$

Chapter 3

Achieving security against stronger adversaries

In this chapter, we study non-malleability in the context of stronger adversaries.

In Section 3.1, we focus on the attack model, and we construct a non-malleable secret sharing scheme that is secure against an adversary that is able to jointly leak from and tamper with the shares. In this model, the partition of the shares can be chosen by the adversary in an adaptive, but restricted, way.

In Section 3.2, we show how to extend the above result in the case of multiple tampering attempts. Namely, we show a technique transforming a generic one-time non-malleable secret sharing scheme into a secret sharing scheme that supports multiple tampering attempts that are bounded *a priori*. Unfortunately, this technique alone is not able to produce a continuously non-malleable secret sharing scheme.

Finally, in Section 3.3, we use a workaround for the above technique and show the first continuously non-malleable secret sharing scheme which is secure against joint partitioning. Furthermore, our construction also achieves leakage-resilience.

3.1 Non-malleability against semi-adaptive partitioning

In this section, we construct a leakage-resilient non-malleable secret sharing scheme which is able to withstand one tampering attempt and many leakage queries. Our scheme is secure in the information-theoretic model (i.e. without computational assumptions) and in the model of semi-adaptive partitioning.

Informally, the idea behind our construction is to first share the message μ into two non-malleable shares (σ_0, σ_1), and then share each σ_i into the desired number of leakage-resilient shares. Then, the shares (σ_0, σ_1) are protected from leakage attempts by the outer leakage-resilient schemes. Furthermore, the two leakage-resilient secret sharing schemes have different access structure, so that the tampering attack can be "split" into two subsets of shares that affect σ_0 and σ_1 independently, thus allowing us to reduce to the security of the non-malleable scheme.

The security analysis of this construction is quite technical. We show the details of our construction in Section 3.1.1, then we give an overview of the security proof in Section 3.1.2 and we give the actual security analysis in Section 3.1.3. Finally, the details on the instantiation can be found in Section 3.1.4.

```
Algorithm: $\mathsf{SS}.\mathsf{Share}$
    Input: $\mu \in \mathcal{M}$
           $(\hat{\sigma}_0, \hat{\sigma}_1) \leftarrow\!\!\$\; \mathsf{SS}_2.\mathsf{Share}(\mu)$
           $(\sigma_{0,1}, \dots, \sigma_{0,n}) \leftarrow\!\!\$\; \mathsf{SS}_0.\mathsf{Share}(\hat{\sigma}_0)$
           $(\sigma_{1,1}, \dots, \sigma_{1,n}) \leftarrow\!\!\$\; \mathsf{SS}_1.\mathsf{Share}(\hat{\sigma}_1)$
           $\forall\ i \in [n]$ :
              $\sigma_i := (\sigma_{0,i}, \sigma_{1,i})$
   Output: $(\sigma_1, \dots, \sigma_n)$

Algorithm: $\mathsf{SS}.\mathsf{Reconstruct}$
    Input: $(\mathcal{I}, \sigma_{\mathcal{I}}) \in \mathcal{A} \times \mathcal{S}$
           $\forall\ i \in \mathcal{I}$ :
              $(\sigma_{0,i}, \sigma_{1,i}) := \sigma_i$
           $\{i_1, \dots, i_{\#\mathcal{I}}\} := \mathcal{I}$
           $\mathcal{J}_{k_1} := \{i_1, \dots, i_{k_1}\}$
           $\hat{\sigma}_0 \leftarrow \mathsf{SS}_0.\mathsf{Reconstruct}(\mathcal{I}, \sigma_{0,\mathcal{I}})$
           $\hat{\sigma}_1 \leftarrow \mathsf{SS}_1.\mathsf{Reconstruct}\left(\mathcal{J}_{k_1}, \sigma_{1,\mathcal{J}_{k_1}}\right)$
           $\mu \leftarrow \mathsf{SS}_2.\mathsf{Reconstruct}([2], (\hat{\sigma}_0, \hat{\sigma}_1))$
   Output: $\mu$
```

Figure 3.1. The Share and Reconstruct algorithms of our construction.

3.1.1 Our construction

Let $\lambda, n \in \mathbb{N}$ be respectively the security parameter and the number of parties. Let $\mathcal{M}$ the set of possible messages. First, we need a 2-out-of-2 one-time non-malleable secret sharing scheme[1] SS_2 to share the message $\mu \in \mathcal{M}$. Then, we need to share each output of SS_2 in n parts. The full algorithm is depicted in Fig. 3.1.

For what concerns the parameters of our scheme, let $\mathcal{A}$ be an access structure for n parties. Let $s_0 = s_0(\lambda) \in \mathbb{N}$. Let $\ell = \ell(\lambda) \in \mathbb{N}$ and $\ell_0 := \ell + 1, \ell_1 := \ell + n \cdot s_0$ be leakage parameters. Let $k = k(\lambda) \in \mathbb{N}$ and let $k_1 := \lfloor\sqrt{k}\rfloor$. Finally, let $\mathcal{S}_{0,1} \times \ldots \times \mathcal{S}_{0,n}$ be a share space such that, for all $i \in [n]$, $\log \#\mathcal{S}_{0,i} \leq s_0$.

We obtain the following.

Theorem 3.1. *Let $n, k, s_0, \ell \in \mathbb{N}, \varepsilon_0, \varepsilon_1, \varepsilon_2 \in [0,1]$ be parameters and let $\mathcal{A}$ be an access structure over n parties. Let $\ell_0 := \ell + 1$, $\ell_1 := \ell + n \cdot s_0$, $k_1 := \lfloor\sqrt{k}\rfloor$, $\varepsilon := 2(\varepsilon_0 + \varepsilon_1) + \varepsilon_2$. Finally, let $\mathcal{S}_{0,1} \times \ldots \times \mathcal{S}_{0,n}$ be a share space such that, for all $i \in [n]$, $\log \#\mathcal{S}_{0,i} \leq s_0$. In the construction depicted in Fig. 3.1, assume that*

- SS_2 *is a 2-out-of-2 one-time non-malleable secret sharing scheme with statistical security ε_2, message space $\mathcal{M}$ and share space $\hat{\mathcal{S}}_0 \times \hat{\mathcal{S}}_1$;*
- SS_0 *is a ℓ_0-bounded leakage resilient secret sharing scheme for access structure $\mathcal{A}$ over $[n]$ parties, with statistical security ε_0 against k-sized adaptive partitioning, message space $\hat{\mathcal{S}}_0$ and share space $\mathcal{S}_{0,1} \times \ldots \times \mathcal{S}_{0,n}$;*

[1] Or, a 2-split-state one-time non-malleable code [ADKO15b].

- SS_1 *is a* ℓ_1*-bounded leakage resilient secret sharing scheme for* k_1*-out-of-*n *access structures, with statistical security* ε_1 *against* (k_1-1)*-sized adaptive partitioning, message space* $\hat{\mathcal{S}}_1$ *and share space* $\mathcal{S}_{1,1} \times \ldots \times \mathcal{S}_{1,n}$.

Then, SS *is a* ℓ*-bounded leakage resilient one-time non-malleable secret sharing scheme for access structure* $\mathcal{A}$ *over* $[n]$ *parties, with statistical security* ε *against* (k_1-1)*-sized semi-adaptive partitioning.*

Remark 3.2. *In the above theorem, the scheme* SS *has the same access structure of* SS_0 *and must contain the access structure of* SS_1 *as well in order to guarantee reconstruction for all the authorized subsets; however, the latter condition does not appear in the theorem and, indeed, we can show that is always satisfied. Notice that* SS_0 *is secure against* k*-sized partitioning, which means that every set* $\mathcal{I} \in \mathcal{A}$ *is such that* $\#\mathcal{I} > k$ *(otherwise, the adversary could choose a partition with one of the sets being authorized). Since the reconstruction threshold for* SS_1 *is* $k_1 = \lfloor\sqrt{k}\rfloor \leq k$*, this means that every* $\mathcal{I} \in \mathcal{A}$ *is also authorized for* SS_1.

3.1.2 Proof overview.

In order to prove Theorem 3.1, we first make some considerations on the tampering query $(\mathcal{T}, \mathcal{B}, f)$ performed by the adversary at the end of the experiment. In particular, we construct two disjoint sets $\mathcal{T}_0^*, \mathcal{T}_1^*$ that are the union of subsets from the partition $\mathcal{B}$. More in detail, we want that the following holds.

1. $\mathcal{T}_0^* \cap \mathcal{T}$ contains at least k_1 elements, so that it can be used as a reconstruction set for SS_1.
2. Each subset $\mathcal{B}_i$ of the partition $\mathcal{B}$ intersects at most one between $\mathcal{T}_0^*$ and $\mathcal{T}_1^*$, so that both leakage and tampering queries can be computed independently on $\mathcal{T}_0^*$ and $\mathcal{T}_1^*$.

After constructing these sets, we answer the leakage and tampering queries, in the original experiment, as follows.

$$\text{Leakage queries:} \quad \left(\begin{pmatrix} \sigma_{0,i} \\ \sigma_{1,i} \end{pmatrix}_{i \in \mathcal{T}_0^*}, \begin{pmatrix} \sigma_{0,i} \\ \sigma_{1,i} \end{pmatrix}_{i \in \mathcal{T}_1^*} \right)$$
$$\text{Tampering queries:} \quad \left(\begin{pmatrix} \sigma_{0,i} \\ \sigma_{1,i} \end{pmatrix}_{i \in \mathcal{T}_0^*}, \begin{pmatrix} \sigma_{0,i} \\ \sigma_{1,i} \end{pmatrix}_{i \in \mathcal{T}_1^*} \right)$$

Then, we proceed by hybrid arguments and, in particular, we construct the following hybrid experiments.

First Hybrid: In the first hybrid experiment, we change how the tampering query is answered. Namely, after the leakage phase we replace all the shares $(\sigma_{0,r})_{r \in \mathcal{T}_1^*}$ with shares $(\sigma^*_{0,r})_{r \in \mathcal{T}_1^*}$ that are valid shares of the same message $\hat{\sigma}_0$ and are consistent with the leakage. The situation becomes the following.

$$\begin{array}{rl} \text{Leakage queries:} & \left(\begin{pmatrix} \sigma_{0,i} \\ \sigma_{1,i} \end{pmatrix}_{i \in \mathcal{T}_0^*}, \begin{pmatrix} \sigma_{0,i} \\ \sigma_{1,i} \end{pmatrix}_{i \in \mathcal{T}_1^*} \right) \\ \text{Tampering queries:} & \left(\begin{pmatrix} \sigma_{0,i} \\ \sigma_{1,i} \end{pmatrix}_{i \in \mathcal{T}_0^*}, \begin{pmatrix} \sigma^*_{0,i} \\ \sigma_{1,i} \end{pmatrix}_{i \in \mathcal{T}_1^*} \right) \end{array}$$

This change does not affect the view of the adversary, since both $(\sigma_{0,r})_{r\in\mathcal{T}_1^*}$ and $(\sigma^*_{0,r})_{r\in\mathcal{T}_1^*}$ come from the same distribution and the only thing that changes is when they are sampled.

Second Hybrid: In the second hybrid experiment, we replace all the shares $(\sigma_{0,i})_{i\in[n]}$ with "dummy" shares $(\sigma'_{0,i})_{i\in[n]}$ of something else. The situation becomes the following.

$$\begin{array}{rl} \text{Leakage queries:} & \left(\begin{pmatrix} \sigma'_{0,i} \\ \sigma_{1,i} \end{pmatrix}_{i \in \mathcal{T}_0^*}, \begin{pmatrix} \sigma'_{0,i} \\ \sigma_{1,i} \end{pmatrix}_{i \in \mathcal{T}_1^*} \right) \\ \text{Tampering queries:} & \left(\begin{pmatrix} \sigma'_{0,i} \\ \sigma_{1,i} \end{pmatrix}_{i \in \mathcal{T}_0^*}, \begin{pmatrix} \sigma^*_{0,i} \\ \sigma_{1,i} \end{pmatrix}_{i \in \mathcal{T}_1^*} \right) \end{array}$$

A reduction to the leakage-resilience of SS_0 proves that this new experiment is ε_0-close to the previous one. The key idea here is to forward the leakage queries to the respective oracle during the leakage phase and, once the adversary outputs its tampering query, obtain all the shares within $\mathcal{T}_0^*$ thanks to the augmented property. After obtaining such shares, the reduction can sample the remaining shares $(\sigma^*_{0,r})_{r\in\mathcal{T}_1^*}$ as in the previous experiment and can thus compute the tampering query.

Third Hybrid: In the third hybrid experiment, we change again how the tampering query is answered. Namely, after the leakage phase we replace all the shares $(\sigma_{1,r})_{r\in\mathcal{T}_0^*}$ with shares $(\sigma^*_{1,r})_{r\in\mathcal{T}_0^*}$ that are valid shares of message $\hat{\sigma}_1$ and are consistent with the leakage. Furthermore, we require that these new shares do not affect the outcome of the tampering query. The situation becomes the following.

$$\begin{array}{rl} \text{Leakage queries:} & \left(\begin{pmatrix} \sigma'_{0,i} \\ \sigma_{1,i} \end{pmatrix}_{i \in \mathcal{T}_0^*}, \begin{pmatrix} \sigma'_{0,i} \\ \sigma_{1,i} \end{pmatrix}_{i \in \mathcal{T}_1^*} \right) \\ \text{Tampering queries:} & \left(\begin{pmatrix} \sigma'_{0,i} \\ \sigma^*_{1,i} \end{pmatrix}_{i \in \mathcal{T}_0^*}, \begin{pmatrix} \sigma^*_{0,i} \\ \sigma_{1,i} \end{pmatrix}_{i \in \mathcal{T}_1^*} \right) \end{array}$$

This is required in order to make the step to the last hybrid possible. As before, this change does not affect the view of the adversary.

Fourth Hybrid: In the fourth hybrid experiment, we replace again all the shares $(\sigma_{1,i})_{i\in[n]}$ with "dummy" shares $(\sigma'_{1,i})_{i\in[n]}$ of something else. The situation becomes the following.

$$\begin{array}{rl} \text{Leakage queries:} & \left(\begin{pmatrix} \sigma'_{0,i} \\ \sigma'_{1,i} \end{pmatrix}_{i \in \mathcal{T}_0^*}, \begin{pmatrix} \sigma'_{0,i} \\ \sigma'_{1,i} \end{pmatrix}_{i \in \mathcal{T}_1^*} \right) \\ \text{Tampering queries:} & \left(\begin{pmatrix} \sigma'_{0,i} \\ \sigma^*_{1,i} \end{pmatrix}_{i \in \mathcal{T}_0^*}, \begin{pmatrix} \sigma^*_{0,i} \\ \sigma'_{1,i} \end{pmatrix}_{i \in \mathcal{T}_1^*} \right) \end{array}$$

A reduction to the leakage-resilience of SS_1 proves that this new experiment is ε_1-close to the previous one. The key idea here is to simulate the tampering query via a leakage query that yields the result of the tampering on all the shares $(\sigma_{0,i})_{i \in \mathcal{T}_0^* \cup \mathcal{T}_1^*}$. This is allowed thanks to the restriction to the size of the shares of SS_0. After the tampering query, the reduction will still be able to produce the shares $(\sigma^*_{1,r})_{r \in \mathcal{T}_0^*}$: this happens because (1) $\mathcal{T}_0^*$ is authorized for SS_1 and thus there is no need for other shares to check consistency and (2) as we said before, the modification introduced in the third hybrid does not affect the outcome of the tampering query. This allows to compute the tampering query offline and, in turns, to complete the reduction.

Since the above defined hybrid experiments are all statistically close, it only remains to show that the last hybrid with bit 0 is statistically close to the same hybrid with bit 1. The situation of how the queries are computed is now the following.

$$\begin{array}{rl} \text{Leakage queries:} & \left(\begin{pmatrix} \sigma'_{0,i} \\ \sigma'_{1,i} \end{pmatrix}_{i \in \mathcal{T}_0^*}, \begin{pmatrix} \sigma'_{0,i} \\ \sigma'_{1,i} \end{pmatrix}_{i \in \mathcal{T}_1^*} \right) \\ \text{Tampering queries:} & \left(\begin{pmatrix} \sigma'_{0,i} \\ \sigma^*_{1,i} \end{pmatrix}_{i \in \mathcal{T}_0^*}, \begin{pmatrix} \sigma^*_{0,i} \\ \sigma'_{1,i} \end{pmatrix}_{i \in \mathcal{T}_1^*} \right) \end{array}$$

In particular, the shares $\hat{\sigma}_0$ and $\hat{\sigma}_1$ are now tampered independently. Indeed, the tampering on the share $\hat{\sigma}_0$ is computed through $\left(\sigma^*_{0,i}\right)_{i \in \mathcal{T}_1^*}$ (and the "dummy" shares $\left(\sigma'_{0,i}\right)_{i \in \mathcal{T}_1^*}$), while the tampering on the shares $\hat{\sigma}_1$ is computed through $\left(\sigma^*_{1,i}\right)_{i \in \mathcal{T}_0^*}$ (and the "dummy" shares $\left(\sigma'_{1,i}\right)_{i \in \mathcal{T}_1^*}$). Because of this, a straightforward reduction to the non-malleability of SS_2 concludes the security proof.

3.1.3 Security analysis.

Before proceeding with the analysis, we introduce some useful notation.

Recall that, after the leakage phase, the adversary sends a single tampering query $(\mathcal{T}, \mathcal{B}, f)$. Let $t := \#\mathcal{T}$ and let $\mathcal{T} := \{r_1, \ldots, r_t\}$. For $i \in [t]$, consider the function $\beta(\mathcal{T}, \mathcal{B}, i)$ such that $r_i \in \mathcal{B}_{\beta(\mathcal{T}, \mathcal{B}, i)}$. When $\mathcal{T}$ and $\mathcal{B}$ are clear from the context, we simply write $\beta(i)$. In other words, $\beta(i)$ is the index of the function in $(f_1, \ldots, f_m)$ such that the i-th share is tampered by $f_{\beta(i)}$.

Starting from the set $\mathcal{T}$, we define the following subsets. Let $\mathcal{T}_{k_1} := \{r_1, \ldots, r_{k_1}\}$, and let

$$\mathcal{T}_0^* := \bigcup_{r \in \mathcal{T}_{k_1}} \mathcal{B}_{\beta(r)} \qquad \text{and} \qquad \mathcal{T}_0 := \mathcal{T}_0^* \cap \mathcal{T}.$$

Intuitively, $\mathcal{T}_0^*$ is the set of all the shares that are tampered with the first k_1 shares of the reconstruction set $\mathcal{T}$, while $\mathcal{T}_0$ is simply the set of the shares in $\mathcal{T}_0^*$ that are used to reconstruct the message.

For the remaining shares, we can define

$$\mathcal{T}_1 := \mathcal{T} \setminus \mathcal{T}_0 \qquad \text{and} \qquad \mathcal{T}_1^* := \bigcup_{r \in \mathcal{T}_1} \mathcal{B}_{\beta(r)}.$$

In this way, $\mathcal{T}_1$ is the set of the remaining shares in the reconstruction set, and $\mathcal{T}_1^*$ is the set of all the shares that are tampered together with the shares in $\mathcal{T}_1$.

This notation is useful because it allows to partition the set of the shares in a good way. More in detail, the set $\mathcal{T}_0$ has at least k_1 shares (indeed, $\{r_1, \dots, r_{k_1}\} \subseteq \mathcal{T}_0$) and the sets $\mathcal{T}_0^*$ and $\mathcal{T}_1^*$ are such that $\mathcal{T}_0^* \cap \mathcal{T}_1^* = \emptyset$. Indeed, if $\mathcal{T}_0^* \cap \mathcal{T}_1^* \neq \emptyset$, then there exists $r \in \mathcal{T}_0^* \cap \mathcal{T}_1^*$ and, therefore, some $\mathcal{B}_i$ such that $r \in \mathcal{B}_i$. Since the sets $\mathcal{T}_0^*$ and $\mathcal{T}_1^*$ are defined as unions of sets $\mathcal{B}_j$, this implies that $\mathcal{B}_i \subseteq \mathcal{T}_0^* \cap \mathcal{T}_1^*$. However, this is impossible, because it implies that there exists $r' \in \mathcal{T}_{k_1} \cap \mathcal{T}_1$ such that $\mathcal{B}_i = \mathcal{B}_{\beta(r')}$, against the definition of $\mathcal{T}_1 = \mathcal{T} \setminus \mathcal{T}_0$ that makes $\mathcal{T}_1$ and $\mathcal{T}_{k_1} \subseteq \mathcal{T}_0$ disjoint.

For the sake of completeness, we rewrite the experimet $\mathbf{Tamper}^{\mathsf{BL}}$, expanding the definition of SS.Share.

```
Experiment:  $\mathbf{Tamper}^{\mathsf{BL}}_{\mathsf{A}}[\lambda, 1, \mu_0, \mu_1, b]$
             $(\hat\sigma_0, \hat\sigma_1) \leftarrow\!\$\ \mathsf{SS}_2.\mathsf{Share}(\mu_b)$          // Share the message.
             $(\sigma_{0,1}, \dots, \sigma_{0,n}) \leftarrow\!\$\ \mathsf{SS}_0.\mathsf{Share}(\hat\sigma_0)$
             $(\sigma_{1,1}, \dots, \sigma_{1,n}) \leftarrow\!\$\ \mathsf{SS}_1.\mathsf{Share}(\hat\sigma_1)$
             $\forall\ r \in [n] : \sigma_r := (\sigma_{0,r}, \sigma_{1,r})$
             $\sigma := (\sigma_1, \dots, \sigma_n)$
             $(\alpha; \mathcal{T}, \mathcal{B}, f) \leftarrow\!\$\ \mathsf{A}_1^{\mathcal{O}_{\mathsf{BL}}(\sigma,\cdot,\cdot)}(1^\lambda)$          // Leakage phase.
             $(\mathcal{B}_1, \dots, \mathcal{B}_m) := \mathcal{B}$
             $(f_1, \dots, f_m) := f$
             $\forall\ i \in [m] : \tilde\sigma_{\mathcal{B}_i} \leftarrow f_i(\sigma_{\mathcal{B}_i})$          // Tampering phase.
             $\tilde\mu \leftarrow \mathsf{SS}.\mathsf{Reconstruct}(\mathcal{T}, \tilde\sigma_{\mathcal{T}})$
             IF $\tilde\mu \in \{\mu_0, \mu_1\} : \tilde\mu \leftarrow \heartsuit$
             $b^* \leftarrow\!\$\ \mathsf{A}_2(\alpha; \tilde\mu)$
     Output: $b^*$
```

The first hybrid experiment. We define the experiment $\mathbf{Hyb}^1_{\mathsf{A}}[\lambda, \mu_0, \mu_1, b]$ to be the same as $\mathbf{Tamper}^{\mathsf{BL}}_{\mathsf{A}}[\lambda, 1, \mu_0, \mu_1, b]$, except for how the tampering query is answered.

Namely, after the leakage phase, $\mathbf{Hyb}^1_{\mathsf{A}}$ samples the shares $(\sigma^*_{0,r})_{r \in \mathcal{T}_1^*}$ such that $(\sigma_{0,r})_{r \in \mathcal{T}_0^*}, (\sigma^*_{0,r})_{r \in \mathcal{T}_1^*}$ is a valid secret sharing of $\hat\sigma_0$ and, moreover, it is consistent with the leakage performed by A during the leakage phase. Then, $\mathbf{Hyb}^1_{\mathsf{A}}$ answers to A's queries as follows.

$$\text{Leakage queries:} \quad \left(\begin{pmatrix} \sigma_{0,r} \\ \sigma_{1,r} \end{pmatrix}_{r \in \mathcal{T}_0^*}, \begin{pmatrix} \sigma_{0,r} \\ \sigma_{1,r} \end{pmatrix}_{r \in \mathcal{T}_1^*} \right)$$

$$\text{Tampering queries:} \quad \left(\begin{pmatrix} \sigma_{0,r} \\ \sigma_{1,r} \end{pmatrix}_{r \in \mathcal{T}_0^*}, \begin{pmatrix} \sigma^*_{0,r} \\ \sigma_{1,r} \end{pmatrix}_{r \in \mathcal{T}_1^*} \right)$$

In the following lemma, we show that the two experiments are equivalent.

Lemma 3.3. *For all* $b \in \{0,1\}$, $\mathbf{Tamper}_{\mathsf{A}}^{\mathsf{BL}}[\lambda, 1, \mu_0, \mu_1, b] \triangleq \mathbf{Hyb}_{\mathsf{A}}^{1}[\lambda, \mu_0, \mu_1, b]$.

Proof. Let, for $i, j \in \{0,1\}$, $(\mathbf{\Sigma}_{i,r})_{r \in \mathcal{T}_j^*}$ be the random variable for the value $(\sigma_{i,r})_{r \in \mathcal{T}_j^*}$, and let $\left(\mathbf{\Sigma}_{0,r}^*\right)_{r \in \mathcal{T}_1^*}$ be the random variable for the value $\left(\sigma_{i,r}^*\right)_{r \in \mathcal{T}_j^*}$.

For a string $\bar{s}$, let $\mathrm{E}_{\bar{s}}$ be the event that $(\mathbf{\Sigma}_{0,r})_{r \in \mathcal{T}_1^*} = \bar{s}$, and let $\mathrm{E}_{\bar{s}}^*$ be the event that $\left(\mathbf{\Sigma}_{0,r}^*\right)_{r \in \mathcal{T}_1^*} = \bar{s}$. Then, $\mathbb{P}\left[\mathrm{E}_{\bar{s}}\right] = \mathbb{P}\left[\mathrm{E}_{\bar{s}}^*\right]$. This is because $\left(\mathbf{\Sigma}_{0,r}^*\right)_{r \in \mathcal{T}_1^*}$ is sampled from the same distribution of $(\mathbf{\Sigma}_{0,r})_{r \in \mathcal{T}_1^*}$ and consistently with $(\mathbf{\Sigma}_{0,r})_{r \in \mathcal{T}_0^*}$ and the leakage performed by A.

With that in mind, we can conclude the lemma by writing

$$\begin{aligned}
&\mathbb{P}\left[\mathbf{Tamper}_{\mathsf{A}}^{\mathsf{BL}}[\lambda, 1, \mu_0, \mu_1, b] = 1\right] - \mathbb{P}\left[\mathbf{Hyb}_{\mathsf{A}}^{1}[\lambda, \mu_0, \mu_1, b] = 1\right] \\
&= \sum_{\bar{s}} \mathbb{P}\left[\mathrm{E}_{\bar{s}}\right] \mathbb{P}\left[\mathbf{Tamper}_{\mathsf{A}}^{\mathsf{BL}}[\lambda, 1, \mu_0, \mu_1, b] = 1 \middle| \mathrm{E}_{\bar{s}}\right] \\
&\qquad - \sum_{\bar{s}} \mathbb{P}\left[\mathrm{E}_{\bar{s}}^*\right] \mathbb{P}\left[\mathbf{Hyb}_{\mathsf{A}}^{1}[\lambda, \mu_0, \mu_1, b] = 1 \middle| \mathrm{E}_{\bar{s}}^*\right] \\
&= \sum_{\bar{s}} \mathbb{P}\left[\mathrm{E}_{\bar{s}}\right] \left(\mathbb{P}\left[\mathbf{Tamper}_{\mathsf{A}}^{\mathsf{BL}}[\lambda, 1, \mu_0, \mu_1, b] = 1 \middle| \mathrm{E}_{\bar{s}}\right]\right. \qquad (3.1)\\
&\qquad \left. - \mathbb{P}\left[\mathbf{Hyb}_{\mathsf{A}}^{1}[\lambda, \mu_0, \mu_1, b] = 1 \middle| \mathrm{E}_{\bar{s}}^*\right]\right) \\
&= 0, \qquad (3.2)
\end{aligned}$$

where Eq. (3.1) comes from the above reasoning and Eq. (3.2) holds because, once fixed $\bar{s}$, if both $\mathrm{E}_{\bar{s}}$ and $\mathrm{E}_{\bar{s}}^*$ happen, then $(\mathbf{\Sigma}_{0,r})_{r \in \mathcal{T}_0^*} = \bar{s} = \left(\mathbf{\Sigma}_{0,r}^*\right)_{r \in \mathcal{T}_0^*}$ and the two experiments are identical. □

The second hybrid experiment. We define experiment $\mathbf{Hyb}_{\mathsf{A}}^{2}[\lambda, \mu_0, \mu_1, b]$ to be the same as $\mathbf{Hyb}_{\mathsf{A}}^{1}[\lambda, \mu_0, \mu_1, b]$, except for the fact that leakage is performed on *fake* shares of σ_0.

