Protecting Cryptographic Memory against Tampering Attack

A dissertation

submitted for the partial fulfillment of

Doctor of Philosophy
by
Pratyay Mukherjee
Department of Computer Science
Aarhus University

August 31, 2015

This is the author's version with minor modifications to the official version

The author thanks Serge Fehr for useful comments on the preliminary draft.

To Madhubanti...

Acknowledgments

First and foremost, I would like to thank my supervisor Jesper Buus Nielsen. Not only he guided me with his vast knowledge, but also he always gave me plenty of freedom on everything, from choosing my research topic to traveling to any conference/research visit. His attitude towards research and life indeed helped me a lot to take important decisions, get rid of confusions and boost confidence. I would like to thank our leader Ivan Damgård, who's name is almost synonymous with the Aarhus Crypto Group. The most amazing thing about Ivan is that while talking to him it is hard to imagine that he indeed is the man with such enormous legacy. If there is any question, however stupid it might be, he is there with an eternal smile to entertain it. I thank Ivan for all the wonderful interactions.

I owe most of my technical learning during my PhD to Sebastian (Faust) and Daniele (Venturi), who were post-docs at Aarhus. It was indeed a dream team with hours of brainstorming day after day (sometimes including Ivan or Jesper) and was extremely fruitful in terms of research. Apart from work, we became the best buddies too and three of us were given the nickname "The Family" (Ivan sometimes played the role of "Godfather" of "The Family"). I thank you guys for all the wonderful memories and giving me a chance to work with you.

I would like to thank Daniel Wichs, who gave me a chance to spend a wonderful year with him at Boston. It was a transitional period for me as at that time I started to work on something new and completely different from the earlier topics I worked on. Nonetheless, his amazing explanations made it quite easy for me to pick up a new area quickly. I learned enormously from the interactions with him on technical ideas and more significantly how to view a problem correctly.

I spent three wonderful months at Warsaw hosted by Stefan Dziembowski. It was quite an amazing experience spending a summer in such a vibrant city. I thank him for giving me chance to spend time there and interact with his group on several interesting stuffs.

I would also like to thank Vipul Goyal, Yevgeniy Dodis, Rishiraj Bhattacharyya and Sanjam Garg for inviting me to visit them and for giving me a opportunities to spend wonderful time with them. Those have been significantly stimulating experiences and had great impact on my technical abilities.

I would like to express gratitude to my amazing officemate Tore, colleagues Claudio and Thomas for helping me to settle down in a new country. Especially, I thank Tore for all the help I got from him in numerous occasions. I would like to thank all the fellow PhD students Antigoni, Bernardo, Carsten, Irene, Michael, Rasmus, Roberto, Peter, Sigurd, Morten, Valerio, Rikke, Angela, Sarah and Pavel; post-docs Tomas, Samuel, Nacho, Mario and Frederic; administrative staffs Dorthe and Ellen for making my life at Aarhus full of fun and filled with wonderful memories.

I could never do anything in my life without the support of my family. I would like to thank my parents and my sister for always extending their support in all I have done, and never asked for anything in return.

Finally, this dissertation would have been incomplete without mentioning the name of the most important person in my life, my wife Madhu. In fact, I hardly have words to express gratitude to her. Indeed, I believe that she has equal, if not more, contribution to this dissertation, as I have, through her constant support, love and sacrifices over the years. Thank you for everything !

Abstract

In this dissertation we investigate the question of protecting cryptographic devices from tampering attacks. Traditional theoretical analysis of cryptographic devices is based on *black-box* models which do not take into account the attacks on the implementations, known as *physical attacks*. In practice such attacks can be executed easily, e.g. by heating the device, as substantiated by numerous works in the past decade. Tampering attacks are a class of such physical attacks where the attacker can change the memory/computation, gains additional (non-black-box) knowledge by interacting with the faulty device and then tries to break the security. Prior works show that generically approaching such problem is notoriously difficult. So, in this dissertation we attempt to solve an easier question, known as memory-tampering, where the attacker is allowed to tamper only with the memory of the device but not the computation. Such weaker model can still be practically useful and moreover, may provide nice building-blocks to tackle full-fledged tampering in future.