Namely, let $\mathsf{Resample}_0$ be the algorithm that takes as input the value σ_0, the shares $(\sigma_{0,r})_{r \in \mathcal{T}_1^*}$ and the leakage Λ from the leakage phase and outputs the shares $\left(\sigma_{0,r}^*\right)_{r \in \mathcal{T}_1^*}$ as defined in $\mathbf{Hyb}_{\mathsf{A}}^{1}[\lambda, \mu_0, \mu_1, b]$. Then, we highlight the differences between $\mathbf{Hyb}_{\mathsf{A}}^{1}[\lambda, \mu_0, \mu_1, b]$ and $\mathbf{Hyb}_{\mathsf{A}}^{2}[\lambda, \mu_0, \mu_1, b]$ in the following table. Here, we make A output the performed leakage Λ as well.

```
Differences:  $\mathbf{Hyb}^1_{\mathsf{A}}[\lambda, \mu_0, \mu_1, b]$  $\mathbf{Hyb}^2_{\mathsf{A}}[\lambda, \mu_0, \mu_1, b]$
              $(\hat{\sigma}_0, \hat{\sigma}_1) \leftarrow\!\!\$\ \mathsf{SS}_2.\mathsf{Share}(\mu_b)$
              $(\sigma_{0,1}, \dots, \sigma_{0,n}) \leftarrow\!\!\$\ \mathsf{SS}_0.\mathsf{Share}(\hat{\sigma}_0)$
              $(\hat{\sigma}'_0, \hat{\sigma}'_1) \leftarrow\!\!\$\ \mathsf{SS}_2.\mathsf{Share}(0)$
              $(\sigma'_{0,1}, \dots, \sigma'_{0,n}) \leftarrow\!\!\$\ \mathsf{SS}_0.\mathsf{Share}(\hat{\sigma}'_0)$
              $(\sigma_{1,1}, \dots, \sigma_{1,n}) \leftarrow\!\!\$\ \mathsf{SS}_1.\mathsf{Share}(\hat{\sigma}_1)$
              $\forall\ r \in [n] : \sigma_r := (\sigma_{0,r}, \sigma_{1,r})$
              $\forall\ r \in [n] : \sigma_r := (\sigma'_{0,r}, \sigma_{1,r})$
              $\sigma := (\sigma_1, \dots, \sigma_n)$
              $(\alpha, \Lambda; \mathcal{T}, \mathcal{B}, f) \leftarrow\!\!\$\ \mathsf{A}_1^{\mathcal{O}_{\mathsf{BL}}(\sigma,\cdot,\cdot)}(1^\lambda)$
              $\left(\sigma^*_{0,r}\right)_{r \in \mathcal{T}^*_1} \leftarrow\!\!\$\ \mathsf{Resample}_0\left(\hat{\sigma}_0, (\sigma_{0,r})_{r \in \mathcal{T}^*_1}, \Lambda\right)$
              $(\sigma_{0,r})_{r \in \mathcal{T}^*_1} \leftarrow \left(\sigma^*_{0,r}\right)_{r \in \mathcal{T}^*_1}$
              $\left(\sigma^*_{0,r}\right)_{r \in \mathcal{T}^*_1} \leftarrow\!\!\$\ \mathsf{Resample}_0\left(\hat{\sigma}_0, \left(\sigma'_{0,r}\right)_{r \in \mathcal{T}^*_1}, \Lambda\right)$
              $\left(\sigma'_{0,r}\right)_{r \in \mathcal{T}^*_1} \leftarrow \left(\sigma^*_{0,r}\right)_{r \in \mathcal{T}^*_1}$
              $(\mathcal{B}_1, \dots, \mathcal{B}_m) := \mathcal{B}$
              $(f_1, \dots, f_m) := f$
              $\forall\ i \in [m] : \tilde{\sigma}_{\mathcal{B}_i} \leftarrow f_i(\sigma_{\mathcal{B}_i})$
              $\tilde{\mu} \leftarrow \mathsf{SS}.\mathsf{Reconstruct}(\mathcal{T}, \tilde{\sigma}_{\mathcal{T}})$
              IF $\tilde{\mu} \in \{\mu_0, \mu_1\} : \tilde{\mu} \leftarrow \heartsuit$
              $b^* \leftarrow\!\!\$\ \mathsf{A}_2(\alpha, \Lambda; \tilde{\mu})$
Output:       $b^*$
```

In other words, $\mathbf{Hyb}^2_{\mathsf{A}}$ answers to A's queries as follows.

$$\text{Leakage queries:} \quad \left(\begin{pmatrix} \sigma'_{0,r} \\ \sigma_{1,r} \end{pmatrix}_{r \in \mathcal{T}^*_0}, \begin{pmatrix} \sigma'_{0,r} \\ \sigma_{1,r} \end{pmatrix}_{r \in \mathcal{T}^*_1} \right)$$

$$\text{Tampering queries:} \quad \left(\begin{pmatrix} \sigma'_{0,r} \\ \sigma_{1,r} \end{pmatrix}_{r \in \mathcal{T}^*_0}, \begin{pmatrix} \sigma^*_{0,r} \\ \sigma_{1,r} \end{pmatrix}_{r \in \mathcal{T}^*_1} \right)$$

In the following lemma, we show that the two experiments are statistically close.

Lemma 3.4. *For all* $b \in \{0,1\}$, $\mathbf{Hyb}^1_{\mathsf{A}}[\lambda, \mu_0, \mu_1, b] \overset{\triangle}{\approx}_{\varepsilon_0(\lambda)} \mathbf{Hyb}^2_{\mathsf{A}}[\lambda, \mu_0, \mu_1, b]$.

Proof. By reduction to leakage-resilience of SS_0. Suppose towards contradiction that there exist messages μ_0, μ_1, a bit $b \in \{0,1\}$ and an adversary A which is able to cause a statistical difference between $\mathbf{Hyb}^1_{\mathsf{A}}[\lambda, \mu_0, \mu_1, b]$ and $\mathbf{Hyb}^2_{\mathsf{A}}[\lambda, \mu_0, \mu_1, b]$ of more than $\varepsilon_0(\lambda)$.

Let $(\hat{\sigma}_0, \hat{\sigma}_1) \leftarrow \mathsf{SS}_2.\mathsf{Share}(\mu_b)$ and let $(\hat{\sigma}'_0, \hat{\sigma}'_1) \leftarrow \mathsf{SS}_2.\mathsf{Share}(0)$. Consider the following reduction $\mathsf{R}^{\mathcal{O}_{\mathsf{BL}}((\sigma^{\mathsf{target}}_{0,r})_{r \in [n]},\cdot,\cdot)}$.

1. Sample $(\sigma_{1,1}, \dots, \sigma_{1,n}) \leftarrow\!\!\$\ \mathsf{SS}_1.\mathsf{Share}(\hat{\sigma}_1)$.

2. Construct the oracle $\mathcal{O}'_{\mathsf{BL}}((\sigma_{0,r}^{\mathsf{target}}, \sigma_{1,r})_{r\in[n]}, \cdot, \cdot)$ that, upon input a leakage query $(\mathcal{B}, g)$, hard-wires values $(\sigma_{1,1}, \dots, \sigma_{1,n})$ into g, queries the oracle $\Lambda \leftarrow\!\!\$\ \mathcal{O}_{\mathsf{BL}}\left((\sigma_{0,r}^{\mathsf{target}})_{r\in[n]}, \mathcal{B}, g\right)$ and finally outputs Λ.
3. Run $(\alpha, \Lambda; \mathcal{T}, \mathcal{B}, f) \leftarrow\!\!\$\ \mathsf{A}_1^{\mathcal{O}'_{\mathsf{BL}}(\sigma,\cdot,\cdot)}(1^\lambda)$.
4. Construct the sets $\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_0^*, \mathcal{T}_1^*$ from $\mathcal{T}, \mathcal{B}$, as described at the beginning of this section.
5. Use 1 bit of leakage to apply the augmented property, using Theorem 2.14, thus obtaining shares $\left(\sigma_{0,r}^{\mathsf{target}}\right)_{r\in\mathcal{T}_0^*}$.
6. For all $j \in \mathcal{T}_0$, compute $(\tilde{\sigma}_{0,r}, \tilde{\sigma}_{1,r})_{r\in\mathcal{B}_{\beta(j)}} = f_{\beta(j)}\left(\left(\sigma_{0,r}^{\mathsf{target}}, \sigma_{1,r}\right)_{r\in\mathcal{B}_{\beta(j)}}\right)$.
7. Sample $\left(\sigma_{0,r}^*\right)_{r\in\mathcal{T}_1^*} \leftarrow\!\!\$\ \mathsf{Resample}_0\left(\hat{\sigma}_0, (\sigma_{0,r})_{r\in\mathcal{T}_0^*}, \Lambda\right)$.
8. For all $j \in \mathcal{T}_1$, compute $(\tilde{\sigma}_{0,r}, \tilde{\sigma}_{1,r})_{r\in\mathcal{B}_{\beta(j)}} = f_{\beta(j)}\left(\left(\sigma_{0,r}^*, \sigma_{1,r}\right)_{r\in\mathcal{B}_{\beta(j)}}\right)$.
9. Compute $\tilde{\sigma}_0 \leftarrow \mathsf{SS}_0.\mathsf{Reconstruct}\,(\mathcal{T}, (\tilde{\sigma}_{0,r})_{r\in\mathcal{T}})$.
10. Compute $\tilde{\sigma}_1 \leftarrow \mathsf{SS}_1.\mathsf{Reconstruct}\left(\mathcal{T}_{k_1}, (\tilde{\sigma}_{1,r})_{r\in\mathcal{T}_{k_1}}\right)$.
11. Compute $\tilde{\mu} \leftarrow \mathsf{SS}_2.\mathsf{Reconstruct}\,([2], (\tilde{\sigma}_0, \tilde{\sigma}_1))$; if $\tilde{\mu} \in \{\mu_0, \mu_1\}$, replace $\tilde{\mu} \leftarrow \heartsuit$.
12. Output the same as $\mathsf{A}_2(\alpha, \Lambda; \tilde{\mu})$.

The analysis is as follows.

Correctness. R perfectly simulates $\mathbf{Hyb}^1$ if $(\sigma_{0,r}^{\mathsf{target}})_{r\in[n]} \leftarrow\!\!\$\ \mathsf{SS}_0.\mathsf{Share}(\hat{\sigma}_0)$ and perfectly simulates $\mathbf{Hyb}^2$ if $(\sigma_{0,r}^{\mathsf{target}})_{r\in[n]} \leftarrow\!\!\$\ \mathsf{SS}_0.\mathsf{Share}(\hat{\sigma}'_0)$. Furthermore, R always outputs the same as A_2, thus retaining the same distinguishing advantage.

Admissibility. The leakage queries performed by R are the same performed by A, which is admissible for ℓ bits of leakage and the same partitioning.[2] Furthermore, in step 5, R performs one bit of leakage to apply the augmented property of leakage-resilient secret sharing schemes and obtain all the challenge shares in $\mathcal{T}_0^*$; since $\mathcal{T}_0^*$ comes from the union of at most k_1 sets of at most $k_1 - 1$ elements each, $\#\mathcal{T}_0^* \leq k_1(k_1 - 1) \leq k_1^2 \leq k$, hence the augmented property can be applied. Since no other leakage is performed, this makes R admissible for $\ell_0 = \ell + 1$ bits of leakage and for $\mathcal{A}\mathsf{daptive}(k)$ partitioning. The lemma follows. □

[2]It is easy to show that any adversary that is admissible for semi-adaptive partitioning is admissible for adaptive partitioning as well.

The third hybrid experiment. We define the experiment $\mathbf{Hyb}^3_{\mathsf{A}}[\lambda, \mu_0, \mu_1, b]$ to be the same as $\mathbf{Hyb}^2_{\mathsf{A}}[\lambda, \mu_0, \mu_1, b]$, except for how the tampering query is answered.

Namely, after the leakage phase, $\mathbf{Hyb}^1_{\mathsf{A}}$ samples the shares $(\sigma^*_{1,r})_{r \in \mathcal{T}^*_0}$ such that the following conditions are met.

1. $\mathsf{SS}_2.\mathsf{Reconstruct}\left((\sigma^*_{1,r})_{r \in \mathcal{T}^*_0}\right) = \hat{\sigma}_1$ and, moreover, it is consistent with the leakage performed by A during the leakage phase.

2. For all $j \in \mathcal{T}_0$, let

$$\left(\tilde{\sigma}^*_{0,r}\right)_{r \in \mathcal{B}_{\beta(j)}} = f_{\beta(j)}\left(\left(\sigma'_{0,r}, \sigma^*_{1,r}\right)_{r \in \mathcal{B}_{\beta(j)}}\right),$$
$$\left(\tilde{\sigma}_{0,r}\right)_{r \in \mathcal{B}_{\beta(j)}} = f_{\beta(j)}\left(\left(\sigma'_{0,r}, \sigma_{1,r}\right)_{r \in \mathcal{B}_{\beta(j)}}\right).$$

   Then, $(\tilde{\sigma}^*_{0,r})_{r \in \mathcal{B}_{\beta(j)}} = (\tilde{\sigma}_{0,r})_{r \in \mathcal{B}_{\beta(j)}}$. In other words, applying the tampering function to $(\sigma^*_{1,r})_{r \in \mathcal{B}_{\beta(j)}}$ or to $(\sigma_{1,r})_{r \in \mathcal{B}_{\beta(j)}}$ leads to the same shares for the scheme SS_0.

Then, $\mathbf{Hyb}^3_{\mathsf{A}}$ answers to A's queries as follows.

$$\text{Leakage queries:} \quad \left(\begin{pmatrix} \sigma'_{0,r} \\ \sigma_{1,r} \end{pmatrix}_{r \in \mathcal{T}^*_0}, \begin{pmatrix} \sigma'_{0,r} \\ \sigma_{1,r} \end{pmatrix}_{r \in \mathcal{T}^*_1} \right)$$
$$\text{Tampering queries:} \quad \left(\begin{pmatrix} \sigma'_{0,r} \\ \sigma^*_{1,r} \end{pmatrix}_{r \in \mathcal{T}^*_0}, \begin{pmatrix} \sigma^*_{0,r} \\ \sigma_{1,r} \end{pmatrix}_{r \in \mathcal{T}^*_1} \right)$$

The following lemma holds.

Lemma 3.5. *For all $b \in \{0,1\}$, $\mathbf{Hyb}^2_{\mathsf{A}}[\lambda, \mu_0, \mu_1, b] \triangleq \mathbf{Hyb}^3_{\mathsf{A}}[\lambda, \mu_0, \mu_1, b]$.*

The proof of this lemma is almost the same of the proof of Lemma 3.3, and thus omitted.

The fourth hybrid experiment. We define experiment $\mathbf{Hyb}^4_{\mathsf{A}}[\lambda, \mu_0, \mu_1, b]$ to be the same as $\mathbf{Hyb}^3_{\mathsf{A}}[\lambda, \mu_0, \mu_1, b]$, except for the fact that leakage is performed on *fake* shares of σ_1.

Namely, let $\mathsf{Resample}_1$ be the algorithm that takes as input the value σ_1, the leakage Λ from the leakage phase and the output $(\tilde{\sigma}_{0,r})_{r \in \mathcal{T}^*_0 \cup \mathcal{T}^*_1}$ of the tampering query and outputs the shares $\left(\sigma^*_{1,r}\right)_{r \in \mathcal{T}^*_0}$ as defined in $\mathbf{Hyb}^3_{\mathsf{A}}[\lambda, \mu_0, \mu_1, b]$. Then, we highlight the differences between $\mathbf{Hyb}^3_{\mathsf{A}}[\lambda, \mu_0, \mu_1, b]$ and $\mathbf{Hyb}^4_{\mathsf{A}}[\lambda, \mu_0, \mu_1, b]$ in the following table. Here, we make A output the performed leakage Λ as well.

```
Differences:   $\mathbf{Hyb}^3_{\mathsf{A}}[\lambda,\mu_0,\mu_1,b]$   $\mathbf{Hyb}^4_{\mathsf{A}}[\lambda,\mu_0,\mu_1,b]$
    $(\hat\sigma_0,\hat\sigma_1) \leftarrow\$ \mathsf{SS}_2.\mathsf{Share}(\mu_b)$
    $(\hat\sigma'_0,\hat\sigma'_1) \leftarrow\$ \mathsf{SS}_2.\mathsf{Share}(0)$
    $(\sigma'_{0,1},\ldots,\sigma'_{0,n}) \leftarrow\$ \mathsf{SS}_0.\mathsf{Share}(\hat\sigma'_0)$
    $(\sigma_{1,1},\ldots,\sigma_{1,n}) \leftarrow\$ \mathsf{SS}_1.\mathsf{Share}(\hat\sigma_1)$
    $\forall\ r \in [n] : \sigma_r := (\sigma'_{0,r},\sigma_{1,r})$
    $(\sigma'_{1,1},\ldots,\sigma'_{1,n}) \leftarrow\$ \mathsf{SS}_1.\mathsf{Share}(\hat\sigma'_1)$
    $\forall\ r \in [n] : \sigma_r := (\sigma'_{0,r},\sigma'_{1,r})$
    $\sigma := (\sigma_1,\ldots,\sigma_n)$
    $(\alpha,\Lambda;\mathcal{T},\mathcal{B},f) \leftarrow\$ \mathsf{A}_1^{\mathcal{O}_{\mathsf{BL}}(\sigma,\cdot,\cdot)}(1^\lambda)$
    $\left(\sigma^*_{0,r}\right)_{r\in\mathcal{T}^*_1} \leftarrow\$ \mathsf{Resample}_0\left(\hat\sigma_0,\left(\sigma'_{0,r}\right)_{r\in\mathcal{T}^*_1},\Lambda\right)$
    $\left(\sigma'_{0,r}\right)_{r\in\mathcal{T}^*_1} \leftarrow \left(\sigma^*_{0,r}\right)_{r\in\mathcal{T}^*_1}$
    $(\mathcal{B}_1,\ldots,\mathcal{B}_m) := \mathcal{B}$
    $(f_1,\ldots,f_m) := f$
    $\forall\ i \in [m] : \tilde\sigma'_{\mathcal{B}_i} \leftarrow f_i(\sigma_{\mathcal{B}_i})$
    $\left(\sigma^*_{1,r}\right)_{r\in\mathcal{T}^*_0} \leftarrow\$ \mathsf{Resample}_1\left(\hat\sigma_0,\Lambda,\left(\tilde\sigma'_{0,r}\right)_{r\in\mathcal{T}^*_0\cup\mathcal{T}^*_1}\right)$
    $(\sigma_{1,r})_{r\in\mathcal{T}^*_0} \leftarrow \left(\sigma^*_{1,r}\right)_{r\in\mathcal{T}^*_0}$
    $\left(\sigma'_{1,r}\right)_{r\in\mathcal{T}^*_0} \leftarrow \left(\sigma^*_{1,r}\right)_{r\in\mathcal{T}^*_0}$
    $\forall\ i \in [m] : \tilde\sigma_{\mathcal{B}_i} \leftarrow f_i(\sigma_{\mathcal{B}_i})$
    $\tilde\mu \leftarrow \mathsf{SS}.\mathsf{Reconstruct}(\mathcal{T},\tilde\sigma_{\mathcal{T}})$
    IF $\tilde\mu \in \{\mu_0,\mu_1\} : \tilde\mu \leftarrow \heartsuit$
    $b^* \leftarrow\$ \mathsf{A}_2(\alpha,\Lambda;\tilde\mu)$
Output:   $b^*$
```

In other words, $\mathbf{Hyb}^4_{\mathsf{A}}$ answers to A's queries as follows.

$$\text{Leakage queries:}\quad \left(\begin{pmatrix}\sigma'_{0,r}\\ \sigma'_{1,r}\end{pmatrix}_{r\in\mathcal{T}^*_0},\begin{pmatrix}\sigma'_{0,r}\\ \sigma'_{1,r}\end{pmatrix}_{r\in\mathcal{T}^*_1}\right)$$

$$\text{Tampering queries:}\quad \left(\begin{pmatrix}\sigma'_{0,r}\\ \sigma^*_{1,r}\end{pmatrix}_{r\in\mathcal{T}^*_0},\begin{pmatrix}\sigma^*_{0,r}\\ \sigma'_{1,r}\end{pmatrix}_{r\in\mathcal{T}^*_1}\right)$$

Lemma 3.6. *For all $b \in \{0,1\}$, $\mathbf{Hyb}^3_{\mathsf{A}}[\lambda,\mu_0,\mu_1,b] \overset{\Delta}{\approx}_{\varepsilon_1(\lambda)} \mathbf{Hyb}^4_{\mathsf{A}}[\lambda,\mu_0,\mu_1,b]$.*

Proof. By reduction to leakage-resilience of SS_1. Suppose towards contradiction that there exist messages μ_0,μ_1, a bit $b \in \{0,1\}$ and an adversary A which is able to cause a statistical difference between $\mathbf{Hyb}^3_{\mathsf{A}}[\lambda,\mu_0,\mu_1,b]$ and $\mathbf{Hyb}^4_{\mathsf{A}}[\lambda,\mu_0,\mu_1,b]$ of more than $\varepsilon_1(\lambda)$.

Let $(\hat\sigma_0,\hat\sigma_1) \leftarrow \mathsf{SS}_2.\mathsf{Share}(\mu_b)$ and let $(\hat\sigma'_0,\hat\sigma'_1) \leftarrow \mathsf{SS}_2.\mathsf{Share}(0)$. Consider the following reduction $\mathsf{R}^{\mathcal{O}_{\mathsf{BL}}\left((\sigma^{\mathsf{target}}_{0,r})_{r\in[n]},\cdot,\cdot\right)}$.

1. Sample $(\sigma'_{0,1},\ldots,\sigma'_{0,n}) \leftarrow\$ \mathsf{SS}_0.\mathsf{Share}(\hat\sigma'_0)$.

2. Construct the oracle $\mathcal{O}'_{\mathsf{BL}}((\sigma'_{0,r},\sigma^{\mathsf{target}}_{1,r})_{r\in[n]},\cdot,\cdot)$ that, upon input a leakage query $(\mathcal{B},g)$, hard-wires values $(\sigma'_{1,1},\dots,\sigma'_{1,n})$ into g, queries the oracle $\Lambda \leftarrow\!\!\$\; \mathcal{O}_{\mathsf{BL}}\left((\sigma^{\mathsf{target}}_{1,r})_{r\in[n]},\mathcal{B},g\right)$ and finally outputs Λ.
3. Run $(\alpha,\Lambda;\mathcal{T},\mathcal{B},f) \leftarrow\!\!\$\; \mathsf{A}_1^{\mathcal{O}'_{\mathsf{BL}}(\sigma,\cdot,\cdot)}(1^\lambda)$.
4. Construct the sets $\mathcal{T}_0,\mathcal{T}_1,\mathcal{T}_0^*,\mathcal{T}_1^*$ from $\mathcal{T},\mathcal{B}$, as described at the beginning of this section.
5. Sample $\left(\sigma^*_{0,r}\right)_{r\in\mathcal{T}_1^*} \leftarrow\!\!\$\; \mathsf{Resample}_0\left(\hat\sigma_0,\left(\sigma'_{0,r}\right)_{r\in\mathcal{T}_0^*},\Lambda\right)$. Notice that this can be done because R sampled the values $\left(\sigma'_{0,r}\right)_{r\in\mathcal{T}_0^*}$ in step 1.
6. Set $\sigma^{**}_{0,r} := \sigma'_{0,r}$ for all $r\in\mathcal{T}_0^*$ and $\sigma^{**}_{0,r} := \sigma^*_{0,r}$ for all $r\in\mathcal{T}_1^*$. Notice that this is well defined since $\mathcal{T}_0^*\cap\mathcal{T}_1^* = \emptyset$.
7. For all $i\in[m]$, construct the following leakage function. $\hat g_i$

Function:	$\hat g_i$
Input:	$\left(\sigma^{\mathsf{target}}_{1,r}\right)_{r\in\mathcal{B}_i}$
	$(\tilde\sigma_{0,r},\tilde\sigma_{1,r})_{r\in\mathcal{B}_i} \leftarrow f_i\left(\left(\sigma^{**}_{0,r},\sigma^{\mathsf{target}}_{1,r}\right)_{r\in\mathcal{B}_i}\right)$
Output:	$(\tilde\sigma_{0,r})_{r\in\mathcal{B}_i}$

8. Query $(\tilde\sigma_{0,r})_{r\in\mathcal{T}_0^*\cup\mathcal{T}_1^*} \leftarrow \mathcal{O}_{\mathsf{BL}}\left((\sigma^{\mathsf{target}}_{1,r})_{r\in[n]},\mathcal{B},(g_1,\dots,g_m)\right)$.
9. Sample $\left(\sigma^*_{1,r}\right)_{r\in\mathcal{T}_0^*} \leftarrow\!\!\$\; \mathsf{Resample}_1\left(\hat\sigma_1,\Lambda,(\tilde\sigma_{1,r})_{r\in\mathcal{T}_1^*}\right)$.
10. For all $j\in\mathcal{T}_0$, compute $(\tilde\sigma_{0,r},\tilde\sigma_{1,r})_{r\in\mathcal{B}_{\beta(j)}} = f_{\beta(j)}\left(\left(\sigma^{**}_{0,r},\sigma_{1,r}\right)_{r\in\mathcal{B}_{\beta(j)}}\right)$.
11. Compute $\tilde\sigma_0 \leftarrow \mathsf{SS}_0.\mathsf{Reconstruct}\left(\mathcal{T},(\tilde\sigma_{0,r})_{r\in\mathcal{T}}\right)$.
12. Compute $\tilde\sigma_1 \leftarrow \mathsf{SS}_1.\mathsf{Reconstruct}\left(\mathcal{T}_{k_1},(\tilde\sigma_{1,r})_{r\in\mathcal{T}_{k_1}}\right)$; recall that $\mathcal{T}_{k_1}\subseteq\mathcal{T}_0$, and we obtained the shares $(\tilde\sigma_{1,r})_{r\in\mathcal{T}_0}$ in step 10.
13. Compute $\tilde\mu \leftarrow \mathsf{SS}_2.\mathsf{Reconstruct}\left([2],(\tilde\sigma_0,\tilde\sigma_1)\right)$; if $\tilde\mu\in\{\mu_0,\mu_1\}$, replace $\tilde\mu\leftarrow\heartsuit$.
14. Output the same as $\mathsf{A}_2(\alpha,\Lambda;\tilde\mu)$.

The analysis is as follows.

Correctness. R perfectly simulates $\mathbf{Hyb}^3$ if $(\sigma^{\mathsf{target}}_{1,r})_{r\in[n]} \leftarrow\!\!\$\; \mathsf{SS}_1.\mathsf{Share}(\hat\sigma_1)$ and perfectly simulates $\mathbf{Hyb}^4$ if $(\sigma^{\mathsf{target}}_{1,r})_{r\in[n]} \leftarrow\!\!\$\; \mathsf{SS}_1.\mathsf{Share}(\hat\sigma'_1)$. Furthermore, R always outputs the same as A_2, thus retaining the same distinguishing advantage.

Admissibility. The leakage queries performed by R are the same performed by A, plus the amount necessary to obtain the values $(\tilde{\sigma}_{0,r})_{r\in\mathcal{T}_0^*\cup\mathcal{T}_1^*}$ in step 8. Therefore, the total amount of leakage performed by R is bounded by

$$\ell_1 = \ell + \sum_{r\in\mathcal{T}_0^*\cup\mathcal{T}_1^*} |\tilde{\sigma}_{0,r}| \leq \ell + n \cdot s_0,$$

hence R is ℓ_1-leakage admissible.

Furthermore, R uses the same partitioning of A, that is $\mathcal{S}\text{emiAdaptive}(k_1 - 1)$; hence, R is also $\mathcal{A}\text{daptive}(k_1 - 1)$ and, therefore, admissible. The lemma follows. □

The final step. Now it only remains to show the following lemma in order to conclude.

Lemma 3.7. $\mathbf{Hyb}^4_{\mathsf{A}}[\lambda,\mu_0,\mu_1,0] \overset{\triangle}{\approx}_{\varepsilon_2(\lambda)} \mathbf{Hyb}^4_{\mathsf{A}}[\lambda,\mu_0,\mu_1,1]$.

Proof. By reduction to non-malleability of SS_2. Suppose towards contradiction that there exist messages μ_0,μ_1 and an adversary A which is able to cause a statistical difference between $\mathbf{Hyb}^4_{\mathsf{A}}[\lambda,\mu_0,\mu_1,0]$ and $\mathbf{Hyb}^4_{\mathsf{A}}[\lambda,\mu_0,\mu_1,1]$ of more than $\varepsilon_2(\lambda)$.

Let $(\hat{\sigma}'_0,\hat{\sigma}'_1) \leftarrow\!\!\$\; \mathsf{SS}_2.\mathsf{Share}(0)$ and let, for $i\in\{0,1\}$, $(\sigma'_{i,1},\ldots,\sigma'_{i,n}) \leftarrow\!\!\$\; \mathsf{SS}_i.\mathsf{Share}(\hat{\sigma}'_i)$. Consider the following reduction $\mathsf{R}^{\mathcal{O}^1_{\mathsf{T}}((\hat{\sigma}_0^{\mathsf{target}},\hat{\sigma}_1^{\mathsf{target}}),\cdot,\cdot,\cdot)}$.

1. Let $\sigma' = ((\sigma'_{0,1},\sigma'_{1,1}),\ldots,(\sigma'_{0,n},\sigma'_{1,n}))$.
2. Run $(\alpha,\Lambda;\mathcal{T},\mathcal{B},f) \leftarrow\!\!\$\; \mathsf{A}_1^{\mathcal{O}_{\mathsf{BL}}(\sigma',\cdot,\cdot)}(1^\lambda)$.
3. Construct the sets $\mathcal{T}_0,\mathcal{T}_1,\mathcal{T}_0^*,\mathcal{T}_1^*$ from $\mathcal{T},\mathcal{B}$, as described at the beginning of this section.
4. Compute, for $j\in\mathcal{T}_1$, $\left(\tilde{\sigma}'_{0,r},\tilde{\sigma}'_{1,r}\right)_{r\in\mathcal{B}_{\beta(j)}} \leftarrow f_{\beta(j)}\left(\left(\sigma'_{0,r},\sigma'_{1,r}\right)_{r\in\mathcal{B}_{\beta(j)}}\right)$
5. Construct the following functions.

```
Function: $\hat{f}_0$
   Input: $\hat{\sigma}_0^{\mathsf{target}}$
          $\left(\sigma^*_{0,r}\right)_{r\in\mathcal{T}_1^*} \leftarrow\!\!\$\; \mathsf{Resample}_0\left(\hat{\sigma}_0^{\mathsf{target}}, \left(\sigma'_{0,r}\right)_{r\in\mathcal{T}_0^*}, \Lambda\right)$
          $\forall\, j\in\mathcal{T}_0 :\ (\tilde{\sigma}_{0,r},\tilde{\sigma}_{1,r})_{r\in\mathcal{B}_{\beta(j)}} \leftarrow f_{\beta(j)}\left(\sigma'_{0,r},\sigma'_{1,r}\right)_{r\in\mathcal{B}_{\beta(j)}}$
          $\forall\, j\in\mathcal{T}_1 :\ (\tilde{\sigma}_{0,r},\tilde{\sigma}_{1,r})_{r\in\mathcal{B}_{\beta(j)}} \leftarrow f_{\beta(j)}\left(\sigma^*_{0,r},\sigma'_{1,r}\right)_{r\in\mathcal{B}_{\beta(j)}}$
          $\tilde{\sigma}_0 \leftarrow \mathsf{SS}_0.\mathsf{Reconstruct}\Big((\mathcal{T},\tilde{\sigma}_{0,r})_{r\in\mathcal{T}}\Big)$
  Output: $\tilde{\sigma}_0$
```

```
Function: $\hat{f}_1$
Input: $\hat{\sigma}_1^{\mathsf{target}}$

    $\left(\sigma^*_{1,r}\right)_{r \in \mathcal{T}_0^*} \leftarrow\$ \mathsf{Resample}_1\left(\hat{\sigma}_1^{\mathsf{target}}, \Lambda, \left(\tilde{\sigma}'_{1,r}\right)_{r \in \mathcal{T}_1^*}\right)$
    $\forall\ j \in \mathcal{T}_0 :\ (\tilde{\sigma}_{0,r}, \tilde{\sigma}_{1,r})_{r \in \mathcal{B}_{\beta(j)}} \leftarrow f_{\beta(j)}\left(\sigma'_{0,r}, \sigma^*_{1,r}\right)_{r \in \mathcal{B}_{\beta(j)}}$
    $\tilde{\sigma}_1 \leftarrow \mathsf{SS}_1.\mathsf{Reconstruct}\Big((\mathcal{T}_{k_1} \tilde{\sigma}_{0,r})_{r \in \mathcal{T}_{k_1}}\Big)$

Output: $\tilde{\sigma}_1$
```

6. Query $\tilde{\mu} \leftarrow \mathcal{O}^1_{\mathsf{T}}\Big((\hat{\sigma}_0^{\mathsf{target}}, \hat{\sigma}_1^{\mathsf{target}}), \mathcal{T}, \mathcal{B}, f\Big)$.
7. Output the same as $\mathsf{A}_2(\alpha, \Lambda; \tilde{\mu})$.