In this dissertation we study different models of memory-tampering and provide a number of solutions with different flavors. Mainly we took two different approaches: (i) securing specific schemes against tampering and (ii) constructing a generic transformation which turns any scheme resilient to tampering. In Chapter 3 we take the first approach and propose several tamper-resilient public-key schemes in a new model which allows arbitrary tampering, but only bounded number of times [DFMV13]. We provide solutions mainly for identification schemes and encryption schemes. The second approach is based on an abstract notion called *non-malleable codes* introduced in an earlier work. In Chapter 5 and 6 we mainly improve the state-of-art of non-malleable codes. In Chapter 5 we provide new constructions of such codes [FMVW14], which implicitly resolve the question of memory-tampering in an important model using the known transformation. In the same chapter we also introduce a new and related notion called *non-malleable key-derivations* which are found to be useful in tamper-resilience as well. Finally in Chapter 6 we strengthen the prior definitions of non-malleable codes by considering continuous tampering [FMNV14]. We provide a construction which satisfies the stronger definition. This strengthening against continuous tampering provides new and better solutions for generic tamper-resilience which removes the requirement of *erasures* that were necessary in earlier transformations. We explicitly present the new transformations based on our continuous notion.

Abstract (in Danish)

I denne afhandling undersøger vi beskyttelse af kryptografiske enheder mod manipulations-angreb. Traditionel teoretisk analyse af kryptografiske enheder er baseret på *black-box* modeller, som ikke tager hensyn til angrebene på de implementeringer, kendt som *fysiske angreb*. I praksis kan sådanne angreb udføres nemt, feks ved opvarmning af enheden, som dokumenteret af talrige artikler i det seneste årti. Manipulations-angreb er en klasse af sådanne fysiske angreb, hvor hackeren kan ændre hukommelsen/beregning, får ekstra (ikke-black-box) viden ved at interagere med den defekte enhed og derefter forsøger at bryde sikkerheden. Tidligere værker viser at det er notorisk vanskeligt at nærme sig sådan et problem generisk. Så i denne afhandling vil vi forsøge at løse et lettere problem, kendt som memory-manipulation, hvor angriberen er tilladt sabotage kun af hukommelsen på enheden, men ikke beregningen. En sådan svagere model kan stadig være praktisk anvendelig og kan desuden være et trinbræt til at tackle fuldgyldigt manipulation i fremtiden.

I denne afhandling undersøger vi forskellige modeller af hukommelses-manipulation og giver en række løsninger med forskellige smagsvarianter. Hovedsageligt tog vi to forskellige tilgange: (i) sikring af specifikke ordninger mod manipulation og (ii) at konstruere en generisk forvandling, der forvandler enhver ordning så denne bliver modstandsdygtig over for manipulation. I kapitel 3 tager vi den første tilgang og foreslår flere tamper-elastiske offentlig-nøgle ordninger i en ny model, som giver mulighed for vilkårlige indgreb, men kun et begrænset antal gange [DFMV13]. Vi leverer løsninger primært til identifikation ordninger og kryptering ordninger. Den anden tilgang er baseret på et abstrakt begreb kaldet *ikkesmedbart koder* indført i en tidligere artikel. I kapitel 5 og 6 vil vi primært forbedre state-of-art af ikkesmedbart koder. I kapitel 5 leverer vi nye konstruktioner af sådanne koder [FMVW14], hvilket implicit løser spørgsmålet om hukommelse-manipulation i en vigtig model, ved hjælp af den kendte transformation. I samme kapitel vil vi også indføre en ny og relateret begreb kaldet *ikkesmedbart nøgle-afledninger* som viser sig at være nyttige i tamper-modstandskraft så godt. Endelig i kapitel 6 styrker vi de forudgående definitioner af ikkesmedbart koder ved at overveje kontinuerlig manipulation [FMNV14]. Vi giver en konstruktion, som opfylder denne stærkere definition. Denne styrkelse mod kontinuerlig manipulation giver nye og bedre løsninger til generisk tamper-modstandsdygtighed, der fjerner kravet om *raderinger*, der var nødvendige i tidligere transformationer. Vi præsenterer de nye transformationer eksplicit på grundlag af vores kontinuerlige begreb.