The analysis is as follows.

Correctness. For $b \in \{0,1\}$, R perfectly simulates $\mathbf{Hyb}^4_{\mathsf{A}}[\lambda, \mu_0, \mu_1, b]$ whenever $(\hat{\sigma}_0^{\mathsf{target}}, \hat{\sigma}_1^{\mathsf{target}}) \leftarrow\$ \mathsf{SS}_2.\mathsf{Share}(\mu_b)$. Furthermore, R always outputs the same as A_2, thus retaining the same distinguishing advantage.

Notice that the leakage is performed by R on the fake shares $((\sigma'_{0,1}, \sigma'_{1,1}), \ldots, (\sigma'_{0,n}, \sigma'_{1,n}))$, hence, R is 0-leakage admissible in the sense that does not perform leakage from the target shares. The lemma follows. □

Proof of Theorem 3.1. The theorem follows from the above lemmas and the triangular inequality. Indeed, for $b \in \{0,1\}$,

$$\begin{aligned}\mathbf{Tamper}^{\mathsf{BL}}_{\mathsf{A}}[\lambda, 1, \mu_0, \mu_1, b] &\triangleq \mathbf{Hyb}^1_{\mathsf{A}}[\lambda, \mu_0, \mu_1, b] \\ &\overset{\triangle}{\approx}_{\varepsilon_0(\lambda)} \mathbf{Hyb}^2_{\mathsf{A}}[\lambda, \mu_0, \mu_1, b] \\ &\triangleq \mathbf{Hyb}^3_{\mathsf{A}}[\lambda, \mu_0, \mu_1, b] \\ &\overset{\triangle}{\approx}_{\varepsilon_1(\lambda)} \mathbf{Hyb}^4_{\mathsf{A}}[\lambda, \mu_0, \mu_1, b]\end{aligned}$$

implies that

$$\mathbf{Tamper}^{\mathsf{BL}}_{\mathsf{A}}[\lambda, 1, \mu_0, \mu_1, b] \overset{\triangle}{\approx}_{\varepsilon_0(\lambda)+\varepsilon_1(\lambda)} \mathbf{Hyb}^4_{\mathsf{A}}[\lambda, \mu_0, \mu_1, b]\,,$$

and, by applying Lemma 3.7,

$$\begin{aligned}&\Delta\Big(\mathbf{Tamper}^{\mathsf{BL}}_{\mathsf{A}}[\lambda, 1, \mu_0, \mu_1, 0]\,, \mathbf{Tamper}^{\mathsf{BL}}_{\mathsf{A}}[\lambda, 1, \mu_0, \mu_1, 1]\Big) \\ &\qquad\qquad \leq 2\,(\varepsilon_0(\lambda) + \varepsilon_1(\lambda)) + \varepsilon_2(\lambda).\end{aligned}$$

□

3.1.4 Instantiation

Using an existing construction of bounded leakage-resilient secret sharing scheme against joint leakage under adaptive partitioning, we obtain the following.

Corollary 3.8. *For every $\ell, n, \lambda \in \mathbb{N}$, every $k_1 \in O(\sqrt{\log(n)})$ and every access structure $\mathcal{A}$ for n parties that can be described by a polynomial-size monotone span program for which authorized sets have size greater than $k = k_1^2 + 1$, there exists a secret sharing scheme with the following parameters, where the message space $\mathcal{M}$ is such that $\#\mathcal{M} \in \Omega(\lambda / \log(\lambda))$.*

Secret sharing scheme SS	
Parties :	n
AccessStructure :	$\mathcal{A}$
MessageSpace :	$\mathcal{M}$
LeakageModel :	Bounded(ℓ)
NonMalleability :	Statistical, $2^{-\Omega(\lambda/\log(\lambda))}$
TamperingAttempts :	1
Partitioning :	SemiAdaptive(k_1)

Proof. It suffices to instantiate $\mathsf{SS}_0, \mathsf{SS}_1, \mathsf{SS}_2$ in Theorem 3.1. Using [KMS19, Theorem 1] and [KMS19, Corollary 2], we can instantiate SS_0 and SS_1 with parameters $\varepsilon_0(\lambda) = \varepsilon_1(\lambda) = 2^{-\Omega(\lambda/\log(\lambda))}, k \in O(\log(n)), s_0 = \mathcal{P}\mathrm{oly}(\lambda)$ and any $\ell_0, \ell_1 \in \mathbb{N}$, which also allows for $k_1 \in O(\sqrt{\log(n)})$. As for SS_2, we can use the split-state non-malleable code in [Li17, Theorem 1.12], which achieves error $2^{-\Omega(\lambda/\log(\lambda))}$. □

3.2 A technique for multiple tampering attempts

In Section 3.1, we constructed a non-malleable secret sharing scheme that is able to withstand a single tampering attempt. In this section, we see how to extend the security of *any* one-time non-malleable secret sharing scheme against joint tampering into a scheme that is secure against multiple tampering attempts. Here, we revisit a compiler from [OPVV18] in the setting of non-malleable secret sharing against joint tampering.

The basic idea is as follows. First, we commit to the message μ using random coins ρ, thus obtaining a cryptographic commitment γ. Then, we secret share the string $\mu \| \rho$ using an auxiliary secret sharing scheme SS_0, thus obtaining shares $\sigma_1, \ldots, \sigma_n$. The final shares are of the form $\sigma_i^* := (\gamma, \sigma_i)$. The reconstruction procedure first checks that all the commitments are equal, then uses the shares σ_i^* to recover $\mu \| \rho$ and finally verifies that $\mu \| \rho$ opens to γ; if any of these checks fails, the reconstruction procedure outputs $\perp$.

The original analysis in [OPVV18] shows that if SS_0 is a bounded leakage-resilient statistically one-time non-malleable secret sharing scheme satisfying additional properties, then the resulting scheme SS is continuously non-malleable. A follow up work [BFV19] shows that the aditional properties can be avoided if SS_0 satisfies a stronger notion of noisy leakage, and further extends the original analysis to any number of parties and to arbitrary access structures.

However, both the proofs in [OPVV18, BFV19] are for the setting of independent tampering. The main idea behind the proof is to reduce the security of SS to the one of SS_0 by using bounded leakage to simulate multiple tampering queries. Unfortunately, this requires a small amount of leakage for each tampering query, and thus the analysis only works in case the number of tampering queries is bounded *a priori.*

```
Algorithm: $\mathsf{SS}.\mathsf{Share}$
    Input: $\mu \in \mathcal{M}$
           $\rho \leftarrow\!\!\$\; \mathcal{R}$
           $\gamma \leftarrow \mathsf{Com}.\mathsf{Commit}(\mu; \rho)$
           $(\sigma_1, \dots, \sigma_n) \leftarrow\!\!\$\; \mathsf{SS}_0.\mathsf{Share}(\mu \| \rho)$
           $\forall\, i \in [n] : \sigma_i^* := (\gamma, \sigma_i)$
   Output: $(\sigma_1^*, \dots, \sigma_n^*)$

Algorithm: $\mathsf{SS}.\mathsf{Reconstruct}$
    Input: $(\mathcal{I}, \sigma_{\mathcal{I}}^*) \in \mathcal{A} \times \mathcal{S}$
           $\forall\, i \in \mathcal{I} : (\gamma_i, \sigma_i) := \sigma_i^*$
           IF $\exists i_1, i_2 \in \mathcal{I} : \gamma_{i_1} \neq \gamma_{i_2}$ : RETURN $\perp$
           $\gamma := \gamma_i$
           $\mu \| \rho \leftarrow \mathsf{SS}_0.\mathsf{Reconstruct}(\sigma_{\mathcal{I}})$
           IF $\mu \| \rho = \perp$ : RETURN $\perp$
           IF $\mathsf{Com}.\mathsf{Commit}(\mu; \rho) \neq \gamma$ : RETURN $\perp$
   Output: $\mu$
```

Figure 3.2. The Share and Reconstruct algorithms of our construction.

We show the compiler in Fig. 3.2, and we prove its security in the theorem below.

Theorem 3.9. *Let $\lambda, \ell, p \in \mathbb{N}$ be parameters and let $\mathcal{A}$ be an arbitrary access structure for n parties without singletons. In the construction depicted in Fig. 3.2, assume that*

- Com *is a perfectly binding and computationally hiding non-interactive commitment scheme;*
- SS_0 *is a ℓ-bounded leakage resilient one-time non-malleable secret sharing scheme realizing access structure $\mathcal{A}$ with statistical security against k-joint semi-adaptive partitioning and message space $\mathcal{M}$ such that $\#\mathcal{M} \in \omega(\log(\lambda))$.*

Then, SS *is a p-time non-malleable secret sharing scheme realizing access structure $\mathcal{A}$ with computational security against k-joint semi-adaptive partitioning, as long as $\ell = p \cdot (\log \#\mathcal{C} + n)$, where $\mathcal{C}$ is the space of the commitments.*

The proof of privacy was already given in [BFV19]; in what follows, we focus on showing non-malleability against joint tampering and semi-adaptive partitioning.

The proof strategy is to construct an hybrid experiment in which we replace the secret sharing of $\mu \| \rho$ with a secret sharing of a random and independent $\hat{\mu} \| \hat{\rho} \leftarrow\!\!\$\; \mathcal{M} \times \mathcal{R}$. Both the original and the hybrid experiments are depicted in Fig. 3.3, in which we also expanded the definition of SS. We first prove that the above experiments are computationally close by induction over the number $p' \leq p$ of tampering queries asked by A; then, a final reduction to the computationally hiding property of Com concludes the proof. The lemma below constitutes the basis of the induction.

```
Differences:  $\mathbf{Tamper}^{\mathsf{BL}}_{\mathsf{A}}[\lambda, p, \mu_0, \mu_1, b]$   $\mathbf{Hyb}_{\mathsf{A}}[\lambda, p, \mu_0, \mu_1, b]$

              $\rho \leftarrow\!\!\$\ \mathcal{R}$
              $\gamma \leftarrow \mathsf{Com}.\mathsf{Commit}(\mu_b; \rho)$
              $(\sigma_1, \ldots, \sigma_n) \leftarrow\!\!\$\ \mathsf{SS}_0.\mathsf{Share}(\mu_b \| \rho)$
              $\hat{\mu} \| \hat{\rho} \leftarrow\!\!\$\ \mathcal{M} \times \mathcal{R}$
              $(\sigma_1, \ldots, \sigma_n) \leftarrow\!\!\$\ \mathsf{SS}_0.\mathsf{Share}(\hat{\mu} \| \hat{\rho})$
              $\forall\ i \in [n] : \sigma^*_i := (\gamma, \sigma_i)$
              $\sigma^* := (\sigma^*_1, \ldots, \sigma^*_n)$
              $b^* \leftarrow\!\!\$\ \mathsf{A}^{\mathcal{O}_{\mathsf{L}}(\sigma,\cdot,\cdot), \mathcal{O}^{p}_{\mathsf{T}}((\mu_0,\mu_1),\sigma,\cdot,\cdot,\cdot)}$
              $b^* \leftarrow\!\!\$\ \mathsf{A}^{\mathcal{O}_{\mathsf{L}}(\sigma,\cdot,\cdot), \mathcal{O}^{p,\star}_{\mathsf{T}}((\mu_0,\mu_1),\sigma,\cdot,\cdot,\cdot)}$
Output:       $b^*$

Differences:  $\mathcal{O}^{p}_{\mathsf{T}}((\mu_0, \mu_1), \sigma, \mathcal{T}, \mathcal{B}, f)$   $\mathcal{O}^{p,\star}_{\mathsf{T}}((\hat{\kappa}, \kappa), (\mu_0, \mu_1), \sigma, \mathcal{T}, \mathcal{B}, f)$
Initial
state:        $p^* := 0, \mathsf{SD} := 0$

              IF $\mathsf{sd} = 1$ : RETURN $\perp$
              IF $p^* = p$ : RETURN $\perp$
              $p^* \leftarrow p^* + 1$
              $\forall\ i \in \mathcal{I} : (\gamma_i, \sigma_i) := \sigma^*_i$
              IF $\exists i_1, i_2 \in \mathcal{I} : \gamma_{i_1} \neq \gamma_{i_2}$ : RETURN $\perp$
              $\gamma := \gamma_i$
              $\mu \| \rho \leftarrow \mathsf{SS}_0.\mathsf{Reconstruct}(\sigma_{\mathcal{I}})$
              IF $\mu \| \rho = \perp$ : RETURN $\perp$
              IF $\mathsf{Com}.\mathsf{Commit}(\mu; \rho) \neq \gamma$ : RETURN $\perp$
              IF $\tilde{\mu} \in \{\mu_0, \mu_1\}$ : RETURN $\heartsuit$
              IF $\tilde{\mu} = \hat{\mu}$ :
                 IF $\tilde{\gamma} = \gamma$ : RETURN $\heartsuit$
                 ELSE : RETURN $\perp$
Output:       $\tilde{\mu}$
```

Figure 3.3. The differences between the original experiment $\mathbf{Tamper}^{\mathsf{L}}$ and the hybrid experiment **Hyb**.

Lemma 3.10. *For all $\mu_0, \mu_1 \in \mathcal{M}$ and all $b \in \{0,1\}$,*

$$\mathbf{Tamper}_{\mathsf{A}}^{\mathsf{BL}}[\lambda, 1, \mu_0, \mu_1, b] \overset{s}{\approx} \mathbf{Hyb}_{\mathsf{A}}[\lambda, 1, \mu_0, \mu_1, b]$$

Proof. By reduction to the statistical leakage-resilient one-time non-malleability of the underlying scheme SS_0. Suppose towards contradiction that there exist two distinct messages $\mu_0, \mu_1 \in \mathcal{M}$ and an *unbounded* adversary A which is able to cause a non-negligible (in λ) statistical difference between the experiments $\mathbf{Tamper}_{\mathsf{A}}^{\mathsf{BL}}[\lambda, 1, \mu_0, \mu_1, b]$ and $\mathbf{Hyb}_{\mathsf{A}}[\lambda, 1, \mu_0, \mu_1, b]$. By an averaging argument, this means that there must exist values $\rho \in \mathcal{R}$ and $\hat{\mu}\|\hat{\rho} \in \mathcal{M} \times \mathcal{R}$ such that A causes the non-negligible statistical difference between the experiments when we fix these particular values in the experiments described in Fig. 3.3. Let $\hat{\mu}_0 := \mu_b\|\rho$, $\hat{\mu}_1 := \hat{\mu}\|\hat{\rho}$ and $\gamma = \mathsf{Com}.\mathsf{Commit}(\mu_0; \rho)$. Finally, let $\sigma = (\sigma_1, \dots, \sigma_n)$ be the target secret sharing of either $\hat{\mu}_0$ or $\hat{\mu}_1$. Without loss of generality, we can assume that A is deterministic.[3]

Consider the following reduction R.

1. Run $b^* \leftarrow\$ \mathsf{A}^{\mathsf{R}}(1^\lambda)$, simulating the oracles as follows.
   - **Tampering oracle:** upon input $(\mathcal{T}, \mathcal{B}, f)$:
     (a) Parse $(\mathcal{B}_1, \dots, \mathcal{B}_m) = \mathcal{B}$ and $(f_1, \dots, f_m) = f$.
     (b) For $i \in [m]$, construct the following leakage function.

```
Function: $\hat{h}_i$
   Input: $\sigma_{\mathcal{B}_i}$
          $(\tilde{\gamma}_j, \tilde{\sigma}_j)_{j \in \mathcal{B}_i} \leftarrow f_i((\gamma, \sigma_j)_{j \in \mathcal{B}_i})$
          IF $\exists j_1, j_2 \in \mathcal{B}_i : \tilde{\gamma}_{j_1} \neq \tilde{\gamma}_{j_2}$ : RETURN $\perp$
          $\tilde{\gamma} := \tilde{\gamma}_j$
  Output: $\tilde{\gamma}$
```

     (c) Query $\tilde{\gamma}_{[m]} \leftarrow \mathcal{O}_{\mathsf{BL}}\left(\mathcal{B}, \hat{h}\right)$, where $\hat{h} = (\hat{h}_1, \dots, \hat{h}_m)$.
     (d) If there exist $i_1, i_2 \in [m]$ such that $\tilde{\gamma}_{i_1} \neq \tilde{\gamma}_{i_2}$, return $\perp$.
     (e) Let $\tilde{\gamma} = \tilde{\gamma}_i$ for some $i \in [m]$.
     (f) Hard-code the value γ in f.
     (g) Query $\tilde{\mu}\|\tilde{\rho} \leftarrow \mathcal{O}_{\mathsf{T}}^{1}(\mathcal{T}, \mathcal{B}, f)$; then:
       - if $\tilde{\mu}\|\tilde{\rho} = \perp$ or $\tilde{\gamma} \neq \mathsf{Com}.\mathsf{Commit}(\tilde{\mu}; \tilde{\rho})$, return $\perp$;
       - if $\tilde{\mu} \in \{\mu_0, \mu_1\}$, return $\heartsuit$;
       - if $\tilde{\mu} = \hat{\mu}$, return $\heartsuit$ if $\tilde{\gamma} = \gamma$ and return $\perp$ otherwise;
       - otherwise, return $\tilde{\mu}$.
2. Output b^*.

For the analysis, we claim that our reduction is perfect with overwhelming probability. More in detail, we only need to check that the answer to A's tampering query is simulated correctly with all but negligible probability. Indeed:

[3] It is always possible to fix the random coins of A in order to maximize its distinguishing advantage.

- If $\mathsf{SS}_0.\mathsf{Reconstruct}(\mathcal{T}, \sigma_{\mathcal{T}}) = \bot$, both the real and the hybrid experiment would return $\bot$, which is perfectly emulated by the reduction.

- If $\mathsf{SS}_0.\mathsf{Reconstruct}(\mathcal{T}, \sigma_{\mathcal{T}}) = \heartsuit$, then the inner secret sharing $\sigma_{\mathcal{T}}$ reconstructs to either $\hat{\mu}_0 = \mu_0 \| \rho$ or $\hat{\mu}_1 = \hat{\mu} \| \hat{\rho}$. Let $\tilde{\gamma}$ be the value of the commitment in the tampered shares. Notice that this value is well defined since all the tampered shares agree on a single commitment; if it was not the case, then the output of the reconstruction would have been $\bot$. Then, there are 4 possible cases: either both experiments output the same $\hat{\mu}_0$ or $\hat{\mu}_1$, or one of the experiments outputs $\hat{\mu}_0$ and the other one outputs $\hat{\mu}_1$.

  We can further reduce the number of cases by considering the fact that the view of A in the original experiment is independent of the value $\hat{\mu}$, except with negligible probability $2^{-\omega(\log(\lambda))} = 1/\#\mathcal{M}$. Therefore, we just consider the following two cases:

  1. Both the real and the hybrid experiment return $\hat{\mu}_0 = \mu_0 \| \rho$.
  2. The real experiment returns $\hat{\mu}_0 = \mu_0 \| \rho$ and the hybrid experiment returns $\hat{\mu}_1 = \hat{\mu} \| \hat{\rho}$.

  In both cases, the output of the two experiment is equal to $\heartsuit$ if $\tilde{\gamma} = \gamma$ and is equal to $\bot$ otherwise. However, this situation is perfectly captured by the reduction, hence the simulation in this case is perfect except with negligible probability.

- If $\mathsf{SS}_0.\mathsf{Reconstruct}(\mathcal{T}, \sigma_{\mathcal{T}}) = \tilde{\mu} \| \tilde{\rho} \notin \{\heartsuit, \bot\}$, it means that, in particular, $\tilde{\mu} \| \tilde{\rho} \notin \{\hat{\mu}_0, \hat{\mu}_1\}$. In this case, both experiments return $\bot$ in case the reconstructed commitment $\tilde{\gamma}$ does not match the opening $\tilde{\mu}, \tilde{\rho}$ and, if not, it means that A produced valid shares of a message $\tilde{\mu} \in \mathcal{M}$. In the latter case, both experiments would output $\heartsuit$ if $\tilde{\mu} \in \{\mu_0, \mu_1\}$ and $\tilde{\mu}$ otherwise, which is perfectly captured by the adversary.

It only remains to discuss the admissibility of R.

The reduction R only performs leakage in step 1c, for a total of $m \cdot |\tilde{\gamma}|$ bits. However, it is possible to make R slightly more leakage-efficient in the following way: instead of leaking the value of $\tilde{\gamma}$ from each subset, the reduction fixes a particular index i^* and only outputs the value of $\tilde{\gamma}$ from the shares in $\mathcal{B}_{i^*}$. Then, R performs a leakage query of 1 bit from each other subset $\mathcal{B}_i$, obtaining, e.g. 1 if the reconstructed commitment in $\mathcal{B}_i$ matches $\tilde{\gamma}$, and 0 otherwise. With this optimization, R only leaks $|\tilde{\gamma}| + (m-1)$ bits to compute step 1c and, since $m \leq n$, R is leakage admissible for $|\tilde{\gamma}| + n - 1$ bits of leakage. The lemma follows. □

The lemma below constitutes the induction step.

Lemma 3.11. *For $p' < p$, all $\mu_0, \mu_1 \in \mathcal{M}$ and all $b \in \{0,1\}$, assume that*

$$\mathbf{Tamper}_{\mathsf{A}}^{\mathsf{BL}}[\lambda, p', \mu_0, \mu_1, b] \overset{\Delta}{\approx} \mathbf{Hyb}_{\mathsf{A}}[\lambda, p', \mu_0, \mu_1, b] .$$

Then,

$$\mathbf{Tamper}_{\mathsf{A}}^{\mathsf{BL}}[\lambda, p'+1, \mu_0, \mu_1, b] \overset{\Delta}{\approx} \mathbf{Hyb}_{\mathsf{A}}[\lambda, p'+1, \mu_0, \mu_1, b] .$$

Proof. Again, by reduction to the statistical leakage-resilient one-time non-malleability of the underlying scheme SS_0. Suppose towards contradiction that there exist two distinct messages $\mu_0, \mu_1 \in \mathcal{M}$ and an *unbounded* adversary A which is able to cause a non-negligible (in λ) statistical difference between the experiments $\mathbf{Tamper}^{\mathsf{BL}}_{\mathsf{A}}[\lambda, p'+1, \mu_0, \mu_1, b]$ and $\mathbf{Hyb}_{\mathsf{A}}[\lambda, p'+1, \mu_0, \mu_1, b]$. By an averaging argument, this means that there must exist values $\rho \in \mathcal{R}$ and $\hat{\mu} \| \hat{\rho} \in \mathcal{M} \times \mathcal{R}$ such that A causes the non-negligible statistical difference between the experiments when we fix these particular values in the experiments described in Fig. 3.3. Let $\hat{\mu}_0 := \mu_b \| \rho$, $\hat{\mu}_1 := \hat{\mu} \| \hat{\rho}$ and $\gamma = \mathsf{Com}.\mathsf{Commit}(\mu_0; \rho)$. Finally, let $\sigma = (\sigma_1, \ldots, \sigma_n)$ be the target secret sharing of either $\hat{\mu}_0$ or $\hat{\mu}_1$. Without loss of generality, we can assume that A is deterministic.

Consider the following reduction R.

1. Run $b^* \leftarrow\!\!\$\ \mathsf{A}^{\mathsf{R}}(1^\lambda)$, simulating the oracles as follows.
   - **Tampering oracle, query $q \le p'$:** upon input $(\mathcal{T}^{(q)}, \mathcal{B}^{(q)}, f^{(q)})$:
     (a) Parse $(\mathcal{B}^{(q)}_1, \ldots, \mathcal{B}^{(q)}_{m^{(q)}}) = \mathcal{B}^{(q)}$ and $(f^{(q)}_1, \ldots, f^{(q)}_{m^{(q)}}) = f^{(q)}$.
     (b) For $i \in [m^{(q)}]$, construct the following leakage function.

```
        Function:  $\hat{h}^{(q)}_i$
           Input:  $\sigma_{\mathcal{B}^{(q)}_i}$
                   $(\tilde{\gamma}_j, \tilde{\sigma}_j)_{j \in \mathcal{B}^{(q)}_i} \leftarrow f_i\left((\gamma, \sigma_j)_{j \in \mathcal{B}^{(q)}_i}\right)$
                   IF $\exists j_1, j_2 \in \mathcal{B}^{(q)}_i : \tilde{\gamma}_{j_1} \neq \tilde{\gamma}_{j_2}$ : RETURN $\bot$
                   $\tilde{\gamma} := \tilde{\gamma}_j$
          Output:  $\tilde{\gamma}$
```

     (c) Query $\tilde{\gamma}_{[m^{(q)}]} \leftarrow \mathcal{O}_{\mathsf{BL}}\left(\mathcal{B}^{(q)}, \hat{h}^{(q)}\right)$, where $\hat{h}^{(q)} = (\hat{h}^{(q)}_1, \ldots, \hat{h}^{(q)}_m)$.
     (d) If there exist $i_1, i_2 \in [m^{(q)}]$ such that $\tilde{\gamma}^{(q)}_{i_1} \neq \tilde{\gamma}^{(q)}_{i_2}$, return $\bot$.
     (e) Let $\tilde{\gamma}^{(q)} = \tilde{\gamma}^{(q)}_i$ for some $i \in [m^{(q)}]$.
     (f) Find by brute force the only opening $\tilde{\mu}^{(q)}, \tilde{\rho}^{(q)}$ of $\tilde{\gamma}^{(q)}$.
     (g) Replace $\tilde{\mu}^{(q)}$ as follows:
       – if $\tilde{\mu}^{(q)} \in \{\mu_0, \mu_1\}$, replace $\tilde{\mu}^{(q)} \leftarrow \heartsuit$;
       – if $\tilde{\mu}^{(q)} = \hat{\mu}$, replace $\tilde{\mu}^{(q)} \leftarrow \heartsuit$ if $\tilde{\gamma}^{(q)} = \gamma$ and $\tilde{\mu}^{(q)} \leftarrow \bot$, triggering a self-destruct, otherwise;
       – otherwise, leave $\tilde{\mu}^{(q)}$ as it is.
     (h) Return $\tilde{\mu}^{(q)}$.
   - **Tampering oracle, query $q = p'+1$:** upon input $(\mathcal{T}^{(q)}, \mathcal{B}^{(q)}, f^{(q)})$:
     (a) Parse $(\mathcal{B}^{(q)}_1, \ldots, \mathcal{B}^{(q)}_{m^{(q)}}) = \mathcal{B}^{(q)}$ and $(f^{(q)}_1, \ldots, f^{(q)}_{m^{(q)}}) = f^{(q)}$.
     (b) Check that the simulation up to the first p' queries did not cause any inconsistency due to the fact that the outcome of the q'-th tampering query should have been $\bot$ because the result $\tilde{\sigma}_{\mathcal{T}^{(q')}}$ of the tampering was not a valid secret sharing.
       i. Without loss of generality, assume that $1 \in \mathcal{T}^{(p'+1)}$ (it is always possible to permute the indices) and that A outputs 0 whenever

it believes that the target secret sharing is distributed as in the real experiment.

ii. Define the special set $\hat{\mathcal{S}} \subseteq \mathcal{S}_1 \times \ldots \times \mathcal{S}_n$ such that $\hat{\mathcal{S}}$ contains all the possible secret sharings of μ_0 and μ_1 that are compatible with the answer to the tampering queries being $\tilde{\mu}^{(1)}, \ldots, \tilde{\mu}^{(p')}$.

iii. Define the following special leakage function $\hat{h}_{\mathsf{check}} : \mathcal{S}_1 \to \{0,1\}$.

- The function hard-wires a description of A, the values γ, μ_0, μ_1, a description of the final tampering query $(\mathcal{T}^{(p'+1)}, \mathcal{B}^{(p'+1)}, f^{(p'+1)})$, the answer to all the previous queries $\tilde{\mu}^{(1)}, \ldots, \tilde{\mu}^{(p')}$ and the set $\hat{\mathcal{S}}$.
- Let $\hat{\sigma}^* := ((\gamma, \sigma_1), (\gamma, \hat{\sigma}_2) \ldots, (\gamma, \hat{\sigma}_n))$ be the target secret sharing for each of the possible set of compatible shares $(\sigma_1, \hat{\sigma}_2, \ldots, \hat{\sigma}_n) \in \hat{\mathcal{S}}$.
- The output of $\hat{h}_{\mathsf{check}}$ is a bit $\tilde{b}$ such that $\tilde{b} = 1$ if and only if $\mathsf{A}(\tilde{\mu}^{(1)}, \ldots, \tilde{\mu}^{(p')}, \tilde{\mu}^*) = 0$ more often when $\hat{\sigma}^*$ is a valid secret sharing of the message μ_0, where

$$\tilde{\mu}^* \leftarrow \mathcal{O}_{\mathsf{T}}\Big((\mu_0, \mu_1), \hat{\sigma}^*, \mathcal{T}^{(p'+1)}, \mathcal{B}^{(p'+1)}, f^{(p'+1)}\Big).$$

iv. Query $\tilde{b} \leftarrow \mathcal{O}_{\mathsf{BL}}\Big((\{1\}, \ldots, \{n\}), (\hat{h}_{\mathsf{check}}, \epsilon, \ldots, \epsilon)\Big)$, where ϵ is the empty string.