Contents

Chapter 1

Introduction

1.1 Physical attacks on Cryptographic devices

With the enormous development in the technology of computation over the past few decades, our lives have become "highly digitalized" now-a-days. A large number of day-to-day activities have been transformed to digitalized computer-based systems. Although these transformations provide us huge transparency and convenience, at the same time they leave novel potential vulnerabilities to frauds, thereby rendering convenience and security to pose as somewhat conflicting goals to achieve. We, as cryptographers, aim to ensure security using mathematical modeling of such real-life systems without significant sacrifice in convenience and transparency.

Among all the "digitalizations", a major transformation took place in the everyday payment-system, in that *smart-card based payment* is now widely-used and (almost) universally accepted. Essentially, these smart cards are small hardware devices that implement specific cryptographic algorithms. Here, the role of cryptography is to ensure *integrity* and *confidentiality* in case of theft by some potentially malicious user, of course without altering the basic functionality of the device. To initiate a formal analysis of security, first we need an adequate mathematical model.

Traditionally, the security of cryptographic algorithms are analyzed in *black-box model*, in that an adversary may execute the device and observe its input-output behavior and then try to "break" using the foreseen knowledge. In particular, such modeling assumes that the device computes the algorithm in a "fully trusted" environment. More formally, the main idea, introduced by Goldwasser and Micali in a ground-breaking work [GM84], is to first develop an attack-model, formally describe the capabilities of an adversary and define what it means "to break" the security. The next step is to construct cryptographic schemes for which, under reasonable assumptions (e.g. factoring is "hard"), one can derive a *formal proof* showing that no adversary with such capabilities can "break" the definition of security. Though this modeling is found to be useful in most of the real-life scenarios, sometimes, e.g. for the case of smart-card theft, it fails to match reality. It is observed that there could be scenarios where a more notorious attacker can attack the algorithm's implementation without hurting the underlying mathematical principles. There are several means to carry out such attacks: e.g. by measuring the electromagnetic radiation generated by the execution (e.g. [HAS10, Kal] etc.), by measuring the power consumption (e.g. [HTH+06, L+02, LST12, KJJ99] etc.), by measuring the sounds (e.g. [ST04, GST14] etc.) produced by the device, by measuring running times (e.g. [Koc95, Koc96] etc.), by injecting some intentional fault (e.g. [BDL97, ABF+03, KQ07, AK96, BDL01, SA02] etc.) and then observing the input-output behavior etc. (c.f. to [Boc10] for many examples of how such physical phenomenons can be exploited in attacks). These kinds of attacks are broadly known as *physical attacks* and they can lead to "total breach" of security even if the proofs remain correct! *So what goes wrong?* Evidently, one can see that the problem lies within the modeling because the black-box model described above does not really capture these kind of physical attacks where the adversary can potentially get *additional information* apart from the input-output behavior of the device.

Over the past decade there has been significant progress towards protecting cryptographic devices against such physical attacks from theoretical perspectives. Numerous works proposed several extensions of the traditional black-box model and constructed cryptographic schemes secure in those stronger models. Theoretically, all such models of physical attacks can be categorized into two main branches: (i) *leakage* where by some means the "passive" attacker gains some partial knowledge[1] about the secret state of the device; and (ii) *tampering* where the attacker "actively" tampers with the device and then observes the input-output behavior of the "faulty" device. The branch dealing with leakage attacks is called *leakage-resilient cryptography* which has been explored by numerous works e.g. [ISW03, MR04, DP08, Pie09, DKL09, FKPR10, FRR+10, BKKV10] and many more. Note that in this model, the adversary is restricted to behave *passively* and can only obtain some additional secret knowledge. On the other hand, in tampering attacks the adversary *actively* interacts with the device by, e.g., inducing deliberate *faults* into the computation and observing its result at the output. Such faults can be induced by several means by e.g. heating up the device, exposing it to infrared radiation, or altering the internal power supply or even clock [AK96, BDL01, SA02]. Tampering attacks were found to have devastating consequences: in a seminal paper, Boneh, DeMillo and Lipton [BDL01] showed that even a *single, random* fault suffices to attack an RSA-based digital signature scheme. Such attacks were also found to be successful against other cryptographic primitives like block ciphers in several other works [BS97, SYO09, MSS06] etc.

The branch dealing with such attacks from theoretical point of view is known as *tamper-resilient cryptography* which is our primary focus area in this dissertation. In particular, we try to develop theoretical models that incorporate such tampering attacks and design cryptographic schemes that are *provably secure* even when implemented on a *tamperable* device. Our analysis is proof-based and involves rigorous mathematical treatment.

1.2 Models of tampering

As a first step we discuss how to model the physical phenomena of tampering. "Modifying a device via tampering" is generally modeled as "applying tampering functions to the computation" with appropriate input/output domain. Notice that, the adversary may not see the entire input-output of the functions as that may contain some secret part. So, in that sense the tampering is partially "blind". Note that, once tampered, the attacker gets oracle access to the faulty algorithm and thus can observe the "tampered" input-output behavior of the device. Let us formalize a bit now. Without loss of generality, assume a framework where some cryptographic functionality with some secret key can be represented as a pair $(\mathcal{C}, \mathsf{st})$ where circuit $\mathcal{C}$ implements the functionality and the secret state st contains the key. For example, $\mathcal{C}$ might be the circuit implementing some block cipher like AES with secret state st. Now, the most general model of tampering we can think of is as follows: the adversary applies an arbitrary tampering function f resulting in a tampered circuit $\mathcal{C}'$ with tampered memory st' to get the faulty device with $(\mathcal{C}', \mathsf{st}') = f(\mathcal{C}, \mathsf{st})$. Then it gets oracle access to $\mathcal{C}'(\mathsf{st}', \cdot)$. However, it is easy to observe that if f tampers in a way such that $\mathcal{C}'$ just outputs st, then there is no hope of achieving security. So we need to put restriction on the type of functions allowed, which is modeled by assuming that the adversary can tamper only with functions from

[1] Note that if the adversary can recover the entire secret state, then there is obviously no counter-measure possible. So the natural assumption in this direction is to consider that the adversary has some partial knowledge which may be as large as 99% of the secret state, depending on the model, but not the entire secret.

some *admissible* family $\mathcal{F}$. In general a model of tampering typically refers to $\mathcal{F}$ that is what is the class of allowed tampering functions. Obviously, the richer the family $\mathcal{F}$ the stronger is the model. Apart from this main feature, our modeling will also depend on other standard features like the computational power of adversary, that is whether the adversary is unbounded or a probabilistic polynomial time (PPT) algorithm.

1.2.1 Generic framework: Tampering with computation

A framework of tampering in its most general form (as discussed above) allows the attacker to tamper with the entire computation: the circuit $\mathcal{C}$ and the secret st. Such model, commonly known as *tampering with computation* has been considered first time by Ishai et al. [IPSW06]. Essentially they build a circuit compiler which takes any circuit implementing some crypto-functionality and compiles it to a tamper-proof circuit. Despite the framework being extremely generic and realistic, the positive results shown in [IPSW06] were not so encouraging. Their construction could only achieve security against some very limited class $\mathcal{F}$, in particular the tampering function is allowed to tamper only a small fraction of wires in each invocation. Later the class $\mathcal{F}$ has been broadened by a few works [FPV11, DSK12, DK14, GIP+14] though all of them suffered from similar limitations on $\mathcal{F}$ making this modeling rather distant from reality of fault attack. Moreover, in spite of achieving weaker result, the analysis on those works involved "heavy tools" like probabilistically checkable proofs (in [DSK12]) rendering the problem of tamper-resilience harder to tackle.

1.2.2 Restrictive framework: Memory-only tampering

The difficulty faced in the general framework necessitated considering a restricted framework. The main hope was (i) the restriction would allow tamper-resilience for "large" and "meaningful" $\mathcal{F}$; (ii) the analysis would be much simpler. Towards that a framework, so-called *memory-only tampering* has been considered, pioneered by the work of Gennaro et al. [GLM+04]. Here the adversary is *not* allowed to tamper with the circuit or computation $\mathcal{C}$ but only with the secret memory st. In particular, this constraints the domain of the functions $f \in \mathcal{F}$ to apply only on st (in contrast to the earlier model where it applies to both $\mathcal{C}$ and st). Although this restricts the power of adversary significantly, it was indeed an interesting model to consider because:

- This is a much *simpler* problem to encounter and can be considered as a *building-block* to realize the generic framework. In fact, similar approaches have been quite successful in the past in the context of leakage-resilience.