(c) For $i \in [m^{(q)}]$, construct the leakage function $\hat{h}_i^{(q)}$ as in the previous tampering queries.

(d) Query $\tilde{\gamma}_{[m^{(q)}]} \leftarrow \mathcal{O}_{\mathsf{BL}}\Big(\mathcal{B}^{(q)}, \hat{h}^{(q)}\Big)$, where $\hat{h}^{(q)} = (\hat{h}_1^{(q)}, \ldots, \hat{h}_m^{(q)})$.

(e) If there exist $i_1, i_2 \in [m^{(q)}]$ such that $\tilde{\gamma}_{i_1}^{(q)} \neq \tilde{\gamma}_{i_2}^{(q)}$, return $\perp$.

(f) Let $\tilde{\gamma}^{(q)} = \tilde{\gamma}_i^{(q)}$ for some $i \in [m^{(q)}]$.

(g) Hard-code the value γ in $f^{(q)}$.

(h) Query $\tilde{\mu}^{(q)} \| \tilde{\rho}^{(q)} \leftarrow \mathcal{O}_{\mathsf{T}}^1\Big(\mathcal{T}^{(q)}, \mathcal{B}^{(q)}, f^{(q)}\Big)$; then:

- if $\tilde{\mu}^{(q)} \| \tilde{\rho}^{(q)} = \perp$ or $\tilde{\gamma}^{(q)} \neq \mathsf{Com}.\mathsf{Commit}\Big(\tilde{\mu}^{(q)}; \tilde{\rho}^{(q)}\Big)$, return $\perp$;
- if $\tilde{\mu}^{(q)} \in \{\mu_0, \mu_1\}$, return $\heartsuit$;
- if $\tilde{\mu}^{(q)} = \hat{\mu}$, return $\heartsuit$ if $\tilde{\gamma}^{(q)} = \gamma$ and return $\perp$ otherwise;
- otherwise, return $\tilde{\mu}^{(q)}$.

(i) Return $\tilde{\mu}^{(q)}$.

2. If $\tilde{b} = 1$, output b^*; else output 0.

The reduction R runs in exponential time. For the analysis, we get that

$$\begin{aligned}
&\left|\mathbb{P}\left[\mathbf{Tamper}^{\mathsf{BL}}_{\mathsf{R}}[\lambda, 1, \mu_0, \mu_1, 0] = 1\right] - \mathbb{P}\left[\mathbf{Tamper}^{\mathsf{BL}}_{\mathsf{R}}[\lambda, 1, \mu_0, \mu_1, 1] = 1\right]\right| \\
&= \left|\mathbb{P}\left[\mathbf{Tamper}^{\mathsf{BL}}_{\mathsf{R}}[\lambda, 1, \mu_0, \mu_1, 0] = 1 \wedge \tilde{b} = 1\right]\right. && (3.3)\\
&\qquad \left.-\mathbb{P}\left[\mathbf{Tamper}^{\mathsf{BL}}_{\mathsf{R}}[\lambda, 1, \mu_0, \mu_1, 1] = 1 \wedge \tilde{b} = 1\right]\right| \\
&\geq \frac{1}{\mathsf{Poly}(\lambda)}\left|\mathbb{P}\left[\mathbf{Tamper}^{\mathsf{BL}}_{\mathsf{R}}[\lambda, 1, \mu_0, \mu_1, 1] = 1 \middle| \tilde{b} = 1\right]\right. && (3.4)\\
&\qquad \left.-\mathbb{P}\left[\mathbf{Tamper}^{\mathsf{BL}}_{\mathsf{R}}[\lambda, 1, \mu_0, \mu_1, 1] = 1 \middle| \tilde{b} = 1\right]\right| \\
&\geq \frac{1}{\mathsf{Poly}(\lambda)}\left(\frac{1}{\mathsf{Poly}(\lambda)} - \mathsf{Negl}(\lambda)\right), && (3.5)
\end{aligned}$$

where:

- (3.3) follows because when $\tilde{b} = 0$ the reduction R always returns 0 unconditionally, and this cancels its distinguishing advantage.
- (3.4) holds for the following reason. The induction hypothesis states that A is not able to cause a non-negligible difference between the two experiments in p' queries, while the contradiction hypothesis states that A is able to cause a non-negligible difference between the two experiments in $p' + 1$ queries. This implies that $\tilde{b} = 1$ with non-negligible probability: otherwise, A would cause a self-destruction earlier in the experiment, thus causing a non-negligible difference in less than $p' + 1$ tampering queries.
- (3.5) holds because an analysis identical to the one in Lemma 3.10 shows that the view of A is perfectly simulated (except with negligible probability) when $\tilde{b} = 1$, and thus R retains essentially the same advantage of A.

In order to conclude the proof, it remains to show that R is ℓ-leakage admissible, for ℓ as in the statement of the theorem, and that R is $\mathsf{SemiAdaptive}(k)$-admissible.

For the leakage, notice that R makes leakage queries in steps 1c, 1(b)iv, 1d. The leakage queries of steps 1c and 1d are the same used to obtain the tampered commitment in Lemma 3.10 and hence the same optimization can be applied; since there are $p' + 1$ such queries, R performs $(p' + 1)(|\tilde{\gamma}| + n - 1)$ bits of leakage. Furthermore, R performs 1 bit of leakage in the query in step 1(b)iv.

Finally, R converts all the tampering queries from A into leakage queries using the same partition, except that R performs no leakage from the subsets that do not intersect the reconstruction set $\mathcal{T}^{(q)}$. Therefore, if A is $\mathsf{SemiAdaptive}(k)$-admissible, for any two $\mathcal{B}_i^{(q)}, \mathcal{B}_{i'}^{(q')}$, either $\mathcal{B}_i^{(q)} = \mathcal{B}_{i'}^{(q')}$ or $\mathcal{B}_i^{(q)} \cap \mathcal{B}_{i'}^{(q')} = \emptyset$, and the same holds when converting the tampering queries into leakage queries; hence, R is $\mathsf{SemiAdaptive}(k)$-admissible as well. This concludes the proof of the lemma. $\square$

Finally, the lemma below allows to conclude the proof of Theorem 3.9.

Lemma 3.12. *For all $\mu_0, \mu_1 \in \mathcal{M}$,*

$$\mathbf{Hyb}^{\mathsf{BL}}_{\mathsf{A}}[\lambda, p, \mu_0, \mu_1, 0] \overset{c}{\approx} \mathbf{Hyb}_{\mathsf{A}}[\lambda, p, \mu_0, \mu_1,]\,.$$

Proof. By reduction to the computationally hiding property of Com. Suppose towards contradiction that there exist two distinct messages μ_0, μ_1 and a PPT adversary A which is able to cause a non-negligible difference between the experiments $\mathbf{Hyb}^{\mathsf{BL}}_{\mathsf{A}}[\lambda, p, \mu_0, \mu_1, 0]$ and $\mathbf{Hyb}_{\mathsf{A}}[\lambda, p, \mu_0, \mu_1,]$.

Let $b \in \{0,1\}, \rho \in \mathcal{R}$ and let $\hat{\gamma} = \mathsf{Com}.\mathsf{Commit}(\mu_b; \rho)$ be the target commitment. Consider the following reduction R.

1. Compute $(\sigma_1, \dots, \sigma_n) \leftarrow\$ \mathsf{SS}_0.\mathsf{Share}(\hat{\mu} \| \hat{\rho})$, where $\hat{\mu} \| \hat{\rho} \leftarrow\$ \mathcal{M} \times \mathcal{R}$.
2. Run $b^* \leftarrow\$ \mathsf{A}^{\mathsf{R}}(1^\lambda)$, simulating the oracles as follows.
   - **Tampering oracle, query q:** upon input $(\mathcal{T}^{(q)}, \mathcal{B}^{(q)}, f^{(q)})$:
     (a) Parse $(\mathcal{B}^{(q)}_1, \dots, \mathcal{B}^{(q)}_{m^{(q)}}) = \mathcal{B}^{(q)}$ and $(f^{(q)}_1, \dots, f^{(q)}_{m^{(q)}}) = f^{(q)}$.
     (b) For all $i \in [m^{(q)}]$, compute
     $$\tilde{\sigma}^*_{\mathcal{B}_i} = (\tilde{\gamma}_j, \tilde{\sigma}_j)_{j \in \mathcal{B}_i} \leftarrow f^{(q)}_i\left((\hat{\gamma}, \sigma_j)_{j \in \mathcal{B}_i}\right)$$
     and let $\tilde{\mu}^{(q)} \| \tilde{\rho}^{(q)} \leftarrow \mathsf{SS}.\mathsf{Reconstruct}\left(\tilde{\sigma}_{\mathcal{T}^{(q)}}\right)$.
     (c) If there exist $i_1, i_2 \in \mathcal{T}^{(q)}$ such that $\tilde{\gamma}_{i_1} \neq \tilde{\gamma}_{i_2}$, return $\perp$ triggering self-destruct.
     (d) If $\tilde{\mu}^{(q)} = \perp$ or $\tilde{\gamma}_1 \neq \mathsf{Com}.\mathsf{Commit}\left(\tilde{\mu}^{(q)}, \tilde{\rho}^{(q)}\right)$, return $\perp$ triggering self-destruct.
     (e) If $\tilde{\mu}^{(q)} \in \{\mu_0, \mu_1\}$, return $\heartsuit$.
     (f) If $\tilde{\mu}^{(q)} = \hat{\mu}$, return $\heartsuit$ if $\tilde{\gamma}^{(q)} = \hat{\gamma}$ and return $\perp$ triggering self-destruct otherwise.
     (g) Else, return $\tilde{\mu}^{(q)}$.
3. Output b^*.

For the analysis, note that the simulation done by R is perfect. In particular, depending on the value $\hat{\gamma}$ being either a commitment to μ_0 or to μ_1, the view of A is identical to the one in either experiment $\mathbf{Hyb}^{\mathsf{BL}}_{\mathsf{A}}[\lambda, p, \mu_0, \mu_1, 0]$ or $\mathbf{Hyb}_{\mathsf{A}}[\lambda, p, \mu_0, \mu_1,]$. Therefore R distinguishes with the same advantage of A. □

Proof of Theorem 3.9. The proof follows by the following chain.

$$\begin{aligned}
\mathbf{Tamper}^{\mathsf{BL}}_{\mathsf{A}}[\lambda, p', \mu_0, \mu_1, 0] &\overset{\Delta}{\approx} \mathbf{Hyb}_{\mathsf{A}}[\lambda, p, \mu_0, \mu_1, 0] \\
&\overset{c}{\approx} \mathbf{Hyb}_{\mathsf{A}}[\lambda, p, \mu_0, \mu_1,] \\
&\overset{\Delta}{\approx} \mathbf{Tamper}^{\mathsf{BL}}_{\mathsf{A}}[\lambda, p', \mu_0, \mu_1, 1]
\end{aligned}$$

□

3.3 Achieving continuous non-malleability

The same technique used in Section 3.2 can be used to achieve the even stronger notion of *continuous* non-malleability. By taking a closer look at the proof technique, we just need a way to extract the common commitment γ from each subset of tampered shares. Unfortunately, this cannot be achieved in the bounded leakage model, nor without further assumptions on the scheme SS_0 in Fig. 3.2. However, by "opening" the black-box, it is possible to overcome this difficulty and obtain the first secret sharing scheme achieving joint noisy leakage resilience and continuous non-malleability in the plain model.

More in detail, we use a modified version of a construction by Goyal, Srinivasan and Zhu [GSZ20], in which we are able to obtain noisy-leakage resilience and two additional properties that are necessary to the proof of security. The details of our construction can be found in Section 3.3.1, while an intuition of the proof can be foun in Section 3.3.2. Finally, the formal proof appears in Section 3.3.3 and an instantiation appears in Section 3.3.4

3.3.1 Our construction

The modified version uses, internally, a 2-split-state non-malleable code (a.k.a. 2-out-of-2 non-malleable secret sharing scheme) SS_2 with asymmetric shares (i.e. $\#\mathcal{S}_{2,0} \neq \#\mathcal{S}_{2,1}$) and two additional properties, listed below.

1. There exists $\hat{\sigma}_0 \in \mathcal{S}_{2,0}$ such that, for all $\mu \in \mathcal{M}$, there exists $\sigma_1 \in \mathcal{S}_{2,1}$ such that $\mathsf{SS}_2.\mathsf{Reconstruct}([2], (\sigma_0, \sigma_1)) = \mu$.

2. There exists $d \in \mathbb{N}$ such that, for all $\mu \in \mathcal{M}$ and all $i \in \{0,1\}$,

$$\widetilde{\mathbb{H}}_\infty(\mathbf{\Sigma}_i|\mathbf{\Sigma}_{1-i}) \geq \mathbb{H}_\infty(\mathbf{\Sigma}_i) - d,$$

   where $\mathbf{\Sigma}_i$ is the random variable for share σ_i. Intuitively, d is a bound on how much does the left (resp. right) share reveal on the right (resp. left) share. This notion is usually known as *d-conditional independence*.

Furthermore, we require the leakage parameters of SS_2 to be different for the two shares, i.e. there exist $\ell_{2,0}, \ell_{2,1} \in \mathbb{N}$ such that given a leakage string $\mathbf{\Lambda}$ from an admissible adversary, it must hold that, for all $i \in \{0,1\}$,

$$\widetilde{\mathbb{H}}_\infty(\mathbf{\Sigma}_i|\mathbf{\Lambda}) \geq \mathbb{H}_\infty(\mathbf{\Sigma}_i) - \ell_i.$$

The construction uses two other secret sharing schemes SS_1 and SS_3. We will instantiate both these schemes as Shamir's secret sharing with *extended* shares. Namely, let $\mu \in \mathbb{F}$ be the shared secret and let p be the randomly sampled polynomial for the sharing procedure. Then, a share $\sigma_i \in \mathcal{S}_i = \mathbb{F}^2$ is the pair $(i, p(i))$ instead of just $p(i)$, and we define the function Party that, upon input $\sigma_i = (i, p(i))$, returns i.

We are now ready to describe our construction. We describe the scheme SS_0 in Fig. 3.4. Our construction is the one in Fig. 3.2 instantiated with the scheme SS_0. The theorem below proves that this construction achieves continuous non-malleability.

Theorem 3.13. *Let $n, t, \ell^*, \ell_1, \ell_2 \in \mathbb{N}$ be parameters with $t > 2n/3$. In the construction depicted in Fig. 3.4, assume that*

- SS_1 *is a t-out-of-n Shamir's secret sharing with extended shares.*

```
Algorithm: $\mathsf{SS}_0.\mathsf{Share}$
    Input: $\mu \in \mathcal{M}$
           $(\sigma_1, \ldots, \sigma_n) \leftarrow\$ \mathsf{SS}_1.\mathsf{Share}(\mu)$
           $\forall\, i \in [n] : (\sigma_{i,0}, \sigma_{i,1}) \leftarrow\$ \mathsf{SS}_2.\mathsf{Share}(\sigma_i)$
           $\forall\, i \in [n] : (\sigma_{i,1,1}, \ldots, \sigma_{i,1,n}) \leftarrow\$ \mathsf{SS}_3.\mathsf{Share}(\sigma_{i,1})$
           $\forall\, i \in [n] : \sigma_i^* := (\sigma_{i,0}, (\sigma_{1,1,i}, \ldots, \sigma_{n,1,i}))$
   Output: $(\sigma_1^*, \ldots, \sigma_n^*)$

Algorithm: $\mathsf{SS}_0.\mathsf{Reconstruct}$
    Input: $(\mathcal{I}, \sigma_{\mathcal{I}}^*) \in \mathcal{A} \times \mathcal{S}$
           $\forall\, i \in \mathcal{I} : (\sigma_{i,0}, (\sigma_{1,1,i}, \ldots, \sigma_{n,1,i})) := \sigma_i^*$
           $\forall\, i \in \mathcal{I} : \sigma_{i,1} \leftarrow \mathsf{SS}_3.\mathsf{Reconstruct}(\mathcal{I}, \sigma_{i,1,\mathcal{I}})$
           $\forall\, i \in \mathcal{I} : \sigma_i \leftarrow \mathsf{SS}_2.\mathsf{Reconstruct}([2], (\sigma_{i,0}, \sigma_{i,1}))$
           IF $\exists i_1, i_2 \in \mathcal{I} : \mathsf{Party}(\sigma_{i_1}) = \mathsf{Party}(\sigma_{i_2})$ : RETURN $\perp$
           $\mu \leftarrow \mathsf{SS}_1.\mathsf{Reconstruct}(\mathcal{I}, \sigma_{\mathcal{I}})$
   Output: $\mu$
```

Figure 3.4. The Share and $\mathsf{Reconstruct}$ algorithms of our construction.

- SS_2 *is an asymmetric* 2*-out-of-*2 (ℓ_1, ℓ_2)*-noisy leakage resilient* t*-time non-malleable secret sharing scheme with statistical security and satisfying the additional properties 1 and 2.*

- SS_3 *is a* t*-out-of-*n *Shamir's secret sharing with extended shares.*

Then, the scheme SS *depicted in Fig. 3.2, instantiated with the scheme* SS_0 *depicted in Fig. 3.4, is a* t*-out-of-*n ℓ^**-noisy-leakage resilient continuously non-malleable code against* $(t-1)$*-joint selective partitioning, as long as*

$$\ell_1 \geq (t-1) \cdot \ell^* + \log \#\mathcal{M} + \log \#\mathcal{C} + d + 1 + \log(\lambda),$$
$$\ell_2 \geq \ell^* + n \cdot (t-1) \cdot \log \#\mathcal{S}_{2,1} + \log \#\mathcal{C} + d + 1 + \log(\lambda).$$

3.3.2 Proof overview

Intuitively, we would like to prove our scheme to be secure by using the results in Section 3.2. Unfortunately, this is not possible in a straightforward way, because our proof strategy requires some additional properties from the underlying schemes. Instead, we open the black-box of SS_0 and we reduce to the security of one of its components. Namely, we would like to reduce to the security of SS_2.

Clearly, this cannot be done with a single hybrid experiment; instead, we construct n hybrid experiments in which we gradually replace the original shares with dummy, randomly sampled shares. Now, when stepping from one hybrid to the next one, we can finally reduce to the security of SS_2 and use the same technique as in Section 3.2.

In this case, continuous non-malleability is made possible by the *conditional independence* property. Roughly speaking, conditional independence says that one of the shares does not reveal too much information on the other share. Here we use the fact that the commitment to the message is prepended to each share: if the commitment values are different, then the secret sharing is invalid; otherwise, it means that the tampered commitment $\tilde{\gamma}$ leaked from $\boldsymbol{\Sigma}_{r,0}$ can also be deterministically computed from $\boldsymbol{\Sigma}_{r,1}$. This reasoning can be extended to several commitments $\tilde{\gamma}^{(1)}, \tilde{\gamma}^{(2)}, \tilde{\gamma}^{(3)}, \ldots$: indeed, until the commitments satisfy $\tilde{\gamma}_0^{(q)} = \tilde{\gamma}^{(q)}$, all of them can be deterministically computed from $\boldsymbol{\Sigma}_{r,0}$, therefore they do not reveal much information from $\boldsymbol{\Sigma}_{r,0}$ as well.

Finally, the last reduction to the computationally hiding property of the commitment scheme is identical to the one in Section 3.2.

3.3.3 Security analysis

In what follows, we make the simplifying assumption that the partition $\mathcal{B}$ used by the attacker only contains two subsets $\mathcal{B}_1, \mathcal{B}_2$. Notice that this restriction is without loss of generality whenever $t > 2n/3$, as in the hypothesis of Theorem 3.13. Indeed, for any partition $\mathcal{B} = (\mathcal{B}_1, \ldots, \mathcal{B}_m)$, it is always possible to find a set $\mathcal{I} \subseteq [m]$ such that

$$\frac{t}{2} \leq \# \bigcup_{i \in \mathcal{I}} \mathcal{B}_i \leq t$$

and then consider the two subsets to be $\hat{\mathcal{B}}_1 := \bigcup_{i \in \mathcal{I}} \mathcal{B}_i$, containing less than t elements by construction, and $\hat{\mathcal{B}}_2 := \bigcup_{i \notin \mathcal{I}} \mathcal{B}_i$, containing at most $n - \#\hat{\mathcal{B}}_1 < 3t/2 - t/2 = t$ elements.

For $r \in [n], p \in \mathbb{N}$, we consider the auxiliary hybrid experiment $\boldsymbol{Hyb}^r[\lambda, p, \mu_0, \mu_1, b]$, described in Fig. 3.5 along with the original experiment in which we expanded the applications of SS and SS_0. Namely, in $\mathbf{Hyb}^r$ we replace the first r shares $(\sigma_1, \ldots, \sigma_r)$ from SS_1 with random and independent values $(\sigma'_1, \ldots, \sigma'_r)$, letting the remaining shares $\sigma'_{r+1}, \ldots, \sigma'_n$ be the same as in the original experiment. Notice that, when $r = 0$, no shares are replaced and, therefore, $\mathbf{Hyb}^r = \mathbf{Tamper}^{\mathsf{NL}}$.

Similarly to the proof in Section 3.2, we will prove by induction that the two experiments are statistically close. The lemma below constitutes the basis of the induction.

Lemma 3.14. *For all $\mu_0, \mu_1 \in \mathcal{M}$, all $b \in \{0, 1\}$ and all $r \in [n]$,*

$$\mathbf{Hyb}^{r-1}[\lambda, 1, \mu_0, \mu_1, b] \overset{\triangle}{\approx} \mathbf{Hyb}^r[\lambda, 1, \mu_0, \mu_1, b] \,.$$

Proof. The difference between the two hybrid experiments lies in the distribution to which is sampled the share σ'_r. Namely, σ'_r is uniformly random in $\mathbf{Hyb}^r$ and it comes from the real share σ_r in $\mathbf{Hyb}^{r-1}$. For $j \in [n]$, let $\beta(j) \in \{1, 2\}$ be the index such that $j \in \mathcal{B}_{\beta(j)}$. The proof proceeds by reduction to leakage-resilient t-time non-malleability of SS_2. Assume that there exists an adversary A which is able to cause a non-negligible difference between the experiments $\mathbf{Hyb}^r$ and $\mathbf{Hyb}^{r-1}$.

Let σ_r and σ'_r be the challenge messages for SS_2 as sampled in $\mathbf{Hyb}^r$, and let $\gamma \leftarrow\!\!\$\ \mathsf{Com}.\mathsf{Commit}(\mu_b)$. Consider the reduction R defined as follows.

- **Shared randomness.** For every $i \in [n] \setminus \{r\}$, sample $\sigma_{i,0}, (\sigma_{i,1,j})_{j \in [n]}$ as in $\mathbf{Hyb}^r$. Then, let $i_0 = \beta(r)$ and $i_1 = 3 - i_0$ (i.e. $r \in \mathcal{B}_{i_0}$ and the other subset is $\mathcal{B}_{i_1}$). Let $\mathcal{J}$ be any set of at least $t - 1$ indices such that $\mathcal{B}_{i_0} \subseteq \mathcal{J} \subseteq [n]$. For all $j \in \mathcal{J}$, sample the shares $\sigma_{r,1,j}$ uniformly at random. Finally, sample the

```
Differences:   $\mathbf{Tamper}^{\mathsf{NL}}_{\mathsf{A}}[\lambda, p, \mu_0, \mu_1, b]$   $\mathbf{Hyb}^{r}_{\mathsf{A}}[\lambda, p, \mu_0, \mu_1, b]$
               $\rho \leftarrow\$ \mathcal{R}$
               $\gamma \leftarrow \mathsf{Com}.\mathsf{Commit}(\mu_b; \rho)$
               $(\sigma_1, \dots, \sigma_n) \leftarrow\$ \mathsf{SS}_1.\mathsf{Share}(\mu_b \| \rho)$
               $(\sigma'_1, \dots, \sigma'_n) \leftarrow\$ \mathcal{S}_{1,1} \times \dots \times \mathcal{S}_{1,n}$
               $\forall\ i > r : \sigma'_i \leftarrow \sigma_i$
               $\forall\ i \in [n]$ :
                  $(\sigma_{i,0}, \sigma_{i,1}) \leftarrow\$ \mathsf{SS}_2.\mathsf{Share}(\sigma_i)$
                  $(\sigma_{i,0}, \sigma_{i,1}) \leftarrow\$ \mathsf{SS}_2.\mathsf{Share}(\sigma'_i)$
                  $(\sigma_{i,1,1}, \dots, \sigma_{i,1,n}) \leftarrow\$ \mathsf{SS}_3.\mathsf{Share}(\sigma_{i,1})$
                  $\sigma^*_i := (\gamma, \sigma_{i,0}, (\sigma_{1,1,1}, \dots, \sigma_{n,1,1}))$
               $\sigma^* := (\sigma^*_1, \dots, \sigma^*_n)$
               $b^* \leftarrow\$ \mathsf{A}^{\mathcal{O}_{\mathsf{L}}(\sigma,\cdot,\cdot), \mathcal{O}^{p}_{\mathsf{T}}((\mu_0,\mu_1),\sigma^*,\cdot,\cdot,\cdot)}$
               $b^* \leftarrow\$ \mathsf{A}^{\mathcal{O}_{\mathsf{L}}(\sigma,\cdot,\cdot), \mathcal{O}^{p,\star}_{\mathsf{T}}((\mu_0,\mu_1),\sigma^*,\cdot,\cdot,\cdot)}$
Output:        $b^*$

Differences:   $\mathcal{O}^{p}_{\mathsf{T}}((\mu_0, \mu_1), \sigma, \mathcal{T}, \mathcal{B}, f)$   $\mathcal{O}^{p,\star}_{\mathsf{T}}((\mu_0, \mu_1), \sigma, \mathcal{T}, \mathcal{B}, f)$
Initial        $p^* := 0, \mathsf{SD} := 0$
state:
               IF $\mathsf{SD} = 1$ : RETURN $\perp$
               IF $p^* = p$ : RETURN $\perp$
               $p^* \leftarrow p^* + 1$
               $(\mathcal{B}_1, \dots, \mathcal{B}_m) := \mathcal{B}$
               $(f_1, \dots, f_m) := f$
               $\forall\ i \in [m] : \tilde{\sigma}^*_{\mathcal{B}_i} = f_i\left(\sigma^*_{\mathcal{B}_i}\right)$
               $\forall\ i \in \mathcal{T} : (\tilde{\gamma}, \tilde{\sigma}_{i,0}, (\tilde{\sigma}_{1,1,1}, \dots, \tilde{\sigma}_{n,1,1})) := \tilde{\sigma}^*_i$
               IF $\exists i_1, i_2 \in \mathcal{I} : \tilde{\gamma}_{i_1} \neq \tilde{\gamma}_{i_2}$ : $\mathsf{SD} \leftarrow 1$; RETURN $\perp$
               $\tilde{\gamma} := \tilde{\gamma}_i$
               $\forall\ i \in \mathcal{T} : \tilde{\sigma}_{i,1} \leftarrow \mathsf{SS}_3.\mathsf{Reconstruct}(\mathcal{T}, \tilde{\sigma}_{i,1,\mathcal{T}})$
               $\forall\ i \in \mathcal{T} : \tilde{\sigma}_i \leftarrow \mathsf{SS}_2.\mathsf{Reconstruct}([2], (\tilde{\sigma}_{i,0}, \tilde{\sigma}_{i,1}))$
               IF $\exists i_1, i_2 \in \mathcal{T} : \mathsf{Party}(\tilde{\sigma}_{i_1}) = \mathsf{Party}(\tilde{\sigma}_{i_2})$ :
                  $\mathsf{SD} \leftarrow 1$; RETURN $\perp$
               $\forall\ i_1, i_2 \in \mathcal{T} : \tilde{\sigma}_{i_1} = \sigma'_{i_2} : \tilde{\sigma}_{i_1} \leftarrow \sigma_{i_2}$
               $\tilde{\mu} \| \tilde{\rho} \leftarrow \mathsf{SS}_0.\mathsf{Reconstruct}(\tilde{\sigma}_{\mathcal{I}})$
               IF $\tilde{\mu} \| \tilde{\rho} = \perp$ : $\mathsf{SD} \leftarrow 1$; RETURN $\perp$
               IF $\mathsf{Com}.\mathsf{Commit}(\tilde{\mu}; \tilde{\rho}) \neq \tilde{\gamma}$ : $\mathsf{SD} \leftarrow 1$; RETURN $\perp$
               IF $\tilde{\mu} \in \{\mu_0, \mu_1\}$ : RETURN $\heartsuit$
Output:        $\tilde{\mu}$
```

Figure 3.5. The differences between the original experiment $\mathbf{Tamper}^{\mathsf{NL}}$ and the hybrid experiment $\mathbf{Hyb}^{r}$.

left share $\hat{\sigma}_0$ given by property 1 of SS_2. At this point, $\hat{\sigma}_0$ and the following values are known:

$$\begin{aligned}
\forall i \in [n] \setminus \mathcal{J} : &\quad \gamma, &\quad \sigma_{i,0}, &\quad (\sigma_{j,1,i})_{j\in[n]\setminus\{r\}}\,, &\quad (3.6)\\
\forall i \in \mathcal{J} \setminus \{r\} : &\quad \gamma, &\quad \sigma_{i,0}, &\quad (\sigma_{j,1,i})_{j\in[n]}\,, &\quad (3.7)\\
i = r : &\quad \gamma, &\quad &\quad (\sigma_{j,1,i})_{j\in[n]}\,, &\quad (3.8)
\end{aligned}$$

and the following values are missing:

$$\begin{aligned}
\forall i \in [n] \setminus \mathcal{J} : &\quad &\quad \sigma_{r,1,i},\\
\forall i \in \mathcal{J} \setminus \{r\} : &\quad &\quad \\
i = r : &\quad \sigma_{i,0}. &\quad
\end{aligned}$$

- Run $b^* \leftarrow \mathsf{A}^{\mathsf{R}}(1^\lambda)$, simulating the oracles as follows.
  - **Leakage oracle:** upon input $((\mathcal{B}_1, \mathcal{B}_2), (g_1, g_2))$, construct the following leakage functions.
    * Let $\hat{g}_0$ be the leakage function which takes as input the value $\sigma_{r,0}$, plugs it in (3.8) and outputs $\Lambda_{i_0} \leftarrow g_{i_0}\left(\sigma^*_{\mathcal{B}_{i_0}}\right)$. Recall that $\mathcal{B}_{i_0} \subseteq \mathcal{J}$, hence $\hat{g}_0$ has all the necessary shares to compute the leakage function.
    * Let $\hat{g}_1$ be the leakage function which takes as input the value $\sigma_{r,1}$, computes the values $(\sigma_{r,1,i})_{i\in[n]\setminus\mathcal{J}}$ from $\sigma_{r,1}$ and $(\sigma_{r,1,i})_{i\in\mathcal{J}}$ and plugs them in (3.6); then, once obtained the values in $(\sigma^*_i)_{i\in[n]\setminus\mathcal{J}}$, appends the values in (3.7) to obtain $(\sigma^*_i)_{i\in\mathcal{B}_{i_1}}$ and finally outputs $\Lambda_{i_1} \leftarrow g_{i_1}\left(\sigma^*_{\mathcal{B}_{i_1}}\right)$.