- It can have some important *practical benefits*:

  - Tamper-resilience in "memory-only" framework implies that now the hardware-designer's job is left to design *only* a tamper-proof circuit as the memory part has already been taken care of theoretically, i.e. on the software level.

  - Moreover, in general the memory of a device changes much more frequently compared to the circuitry and thus designing a scheme which is tamper-proof against memory-tampering reduces the production cost significantly.

This line of research drew attentions from a significant number of researchers in the past. To protect against memory-only tampering attack, two major approaches[2] have been considered in the literature:

Approach-1: Specific Tamper-resilient Schemes. One approach is to build specific crypto-schemes resilient to memory-tampering attack. For example constructing tamper-resilient public-key encryption scheme or symmetric-key primitives like pseudo-random function (PRF) etc. In this approach, the ideas mostly differ from each other which is naturally due to the fundamental differences between the properties of different primitives. This direction of research attracted significant attention in the last decade and was called Related-Key-Attack (RKA in short)-security in majority of these works. Bellare and Kohno [BK03] initiated the theoretical study of related-key attacks. Their results, mainly focused on symmetric key primitives (e.g. PRP, PRF), proposed various block-cipher based constructions which are RKA-secure against certain restricted classes of tampering functions. Their constructions were further improved by [Luc04, BC10]. Following these works other cryptographic primitives were constructed that are provably secure against certain classes of related key attacks. Most of these works consider rather restricted tampering functions that, e.g., can be described by a linear or affine function [BK03, Luc04, BC10, AHI11, Pie12, Wee12, BR13, ABP15]. A few important exceptions are described below.

In [BPT12] the authors showed how to go beyond the linear barrier by extending the class of allowed tampering functions to the class of polynomials of bounded degree for a number of public-key primitives. In [ABPP14] Abdalla et al. constructed PRFs against a similar class of functions. Also, the work of Goyal, O'Neill and Rao [GOR11] considered polynomial relations that are induced to the inputs of a hash function. Finally Bellare, Cash and Miller [BCM11] developed a framework to transfer RKA security from a pseudorandom function to other primitives (including many public key primitives).

Apart from RKA-based works, in a beautiful work, Kalai Kanukrthi and Sahai [KKS11] provided constructions of tamper-resilient encryption and signature schemes. We take this approach in the work discussed in Chapter 3.

Approach-2: Generic Tamper-resilience (via Non-malleable Codes) Another approach attempts to construct a *generic* compiler which transforms *any* cryptographic functionality into a "hardened" functionality such that the later is protected against memory-tampering. Although, this direction of work has been first considered by Gennaro et al. in [GLM+04], it gained more popularity with the proposal of the beautiful abstraction of non-malleable codes by Dziembowski, Pietrzak and Wichs [DPW10]. In that paper they proposed a simple tamper-resilient (generic) compiler which uses non-malleable codes as the main ingredient. Moreover, they showed a feasibility result which proves existence of such codes[3] against very large $\mathcal{F}$[4] and also proposed a concrete efficient construction in much weaker model (i.e. much smaller $\mathcal{F}$) leaving a great scope

[2]These major approaches are not specific to memory-tampering. It is possible to consider them in the context of tampering with computation as well. However in the past only the generic approach has been considered for that direction.

[3]They used an excellent probabilistic method argument which, however, resulted in an inefficient construction. Nonetheless, due to inherent impossibility of non-malleable codes for all functions, this result is very important and provides huge optimism for further investigations.

[4]Notice that similar to the tampering models, non-malleable codes are also defined with respect to some family of functions $\mathcal{F}$ (and w.r.t. the computing power of the adversary).

for improvement. After that, there have been a series of works [LL12, DKO13, CKM11, ADL14, ADKO15a, ADKO15b, CG14b, CG14a, AGM+15b, AGM+15a] focusing on improving constructions of non-malleable codes, which implicitly improve the state-of-art of tamper-resilience as that automatically implies to a better protections (a richer $\mathcal{F}$) against memory-tampering via the generic compiler [DPW10].

Informally, non-malleable codes are defined with respect to a family of functions $\mathcal{F}$ with the following guarantee: if the codeword is tampered with any function $f \in \mathcal{F}$ then the tampered codeword is either invalid (that is error/tampering has been detected) or will contain an "unrelated value". Once we have that guarantee the tamper-resilient compiler [DPW10] works as follows: for a cryptographic circuit $\mathcal{C}$ with some secret st, encode st with any non-malleable code to produce a codeword $c \leftarrow \mathsf{Enc}(\mathsf{st})$. Then the hardened functionality $\widehat{\mathcal{C}}$ works by just decoding c to get the st and then running $\mathcal{C}(\mathsf{st}, \cdot)$ on the input. However, if the decoding returns an invalid value, then the functionality stops working or "self-destructs". Clearly security measure of such tampered functionality depends on the class of functions tolerated by the underlying non-malleable codes.

In Chapter 4 we provide formal definitions and some more details on non-malleable codes. Our results from Chapter 5 and Chapter 6 are mostly about new definitions and constructions of non-malleable codes.

1.2.3 A few remarks

No tampering with randomness. Importantly, we remark that in this thesis we always assume that the randomness required in the computation[5] is *not* tamperable. In [ACM+14] Austrin et al. showed that even very little tampering with the randomness leads to break securities of majority of the crypto-schemes. That direction of research is an interesting one, albeit not in the scope of this dissertation.

Considering tampering and leakage together. In reality it is "unreasonable" to assume that the attacker can only tamper with the device, but gets no leakage. Therefore, we put additional effort in order to account for *tampering and leakage together*. In this dissertation, some of our models consider explicit leakage (e.g. results in Chapter 3), whereas others can be easily extended to a joint tampering-leakage model. For example the generic approach to construct a tamper-resilient compiler via non-malleable codes does not consider leakage explicitly. However, due to its black-box nature, if the underlying scheme is leakage-resilient then the transformation maintains that, thereby enabling modular treatment.

Deterministic vs Probabilistic Tampering Throughout we assume the tampering functions chosen by the adversary to be deterministic. This is without loss of generality as instead of using a randomized function, the (randomized) adversary can hard-wire the randomness into the function[6].

1.3 Our contribution

In this dissertation we propose different means of protecting cryptographic devices from memory-tampering attack. It is mainly based on three separate works. We briefly summarize them below.

[5] Majority of cryptographic primitives are necessarily randomized.

[6] In general, a tampering function is implemented as a circuit and hence a randomized adversary can easily sample a randomness and hard-wire it into the tampering circuit.

1.3.1 Bounded-Tamper Resilience

In this work [DFMV13] we mainly propose a new tampering model called *bounded tampering* and provide constructions of specific public-key schemes like identification schemes (and thereby signatures via a "standard transformation") and encryptions in this model. This type of modeling can also be considered as an extension of previous RKA-models: from that perspective we show that it is possible to go beyond the *algebraic barrier* and achieve security against *arbitrary* key relations, albeit by restricting the number of tampering queries[7]. The latter restriction is necessary in case of arbitrary key relations, as otherwise a generic attack of Gennaro et al. [GLM+04] shows how to recover the key of almost any cryptographic primitive. We summarize the main results in this work below.

1. We show that standard identification and signature schemes constructed from a large class of Σ-protocols (including the Okamoto scheme, for instance) are secure even if the adversary can *arbitrarily* tamper with the prover's state a *bounded* number of times. Interestingly, for the Okamoto scheme we can allow also independent tampering with the public parameters (e.g. the prime modulus).

2. We show a *bounded* tamper resilient CCA-secure public key cryptosystem based on the DDH assumption. We first define a weaker CCA-like security notion that we can instantiate based on DDH, and then we give a general compiler that yields CCA security with tamper and leakage resilience. This requires a public *tamper-proof* common reference string (CRS).

3. Finally, we explain how to boost bounded tampering (as in 1. and 2. above) to *continuous* tampering, in the so-called *floppy model* where each user has a personal hardware token (which is not subject to tampering/leakage) which can be used to *refresh* the secret key.

We believe that bounded tampering is a meaningful and interesting alternative to avoid known impossibility results and can provide important insights into the security of existing standard cryptographic schemes. All of the above results also considers bounded leakage and hence the model is called *Bounded Leakage and Tampering* (BLT).
This works is discussed in Chapter 3.