    Then, query $\Lambda_1\|\Lambda_2 \leftarrow \mathcal{O}_{\mathsf{NL}}((\mathcal{B}_1, \mathcal{B}_2), (\hat{g}_1, \hat{g}_2))$ and return $\Lambda_1\|\Lambda_2$.
  - **Tampering oracle:** upon input $(\mathcal{T}, (\mathcal{B}_1, \mathcal{B}_2), (f_1, f_2))$, construct the following leakage functions.
    * Let $\hat{h}_0$ be the leakage function that computes $\sigma^*_{\mathcal{B}_{i_0}}$ as in $\hat{g}_0$, computes $\tilde{\sigma}^*_{\mathcal{B}_{i_0}} = f_{i_0}\left(\sigma^*_{\mathcal{B}_{i_0}}\right)$, checks whether all the tampered commitments $\tilde{\gamma}_i$ within $\mathcal{T} \cap \mathcal{B}_{i_0}$ agree on a single value $\tilde{\gamma}_0$ (and output $\perp$ otherwise) and finally outputs the values $\tilde{\gamma}_0, (\tilde{\sigma}_{j,1,i})_{i\in\mathcal{B}_{i_0}, j\in\mathcal{T}}$.
    * Let $\hat{h}_1$ be the leakage function that computes $\sigma^*_{\mathcal{B}_{i_1}}$ as in $\hat{g}_1$, computes $\tilde{\sigma}^*_{\mathcal{B}_{i_1}} = f_{i_1}\left(\sigma^*_{\mathcal{B}_{i_1}}\right)$, checks whether all the tampered commitments $\tilde{\gamma}_i$ within $\mathcal{T} \cap \mathcal{B}_{i_1}$ agree on a single value $\tilde{\gamma}_1$ (and output $\perp$ otherwise) and finally outputs $\tilde{\gamma}_1$.

    Then, query $\tilde{\gamma}_0, (\tilde{\sigma}_{j,1,i})_{i\in\mathcal{B}_{i_0}, j\in\mathcal{T}}, \tilde{\gamma}_1 \leftarrow \mathcal{O}_{\mathsf{NL}}\Big((\mathcal{B}_1, \mathcal{B}_2), (\hat{h}_1, \hat{h}_2)\Big)$. Return $\perp$ if $\tilde{\gamma}_0 \neq \tilde{\gamma}_1$ or some of the received values is $\perp$ and otherwise let $\tilde{\gamma} := \tilde{\gamma}_0 = \tilde{\gamma}_1$. After that, construct the following tampering functions.
    * For all $i \in \mathcal{T}$, let $\hat{f}_{i,0}$ be the function that computes $\sigma^*_{\mathcal{B}_{i_0}}$ as in $\hat{g}_0$, computes $\tilde{\sigma}^*_{\mathcal{B}_{i_0}} = f_{i_0}\left(\sigma^*_{\mathcal{B}_{i_0}}\right)$ and outputs $\tilde{\sigma}_{i,0}$ if $i \in \mathcal{B}_{i_0}$ and $\hat{\sigma}_0$ otherwise.

  - For all $i \in \mathcal{T}$, let $\hat{f}_{i,1}$ be the function that computes $\sigma^*_{\mathcal{B}_{i_1}}$ as in $\hat{g}_1$, computes $\tilde{\sigma}^*_{\mathcal{B}_{i_1}} = f_{i_1}\left(\sigma^*_{\mathcal{B}_{i_1}}\right)$ and uses them, along with the previously leaked values, to reconstruct $\tilde{\sigma}_{i,1}$ and finally outputs $\tilde{\sigma}_{i,1}$ if $i \in \mathcal{B}_{i_1}$ and a share $\hat{\sigma}_1$ such that $\mathsf{SS}_2.\mathsf{Reconstruct}([2], (\hat{\sigma}_0, \hat{\sigma}_1)) = \mathsf{SS}_2.\mathsf{Reconstruct}([2], (\tilde{\sigma}_{i,0}, \tilde{\sigma}_{i,1}))$ otherwise.

 Then, query $\tilde{\sigma}_i \leftarrow \mathcal{O}^1_{\mathsf{T}}\left(\hat{f}_{i,0}, \hat{f}_{i,1}\right)$ for all $i \in \mathcal{T}$, and:

  - replace $\tilde{\sigma}_i$ with σ_r if $\tilde{\sigma}_i = \heartsuit$;
  - replace $\tilde{\sigma}_i$ with σ_j if there exists $j \in [n]$ such that $\tilde{\sigma}_i = \sigma'_j$;
  - return $\perp$ if there exist distinct $i_1, i_2 \in \mathcal{T}$ such that $\mathsf{Party}(\sigma_{i_1}) = \mathsf{Party}(\sigma_{i_2})$.

 Finally, reconstruct $\tilde{\mu}\|\tilde{\rho} \leftarrow \mathsf{SS}_1.\mathsf{Reconstruct}(\mathcal{T}, \tilde{\sigma}_{\mathcal{T}})$, check that $\tilde{\gamma} = \mathsf{Com}.\mathsf{Commit}(\tilde{\mu}; \tilde{\rho})$ (and return $\perp$ otherwise), replace $\tilde{\mu} \leftarrow \heartsuit$ if $\tilde{\mu} \in \{\mu_0, \mu_1\}$ and return $\tilde{\mu}$.

- Output b^*.

For the analysis, call BAD_i the event that one tampering query modifies the shares so that the tampered value $(\tilde{\sigma}_{i,0}, \tilde{\sigma}_{i,1})$ is a valid encoding of σ'_r. Since σ'_r is uniformly random, the probability of BAD_i in $\mathbf{Hyb}^{r-1}$ is $O(2^{-\lambda})$. Furthermore, R perfectly simulates $\mathbf{Hyb}^{r-1}$ if the target codeword encodes σ_r and $\text{BAD} = \bigcup_i \text{BAD}_i$ does not happen. On the other hand, the reduction perfectly simulates $\mathbf{Hyb}^r$ if the target codeword encodes σ'_r. Indeed, the auxiliary information leaked by the functions $\hat{h}_0, \hat{h}_1$ allows R to correctly compute the tampering query, yielding a perfect simulation.

To conclude the proof, it only remains to show that the constraints on the leakage hold. The reduction performs the same leakage performed by A, plus the one performed by the functions $\hat{h}_0, \hat{h}_1$, obtaining two commitments $\tilde{\gamma}_0, \tilde{\gamma}_1$ and the tampered shares $(\tilde{\sigma}_{j,1,i})_{i \in \mathcal{B}_{i_0}, j \in \mathcal{T}}$. The overall amout of leakage is thus

$$\ell^* + \#\mathcal{B}_{i_0} \cdot \#\mathcal{T} \cdot s_1 + |\tilde{\gamma}| \leq \ell^* + (t-1) \cdot t \cdot s_1 \leq \ell_0$$

bits from the left side, where $s_1 = \log \#\mathcal{S}_1$, and

$$\ell^* + |\tilde{\gamma}| \leq \ell_1$$

bits from the right side. The lemma follows. □

The lemma below constitutes the inductive step.

Lemma 3.15. *For $p' \in \mathbb{N}$, all $r \in [n]$, all $\mu_0, \mu_1 \in \mathcal{M}$ and all $b \in \{0,1\}$, assume that*

$$\mathbf{Hyb}^{r-1}[\lambda, p, \mu_0, \mu_1, b] \overset{\Delta}{\approx} \mathbf{Hyb}^r[\lambda, p, \mu_0, \mu_1, b]\,.$$

Then,

$$\mathbf{Hyb}^{r-1}[\lambda, p+1, \mu_0, \mu_1, b] \overset{\Delta}{\approx} \mathbf{Hyb}^r[\lambda, p+1, \mu_0, \mu_1, b]\,.$$

Proof. Again, he difference between the two hybrid experiments lies in the distribution to which is sampled the share σ'_r. Namely, σ'_r is uniformly random in $\mathbf{Hyb}^r$ and it comes from the real share σ_r in $\mathbf{Hyb}^{r-1}$. For $j \in [n]$, let $\beta(j) \in \{1,2\}$ be the index such that $j \in \mathcal{B}_{\beta(j)}$. The proof proceeds by reduction to leakage-resilient t-time non-malleability of SS_2. Assume that there exists an adversary A which is able to cause a non-negligible difference between the experiments $\mathbf{Hyb}^r$ and $\mathbf{Hyb}^{r-1}$.

Let σ_r and σ'_r be the challenge messages for SS_2 as sampled in $\mathbf{Hyb}^r$, and let $\gamma \leftarrow\$ \mathsf{Com.Commit}(\mu_b)$. Consider the reduction R defined as follows.

- **Shared randomness.** Everything as in Lemma 3.14. For every $i \in [n] \setminus \{r\}$, sample $\sigma_{i,0}, (\sigma_{i,1,j})_{j\in[n]}$ as in $\mathbf{Hyb}^r$. Then, let $i_0 = \beta(r)$ and $i_1 = 3 - i_0$ (i.e. $r \in \mathcal{B}_{i_0}$ and the other subset is $\mathcal{B}_{i_1}$). Let $\mathcal{J}$ be any set of at least $t-1$ indices such that $\mathcal{B}_{i_0} \subseteq \mathcal{J} \subseteq [n]$. For all $j \in \mathcal{J}$, sample the shares $\sigma_{r,1,j}$ uniformly at random. Finally, sample the left share $\hat{\sigma}_0$ given by property 1 of SS_2. At this point, $\hat{\sigma}_0$ and the following values are known:

$$\forall i \in [n] \setminus \mathcal{J}: \quad \gamma, \quad \sigma_{i,0}, \quad (\sigma_{j,1,i})_{j\in[n]\setminus\{r\}}, \tag{3.9}$$

$$\forall i \in \mathcal{J} \setminus \{r\}: \quad \gamma, \quad \sigma_{i,0}, \quad (\sigma_{j,1,i})_{j\in[n]}, \tag{3.10}$$

$$i = r: \quad \gamma, \quad (\sigma_{j,1,i})_{j\in[n]}, \tag{3.11}$$

  and the following values are missing:

$$\forall i \in [n] \setminus \mathcal{J}: \quad \sigma_{r,1,i},$$
$$\forall i \in \mathcal{J} \setminus \{r\}:$$
$$i = r: \quad \sigma_{i,0}.$$

- Run $b^* \leftarrow \mathsf{A}^{\mathsf{R}}(1^\lambda)$, simulating the oracles as follows.
  - **Leakage oracle:** as in the proof of Lemma 3.14. Call Λ the final transcript corresponding to all leakage queries.
  - **Tampering oracle, query $q < p'$:** upon input $(\mathcal{T}^{(q)}, (\mathcal{B}_1, \mathcal{B}_2), (f_1^{(q)}, f_2^{(q)}))$, construct the following leakage functions.
    * For all $i \in \mathcal{T}$, let $\hat{g}_{i,0}^{(q)}$ be the function that computes $\sigma^*_{\mathcal{B}_{i_0}}$ as in Lemma 3.14, function $\hat{g}_0$, computes $\tilde{\sigma}^*_{\mathcal{B}_{i_0}} = f_{i_0}\left(\sigma^*_{\mathcal{B}_{i_0}}\right)$ and outputs $\tilde{\gamma}$.
    * For all $i \in \mathcal{T}$, let $\hat{g}_{i,1}^{(q)}$ be the function that computes $\sigma^*_{\mathcal{B}_{i_1}}$ as in Lemma 3.14, function $\hat{g}_1$, computes $\tilde{\sigma}^*_{\mathcal{B}_{i_1}} = f_{i_1}\left(\sigma^*_{\mathcal{B}_{i_1}}\right)$ and outputs $\tilde{\gamma}$.

    Then, query $\tilde{\gamma}_0^{(q)}, \tilde{\gamma}_1^{(q)} \leftarrow \mathcal{O}_{\mathsf{NL}}\left((\mathcal{B}_1, \mathcal{B}_2), (\hat{g}_1^{(q)}, \hat{g}_2^{(q)})\right)$, return $\perp$ and trigger self-destruct if $\tilde{\gamma}_0^{(q)} \neq \tilde{\gamma}_1^{(q)}$ or one of the commitment equals bottom, and otherwise find by brute force the correct opening $\tilde{\mu}^{(q)}$ to the above commitment as in Lemma 3.11, replace $\tilde{\mu}^{(q)} \leftarrow \heartsuit$ if $\tilde{\mu}^{(q)} \in \{\mu_0, \mu_1\}$ and finally return $\tilde{\mu}^{(q)}$ (or $\perp$ if no opening is found).
  - **Tampering oracle, query $q = p' + 1$:** upon input $(\mathcal{T}^{(q)}, (\mathcal{B}_1, \mathcal{B}_2), (f_1^{(q)}, f^{(q)})_2)$, check that the simulation up to the first p' queries did not cause any inconsistency due to the fact that the outcome of the q'-th query should have been $\perp$ because the result $\tilde{\sigma}_{\mathcal{T}^{(q)}}$ of the tampering was not a valid secret sharing. This step is very similar to the corresponding one in

the proof of Lemma 3.11, with the only difference that the consistency check is run for the scheme SS_2.

After the consistency check sets a bit $\tilde{b} = 1$ if the simulation results to be consistent and $\tilde{b} = 0$ otherwise, query the same leakage functions $\hat{h}_0, \hat{h}_1$ and tampering function $\hat{f}_{i,0}, \hat{f}_{i,1}$ and compute the same tampered message $\tilde{\mu}$ as in the proof of Lemma 3.14.

- Output b^* if $\tilde{b} = 1$ and output 0 otherwise.

The reduction runs in exponential time. An analysis identical to the one in the proof of Lemma 3.15 shows that the simulation is perfect except with negligible probability and that R retains essentially the same advantage of A.

In order to conclude the proof, it remains to show that R is (ℓ_0, ℓ_1)-leakage admissible for ℓ_0, ℓ_1 as in the statement of the theorem. The reduction R makes leakage queries in the following cases:

1. by forwarding the leakage queries of A, for a total of ℓ^* bits;
2. when obtaining the tampered commitments $\tilde{\gamma}_0^{(q)}, \tilde{\gamma}_1^{(q)}$, for a total of $|\tilde{\gamma}_i|$ bits from each share and for each query;
3. when obtaining the same auxiliary information as in Lemma 3.14 to answer the last tampering query, for a total of at most $t \cdot (t-1) \cdot s_1 + 1$ bits;
4. for running the consistency check, for a total of 1 bit.

The d-conditional independence property (i.e. property 2) of SS_2 allows to minimize the cost of the leakage performed in Item 2. Let $q^* \in \mathbb{N}$ be the index of the tampering query, if any, where either $\tilde{\gamma}_0^{(q^*)} \neq \tilde{\gamma}_1^{(q^*)}$ or at least one of the two commitments equals $\perp$, and set $q^* = p' + 1$ in case that never happens. Notice that q^* is a random variable, which we denote by $\boldsymbol{q}^*$. Clearly, such leakage queries are executed exactly $\min\{q^*, p'\}$ times. For each $i \in \{0, 1\}$, let $\boldsymbol{\Lambda}_i'$ be the random variable corresponding to the leakage performed by R on the share $\sigma_{r,i}$ of the target secret sharing $\boldsymbol{\Sigma}_{r,0}, \boldsymbol{\Sigma}_{\boldsymbol{r},\mathbf{1}}$, and let $\boldsymbol{\Lambda}_i$ be the random variable corresponding to the leakage performed by A on the same share. We can write:

$$
\begin{aligned}
&\widetilde{\mathbb{H}}_\infty \left(\boldsymbol{\Sigma}_{r,0} | \boldsymbol{\Lambda}_0'\right) \\
&\geq \widetilde{\mathbb{H}}_\infty \left(\boldsymbol{\Sigma}_{r,0} | \boldsymbol{\Sigma}_{r,1}, \boldsymbol{\Lambda}_0'\right) && (3.12)\\
&\geq \widetilde{\mathbb{H}}_\infty \left(\boldsymbol{\Sigma}_{r,0} \middle| \boldsymbol{\Sigma}_{r,1}, \left(\hat{h}_0^{(i)}(\boldsymbol{\Sigma}_{r,0})\right)_{i \in [\boldsymbol{q}^*] \cup \{p'+1\}}, \hat{h}_{\mathsf{check}}(\boldsymbol{\Sigma}_{r,0}), \boldsymbol{\Lambda}_0\right) && (3.13)\\
&\geq \widetilde{\mathbb{H}}_\infty \left(\boldsymbol{\Sigma}_{r,0} \middle| \boldsymbol{\Sigma}_{r,1}, \left(\hat{h}_0^{(i)}(\boldsymbol{\Sigma}_{r,0})\right)_{i \in [\boldsymbol{q}^*] \cup \{p'+1\}}\right) - \ell^* - 1 && (3.14)\\
&\geq \widetilde{\mathbb{H}}_\infty \left(\boldsymbol{\Sigma}_{r,0} \middle| \boldsymbol{\Sigma}_{r,1}, \boldsymbol{q}^*, \hat{h}_0^{(\boldsymbol{q})}(\boldsymbol{\Sigma}_{r,0}), \hat{h}_0^{(p'+1)}(\boldsymbol{\Sigma}_{r,0})\right) - \ell^* - 1 && (3.15)\\
&\geq \widetilde{\mathbb{H}}_\infty \left(\boldsymbol{\Sigma}_{r,0} | \boldsymbol{\Sigma}_{r,1}, \boldsymbol{q}^*\right) - |\tilde{\gamma}| - t \cdot (t-1) \cdot s_1 - \ell^* - 1 && (3.16)\\
&\geq \widetilde{\mathbb{H}}_\infty \left(\boldsymbol{\Sigma}_{r,0} | \boldsymbol{\Sigma}_{r,1}\right) - O(\log(\lambda)) - |\tilde{\gamma}| - t \cdot (t-1) \cdot s_1 - \ell^* - 1 && (3.17)\\
&\geq \widetilde{\mathbb{H}}_\infty \left(\boldsymbol{\Sigma}_{r,0}\right) - d - O(\log(\lambda)) - |\tilde{\gamma}| - t \cdot (t-1) \cdot s_1 - \ell^* - 1, && (3.18)
\end{aligned}
$$

where, in the above derivation:

- (3.12) follows by the fact that further conditioning on another random variable can only *reduce* the conditional average min-entropy;
- (3.13) follows by how the leakage $\mathbf{\Lambda}'_i$ is constructed;
- (3.14) follows from the *chain rule* (i.e. Lemma 2.1) as $\mathbf{\Lambda}_i$ is the leakage perfomed by an ℓ^*-admissible adversary and $\hat{h}_{\mathsf{check}}$ only leaks 1 bit;
- (3.15) holds because, for each $q < q^*$, $\tilde{\gamma}_0^{(q)} = \tilde{\gamma}_1^{(q)}$ and therefore $\tilde{\gamma}_0^{(q)}$ can be computed as a deterministic function of $rv\Sigma_{r,1}$ and $\boldsymbol{q}^*$;
- (3.16) and (3.17) follow again from the *chain rule*, since $|q^*| = O(\log(\lambda))$ and either $\boldsymbol{q}^* = p' + 1$, in which case the reduction leaks all the necessary information to construct the last tampering query for up to $|\tilde{\gamma}| + t \cdot (t-1) \cdot s_1$, or $\boldsymbol{q}^* \leq p'$, in which case only the commitment of length $|\tilde{\gamma}|$ is leaked;
- finally, (3.18) follows by d-conditional independence.

An almost identical analysis holds for the leakage from the right share, with the only difference that we do not leak any auxiliary information from $\mathbf{\Sigma}_{r,1}$ and therefore the min-entropy drop only amounts to $d + O(\log(\lambda)) + |\tilde{\gamma}| + \ell^* + 1$ bits. The lemma follows. □

Finally, we can conclude the proof of the main theorem of this section.

Proof of Theorem 3.13. By applying Lemma 3.14 and Lemma 3.15, we proved that for all $r \in [n]$, all $\mu_0, \mu_1 \in \mathcal{M}$ and all $b \in \{0,1\}$,

$$\mathbf{Hyb}^{r-1}[\lambda, \infty, \mu_0, \mu_1, b] \overset{\Delta}{\approx} \mathbf{Hyb}^{r}[\lambda, \infty, \mu_0, \mu_1, b]\,,$$

ultimately proving that

$$\mathbf{Tamper}^{\mathsf{NL}}[\lambda, \infty, \mu_0, \mu_1, b] \overset{\Delta}{\approx} \mathbf{Hyb}^{n}[\lambda, \infty, \mu_0, \mu_1, b]\,.$$

A proof identical to the one in Theorem 3.9 shows that for all $\mu_0, \mu_1 \in \mathcal{M}$,

$$\mathbf{Hyb}^{n}[\lambda, \infty, \mu_0, \mu_1, 0] \overset{c}{\approx} \mathbf{Hyb}^{n}[\lambda, \infty, \mu_0, \mu_1, 1]\,.$$

The theorem then follows by repeatedly applying the triangular inequality. □

3.3.4 Instantiation

By instantiating Theorem 3.13, we obtain the following.

Corollary 3.16. *Assuming the existence of one-to-one one-way functions, for all $n, t, \ell \in \mathbb{N}$ with $t > 2n/3$, there is a construction of a t-out-of-n secret sharing scheme satisfying noisy-leakage resilient continuous non-malleability under selective k-joint leakage and tampering attacks, where $k = t - 1$.*

Proof. The existence of the inner non-malleable code is given by Corollary A.4, whose proof can be found in Appendix A. Furthermore, perfectly binding and computationally hiding commitment schemes can be instantiated from one-to-one one-way functions [GMW87]. □

Chapter 4

Space efficiency

In this chapter, we discuss about the space efficiency of non-malleable secret sharing schemes.

In Section 4.1, we show two lower bounds on the size of the shares of a non-malleable secret sharing scheme. Our first result shows that every (one-time) non-malleable secret sharing scheme is indeed a leakage-resilient non-malleable secret sharing scheme. The proof of this fact is quite simple: we construct a reduction from leakage-resilient non-malleability to non-malleability, which tries to guess in advance the answer to the leakage queries and then uses the tampering query to check whether the answer was correct or wrong. However, this fact allows to extend a known bound on the size of the shares of leakage-resilient secret sharing schemes [NS20] to the case of non-malleable secret sharing scheme. Our second result is an upper bound on the maximal achievable rate[1] of any *continuously* non-malleable secret sharing scheme against joint tampering with many shares.

In Section 4.2, we show a compiler that transforms any continuously non-malleable secret sharing scheme with poor rate into a continuously non-malleable secret sharing scheme with good rate. Our construction is the same as Krawczyk's in [Kra94], except for two small differences. First, the reconstruction threshold of the information dispersal scheme is smaller than the reconstruction threshold of the secret sharing scheme. This means that we achieve smaller rate as well, but this modification is necessary in order to prove our result. Second, we additionally check that all the information dispersal shares in the selected reconstruction set are consistent with the shared information. Again, this is needed for the security proof to go through.

4.1 Upper and lower bounds

We start by showing the following result that allows to trade off non-malleability for leakage resilience.

Theorem 4.1. *Let $\lambda, \ell = \ell(\lambda) \in \mathbb{N}, \varepsilon = \varepsilon(\lambda) \in [0,1]$ be parameters such that $\ell \leq -\log(\varepsilon)$. Let SS be a statistical one-time non-malleable secret sharing scheme against selective partitioning with statistical security ε. Then, SS is also a statistical*

[1]Which, in turn, is a lower bound on the size of the shares which depends on the size of the shared secret.

ℓ-bounded leakage-resilient one-time non-malleable secret sharing scheme against selective partitioning with statistical security $2^{\ell}\varepsilon$.

Proof. By reduction to non-malleability. Suppose towards contradiction that there exist messages μ_0, μ_1 and an adversary $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$ which is able to cause a statistical difference between the experiment $\mathbf{Tamper}^{\mathsf{BL}}_{\mathsf{A}}[\lambda, 1, \mu_0, \mu_1, 0]$ and the experiment $\mathbf{Tamper}^{\mathsf{BL}}_{\mathsf{A}}[\lambda, 1, \mu_0, \mu_1, 1]$ of more than $2^{\ell}\varepsilon$. The adversary A partitions, internally, the set of the shares into m subsets $\mathcal{B}_1, \ldots, \mathcal{B}_m$. Furthermore, A has access to a leakage oracle accepting queries of the form $(\mathcal{B}, g)$, where $\mathcal{B} = (\mathcal{B}_1, \ldots, \mathcal{B}_m)$ and $g = (g_1, \ldots, g_m)$.

Consider the following reduction $\mathsf{R} = (\mathsf{R}_1, \mathsf{R}_2)$. We just describe the algorithm R_1, letting $\mathsf{R}_2 = \mathsf{A}_2$.

1. Construct the (stateful) oracle $\mathcal{O}'_{\mathsf{BL}}(\cdot, \cdot)$ which, upon input a leakage query $(\mathcal{B}, g)$, where, for all $i \in [m]$, g_i is of the form $g_i : \mathcal{S}_{\mathcal{B}_i} \to \{0,1\}^{\ell'_i}$ for some $\ell'_i \in \mathbb{N}$, outputs a randomly sampled $\Lambda'_i \subseteq \{0,1\}^{\ell'_i}$. The state $(\Lambda, \mathfrak{G})$ of the oracle is an initially empty string. After the output, the string $\Lambda'_1 \| \ldots \| \Lambda'_m$ is appended to Λ and (a description of) the function g is appended to $\mathfrak{G}$.

2. Run $(\alpha; \mathcal{T}, \mathcal{B}, f) \leftarrow\!\!\$\ \mathsf{A}_1^{\mathcal{O}'_{\mathsf{BL}}(\cdot,\cdot)}(1^{\lambda})$.

3. Extract the state $(\Lambda, \mathfrak{G})$ from the oracle, writing $\mathfrak{G} = (g^{(1)}, \ldots, g^{(N)})$ and $\Lambda = (\Lambda^{(1)}, \ldots, \Lambda^{(N)})$, where $N \in \mathbb{N}$ is the number of leakage queries performed by A_1.

4. For all $i \in [m]$, construct the tampering function $\hat{f}_i$ as follows.

```
Function: $\hat{f}_i$
   Input: $\sigma_{\mathcal{B}_i}$
          $\forall\ j \in [N]$ :
              IF $g_i^{(j)}(\sigma_{\mathcal{B}_i}) \neq \Lambda_i^{(j)}$ : RETURN $\perp$
          $\tilde{\sigma}_{\mathcal{B}_i} \leftarrow f_i(\sigma_{\mathcal{B}_i})$
  Output: $\tilde{\sigma}_{\mathcal{B}_i}$
```

5. Output $(\alpha; \mathcal{T}, \mathcal{B}, \hat{f})$, where $\hat{f} = (\hat{f}_1, \ldots, \hat{f}_m)$.

For the analysis, we now compute the distinguishing advantage of R. For $b \in \{0,1\}$, let $\textsc{Hit}_b$ be the event that the guess on the leakage was correct in $\mathbf{Tamper}^{\mathsf{BL}}_{\mathsf{A}}[\lambda, 1, \mu_0, \mu_1, b]$ and let $\textsc{Miss}_b$ be the event that the guess on the leakage was wrong.

First, we notice that, for all $b \in \{0,1\}$,

$$\begin{aligned}
\mathbb{P}[\textsc{Hit}_b] &= \sum_{\Lambda \in \{0,1\}^{\ell}} \mathbb{P}\left[\boldsymbol{U}_{\ell} = \Lambda \wedge g\left(\boldsymbol{\Sigma}^{(b)}\right) = \Lambda\right] \\
&= 2^{-\ell} \sum_{\Lambda \in \{0,1\}^{\ell}} \mathbb{P}\left[g\left(\boldsymbol{\Sigma}^{(b)}\right) = \Lambda\right] \\
&= 2^{-\ell},
\end{aligned}$$

where $\boldsymbol{U}_\ell$ is the uniform random variable over $\{0,1\}^\ell$, $\boldsymbol{\Sigma}^{(b)} = \mathsf{SS}.\mathsf{Share}(\mu_b)$ is the random variable for a secret sharing of message μ_b, and $g(\sigma)$ denotes the overall leakage performed over σ. The above holds because the random variables $\boldsymbol{U}_\ell$ and $\boldsymbol{\Sigma}^{(b)}$ are indepenent, hence the events $\boldsymbol{U}_\ell = \Lambda$ and $\boldsymbol{\Sigma}^{(b)} = \Lambda$ are independent as well. Since $\mathbb{P}[\textsc{Hit}_b]$ does not depend on b, this means that $\mathbb{P}[\textsc{Hit}_0] = \mathbb{P}[\textsc{Hit}_1]$.