1.3.2 Efficient Non-Malleable Codes, Key-derivations and their applications in Tamper-resilience

It is easy to see that one *cannot* construct *efficient* non-malleable codes that are secure against *all efficient* tampering functions f: the attacker can use the particular function f_{bad} which has the encoding/decoding algorithm hard-coded into it (such function must exists in the set of *all* efficient functions) and then it first decodes the codeword to obtain the message and re-encodes a related message. However, in [FMVW14] we show "the next best thing": for any bound s (polynomial in the security parameter) given a-priori, there is an efficient non-malleable code that protects against all tampering functions f computable by a circuit of size s. More generally, for any family of tampering functions $\mathcal{F}$ of size $|\mathcal{F}| \leq 2^s$, there is an efficient non-malleable code that protects against all $f \in \mathcal{F}$. The *rate* of our codes, defined as the ratio of message to codeword size, approaches 1.

[7] It is worth noting here that RKA-models allow unbounded tampering. So our model can be considered as complimentary to the traditional RKA-models.

Our results are information-theoretic and our main proof technique relies on a careful probabilistic method argument using limited independence. As a result, we get an efficiently samplable family of efficient codes, such that a random member of the family is non-malleable with overwhelming probability. Alternatively, we can view the result as providing an efficient non-malleable code in the "common reference string" (CRS) model. So, using this code in the above compiler we can get a tamper-resilient functionality in the CRS model which protects against unbounded tampering adversary choosing function from a "small enough" class of functions.

In the same work we also introduce a new and related notion of non-malleable key derivation, which uses randomness x to derive a secret key $y = h(x)$ in such a way that, even if x is tampered to a different value $x' = f(x)$, the derived key $y' = h(x')$ does not reveal any information about y. Our results for non-malleable key derivation are analogous to those for non-malleable codes. Interestingly, this found to be useful to protect stream-ciphers from tampering attacks. Moreover, we show a generic compiler to protect against one-time tampering (a more restricted version of bounded-tampering) using non-malleable key-derivation as a black-box. This shows that in certain scenarios this can be preferable to protect against tampering.

As a useful tool in our analysis, we rely on the notion of "leakage-resilient storage" of Davì, Dziembowski and Venturi [DDV10] and, as a result of independent interest, we also significantly improve on the parameters of such schemes.
This work is discussed in Chapter 5.

1.3.3 Tamper-resilience using Continuous Non-Malleable Codes

In another work [FMNV14] we introduce an extension of the standard non-malleability security notion – so-called *continuous* non-malleability – where we allow the adversary to tamper *continuously* with an encoding. This is in contrast to the standard notion of non-malleable codes where the adversary is allowed to tamper with an encoding only *once*. We show how to construct continuous non-malleable codes in the popular "split-state" model. In a split-state model a codeword c consists of two parts c_0, c_1 and each of them can be tampered *independently* but by *arbitrary* tampering functions. We outline the main results of this work below:

1. We propose a new *uniqueness* requirement of split-state codes which states that it is computationally hard to find two codewords $c = (c_0, c_1)$ and $c' = (c_0, c_1')$ such that both codewords are valid, but one of the parts (c_0 here) is the same. A simple attack shows that *uniqueness is necessary* to achieve continuous non-malleability in the split-state model. Moreover, we illustrate that none of the existing constructions satisfies our uniqueness property and hence is secure in the continuous setting. This can be extended to show that it is *impossible* to construct information theoretically secure continuous non-malleable codes in the split-state model as having one contradicts with the uniqueness property.

2. We construct a split-state code satisfying continuous non-malleability. Our scheme is based on the inner product function, collision-resistant hashing and non-interactive zero-knowledge proofs of knowledge and requires a tamper-proof CRS.