Now, we proceed by studying the distinguishing advantage. In order to make notation shorter, for $b \in \{0,1\}|$, let $\boldsymbol{T}[b] := \mathbf{Tamper}_{\mathsf{R}}[\lambda, 1, \mu_0, \mu_1, b]$ and let $\boldsymbol{T}^{\mathsf{BL}}[b] := \mathbf{Tamper}^{\mathsf{BL}}_{\mathsf{A}}[\lambda, 1, \mu_0, \mu_1, b]$. We have:

$$\begin{aligned}
\Delta(\boldsymbol{T}[0], \boldsymbol{T}[1]) &= |\mathbb{P}[\textsc{Hit}_0]\,\mathbb{P}[\boldsymbol{T}[0] = 1|\textsc{Hit}_0] - \mathbb{P}[\textsc{Hit}_1]\,\mathbb{P}[\boldsymbol{T}[1] = 1|\textsc{Hit}_1] &(4.1)\\
&\quad +\mathbb{P}[\textsc{Miss}_0]\,\mathbb{P}[\boldsymbol{T}[0] = 1|\textsc{Miss}_0] - \mathbb{P}[\textsc{Miss}_1]\,\mathbb{P}[\boldsymbol{T}[1] = 1|\textsc{Miss}_1]| \\
&= |\mathbb{P}[\textsc{Hit}_0]\,\mathbb{P}[\boldsymbol{T}[0] = 1|\textsc{Hit}_0] - \mathbb{P}[\textsc{Hit}_1]\,\mathbb{P}[\boldsymbol{T}[1] = 1|\textsc{Hit}_1]| &(4.2)\\
&= 2^{-\ell}\,|\mathbb{P}[\boldsymbol{T}[0] = 1|\textsc{Hit}_0] - \mathbb{P}[\boldsymbol{T}[1] = 1|\textsc{Hit}_1]| &(4.3)\\
&= 2^{-\ell}\left|\mathbb{P}\left[\boldsymbol{T}^{\mathsf{BL}}[0] = 1\right] - \mathbb{P}\left[\boldsymbol{T}^{\mathsf{BL}}[1] = 1\right]\right| &(4.4)\\
&\geq 2^{-\ell}2^{\ell}\varepsilon = \varepsilon
\end{aligned}$$

In the above derivation, (4.1) follows from the law of total probability; (4.2) comes from the fact that, when $\textsc{Miss}_b$ happens, the view of A (i.e. the leakage Λ and the output $\perp$ of the tampering query) is independent of the target secret sharing, and thus its distinguishing advantage is 0; (4.3) follows from $\mathbb{P}[\textsc{Hit}_b] = 2^{-\ell}$; finally, (4.4) holds because, when $\textsc{Hit}_b$ happens, the reduction perfectly simulates the experiment and thus R retains the same distinguishing advantage of A, that is at least $2^\ell\varepsilon$. This implies that R is able to break non-malleability with advantage at least ε, against the hypothesis of security. The proof follows. □

An immediate corollary of the above theorem is a lower bound for the size of the shares. Nielsen and Simkin proved in [NS20] the following result.

Theorem 4.2 ([NS20], Theorem 2)**.** *Let $n, t, \hat{t}, s, \ell \in \mathbb{N}$ be parameters. Let SS be a t-out-of-n secret sharing scheme with statistical security against ℓ bits of bounded leakage* from each share[2]. *Let $\hat{t}$ be the minimum number such that, for all $\mu \in \mathcal{M}$, $\mathsf{SS}.\mathsf{Share}(\mu)$ is uniquely determined by $(\mathsf{SS}.\mathsf{Share}(\mu))_{\mathcal{T}}$ for any $\mathcal{T}$ such that $\#\mathcal{T} \geq \hat{t}$. Finally, for the share space $\mathcal{S}_1 \times \ldots \times \mathcal{S}_n$, let, for all $i \in [n]$, $s = \log \#\mathcal{S}_i$. Then,*

$$s \geq \frac{\ell(n-t)}{\hat{t}}.$$

By applying the above theorem to Theorem 4.1, we achieve the following.

Corollary 4.3. *Let $n, t, \hat{t}, s \in \mathbb{N}, \varepsilon \in [0,1]$ be parameters. Let SS be a t-out-of-n statistical one-time non-malleable secret sharing scheme. Let $\hat{t}$ be the minimum number such that, for all $\mu \in \mathcal{M}$, $\mathsf{SS}.\mathsf{Share}(\mu)$ is uniquely determined by $(\mathsf{SS}.\mathsf{Share}(\mu))_{\mathcal{T}}$ for any $\mathcal{T}$ such that $\#\mathcal{T} \geq \hat{t}$. Finally, for the share space $\mathcal{S}_1 \times \ldots \times \mathcal{S}_n$, let, for all $i \in [n]$, $s = \log \#\mathcal{S}_i$. Then,*

$$s \geq \frac{(\log(1/\varepsilon) - 1)(1 - t/n)}{\hat{t}}.$$

[2] Notice that leakage-resilience against $n\ell$ bits of overall bounded leakage implies leakage-resilience against ℓ bits of leakage from each share; however, the converse is necessarily true.

Notice that $\hat{t}$ is a simplified notion of entropy. If $t = \hat{t}$, then any authorized set is able to reconstruct all the remaining shares, meaning that those shares have no entropy left. An example of a scheme with $t = \hat{t}$ is Shamir's secret sharing, which has been proved to have a small degree of leakage resilience in [BDIR18].

Remark 4.4. *We stated Theorem 4.1 in the information-theoretic setting. Taking a closer look, Theorem 4.1 also works in the computational setting; in this case, however, we are able to prove resilience only against $\ell = O(\log(\lambda))$ bits. Nonetheless, Theorem 4.2 only works in the information-theoretic setting. Indeed, Nielsen and Simkin showed in [NS20] a construction in the computational setting (and under the random oracle assumption) that breaks such bound.*

Finally, we prove the following upper bound.

Theorem 4.5. *Let $n, t, k, \rho \in \mathbb{N}$ be parameters. Let SS be a t-out-of-n continuously non-malleable secret sharing with security against selective k-sized partitioning and rate ρ. If $k > t/2$, then $\rho \leq t - k$.*

Proof. For simplicity, we assume that the attacker always uses the same reconstruction set $\mathcal{T}$ across all the tampering queries. Notice that this allows to prove a slightly stronger statement, since the upper bound holds even with this restriction. Furthermore, we also assume that $\mathsf{SS}.\mathsf{ShareSpace} = \mathcal{S}_1 \times \ldots \times \mathcal{S}_n$ with $\#\mathcal{S}_i = \#\mathcal{S}_j$ for all $i, j \in [n]$; a generalization of this is immediate.

Consier the following commitment scheme Com.

```
      Algorithm:  $\mathsf{Com}.\mathsf{Commit}$
          Input:  $(\mu; \rho) \in \mathcal{M} \times \mathcal{R}$
                  $(\sigma_1, \ldots, \sigma_n) \leftarrow\$ \mathsf{SS}.\mathsf{Share}(\mu; \rho)$
                  $\gamma \leftarrow (\sigma_1, \ldots, \sigma_{t-k})$
         Output:  $\gamma$
```

In a moment we prove that the above commitment scheme is perfectly binding; first, we show why this implies the stated lower bound. Indeed, the fact that Com is a perfectly binding commitment implies that $\mathsf{Com}.\mathsf{Commit}$ is an injective function, hence $|\mu| \leq |\gamma|$. By letting $s = \log \#\mathcal{S}_1$, the rate satisfies

$$\rho = \frac{|\mu|}{s} \leq \frac{|\gamma|}{s} = \frac{|(\sigma_1, \ldots, \sigma_{t-k})|}{s} \leq t - k.$$

Com is perfectly binding. Assume, towards a contradiction, that Com is not perfectly binding. Namely, there exist a commitment γ and two openings $(\mu^{(0)}, \rho_0)$ and $(\mu^{(1)}, \rho_1)$ such that $\mu^{(0)} \neq \mu^{(1)}$ and $\gamma = \mathsf{Com}.\mathsf{Commit}\left(\mu^{(0)}; \rho_0\right) = \mathsf{Com}.\mathsf{Commit}\left(\mu^{(1)}; \rho_1\right)$.

Let $\mu_0^*, \mu_1^* \in \mathcal{M}$ be any two distinct messages, and denote by $\sigma = (\sigma_1, \ldots, \sigma_n)$ the target secret sharing of μ_b^* in the experiment $\mathbf{Tamper}_{\mathsf{R}}[\lambda, \infty, \mu_0^*, \mu_1^*, b]$, where λ is the security parameter. Furthermore, for better readability, let $\ell = |\sigma_{t-k+1}| + \ldots + |\sigma_t|$. Consider the following PPT adversary A against continuous non-malleablilty of SS.

1. Compute, for $b' \in \{0, 1\}$, $(\sigma_0^{(b')}, \ldots, \sigma_n^{(b')}) \leftarrow\$ \mathsf{SS}.\mathsf{Share}\left(\mu^{(b')}; \rho_{b'}\right)$.

2. By hypothesis, we have that, for all $i \in [t-k]$, $\sigma_i^{(0)} = \sigma_i^{(1)}$.
3. Set $\mathcal{T} := [t]$, $\mathcal{B}_1 := [t-k]$, $\mathcal{B}_2 := [t] \setminus [t-k]$, $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$.
4. Let f_1 be the tampering function that always outputs $(\sigma_j^{(0)})_{j \in [t-k]}$.
5. Let, for $j \in [\ell]$, $f_2^{(j)}$ be the tampering function that outputs $(\sigma_j^{(b')})_{j \in \mathcal{B}_2}$ whenever the j-th bit of the string $(\sigma_i)_{i \in \mathcal{B}_2}$ is $b' \in \{0,1\}$.
6. For $j \in [\ell]$, query $\tilde{\mu} \leftarrow \mathcal{O}_{\mathsf{T}}^{\infty}\left(\sigma, \mathcal{T}, \mathcal{B}, f^{(j)}\right)$, where $f^{(j)} = (f_1, f_2^{(j)})$.
7. For $j \in [\ell]$, let $\alpha_j = b'$ if $\tilde{\mu} = \mu^{(b')}$; notice that, by how the queries are constructed, the tampering function always outputs either a secret sharing of $\mu^{(0)}$ or a secret sharing of $\mu^{(1)}$.
8. Parse the string $\alpha_1 \| \dots \| \alpha_\ell$ as $(\sigma_{t-k+1}, \dots, \sigma_t)$.
9. Construct the tampering query $(\mathcal{T}, (f'_1, f'_2))$ where f'_2 is the identity function and f'_1 either acts as the identity or outputs invalid values depending on whether the reconstructed message is μ_0^* or μ_1^*. Notice that f'_1 takes as input the values $(\sigma_i)_{i \in \mathcal{B}_1}$ and can hard-wire the values $(\sigma_i)_{i \in \mathcal{B}_2}$, hence it is able to reconstruct the message.
10. Query $\tilde{\mu} \leftarrow \mathcal{O}_{\mathsf{T}}^{\infty}(\sigma, \mathcal{T}, \mathcal{B}, (f'_1, f'_2))$.
11. Output 0 if $\tilde{\mu} = \heartsuit$ (i.e. f'_1 acted as the identity) and 1 otherwise.

Notice that, since $k > t/2$, $t - k < k$, therefore $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ is a k-sized partition of $\mathcal{T}$. Furthermore, the reduction obtains the missing shares and is able to run the full reconstruction algorithm inside one of the tampering functions, thus breaking continuous non-malleability with overwhelming probability. This concludes the proof. □

4.2 Achieving optimal rate

The following construction needs a t-out-of-n continuously non-malleable secret sharing scheme SS_0, a t^*-out-of-n information dispersal scheme ID and a secret-key encryption scheme SKE. The formal algorithms of the resulting scheme SS is depicted in Fig. 4.1. Informally, the algorithm works as follows. The secret message is initially encrypted using the encryption scheme SKE. Then, the secret key is shared via the poor-rate continuously non-malleable secret sharing scheme, while the ciphertext is shared via the information dispersal scheme. The final share is then the share of the secret sharing along with one share of the information dispersal.

Proof overview. The security proof proceeds by a hybrid argument. In particular, we show that the original $\mathbf{Tamper}_{\mathsf{A}}^{\mathsf{L}}$ is close to an hybrid experiment $\mathbf{Hyb}_{\mathsf{A}}^{1}$ in which we replace the secret sharing of the key with a secret sharing of another unrelated key. Intuitively, this follows by a reduction to the security of the underlying secret sharing scheme; however, we need to take care of a few challenges.

```
Algorithm:  $\mathsf{SS.Share}$
    Input:  $\mu \in \mathcal{M}$
            $\kappa \leftarrow\!\!\$\ \mathcal{K}$
            $\gamma \leftarrow\!\!\$\ \mathsf{SKE.Encrypt}(\kappa, \mu)$
            $(\gamma_1, \ldots, \gamma_n) \leftarrow\!\!\$\ \mathsf{ID.Share}(\gamma)$
            $(\kappa_1, \ldots, \kappa_n) \leftarrow\!\!\$\ \mathsf{SS}_0.\mathsf{Share}(\kappa)$
            $\forall\ i \in [n] : \sigma_i := (\kappa_i, \gamma_i)$
   Output:  $(\sigma_1, \ldots, \sigma_n)$

Algorithm:  $\mathsf{SS.Reconstruct}$
    Input:  $(\mathcal{I}, \sigma_{\mathcal{I}}) \in \mathcal{A} \times \mathcal{S}$
            $\forall\ i \in \mathcal{I} : (\kappa_i, \gamma_i) := \sigma_i$
            $\kappa \leftarrow \mathsf{SS}_0.\mathsf{Reconstruct}(\mathcal{I}, \kappa_{\mathcal{I}})$
            $\gamma \leftarrow \mathsf{ID.Reconstruct}(\mathcal{I}, \gamma_{\mathcal{I}})$
            IF $(\mathsf{ID.Share}(\gamma))_{\mathcal{I}} \neq \gamma_{\mathcal{I}}$ : RETURN $\perp$
            $\mu \leftarrow \mathsf{SKE.Decrypt}(\kappa, \gamma)$
   Output:  $\mu$
```

Figure 4.1. The Share and Reconstruct algorithms of our construction.

Namely, the reduction to convert a query which tampers with the values (κ_i, γ_i) into a query which just tampers with κ_i. A common solution is to hard-wire the values γ_i, which are known to the reduction, into the tampering function; however, in this way we cannot retrieve the tampered ciphertext $\tilde{\gamma}$. Our solution is to let $t^* < t$ and then reconstruct and leak, somehow, the tampered ciphertext from the shares γ_i. This solution has the small drawback of requiring that, in each tampering query, either zero or at least t^* shares from the same subset of the partition $\mathcal{B}$ are in the reconstruction set $\mathcal{T}$. However, we want to stress that this is still a reasonable assumption because it captures, for instance, the case in which the attacker selects a tampering set $\mathcal{T}$ and splits it into two subsets $\mathcal{B}_1$ and $\mathcal{B}_2$ of equal size greater than t^*.

There is another problem: how exactly do we retrieve the tampered ciphertext? One solution could be by using leakage queries; however, this is infeasible in that, even leaking one bit per tampering query, we could still end up leaking too many bits. Instead, we observe that the reconstructed ciphertext should be the same in each subset of the partition. This, along with the fact that SS_0 is continuously non-malleable, allows to use tampering queries to leak information from the shares. More in detail, by pre-computing a secret sharing of messages $\mu^{(0)}, \mu^{(1)}$, we can leak the j-th bit of the ciphertext by reconstructing the ciphertext in each subset of the partition and then replacing all the shares with shares of $\mu^{(b)}$ depending on the value b of the bit. This technique allows to retrieve the tampered ciphertext $\tilde{\gamma}$ and, in turns, allows the reduction to compute the tampering query without requiring additional leakage or complex leakage models.

Security analysis. Before proceeding with the security analysis of the scheme in Fig. 4.1, we prove a general fact of continuously non-malleable secret sharing schemes. Basically, the following lemma shows that if some information is common to all the groups of shares of a certain size, then such information can be given to the adversary for free.

Lemma 4.6. *Let $n, k, t, t^*, \ell \in \mathbb{N}$ be parameters and let* SS *be a t-out-of-n continuously non-malleable secret sharing scheme with security against k-sized partitioning. Consider an oracle $\mathcal{O}^*(\sigma, \cdot, \cdot, \cdot)$ defined as follows. The oracle takes as input the tuple $(\mathcal{T}, \mathcal{B}, g)$, where*

- *$\mathcal{T} \subseteq [n]$ is a set with at least t elements;*
- *$\mathcal{B} = (\mathcal{B}_1, \ldots, \mathcal{B}_m)$ is a k-sized partition;*
- *$g = (g_i)_{i \in \mathcal{I}}$ is a tuple of functions such that $g_i : \mathcal{B}_i \cap \mathcal{T} \to \{0,1\}^\ell$.*

The oracle outputs $\perp$ if a self-destruct has already been triggered; otherwise, it computes, for all $i \in \mathcal{I}$, $g_i(\sigma_{\mathcal{B}_i \cap \mathcal{T}})$, where $\sigma_{\mathcal{B}_i \cap \mathcal{T}}$ is the tuple of shares within $\mathcal{B}_i \cap \mathcal{T}$, and outputs a value Λ if $\Lambda = g_i(\sigma_{\mathcal{B}_i \cap \mathcal{T}})$ for all $i \in \mathcal{I}$, and outputs $\perp$, triggering self-destruct, if there exist $i_1, i_2 \in \mathcal{I}$ such that $g_{i_1}(\sigma_{\mathcal{B}_{i_1} \cap \mathcal{T}}) \neq g_{i_2}(\sigma_{\mathcal{B}_{i_2} \cap \mathcal{T}})$. Then, SS *retains the same security, including leakage and tampering parameters, even against adversaries that, additionally, have free access to $\mathcal{O}^*(\sigma, \cdot, \cdot, \cdot)$.*

Proof. It suffices to show that the access to $\mathcal{O}^*$ can be simulated for free through a series of tampering queries. Let $\mu_0, \mu_1, \mu^{(0)}, \mu^{(1)} \in \mathcal{M}$ be messages such that $\mu^{(0)}, \mu^{(1)}$ are different each other and different from μ_0, μ_1, let $\sigma = (\sigma_1, \ldots, \sigma_n) \leftarrow\!\!\$\ \mathsf{SS}.\mathsf{Share}(\mu_b)$ be the target secret sharing, let $\mathcal{O}^{\infty}_{\mathsf{T}}((\mu_0, \mu_1), \sigma, \cdot, \cdot, \cdot)$ be the tampering oracle and let $(\mathcal{T}, \mathcal{B}, g), \mathcal{I}, \ell$ be as in the statement of the lemma. Finally, let, for $b \in \{0,1\}$, $(\sigma_1^{(b)}, \ldots, \sigma_n^{(b)}) \leftarrow\!\!\$\ \mathsf{SS}.\mathsf{Share}\big(\mu^{(b)}\big)$ be precomputed shares. For all $i \in \mathcal{I}$ and all $j \in [\ell]$, consider the following functions.

```
Function: $f_i^{(j)}$
   Input: $\sigma_{\mathcal{B}_i} \in \mathcal{S}_{\mathcal{B}_i}$
          $\Lambda_i \leftarrow g_i(\sigma_{\mathcal{B}_i \cap \mathcal{T}})$
          $b_i^{(1)} \| \ldots \| b_i^{(\ell)} := \Lambda_i$    // Parse $\Lambda_i$ as a binary string.
  Output: $\sigma_{\mathcal{B}_i}^{(b_i^{(j)})}$
```

```
Function: $\hat{f}_i$
   Input: $(\Lambda, \sigma_{\mathcal{B}_i}) \in \{0,1\}^\ell \times \mathcal{S}_{\mathcal{B}_i}$
          $\Lambda_i \leftarrow g_i(\sigma_{\mathcal{B}_i \cap \mathcal{T}})$
          IF $\Lambda \neq \Lambda_i$ : $\tilde{\sigma}_{\mathcal{B}_i} := \perp$
          ELSE : $\tilde{\sigma}_{\mathcal{B}_i} := \sigma_{\mathcal{B}_i}^{(0)}$
  Output: $\tilde{\sigma}_{\mathcal{B}_i}$
```

Finally, let $f_i^{(j)}$ and $\hat{f}_i$ be the identity function for all $i \notin \mathcal{I}$ and let $f^{(j)} = (f_i^{(j)})_{i \in \mathcal{I}}$ and $\hat{f} = (\hat{f}_i)_{i \in \mathcal{I}}$. The oracle $\mathcal{O}^*$ can be simulated as follows.

1. Query, for all $j \in [\ell]$, $\tilde{\mu}^{(j)} \leftarrow \mathcal{O}^{\infty}_{\mathsf{T}}\Big((\mu_0, \mu_1), \sigma, \mathcal{T}, \mathcal{B}, f^{(j)}\Big)$; if, at some time, $\tilde{\mu}^{(j)} = \perp$, stop and output $\perp$.

2. For all $j \in [\ell]$, let $b^{(j)}$ be such that $\tilde{\mu}^{(j)} = \mu^{(b^{(j)})}$; if $\tilde{\mu}^{(j)}$ happens to be a different value, stop, trigger self-destruct (e.g. by sending a query which always outputs invalid shares) and output $\perp$.
3. Let $\Lambda := b^{(1)} \| \ldots \| b^{(\ell)}$.
4. Query $\tilde{\mu} \leftarrow \mathcal{O}_{\mathsf{T}}^{\infty}\Big((\mu_0, \mu_1), \sigma, \mathcal{T}, \mathcal{B}, \hat{f}_i(\Lambda, \cdot)\Big)$.
5. If $\tilde{\mu} = \mu^{(0)}$, output Λ; otherwise, trigger self-destruct and output $\perp$.

It is easy to show that, if there exists $\Lambda \in \{0,1\}^{\ell}$ such that $\Lambda = g_i(\sigma_{\mathcal{B}_i \cap \mathcal{T}})$ for all $i \in \mathcal{I}$, the oracle always outputs the correct value Λ. Indeed, the function $f_i^{(j)}$ has access to the shares $\sigma_{\mathcal{B}_i}$ and therefore is able to compute $g_i(\sigma_{\mathcal{B}_i \cap \mathcal{T}})$ and output a set of shares which identifies the j-th bit of Λ. Furthermore, function $\hat{f}_i$ is able to compute again $g_i(\sigma_{\mathcal{B}_i \cap \mathcal{T}})$ and output a valid set of shares only in case the equality with Λ holds, outputting an invalid set of shares otherwise.

Finally, $\mathcal{O}^*$ can be computed in polynomial time and the only resources used by $\mathcal{O}^*$ are tampering queries, which, in the context of continuous non-malleability, are free. $\square$

Now we are ready to prove the main result of this section.

Theorem 4.7. *Let $n, t, t^*, k, \ell \in \mathbb{N}$ be parameters. In the construction depicted in Fig. 4.1, assume that*

- ID *is a t^*-out-of-n information dispersal scheme ;*
- SKE *is a secret-key encryption scheme secure against chosen ciphertext attacks;*
- SS_0 *is a t-out-of-n leakage-resilient continuously non-malleable secret sharing scheme against k joint leakage and tampering attacks.*

Then, SS *is a t-out-of-n leakage-resilient continuously non-malleable secret sharing scheme for n parties against k joint leakage and tampering attacks, with the same leakage model as* SS_0*, under the following restriction: each tampering query $(\mathcal{T}, \mathcal{B}, f)$ output by the attacker is such that, for all subsets $\mathcal{B}_i$ of the partition $\mathcal{B}$, either $\mathcal{B}_1 \cap \mathcal{T} = \emptyset$ or $\#\mathcal{B}_1 \cap \mathcal{T} \geq t^*$. Furthermore,* $\mathsf{SS}.\mathsf{AsymptoticRate} = t^*$.

The proof proceeds by hybrid argument. In particular, we argue that the original experiment is computationally close to an hybrid experiment in which we replace the secret sharing of the key κ with a secret sharing of an unrelated key $\hat{\kappa}$. This experiment is depicted in Fig. 4.2 along with the original experiment $\mathbf{Tamper}^{\mathsf{L}}$, in which we expanded the algorithms related to SS. The lemma below states that the two experiments are computationally indistinguishable.

Lemma 4.8. *For all $\lambda \in \mathbb{N}$, $\mu_0, \mu_1 \in \mathcal{M}$ and all $b \in \{0,1\}$, it holds that*

$$\mathbf{Tamper}^{\mathsf{L}}_{\mathsf{A}}[\lambda, \infty, \mu_0, \mu_1, b] \overset{c}{\approx} \mathbf{Hyb}_{\mathsf{A}}[\lambda, \mu_0, \mu_1, b]\,.$$

```
Differences:  $\mathbf{Tamper}^{\mathsf{L}}_{\mathsf{A}}[\lambda, \infty, \mu_0, \mu_1, b]$   $\mathbf{Hyb}_{\mathsf{A}}[\lambda, \mu_0, \mu_1, b]$

              $\kappa \leftarrow_\$ \mathcal{K}$
              $(\kappa_1, \ldots, \kappa_n) \leftarrow_\$ \mathsf{SS}_0.\mathsf{Share}(\kappa)$
              $\hat{\kappa} \leftarrow_\$ \mathcal{K}$
              $(\kappa_1, \ldots, \kappa_n) \leftarrow_\$ \mathsf{SS}_0.\mathsf{Share}(\kappa)$
              $\gamma \leftarrow_\$ \mathsf{SKE}.\mathsf{Encrypt}(\kappa, \mu_b)$
              $(\gamma_1, \ldots, \gamma_n) \leftarrow_\$ \mathsf{ID}.\mathsf{Share}(\gamma)$
              $\forall\ i \in [n] : \sigma_i := (\kappa_i, \gamma_i)$
              $\sigma := (\sigma_1, \ldots, \sigma_n)$
              $b^* \leftarrow_\$ \mathsf{A}^{\mathcal{O}_{\mathsf{L}}(\sigma,\cdot,\cdot),\mathcal{O}_{\mathsf{T}}(\kappa,(\mu_0,\mu_1),\sigma,\cdot,\cdot,\cdot),\mathcal{O}^*(\sigma,\cdot,\cdot,\cdot)}$
              $b^* \leftarrow_\$ \mathsf{A}^{\mathcal{O}_{\mathsf{L}}(\sigma,\cdot,\cdot),\mathcal{O}^{\star}_{\mathsf{T}}((\hat{\kappa},\kappa),(\mu_0,\mu_1),\sigma,\cdot,\cdot,\cdot),\mathcal{O}^*(\sigma,\cdot,\cdot,\cdot)}$
     Output:  $b^*$

Differences:  $\mathcal{O}_{\mathsf{T}}((\mu_0, \mu_1), \sigma, \mathcal{T}, \mathcal{B}, f)$   $\mathcal{O}^{\star}_{\mathsf{T}}((\hat{\kappa}, \kappa), (\mu_0, \mu_1), \sigma, \mathcal{T}, \mathcal{B}, f)$
     Initial  $\mathsf{SD} := 0$
      state:

              IF $\mathsf{sd} = 1$ : RETURN $\perp$
              $(\sigma_1, \ldots, \sigma_n) := \sigma$
              $(\mathcal{B}_1, \ldots, \mathcal{B}_m) := \mathcal{B}$
              $(f_1, \ldots, f_m) := f$
              $\forall\ i \in [m] : \tilde{\sigma}_{\mathcal{B}_i} \leftarrow f_i(\sigma_{\mathcal{B}_i})$
              $\forall\ i \in [n] : (\tilde{\kappa}_i, \tilde{\gamma}_i) := \tilde{\sigma}_i$
              $\tilde{\kappa} \leftarrow \mathsf{SS}_0.\mathsf{Reconstruct}(\tilde{\kappa}_{\mathcal{T}})$
              $\tilde{\gamma} \leftarrow \mathsf{ID}.\mathsf{Reconstruct}(\tilde{\gamma}_{\mathcal{T}})$
              IF $(\mathsf{ID}.\mathsf{Share}(\gamma))_{\mathcal{T}} \neq \tilde{\gamma}_{\mathcal{T}}$ : $\mathsf{SD} \leftarrow 1$, RETURN $\perp$
              IF $\tilde{\kappa} = \perp$ : $\mathsf{SD} \leftarrow 1$, RETURN $\perp$
              IF $\tilde{\kappa} = \hat{\kappa}$ : $\tilde{\kappa} \leftarrow \kappa$
              $\tilde{\mu} \leftarrow \mathsf{SKE}.\mathsf{Decrypt}(\tilde{\kappa}, \tilde{\gamma})$
              IF $\tilde{\mu} = \perp$ : $\mathsf{SD} \leftarrow 1$, RETURN $\perp$
              IF $\tilde{\mu} \in \{\mu_0, \mu_1\}$ : RETURN $\heartsuit$
     Output:  $\tilde{\mu}$
```

Figure 4.2. The differences between the original experiment $\mathbf{Tamper}^{\mathsf{L}}$ and the hybrid experiment $\mathbf{Hyb}^{1}$.

Proof. By reduction to leakage-resilient continuous non-malleability of SS_0. Suppose towards contradiction that there exist messages μ_0, μ_1, a bit $b \in \{0,1\}$ and an admissible PPT adversary A which is able to cause a non-negligible difference between the two experiments $\mathbf{Tamper}^{\mathsf{L}}_{\mathsf{A}}[\lambda, \infty, \mu_0, \mu_1, b]$ and $\mathbf{Hyb}_{\mathsf{A}}[\lambda, \mu_0, \mu_1, b]$.

Consider the following reduction R trying to cause a non-negligible difference between the two experiments $\mathbf{Tamper}^{\mathsf{L}}_{\mathsf{R}}[\lambda, \infty, \kappa, \hat{\kappa}, b^*]$ for $b^* \in \{0,1\}$, where $\kappa, \hat{\kappa} \in \mathcal{K}$. The reduction R simulates the oracles $\mathcal{O}_{\mathsf{T}}$ and $\mathcal{O}_{\mathsf{L}}$ by itself; furthermore, R has free access to $\mathcal{O}^*$ thanks to Lemma 4.6.

1. Compute $\gamma \leftarrow\!\!\$\ \mathsf{SKE}.\mathsf{Encrypt}(\kappa, \mu_b)$.
2. Compute $(\gamma_1, \ldots, \gamma_n) \leftarrow\!\!\$\ \mathsf{ID}.\mathsf{Share}(\gamma)$.
3. Run $b^* \leftarrow\!\!\$\ \mathsf{A}^{\mathsf{R}}(1^\lambda)$, simulating the oracles as follows.
   - **Leakage oracle:** upon input $(\mathcal{B}, g)$:
     (a) Query $\Lambda \leftarrow \mathcal{O}_{\mathsf{L}}(\mathcal{B}, g)$.
     (b) Return Λ.
   - **Tampering oracle:** upon input $(\mathcal{T}, \mathcal{B}, f)$:
     (a) Parse $(\mathcal{B}_1, \ldots, \mathcal{B}_m) = \mathcal{B}$ and $(f_1, \ldots, f_m) = f$.
     (b) Let $\mathcal{I} \subseteq [m]$ be such that $i \in \mathcal{I}$ if and only if $\mathcal{B}_i \cap \mathcal{T} \neq \emptyset$.
     (c) For all $i \in \mathcal{I}$, consider the following function with hard-wired values $(\gamma_1, \ldots, \gamma_n)$.

```
Function: $\hat{h}_i$
   Input: $\kappa_{\mathcal{B}_i}$
          IF $\mathcal{B}_i \cap \mathcal{T} = \emptyset$ : RETURN $\gamma$
          $(\tilde{\kappa}_j, \tilde{\gamma}_j)_{j \in \mathcal{B}_i} \leftarrow f_i\left((\kappa_j, \gamma_j)_{j \in \mathcal{B}_i}\right)$
          $\tilde{\gamma} \leftarrow \mathsf{ID}.\mathsf{Reconstruct}(\mathcal{B}_i \cap \mathcal{T}, \tilde{\gamma}_{\mathcal{B}_i \cap \mathcal{T}})$
  Output: $\tilde{\gamma}$
```

     Notice that, because of the additional restriction, $\#\mathcal{B}_i \cap \mathcal{T} \geq t^*$, hence the value $\tilde{\gamma}$ can be reconstructed.
     (d) Query $\tilde{\gamma} \leftarrow \mathcal{O}^*(\mathcal{B}, \mathcal{T}, f)$.
     (e) Let $\hat{f}$ be the function f with hard-wired values $\gamma_1, \ldots, \gamma_n$ which only outputs the first component of each share, i.e. $(\tilde{\kappa}_1, \ldots, \tilde{\kappa}_n)$.
     (f) Query $\tilde{\kappa} \leftarrow \mathcal{O}_{\mathsf{T}}\left(\mathcal{B}, \mathcal{T}, \hat{f}\right)$.
     (g) If $\tilde{\gamma} = \bot$ or $\tilde{\kappa} = \bot$, return $\bot$.
     (h) If $\tilde{\kappa} = \heartsuit$, set $\tilde{\kappa} \leftarrow \kappa$.
     (i) Compute $\tilde{\mu} = \mathsf{SKE}.\mathsf{Decrypt}(\tilde{\kappa}, \tilde{\gamma})$.
     (j) Return $\tilde{\mu}$.
4. Output b^*.

We now proceed with the analysis of the reduction. Let Bad be the event, defined over the probability space of the original experiment, that happens whenever a query to the tampering oracle triggers the condition $\tilde{\kappa} = \hat{\kappa}$. It is easy to see that, if Bad does not happen, the reduction is perfect: R perfectly emulates the view of A

in both experiments and outputs the same as A, thus retaining the same advantage. On the other side, since $\hat{\kappa}$ is random and independent of κ, a union bound shows that $\mathbb{P}[\textsc{Bad}] \leq \mathcal{P}\mathrm{oly}(\lambda) \cdot 2^{-\lambda}$. The lemma follows. □

The lemma below allows to conclude the proof of Theorem 4.7.

Lemma 4.9. *For all $\lambda \in \mathbb{N}$ and all $\mu_0, \mu_1 \in \mathcal{M}$ it holds that*

$$\mathbf{Hyb}_{\mathsf{A}}[\lambda, \mu_0, \mu_1, 0] \overset{c}{\approx} \mathbf{Hyb}_{\mathsf{A}}[\lambda, \mu_0, \mu_1, 1]\,.$$

Proof. By reduction to the security of SKE against chosen ciphertext attacks. Suppose towards contradiction that there exist messages μ_0, μ_1, a bit $b \in \{0,1\}$ and an admissible PPT adversary A which is able to cause a non-negligible difference between the two experiments $\mathbf{Hyb}_{\mathsf{A}}[\lambda, \mu_0, \mu_1, 0]$ and $\mathbf{Hyb}_{\mathsf{A}}[\lambda, \mu_0, \mu_1, 1]$.

Consider the following reduction $\mathsf{R} = (\mathsf{R}_1, \mathsf{R}_2)$ trying to cause a non-negligible difference between the two experiments $\mathbf{Ind}^{\mathsf{CCA2}}_{\mathsf{R}}[\lambda, b]$ for $b \in \{0,1\}$. The algorithm R_1 simply outputs the two messages μ_0, μ_1 The algorithm R_2 receives the challenge ciphertext $\tilde{\gamma}$ and proceeds as follows.

1. Sample $\hat{\kappa} \leftarrow\!\!\$\ \mathcal{K}$.
2. Compute $(\gamma_1, \ldots, \gamma_n) \leftarrow\!\!\$\ \mathsf{ID}.\mathsf{Share}(\gamma)$.
3. Compute $(\kappa_1, \ldots, \kappa_n) \leftarrow\!\!\$\ \mathsf{SS}_0.\mathsf{Share}(\hat{\kappa})$.
4. Run $b^* \leftarrow\!\!\$\ \mathsf{A}^{\mathsf{R}}(1^\lambda)$, simulating the oracles as follows.
   - **Leakage oracle:** answer as in $\mathbf{Hyb}_{\mathsf{A}}$.
   - **Tampering oracle:** upon input $(\mathcal{T}, \mathcal{B}, f)$:
     (a) Compute, for all $i \in [m]$, $(\tilde{\kappa}_j, \tilde{\gamma}_j)_{j \in \mathcal{B}_i} \leftarrow f_i\left((\kappa_j, \gamma_j)_{j \in \mathcal{B}_i}\right)$.
     (b) Compute $\tilde{\gamma} \leftarrow \mathsf{ID}.\mathsf{Reconstruct}(\mathcal{T}, \tilde{\gamma}_{\mathcal{T}})$.
     (c) Check that $(\mathsf{ID}.\mathsf{Share}(\tilde{\gamma}))_{\mathcal{T}} = \tilde{\gamma}_{\mathcal{T}}$, and return $\perp$ otherwise.
     (d) Compute $\tilde{\kappa} \leftarrow \mathsf{SS}_0.\mathsf{Reconstruct}(\mathcal{T}, \tilde{\kappa}_{\mathcal{T}})$, and return $\perp$ if $\tilde{\kappa} = \perp$.
     (e) If $\tilde{\kappa} \neq \hat{\kappa}$, compute $\tilde{\mu} \leftarrow \mathsf{SKE}.\mathsf{Decrypt}(\tilde{\kappa}, \tilde{\gamma})$; otherwise, query $\tilde{\mu} \leftarrow \mathcal{O}_{\mathsf{Dec}}(\tilde{\gamma})$.
     (f) If $\tilde{\mu} \in \{\mu_0, \mu_1\}$, set $\tilde{\mu} \leftarrow \heartsuit$.
     (g) Return $\tilde{\mu}$.
5. Output b^*.

For the analysis, notice that the reduction is perfect: $\mathcal{R}$ perfectly simulates the experiment $\mathbf{Hyb}_{\mathsf{A}}$ and outputs the same as A, thus retaining the same advantege. The lemma follows. □

Proof of Theorem 4.7. The security proof follows from the chain

$$\begin{aligned}\mathbf{Tamper}^{\mathsf{L}}_{\mathsf{A}}[\lambda, \infty, \mu_0, \mu_1, 0] &\overset{c}{\approx} \mathbf{Hyb}_{\mathsf{A}}[\lambda, \mu_0, \mu_1, 0] \\ &\overset{c}{\approx} \mathbf{Hyb}_{\mathsf{A}}[\lambda, \mu_0, \mu_1, 1] \\ &\overset{c}{\approx} \mathbf{Tamper}^{\mathsf{L}}_{\mathsf{A}}[\lambda, \infty, \mu_0, \mu_1, 1]\,.\end{aligned}$$

As for the rate, the length of the key κ of the encryption scheme SKE only depends on the security parameter λ and, therefore, size of the shares of SS only depends on λ, on the number n of parties and on the tolerated leakage ℓ. Let $s_0(\lambda, n, \ell)$ be the size of one share of SS_0. For the ciphertext, it is possible to obtain $|\gamma| = |\mu| + O(\lambda)$, hence

$$|\gamma_i| = \frac{|\gamma|}{t^*} = \frac{|\mu| + O(\lambda)}{t^*}.$$

Putting everything together, we obtain

$$s(\lambda, n, \ell, |\mu|) = s_0(\lambda, n, \ell) + \frac{|\mu| + O(\lambda)}{t^*},$$

where $s(\lambda, n, \ell, |\mu|)$ is the size of one share of SS. This means that the asymptotic rate of SS is

$$\begin{aligned}\inf_{\lambda\in\mathbb{N}} \lim_{|\mu|\to\infty} \frac{|\mu|}{s(\lambda, n, \ell, |\mu|)} &= \inf_{\lambda\in\mathbb{N}} \lim_{|\mu|\to\infty} \frac{|\mu|}{s_0(\lambda, n, \ell) + \frac{|\mu|+O(\lambda)}{t^*}} \\ &= \inf_{\lambda\in\mathbb{N}} \lim_{|\mu|\to\infty} \frac{t^* \cdot |\mu|}{t^* \cdot \mathcal{P}\mathrm{oly}(\lambda, n, \ell) + |\mu| + O(\lambda)} \\ &= t^*.\end{aligned}$$

This completes the proof. □

Rate optimality. We stress that, when $k = t - 1$, Theorem 4.5 says that the rate of a continuously non-malleable secret sharing scheme against joint tampering with at most k shares is 1. This is not in contrast with the fact that our compiler in Theorem 4.7 achieves rate larger than 1, as the latter only holds under the additional restriction on the way the attacker can manipluate the shares. Nevertheless, it is possible to adapt the proof of Theorem 4.5 so that it captures this settings, thus showing that our rate compiler achieves the best possible rate whenever $t^* < t/2$.

More in detail, we change the definition of $\mathsf{Com}.\mathsf{Commit}$ so that it now outputs the value $\gamma = (\sigma_1, \ldots, \sigma_{t^*})$, and we adjust the opening procedure accordingly. The proof that Com is perfectly binding is identical to the one in Theorem 4.5, except that now we define $\ell := |(\sigma_{t^*+1}, \ldots, \sigma_t)$ and, moreover, the adversary attacking continuous non-malleability sets $\mathcal{B}_1 = [t^*]$ and $\mathcal{B}_2 = [t] \setminus [t^*]$ and parses the string $\alpha_1 \| \ldots \| \alpha_\ell$ as $(\sigma_{t^*+1}, \ldots, \sigma_t)$. Now $\#\mathcal{B}_1 = t^*$ and $\#\mathcal{B}_2 = t - t^* \geq 2t^* - t^* = t^*$, being $t^* < t/2$ by hypothesis. Finally, since $t^* \leq t - 1$, the adversary is admissible, which concludes the updated proof of Theorem 4.5.

Instantiation. By instantiating Theorem 4.7 with $t^* = 1$, we obtain the following.

Corollary 4.10. *Assuming the existence of one-to-one one-way functions, for all $n, t, \ell \in \mathbb{N}$ with $t > 2n/3$, there is a construction of a t-out-of-n secret sharing scheme satisfying noisy leakage-resilient continuous non-malleability under selective k-joint leakage and tampering attacks. Furthermore, the scheme achieves asymptotic rate* 1*, which is optimal.*

Proof. It is well known that IND-CCA secure secret-key encryption schemes can be constructed in a black-box way from any one-way function, whereas the information dispersal can be instantiated using linear algebra over finite fields [Rab89]. As for the continuously non-malleable secret sharing scheme, we can take the one given by Corollary 3.16. Finally, when applying Theorem 4.7 with $t^* = 1$, the restriction on the tampering queries disappears[3] and we obtain the standard definition of continuous non-malleability against $(t-1)$-joint tampering attacks. Since $t^* = 1$, the asymptotic rate of the construction is 1, which, by Theorem 4.5, is optimal. □

[3]Indeed, any subset either contains at least $t^* = 1$ shares in $\mathcal{T}$ or does not contain any share in $\mathcal{T}$.

Chapter 5

Conclusions and problems

In Chapter 3 we achieved the following two constructions:

- A non-malleable code that is secure in a slightly stronger model than selective partitioning, in Section 3.1 and Section 3.2.
- The first continuously non-malleable secret sharing scheme against joint leakage and tampering, in Section 3.3.

An interesting open problem would be to get the best of both worlds and, at the same time, remove the restrictions on the adaptivity of the adversary, thus achieving the first continuously non-malleable secret sharing scheme against joint *adaptive* partitioning. Unfortunately, this is not an easy task, since a fully adaptive adversary is able to, intuitively, create correlation between possibly all the shares. Even if a construction is secure, fact this makes the security proof way harder.

Another interesting direction is towards the efficiency of non-malleable secret sharing schemes. In Chapter 4, we show some limitations given by the requirement of continuous non-malleability, and then we show how to construct a scheme with a good *asymptotic rate*. However, our construction has a limitation on the capabilities of the adversary. We leave open to discover how to remove such limitation. Furthermore, we leave also open to find a scheme with a good *concrete* rate. One disadvantage of the asymptotic rate is that it measures the space efficiency of the secret sharing for large secrets; however, our construction is not able to retain such space efficiency even for small secrets.

Appendix A

Instantiating the non-malleable code

Here, we explain how to obtain the noisy-leakage resilient p-time non-malleable asymmetric split-state code (a.k.a. 2-out-of-2 secret sharing scheme) with the additional properties 1 and 2 needed to instantiate the scheme in Section 3.3. Our construction exploits leakage-resilient asymmetric split-state codes, as recently introduced by Ball, Guo and Wichs [BGW19] and generalized to the noisy-leakage setting by Brian, Faonio and Venturi [BFV19].

In what follows, we refer to the schemes as (split-state) *codes* instead of 2-out-of-2 secret sharing schemes. Furthermore, the reconstruction algorithms implicitly take [2] as the first argument, as this is the only possible reconstruction set, and we assume the existence of an algorithm Share^*_b, for $b \in \{0,1\}$, which computes the share σ_b from μ and the share σ_{1-b}. We will later prove that this algorithm indeed exists. The construction uses three split-state codes SS, SS_0, SS_1 and it is depicted in Fig. A.1. The theorem below shows that the resulting SS^* is the non-malleable code that we are searching for.

Theorem A.1. *For all $i, j \in \{0,1\}$, let $p, s_i, s_{i,j}, \ell_i, \ell_{i,j} \in \mathbb{N}$ and $\varepsilon, \varepsilon_i \in [0,1]$ be parameters such that*

- $s_1 < s_0$;
- $s_{0,1} < s_{0,0}$, $\ell_{0,0} \geq \ell_0 + p \cdot s_{1,1}$ *and* $\ell_{0,1} \geq \ell_1$;
- $s_{1,1} < s_{1,0}$, $\ell_{1,0} \geq \ell_1 + p \cdot s_{0,1}$ *and* $\ell_{1,1} \geq \ell_0$.

Assume that:

- SS *is an asymmetric p-time non-malleable code with security ε and share space $\mathcal{S}_0 \times \mathcal{S}_1$ such that* $\log \#\mathcal{S}_0 = s_0$ *and* $\log \#\mathcal{S}_1 = s_1$;
- SS_0 *is an asymmetric $(\ell_{0,0}, \ell_{0,1})$-noisy leakage resilient code with security ε_0 and share space $\mathcal{S}_{0,0} \times \mathcal{S}_{0,1}$ such that* $\log \#\mathcal{S}_{0,0} = s_{0,0}$ *and* $\log \#\mathcal{S}_{0,1} = s_{0,1}$.
- SS_1 *is an asymmetric $(\ell_{1,0}, \ell_{1,1})$-noisy leakage resilient code with security ε_1 and share space $\mathcal{S}_{1,0} \times \mathcal{S}_{1,1}$ such that* $\log \#\mathcal{S}_{1,0} = s_{1,0}$ *and* $\log \#\mathcal{S}_{1,1} = s_{1,1}$.

```
Algorithm: $\mathsf{SS}^*.\mathsf{Share}$
    Input: $\mu \in \mathcal{M}$
           $(\sigma_0, \sigma_1) \leftarrow\!\!\$\ \mathsf{SS}(\mu)$
           $(\sigma_{0,0}, \sigma_{0,1}) \leftarrow\!\!\$\ \mathsf{SS}_0(\sigma_0)$
           $(\sigma_{1,0}, \sigma_{1,1}) \leftarrow\!\!\$\ \mathsf{SS}_1(\sigma_1)$
           $\sigma_0^* \leftarrow (\sigma_{0,0}, \sigma_{1,1})$
           $\sigma_1^* \leftarrow (\sigma_{1,0}, \sigma_{0,1})$
   Output: $(\sigma_0^*, \sigma_1^*)$

Algorithm: $\mathsf{SS}^*.\mathsf{Reconstruct}$
    Input: $([2], (\sigma_0^*, \sigma_1^*)) \in \mathcal{A} \times \mathcal{S}$
           $(\sigma_{0,0}, \sigma_{1,1}) := \sigma_0^*$
           $(\sigma_{1,0}, \sigma_{0,1}) := \sigma_1^*$
           $\sigma_0 \leftarrow \mathsf{SS}_0.\mathsf{Reconstruct}(\sigma_{0,0}, \sigma_{0,1})$
           $\sigma_1 \leftarrow \mathsf{SS}_1.\mathsf{Reconstruct}(\sigma_{1,0}, \sigma_{1,1})$
           $\mu \leftarrow \mathsf{SS}.\mathsf{Reconstruct}(\sigma_0, \sigma_1)$
   Output: $\mu$
```

Figure A.1. The Share and Reconstruct algorithms of our construction.

Then, SS^* *is an asymmetric* (ℓ_0, ℓ_1)*-noisy leakage resilient* p*-times non-malleable code with security* $\varepsilon + 2(\varepsilon_0 + \varepsilon_1)$ *and share space* $\mathcal{S}_0^* \times \mathcal{S}_1^*$ *such that* $\mathcal{S}_0^* = \mathcal{S}_{0,0} \times \mathcal{S}_{1,1}$ *and* $\mathcal{S}_1^* = \mathcal{S}_{0,1} \times \mathcal{S}_{1,0}$.

The proof of the above theorem uses a hybrid strategy similar to the one in Theorem 3.1 and goes along the same lines of the proof of Theorem 7 in [BFV19] for the case of 2-out-of-2 secret sharing, the only difference being that, this time, SS is p-time non-malleable instead of one-time non-malleable, and we use different parameters for $\mathsf{SS}_0, \mathsf{SS}_1$. More in detail, the hybrid experiments are the same as in [BFV19] with the only difference that we have to leak $2p$ tampered values (namely, $\tilde{\sigma}_{1,1}^{(j)}, \tilde{\sigma}_{0,1}^{(j)}$ for $j \in [p]$) instead of only two. However, our choice of parameters allows to do so since

$$\ell_{0,0} \geq \ell_0 + p \cdot s_{1,1} \quad \text{and} \quad \ell_{1,0} \geq \ell_1 + p \cdot s_{0,1}.$$

For the above similarities with [BFV19], we will just state the relevant details.

Proof. Let $\mu_0, \mu_1 \in \mathcal{M}$, let $\lambda \in \mathbb{N}$ and let $b \in \{0, 1\}$. Consider the following hybrid experiments.

- Let $\mathbf{Hyb}^1_{\mathsf{A}}[\lambda, \mu_0, \mu_1, b]$ be the same as $\mathbf{Tamper}^{\mathsf{NL}}[\lambda, p, \mu_0, \mu_1, b]$, except that, before applying the tampering queries, we re-sample the share $\sigma'_{0,0}$ (resp. $\sigma_{1,0}$) in such a way that:
  1. the reconstruction with $\sigma'_{0,0}, \sigma_{0,1}$ (resp. $\sigma'_{1,0}, \sigma_{1,1}$) leads to the value σ_0 (resp. σ_1);
  2. it is consisted with the leakage performed;
  3. for all the tampering queries $j \in [p]$, $f_0^{(j)}\left(\sigma'_{0,0}, \sigma_{1,1}\right) = f_0^{(j)}(\sigma_{0,0}, \sigma_{1,1})$ (resp. $f_0^{(j)}\left(\sigma'_{1,0}, \sigma_{0,1}\right) = f_1^{(j)}(\sigma_{1,0}, \sigma_{0,1})$)

We argue that $\mathbf{Hyb}^1_{\mathsf{A}}[\lambda, \mu_0, \mu_1, b]$ and $\mathbf{Tamper}^{\mathsf{NL}}[\lambda, p, \mu_0, \mu_1, b]$ are identically distributed since, the values $\sigma'_{0,0}$ an $\sigma'_{1,0}$ are sampled from the same distribution where the original values are defined.

- Let $\mathbf{Hyb}^2_{\mathsf{A}}[\lambda, \mu_0, \mu_1, b]$ be the same as $\mathbf{Hyb}^1_{\mathsf{A}}[\lambda, \mu_0, \mu_1, b]$ except that all the leakage queries are applied to the shares $(\hat{\sigma}_{0,0}, \sigma_{1,1})$ and $(\hat{\sigma}_{1,0}, \sigma_{0,1})$, where $(\hat{\sigma}_{0,0}, \sigma_{0,1})$ and $(\hat{\sigma}_{1,0}, \sigma_{1,1})$ are valid shares of dummy messages. Here, we can reduce by two consecutive steps to the noisy-leakage resilience of SS_0 and SS_1.

Notice that the above hybrids, and therefore the indistinguishability proofs, closely resemble the ones in Theorem 3.1. Now assume that there exists an adversary A able to cause non-negligible difference between $\mathbf{Hyb}^2_{\mathsf{A}}[\lambda, \mu_0, \mu_1, 0]$ and $\mathbf{Hyb}^2_{\mathsf{A}}[\lambda, \mu_0, \mu_1, 1]$ and consider the following reduction R.

1. Sample $\sigma_{0,1} \leftarrow\!\!\$\; \mathcal{S}_{0,1}, \sigma_{1,1} \leftarrow\!\!\$\; \mathcal{S}_{1,1}$.

2. Sample $\hat{\sigma}_{0,0} \leftarrow\!\!\$\; \mathsf{SS}_0.\mathsf{Share}^*_0(0^{s_0}, \sigma_{0,1})$ and $\hat{\sigma}_{1,0} \leftarrow\!\!\$\; \mathsf{SS}_1.\mathsf{Share}^*_1(0^{s_1}, \sigma_{1,1})$, where, for $i \in \{0,1\}$, $\mathsf{SS}_i.\mathsf{Share}^*_i$ is a special procedure that takes as input the message and one share and samples the remaining share so that the result is a valid encoding of the input message.

3. Sample random coins ρ_0, ρ_1.

4. Run $b^* \leftarrow\!\!\$\; \mathsf{A}^{\mathsf{R}}(1^\lambda)$, simulating the oracles as follows.

   - **Leakage oracle:** answer by computing the leakage queries on $(\hat{\sigma}_{0,0}, \sigma_{1,1})$ and $(\hat{\sigma}_{1,0}, \sigma_{0,1})$.
   - **Tampering oracle, query $q \in [p]$:** upon input $(f_0^{(q)}, f_1^{(q)})$, compute $(\tilde{\sigma}^{(q)}_{0,0}, \tilde{\sigma}^{(q)}_{1,1}) = f_0^{(q)}(\hat{\sigma}_{0,0}, \hat{\sigma}_{1,1})$ and $(\tilde{\sigma}^{(q)}_{1,0}, \tilde{\sigma}^{(q)}_{2,1}) = f_1^{(q)}(\hat{\sigma}_{1,0}, \hat{\sigma}_{0,1})$.
   - **Tampering oracle, answer to $q \in [p]$:** Construct the tampering query $\hat{f}^{(q)} = (\hat{f}^{(q)}_0, \hat{f}^{(q)}_1)$ where $\hat{f}^{(q)}_0$, upon input σ_0 (resp. σ_1), proceeds as follows:
     (a) Using randomness ρ_0 (resp. ρ_1), sample the share $\sigma'_{0,0}$ (resp. $\sigma'_{1,0}$) such that $\mathsf{SS}_0.\mathsf{Reconstruct}\left(\sigma'_{0,0}, \sigma_{0,1}\right) = \sigma_0$ (resp. $\mathsf{SS}_1.\mathsf{Reconstruct}\left(\sigma'_{1,0}, \sigma_{1,1}\right) = \sigma_1$) and $\sigma'_{0,0}$ (resp. $\sigma'_{1,0}$) is consistent with all the leakage done by A and with the tampered values $\tilde{\sigma}^{(1)}_{1,1}, \ldots, \tilde{\sigma}^{(p)}_{1,1}$ (resp. $\tilde{\sigma}^{(1)}_{0,1}, \ldots, \tilde{\sigma}^{(p)}_{0,1}$).
     (b) Compute values $(\tilde{\sigma}'_{0,0}, \tilde{\sigma}^{(j)}_{1,1}) \leftarrow f_0^{(q)}(\sigma'_{0,0}, \sigma_{1,1})$ (resp. $(\tilde{\sigma}'_{1,0}, \tilde{\sigma}^{(j)}_{0,1}) \leftarrow f_1^{(q)}(\sigma'_{1,0}, \sigma_{0,1})$).
     (c) Output $\tilde{\sigma}_0 \leftarrow \mathsf{SS}_0.\mathsf{Reconstruct}\left(\tilde{\sigma}'_{0,0}, \tilde{\sigma}^{(q)}_{1,1}\right)$ (resp. $\tilde{\sigma}_1 \leftarrow \mathsf{SS}_1.\mathsf{Reconstruct}\left(\tilde{\sigma}'_{1,0}, \tilde{\sigma}^{(q)}_{0,1}\right)$).

     Then query $\tilde{\mu}^{(q)} \leftarrow \mathcal{O}^p_{\mathsf{T}}\left(\hat{f}^{(q)}\right)$ and return $\tilde{\mu}^{(q)}$.

For the analysis, note that the reuction is perfect and, in particular, samples a new valid codeword that is consistent with the view of the adversary A and encoes the message μ_b as in the real experiment. This conclues the proof. $\square$

Finally, we need to show that the scheme SS^* is able to achieve the properties 1 and 2 needed to instantiate Theorem 3.13. The lemma below states that if the underlying non-malleable code SS satisfies the additional property 1, so does the scheme SS^*.

Lemma A.2. *Suppose that there exists σ_0 such that, for all $\mu \in \mathcal{M}$, there exists σ_1 such that $\mathsf{SS}.\mathsf{Reconstruct}(\sigma_0, \sigma_1) = \mu$. Then, there exists σ_0^* such that, for all $\mu \in \mathcal{M}$, there exists σ_1^* such that $\mathsf{SS}.\mathsf{Reconstruct}(\sigma_0^*, \sigma_1^*) = \mu$.*

Proof. Let σ_0 be such that, for all $\mu \in \mathcal{M}$, there exists σ_1 such that $\mathsf{SS}.\mathsf{Reconstruct}(\sigma_0, \sigma_1) = \mu$. Then, it is possible to fix $\sigma_{1,1}$ and $\sigma_{0,1}$ and compute $\sigma_{0,0} \leftarrow\!\!\$\ \mathsf{SS}.\mathsf{Share}_0^*(\sigma_0, \sigma_{0,1})$. The new shares will be

- $\sigma_0^* = (\sigma_{0,0}, \sigma_{1,1})$

and, once fixed σ_0^* and $\mu \in \mathcal{M}$ and computed $\sigma_{1,0} \leftarrow\!\!\$\ \mathsf{SS}.\mathsf{Share}_1^*(\sigma_1, \sigma_{1,1})$,

- $\sigma_1^* = (\sigma_{1,0}, \sigma_{0,1})$.

□

The property 2 is a bit more delicate because, even if SS_0 and SS_1 achieve it, the random variables $(\mathbf{\Sigma}_{0,0}, \mathbf{\Sigma}_{1,1})$ and $(\mathbf{\Sigma}_{1,0}, \mathbf{\Sigma}_{0,1})$ are defined by sharing $\mathbf{\Sigma}_0$ and $\mathbf{\Sigma}_1$, which are related distributions. Instead, here we use a non-blackbox approach and prove that the asymmetric code given by Appendix A of [BFV19], which we describe below, allows SS^* to achieve the aforementioned property.

Let Ext be a seeded extractor with d bits of source, r bits of seed an m bits of output, and let $\mathsf{2Ext}$ be a two-source extractor with s_2 bits from each source and r bits of output. The construction of the leakage-resilient code $\mathsf{SS}_\star$ is depicted in Fig. A.2. This construction has already been proven to be an asymmetric (ℓ_1, ℓ_2)-leakage resilient code with security ε for an appropriate choice of the parameters d and r; for more details, see [BGW19, BFV19]. Furthermore, in Fig. A.2 we also give the definition for the alternate sharing procedure Share_1^*.

The following lemma proves that this construction also satisfies conditional independence.

Lemma A.3. *Let SS_0 and SS_1 be two instances of $\mathsf{SS}_\star$ with appropriate parameters. Then, it holds that, for all $\mu \in \mathcal{M}$, the construction depicted in Fig. A.1,*

$$\widetilde{\mathbb{H}}_\infty\left(\mathbf{\Sigma}_0^* | \mathbf{\Sigma}_1^*\right) \geq \mathbb{H}_\infty\left(\mathbf{\Sigma}_0^*\right) - d \quad \text{and} \quad \widetilde{\mathbb{H}}_\infty\left(\mathbf{\Sigma}_1^* | \mathbf{\Sigma}_0^*\right) \geq \mathbb{H}_\infty\left(\mathbf{\Sigma}_1^*\right) - d,$$

where $d = s_0 + s_1$ and $(\mathbf{\Sigma}_0^*, \mathbf{\Sigma}_1^*) = \mathsf{SS}^*.\mathsf{Share}(\mu)$ is the random variable of the shares of μ.

Proof. For all messages $\mu \in \mathcal{M}$, let $(\mathbf{\Sigma}_0^*, \mathbf{\Sigma}_1^*) = \mathsf{SS}^*.\mathsf{Share}(\mu)$ be the random variable relative to a non-malleable encoding of μ, and let

- $(\mathbf{\Sigma}_{0,0}, \mathbf{\Sigma}_{0,1}) = \mathsf{SS}_0.\mathsf{Share}(\mathbf{\Sigma}_0)$,
- $(\mathbf{\Sigma}_{1,0}, \mathbf{\Sigma}_{1,1}) = \mathsf{SS}_1.\mathsf{Share}(\mathbf{\Sigma}_1)$

```
Algorithm: $\mathsf{SS}_\star.\mathsf{Share}$
Input: $\mu \in \mathcal{M}$
    $\sigma_2 \leftarrow_\$ \{0,1\}^{s_1}$
    $x \leftarrow_\$ \{0,1\}^{d}$
    $y \leftarrow_\$ \{0,1\}^{s_2}$
    $\rho \leftarrow \mathsf{2Ext}(\sigma_2, y)$
    $z \leftarrow \mathsf{2Ext}(x, \rho) \oplus \mu$
    $\sigma_1 \leftarrow (x, y, z)$
Output: $(\sigma_1, \sigma_2)$

Algorithm: $\mathsf{SS}_\star.\mathsf{Share}_1^*$
Input: $(\mu, \sigma_2) \in \mathcal{M} \times \mathcal{S}_2$
    $x \leftarrow_\$ \{0,1\}^{d}$
    $y \leftarrow_\$ \{0,1\}^{s_2}$
    $\rho \leftarrow \mathsf{2Ext}(\sigma_2, y)$
    $z \leftarrow \mathsf{Ext}(x, \rho) \oplus \mu$
    $\sigma_1 \leftarrow (x, y, z)$
Output: $\sigma_1$

Algorithm: $\mathsf{SS}_\star.\mathsf{Reconstruct}$
Input: $(\sigma_1, \sigma_2) \in \{0,1\}^{s_1} \times \{0,1\}^{s_2}$
    $(x, y, z) := \sigma_1$
    $\mu \leftarrow z \oplus \mathsf{Ext}(x, \mathsf{2Ext}(\sigma_1, y))$
Output: $\mu$
```

Figure A.2. The Share and Reconstruct algorithms of $\mathsf{SS}_\star$.

be the respective random variables relative to the underlying leakage-resilient encodings. In particular, let, for $i \in \{0,1\}$, $(\boldsymbol{X}_i, \boldsymbol{Y}_i, \boldsymbol{Z}_i) = \boldsymbol{\Sigma}_{i,0}$ be the random variables relative to the values x, y, z in the two instantiations of the scheme in Fig. A.2. Finally, let d_0, d_1 be the parameter d in SS_0 and SS_1 respectively. Then,

$$\begin{aligned}
\widetilde{\mathbb{H}}_\infty(\boldsymbol{\Sigma}_0^*|\boldsymbol{\Sigma}_1^*) &= \widetilde{\mathbb{H}}_\infty((\boldsymbol{X}_0, \boldsymbol{Y}_0, \boldsymbol{Z}_0), \boldsymbol{\Sigma}_{1,1}|(\boldsymbol{X}_1, \boldsymbol{Y}_1, \boldsymbol{Z}_1), \boldsymbol{\Sigma}_{0,1}) \\
&\geq \widetilde{\mathbb{H}}_\infty(\boldsymbol{X}_0, \boldsymbol{Y}_0, \boldsymbol{Z}_0, \boldsymbol{\Sigma}_{1,1}|\boldsymbol{X}_1, \boldsymbol{Y}_1\boldsymbol{\Sigma}_{0,1}) - |\boldsymbol{Z}_1| && \text{(A.1)} \\
&\geq \widetilde{\mathbb{H}}_\infty(\boldsymbol{X}_0, \boldsymbol{Y}_0, \boldsymbol{\Sigma}_{1,1}|\boldsymbol{X}_1, \boldsymbol{Y}_1\boldsymbol{\Sigma}_{0,1}) - |\boldsymbol{Z}_1| && \text{(A.2)} \\
&\geq \widetilde{\mathbb{H}}_\infty(\boldsymbol{X}_0, \boldsymbol{Y}_0, \boldsymbol{\Sigma}_{1,1}) - |\boldsymbol{Z}_1| && \text{(A.3)} \\
&= s_{0,0} - s_0 + s_{1,1} - s_1 && \text{(A.4)} \\
&= \mathbb{H}_\infty(\boldsymbol{\Sigma}_0^*) - s_0 - s_1. && \text{(A.5)}
\end{aligned}$$

In the above derivation,

- (A.1) follows from the application of the *chain lemma*;
- in (A.2) we simply removed the random variable $\boldsymbol{Z}_0$;
- (A.3) holds because now the random variables $\boldsymbol{X}_0, \boldsymbol{Y}_0, \boldsymbol{\Sigma}_{1,1}$ are independent of $\boldsymbol{X}_1, \boldsymbol{Y}_1, \boldsymbol{\Sigma}_{0,1}$;
- (A.4) follows from the fact that $x_0, y_0, \sigma_{0,0}$ are randomly sampled ant that $|(x_0, y_0)| = s_{0,0} - |z_0|$, where $|z_0| = s_0$;
- finally, (A.5) follows from the fact that $\boldsymbol{\Sigma}_0^* = (\boldsymbol{\Sigma}_{0,0}, \boldsymbol{\Sigma}_{1,1})$ is uniformly distributed over $\{0,1\}^{s_{0,0}} \times \{0,1\}^{s_{1,1}}$.

A similar analysis shows that

$$\begin{aligned}
\widetilde{\mathbb{H}}_\infty(\boldsymbol{\Sigma}_1^*|\boldsymbol{\Sigma}_0^*) &= \widetilde{\mathbb{H}}_\infty((\boldsymbol{X}_1, \boldsymbol{Y}_1, \boldsymbol{Z}_1), \boldsymbol{\Sigma}_{0,1}|(\boldsymbol{X}_0, \boldsymbol{Y}_0, \boldsymbol{Z}_0), \boldsymbol{\Sigma}_{1,1}) \\
&\geq \widetilde{\mathbb{H}}_\infty(\boldsymbol{X}_1, \boldsymbol{Y}_1, \boldsymbol{Z}_1, \boldsymbol{\Sigma}_{0,1}|\boldsymbol{X}_0, \boldsymbol{Y}_0\boldsymbol{\Sigma}_{1,1}) - |\boldsymbol{Z}_0| \\
&\geq \widetilde{\mathbb{H}}_\infty(\boldsymbol{X}_1, \boldsymbol{Y}_1, \boldsymbol{\Sigma}_{0,1}|\boldsymbol{X}_0, \boldsymbol{Y}_0\boldsymbol{\Sigma}_{1,1}) - |\boldsymbol{Z}_0| \\
&\geq \widetilde{\mathbb{H}}_\infty(\boldsymbol{X}_1, \boldsymbol{Y}_1, \boldsymbol{\Sigma}_{0,1}) - |\boldsymbol{Z}_0| \\
&= s_{1,0} - s_1 + s_{0,1} - s_0 \\
&= \mathbb{H}_\infty(\boldsymbol{\Sigma}_1^*) - s_1 - s_0.
\end{aligned}$$

□

Corollary A.4. *For all $s_0, s_1, \ell_0, \ell_1, p \in \mathbb{N}$ and all $\varepsilon \in [0,1]$, there exists a construction of an asymmetric (ℓ_0, ℓ_1)-noisy leakage-resilient p-time non-malleable code with statistica security ε and with share space $\mathcal{S}_0 \times \mathcal{S}_1$ such that $\log \#\mathcal{S}_0 = s_0$ and $\log \#\mathcal{S}_1 = s_1$. Furthermore, such construction satisfies both property 1 and property 2 required for Theorem 3.13.*

Proof. Corollary 5.7 of [GSZ20] shows that, for all $n_1, n_2 \in \mathbb{N}$ and all polynomials p' there exists a two-source p-time ε-non-malleable extractor for sources of full-entropy of size n_1, n_2, where $p = n_2^{\Omega(1)}$, $n_1 = 4n_2 + p'(n_2)$ and $\varepsilon = 2^{-n_2^{\Omega(1)}}$. This scheme has efficient pre-image sampleability and further satisfies the additional property described in the hypothesis of Lemma A.2. By the known connection between (leakage-resilient) non-malleable extractors with efficient pre-image sampleability and (leakage-resilient) non-malleable codes, we obtain a p-time non-malleable code with statistical security of $\varepsilon \cdot 2^{p|\mu|+1}$.

Additionally, we note that by our setting of parameters in Theorem A.1 we can have $\ell_0 \geq s_1^*$ so long as the underlying schemes SS_0 and SS_1 allow to arbitrarily set the parameters of hte leakage and of the codeword size, which is the case thanks to Theorem 6 of [BFV19].

The proof follows by applying Lemma A.2 and Lemma A.3. $\square$

Bibliography

[ADKO15a] Divesh Aggarwal, Yevgeniy Dodis, Tomasz Kazana, and Maciej Obremski. Non-malleable reductions and applications. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 459–468, Portland, OR, USA, June 14–17, 2015. ACM Press.

[ADKO15b] Divesh Aggarwal, Stefan Dziembowski, Tomasz Kazana, and Maciej Obremski. Leakage-resilient non-malleable codes. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015: 12th Theory of Cryptography Conference, Part I*, volume 9014 of *Lecture Notes in Computer Science*, pages 398–426, Warsaw, Poland, March 23–25, 2015. Springer, Heidelberg, Germany.

[ADL14] Divesh Aggarwal, Yevgeniy Dodis, and Shachar Lovett. Non-malleable codes from additive combinatorics. In David B. Shmoys, editor, *46th Annual ACM Symposium on Theory of Computing*, pages 774–783, New York, NY, USA, May 31 – June 3, 2014. ACM Press.

[ADN$^+$19] Divesh Aggarwal, Nico Döttling, Jesper Buus Nielsen, Maciej Obremski, and Erick Purwanto. Continuous non-malleable codes in the 8-split-state model. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 531–561, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany.

[BBB$^+$10] Alessandro Barenghi, Guido Bertoni, Luca Breveglieri, Mauro Pellicioli, and Gerardo Pelosi. Low voltage fault attacks to AES and RSA on general purpose processors. Cryptology ePrint Archive, Report 2010/130, 2010. https://eprint.iacr.org/2010/130.

[BC94] Daniel Pierre Bovet and Pierluigi Crescenzi. *Introduction to the Theory of Complexity*, volume 7. Prentice Hall London, 1994.

[BCG$^+$15] Antonella Barletta, Christian Callegari, Stefano Giordano, Michele Pagano, and Gregorio Procissi. Privacy preserving smart grid communications by verifiable secret key sharing. In *2015 International Conference on Computing and Network Communications (CoCoNet)*, pages 199–204, 2015.

[BDG$^+$18] Marshall Ball, Dana Dachman-Soled, Siyao Guo, Tal Malkin, and Li-Yang Tan. Non-malleable codes for small-depth circuits. In Mikkel Thorup, editor, *59th Annual Symposium on Foundations of Computer Science*, pages 826–837, Paris, France, October 7–9, 2018. IEEE Computer Society Press.

[BDIR18] Fabrice Benhamouda, Akshay Degwekar, Yuval Ishai, and Tal Rabin. On the local leakage resilience of linear secret sharing schemes. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 531–561, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

[BDKM16] Marshall Ball, Dana Dachman-Soled, Mukul Kulkarni, and Tal Malkin. Non-malleable codes for bounded depth, bounded fan-in circuits. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 881–908, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.

[BDKM18] Marshall Ball, Dana Dachman-Soled, Mukul Kulkarni, and Tal Malkin. Non-malleable codes from average-case hardness: AC^0, decision trees, and streaming space-bounded tampering. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 618–650, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

[BFMV22] Gianluca Brian, Sebastian Faust, Elena Micheli, and Daniele Venturi. Continuously non-malleable codes against bounded-depth tampering. Cryptology ePrint Archive, Report 2022/1231, 2022. https://eprint.iacr.org/2022/1231.

[BFO$^+$20] Gianluca Brian, Antonio Faonio, Maciej Obremski, Mark Simkin, and Daniele Venturi. Non-malleable secret sharing against bounded joint-tampering attacks in the plain model. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 127–155, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany.

[BFV19] Gianluca Brian, Antonio Faonio, and Daniele Venturi. Continuously non-malleable secret sharing for general access structures. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part II*, volume 11892 of *Lecture Notes in Computer Science*, pages 211–232, Nuremberg, Germany, December 1–5, 2019. Springer, Heidelberg, Germany.

[BFV21] Gianluca Brian, Antonio Faonio, and Daniele Venturi. Continuously non-malleable secret sharing: Joint tampering, plain model and capacity.

In Kobbi Nissim and Brent Waters, editors, *TCC 2021: 19th Theory of Cryptography Conference, Part II*, volume 13043 of *Lecture Notes in Computer Science*, pages 333–364, Raleigh, NC, USA, November 8–11, 2021. Springer, Heidelberg, Germany.

[BG03] Andreas Blass and Yuri Gurevich. Algorithms: A quest for absolute definitions. *Bull. EATCS*, 81:195–225, 2003.

[BGG$^+$18] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 565–596, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 509–539, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th Annual ACM Symposium on Theory of Computing*, pages 1–10, Chicago, IL, USA, May 2–4, 1988. ACM Press.

[BGW19] Marshall Ball, Siyao Guo, and Daniel Wichs. Non-malleable codes for decision trees. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 413–434, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.

[Bih94] Eli Biham. New types of cryptanalytic attacks using related keys. *Journal of Cryptology*, 7(4):229–246, December 1994.

[BK09] Alex Biryukov and Dmitry Khovratovich. Related-key cryptanalysis of the full AES-192 and AES-256. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 1–18, Tokyo, Japan, December 6–10, 2009. Springer, Heidelberg, Germany.

[Bla79] G. R. Blakley. Safeguarding cryptographic keys. *Proceedings of AFIPS 1979 National Computer Conference*, 48:313–317, 1979.

[Blu81] Manuel Blum. Coin flipping by telephone. In Allen Gersho, editor, *Advances in Cryptology – CRYPTO'81*, volume ECE Report 82-04, pages 11–15, Santa Barbara, CA, USA, 1981. U.C. Santa Barbara, Dept. of Elec. and Computer Eng.

[BS03] Johannes Blömer and Jean-Pierre Seifert. Fault based cryptanalysis of the advanced encryption standard (AES). In Rebecca Wright, editor, *FC 2003: 7th International Conference on Financial Cryptography*, volume 2742 of *Lecture Notes in Computer Science*, pages 162–181, Guadeloupe, French West Indies, January 27–30, 2003. Springer, Heidelberg, Germany.

[BS19] Saikrishna Badrinarayanan and Akshayaram Srinivasan. Revisiting non-malleable secret sharing. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 593–622, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany.

[CCHM19] Binyi Chen, Yilei Chen, Kristina Hostáková, and Pratyay Mukherjee. Continuous space-bounded non-malleable codes from stronger proofs-of-space. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 467–495, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.

[CDM00] Ronald Cramer, Ivan Damgård, and Ueli M. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 316–334, Bruges, Belgium, May 14–18, 2000. Springer, Heidelberg, Germany.

[CFV19] Sandro Coretti, Antonio Faonio, and Daniele Venturi. Rate-optimizing compilers for continuously non-malleable codes. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *ACNS 19: 17th International Conference on Applied Cryptography and Network Security*, volume 11464 of *Lecture Notes in Computer Science*, pages 3–23, Bogota, Colombia, June 5–7, 2019. Springer, Heidelberg, Germany.

[CG14a] Mahdi Cheraghchi and Venkatesan Guruswami. Capacity of non-malleable codes. In Moni Naor, editor, *ITCS 2014: 5th Conference on Innovations in Theoretical Computer Science*, pages 155–168, Princeton, NJ, USA, January 12–14, 2014. Association for Computing Machinery.

[CG14b] Mahdi Cheraghchi and Venkatesan Guruswami. Non-malleable coding against bit-wise and split-state tampering. In Yehuda Lindell, editor, *TCC 2014: 11th Theory of Cryptography Conference*, volume 8349 of *Lecture Notes in Computer Science*, pages 440–464, San Diego, CA, USA, February 24–26, 2014. Springer, Heidelberg, Germany.

[CGL16] Eshan Chattopadhyay, Vipul Goyal, and Xin Li. Non-malleable extractors and codes, with their many tampered extensions. In Daniel Wichs and Yishay Mansour, editors, *48th Annual ACM Symposium on Theory of Computing*, pages 285–298, Cambridge, MA, USA, June 18–21, 2016. ACM Press.

[CGMA85] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *26th Annual Symposium on Foundations of Computer Science*, pages 383–395, Portland, Oregon, October 21–23, 1985. IEEE Computer Society Press.

[CKOS21] Nishanth Chandran, Bhavana Kanukurthi, Sai Lakshmi Bhavana Obbattu, and Sruthi Sekar. Adaptive extractors and their application to leakage resilient secret sharing. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part III*, volume 12827 of *Lecture Notes in Computer Science*, pages 595–624. Springer, 2021.

[CKOS22] Nishanth Chandran, Bhavana Kanukurthi, Sai Lakshmi Bhavana Obbattu, and Sruthi Sekar. Short leakage resilient and non-malleable secret sharing schemes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part I*, volume 13507 of *Lecture Notes in Computer Science*, pages 178–207. Springer, 2022.

[CKR16] Nishanth Chandran, Bhavana Kanukurthi, and Srinivasan Raghuraman. Information-theoretic local non-malleable codes and their applications. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A: 13th Theory of Cryptography Conference, Part II*, volume 9563 of *Lecture Notes in Computer Science*, pages 367–392, Tel Aviv, Israel, January 10–13, 2016. Springer, Heidelberg, Germany.

[CL17] Eshan Chattopadhyay and Xin Li. Non-malleable codes and extractors for small-depth circuits, and affine functions. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *49th Annual ACM Symposium on Theory of Computing*, pages 1171–1184, Montreal, QC, Canada, June 19–23, 2017. ACM Press.

[CL18] Eshan Chattopadhyay and Xin Li. Non-malleable codes, extractors and secret sharing for interleaved tampering and composition of tampering. Cryptology ePrint Archive, Report 2018/1069, 2018. https://eprint.iacr.org/2018/1069.

[CLFT14] Franck Courbon, Philippe Loubet-Moundi, Jacques J. A. Fournier, and Assia Tria. Adjusting laser injections for fully controlled faults. In Emmanuel Prouff, editor, *COSADE 2014: 5th International Workshop on Constructive Side-Channel Analysis and Secure Design*, volume 8622 of *Lecture Notes in Computer Science*, pages 229–242, Paris, France, April 13–15, 2014. Springer, Heidelberg, Germany.

[CMD$^+$18] Brice Colombier, Alexandre Menu, Jean-Max Dutertre, Pierre-Alain Moëllic, Jean-Baptiste Rigaud, and Jean-Luc Danger. Laser-induced single-bit faults in flash memory: Instructions corruption on a 32-bit microcontroller. Cryptology ePrint Archive, Report 2018/1042, 2018. https://eprint.iacr.org/2018/1042.

[CML+11] Gaetan Canivet, Paolo Maistri, Régis Leveugle, Jessy Clédière, Frédéric Valette, and Marc Renaudin. Glitch and laser fault attacks onto a secure AES implementation on a SRAM-based FPGA. *Journal of Cryptology*, 24(2):247–268, April 2011.

[CZ14] Eshan Chattopadhyay and David Zuckerman. Non-malleable codes against constant split-state tampering. In *55th Annual Symposium on Foundations of Computer Science*, pages 306–315, Philadelphia, PA, USA, October 18–21, 2014. IEEE Computer Society Press.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DKO13] Stefan Dziembowski, Tomasz Kazana, and Maciej Obremski. Non-malleable codes from two-source extractors. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 239–257, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

[DORS03] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. Cryptology ePrint Archive, Report 2003/235, 2003. https://eprint.iacr.org/2003/235.

[DPW10] Stefan Dziembowski, Krzysztof Pietrzak, and Daniel Wichs. Non-malleable codes. In Andrew Chi-Chih Yao, editor, *ICS 2010: 1st Innovations in Computer Science*, pages 434–452, Tsinghua University, Beijing, China, January 5–7, 2010. Tsinghua University Press.

[FHMV17] Sebastian Faust, Kristina Hostáková, Pratyay Mukherjee, and Daniele Venturi. Non-malleable codes for space-bounded tampering. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 95–126, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[FMNV14] Sebastian Faust, Pratyay Mukherjee, Jesper Buus Nielsen, and Daniele Venturi. Continuous non-malleable codes. In Yehuda Lindell, editor, *TCC 2014: 11th Theory of Cryptography Conference*, volume 8349 of *Lecture Notes in Computer Science*, pages 465–488, San Diego, CA, USA, February 24–26, 2014. Springer, Heidelberg, Germany.

[FMS01] Scott R. Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the key scheduling algorithm of RC4. In Serge Vaudenay and Amr M. Youssef, editors, *SAC 2001: 8th Annual International Workshop on Selected*

Areas in Cryptography, volume 2259 of *Lecture Notes in Computer Science*, pages 1–24, Toronto, Ontario, Canada, August 16–17, 2001. Springer, Heidelberg, Germany.

[FNSV18] Antonio Faonio, Jesper Buus Nielsen, Mark Simkin, and Daniele Venturi. Continuously non-malleable codes with split-state refresh. 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 121–139, Leuven, Belgium, July 2–4, 2018. Springer, Heidelberg, Germany.

[FP01] Pierre-Alain Fouque and David Pointcheval. Threshold cryptosystems secure against chosen-ciphertext attacks. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 351–368, Gold Coast, Australia, December 9–13, 2001. Springer, Heidelberg, Germany.

[FV19] Antonio Faonio and Daniele Venturi. Non-malleable secret sharing in the computational setting: Adaptive tampering, noisy-leakage resilience, and improved rate. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 448–479, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.

[GA03] Sudhakar Govindavajhala and Andrew W. Appel. Using memory errors to attack a virtual machine. In *2003 IEEE Symposium on Security and Privacy*, pages 154–165, Berkeley, CA, USA, May 11–14, 2003. IEEE Computer Society Press.

[GGD17] Oscar M. Guillen, Michael Gruber, and Fabrizio De Santis. Low-cost setup for localized semi-invasive optical fault injection attacks - how low can we go? In Sylvain Guilley, editor, *COSADE 2017: 8th International Workshop on Constructive Side-Channel Analysis and Secure Design*, volume 10348 of *Lecture Notes in Computer Science*, pages 207–222, Paris, France, April 13–14, 2017. Springer, Heidelberg, Germany.

[GK18a] Vipul Goyal and Ashutosh Kumar. Non-malleable secret sharing. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *50th Annual ACM Symposium on Theory of Computing*, pages 685–698, Los Angeles, CA, USA, June 25–29, 2018. ACM Press.

[GK18b] Vipul Goyal and Ashutosh Kumar. Non-malleable secret sharing for general access structures. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 501–530, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *14th*

Annual ACM Symposium on Theory of Computing, pages 365–377, San Francisco, CA, USA, May 5–7, 1982. ACM Press.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261, Paris, France, May 14–16, 2001. Springer, Heidelberg, Germany.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City, NY, USA, May 25–27, 1987. ACM Press.

[GSZ20] Vipul Goyal, Akshayaram Srinivasan, and Chenzhi Zhu. Multi-source non-malleable extractors and applications. Cryptology ePrint Archive, Report 2020/157, 2020. https://eprint.iacr.org/2020/157.

[GSZ21] Vipul Goyal, Akshayaram Srinivasan, and Chenzhi Zhu. Multi-source non-malleable extractors and applications. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 468–497, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany.

[Gur20] Mordechai Guri. AIR-FI: generating covert wi-fi signals from air-gapped computers. *CoRR*, abs/2012.06884, 2020.

[HJKY95] Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In Don Coppersmith, editor, *Advances in Cryptology – CRYPTO'95*, volume 963 of *Lecture Notes in Computer Science*, pages 339–352, Santa Barbara, CA, USA, August 27–31, 1995. Springer, Heidelberg, Germany.

[HS14] Michael Hutter and Jörn-Marc Schmidt. The temperature side channel and heating fault attacks. Cryptology ePrint Archive, Report 2014/190, 2014. https://eprint.iacr.org/2014/190.

[Jen] Andrew Jenner. 8086 microcode disassembled. Blog post dated 09.2020. URL: https://www.reenigne.org/blog/8086-microcode-disassembled/.

[KHF+19] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas

Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy*, pages 1–19, San Francisco, CA, USA, May 19–23, 2019. IEEE Computer Society Press.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Heidelberg, Germany.

[KJP14] Raghavan Kumar, Philipp Jovanovic, and Ilia Polian. Precise fault-injections using voltage and temperature manipulation for differential cryptanalysis. Cryptology ePrint Archive, Report 2014/782, 2014. https://eprint.iacr.org/2014/782.

[KLT18] Aggelos Kiayias, Feng-Hao Liu, and Yiannis Tselekounis. Non-malleable codes for partial functions with manipulation detection. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 577–607, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

[KMS18] Ashutosh Kumar, Raghu Meka, and Amit Sahai. Leakage-resilient secret sharing. Cryptology ePrint Archive, Report 2018/1138, 2018. https://eprint.iacr.org/2018/1138.

[KMS19] Ashutosh Kumar, Raghu Meka, and Amit Sahai. Leakage-resilient secret sharing against colluding parties. In David Zuckerman, editor, *60th Annual Symposium on Foundations of Computer Science*, pages 636–660, Baltimore, MD, USA, November 9–12, 2019. IEEE Computer Society Press.

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113, Santa Barbara, CA, USA, August 18–22, 1996. Springer, Heidelberg, Germany.

[KOS17] Bhavana Kanukurthi, Sai Lakshmi Bhavana Obbattu, and Sruthi Sekar. Four-state non-malleable codes with explicit constant rate. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part II*, volume 10678 of *Lecture Notes in Computer Science*, pages 344–375, Baltimore, MD, USA, November 12–15, 2017. Springer, Heidelberg, Germany.

[Kra94] Hugo Krawczyk. Secret sharing made short. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 136–146, Santa Barbara, CA, USA, August 22–26, 1994. Springer, Heidelberg, Germany.

[KSW96] John Kelsey, Bruce Schneier, and David Wagner. Key-schedule cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 237–251, Santa Barbara, CA, USA, August 18–22, 1996. Springer, Heidelberg, Germany.

[KSW97] John Kelsey, Bruce Schneier, and David Wagner. Related-key cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA. In Yongfei Han, Tatsuaki Okamoto, and Sihan Qing, editors, *ICICS 97: 1st International Conference on Information and Communication Security*, volume 1334 of *Lecture Notes in Computer Science*, pages 233–246, Beijing, China, November 11–14, 1997. Springer, Heidelberg, Germany.

[KY02] Jonathan Katz and Moti Yung. Threshold cryptosystems based on factoring. In Yuliang Zheng, editor, *Advances in Cryptology – ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 192–205, Queenstown, New Zealand, December 1–5, 2002. Springer, Heidelberg, Germany.

[Li17] Xin Li. Improved non-malleable extractors, non-malleable codes and independent source extractors. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *49th Annual ACM Symposium on Theory of Computing*, pages 1144–1156, Montreal, QC, Canada, June 19–23, 2017. ACM Press.

[Liu68] Chung Laung Liu. *Introduction to Combinatorial Mathematics*. Computer science series. McGraw-Hill, 1968.

[LL12] Feng-Hao Liu and Anna Lysyanskaya. Tamper and leakage resilience in the split-state model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 517–532, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.

[LSG$^+$18] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018: 27th USENIX Security Symposium*, pages 973–990, Baltimore, MD, USA, August 15–17, 2018. USENIX Association.

[LY11] Benoît Libert and Moti Yung. Adaptively secure non-interactive threshold cryptosystems. In Luca Aceto, Monika Henzinger, and Jiri Sgall, editors, *ICALP 2011: 38th International Colloquium on Automata, Languages and Programming, Part II*, volume 6756 of *Lecture Notes in Computer Science*, pages 588–600, Zurich, Switzerland, July 4–8, 2011. Springer, Heidelberg, Germany.

[NS20] Jesper Buus Nielsen and Mark Simkin. Lower bounds for leakage-resilient secret sharing. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 556–577, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany.

[OPVV18] Rafail Ostrovsky, Giuseppe Persiano, Daniele Venturi, and Ivan Visconti. Continuously non-malleable codes in the split-state model from minimal assumptions. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 608–639, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

[Pat16] Kinjal Patel. Secure multiparty computation using secret sharing. In *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*, pages 863–866, 2016.

[Rab89] Michael O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *J. ACM*, 36(2):335–348, apr 1989.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978.

[Sha49] Claude E. Shannon. Communication theory of secrecy systems. *Bell Systems Technical Journal*, 28(4):656–715, 1949.

[Sha79] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

[Shi] Ken Shirriff. A look at the die of the 8086 processor. Blog post dated 06.2020. URL: https://www.righto.com/2020/06/a-look-at-die-of-8086-processor.html.

[SIR02] Adam Stubblefield, John Ioannidis, and Aviel D. Rubin. Using the Fluhrer, Mantin, and Shamir attack to break WEP. In *ISOC Network and Distributed System Security Symposium – NDSS 2002*, San Diego, CA, USA, February 6–8, 2002. The Internet Society.

[SV19] Akshayaram Srinivasan and Prashant Nalini Vasudevan. Leakage resilient secret sharing and applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 480–509, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.

[SW05] Geoff Sullivan and Frode Weierud. Breaking german army ciphers. *Cryptologia*, 29(3):193–232, 2005.

[TW88] Martin Tompa and Heather Woll. How to share a secret with cheaters. *Journal of Cryptology*, 1(2):133–138, June 1988.

[Van09] Vanish: Increasing data privacy with Self-Destructing data. In *18th USENIX Security Symposium (USENIX Security 09)*, Montreal, Quebec, August 2009. USENIX Association.

[WHH$^+$10] Scott Wolchok, Owen S. Hofmann, Nadia Heninger, Edward W. Felten, J. Alex Halderman, Christopher J. Rossbach, Brent Waters, and Emmett Witchel. Defeating vanish with low-cost sybil attacks against large DHTs. In *ISOC Network and Distributed System Security Symposium – NDSS 2010*, San Diego, CA, USA, February 28 – March 3, 2010. The Internet Society.