3. Finally using such codes we construct a new memory-tampering compiler which works even when there is *no erasure* available and the adversary is allowed to make many copies of the original codeword and save them into the memory. Previous compilers requires to *perfectly erase* the entire memory after each execution and does not consider setting where the size

Results	Type of Tamp.	No. of Tamp	Self-destruct	Schemes	CRS	Adversary	Erasure	Technique
Chapter 3	Arbitrary	Bounded	No	ID-schemes, PKE	Not reqd.	PPT	No	LR
Chapter 5	poly-size circuits	Unbounded	Yes	Any	Required	Unbounded	Yes	NMC
	poly-size circuits	Unbounded	No	Stream Cipher	Required	PPT	No.	NMKD
Chapter 6	Split-state	Unbounded	Yes	Any	Required	PPT	No	NMC

Table 1.1: An overview of our main results. LR refers to the fact that the techniques in Chapter 3 are based on techniques from leakage-resilience (from very high-level). We emphasize that this table does *not* exactly capture all our result, but only gives a high-level comparative overview.

of codeword is much smaller than the entire memory. Our compiler using continuous non-malleable codes avoid these restrictions.

This work is presented in Chapter 6.

Comparison among our main results. In Table 1.1 we provide a high-level comparison of our main results based on different features of tamper-resilience. However, we oversimplify some results to maintain compactness of the table. For exact results we refer to the corresponding chapters.

1.4 Roadmap

In Chapter 2 we provide some basic notations and well-known definitions which are used later in the dissertation. Then in Chapter 3 we describe our results on bounded-tamper resilience [DFMV13]. Chapter 4 provides an introduction to non-malleable codes including formal definitions, a brief description of the generic compiler [DPW10] and a brief survey on the recent results related to non-malleable codes. Next, in Chapter 5 we present our construction of efficient non-malleable codes (and key-derivations) against poly-size circuit and some applications of non-malleable key-derivations [FMVW14]. Finally in Chapter 6 we provide the results on continuous non-malleable codes and the new compiler for tamper-resilience without erasure [FMNV14]. The dissertation is concluded in Chapter 7 with brief descriptions of some subsequent works based on the results presented here and some prospective future directions.

Chapter 2

Preliminaries

In this chapter we provide some common notations and basic definitions of known cryptographic primitives used throughout the rest of the thesis. However, chapter-specific definitions and notations are left out and provided in the corresponding chapters.

2.1 Some basic notations

We let $\mathbb{N}$ be the set of naturals. For $n \in \mathbb{N}$, we write $[n] := \{1, \ldots, n\}$. Let X, Y be random variables with supports $S(X), S(Y)$, respectively[1] The statistical distance between two random variables X and Y is defined as:

$$\mathbf{SD}(X,Y) \stackrel{\text{def}}{=} \frac{1}{2} \sum_{s \in S(X) \cup S(Y)} |\Pr[X = s] - \Pr[Y = s]|$$

We write $X \approx_\varepsilon Y$ and say that X and Y are ε-statistically close to denote that $\mathbf{SD}(X,Y) \leq \varepsilon$. We let U_n denote the uniform distribution over $\{0,1\}^n$. We use the notation $x \leftarrow X$ to denote the process of sampling a value x according to the distribution X. If A is a randomized algorithm, we write $\mathsf{A}(x;r)$ to denote the execution of A on input x with random coins r. We let $\mathsf{A}(x)$ denote a random variable over the random coins r. Expectation of a random variable X is denoted by $\mathbf{E}[X]$.

Throughout the thesis we denote the security parameter by $\lambda \in \mathbb{N}$. A function $\delta(\lambda)$ is called *negligible* in λ (or simply negligible) if it vanishes faster than the inverse of *any* polynomial in λ, i.e., $\delta(\lambda) = \lambda^{-\omega(1)}$. An algorithm S is called *probabilistic polynomial time* (PPT) if for any input $x \in \{0,1\}^*$ the computation of $\mathsf{S}(x)$ terminates in at most $poly(|x|)$ steps and S is randomized. Two random variables X and Y are computationally indistinguishable means that there exists no PPT algorithm D (also called distinguisher) which can distinguish between X and Y with non-negligible probability.

Vectors are denoted in bold. Given a vector $\mathbf{x} = (x_1, \ldots, x_\ell)$ and some integer a, we write $a^{\mathbf{x}}$ for the vector $(a^{x_1}, \ldots, a^{x_\ell})$. The inner product of $\mathbf{x} = (x_1, \ldots, x_\ell)$ and $\mathbf{y} = (y_1, \ldots, y_\ell)$ is $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{\ell} x_i \cdot y_i$.

2.2 Basics of information theory.

The min-entropy of a random variable X over a set $\mathcal{X}$ is defined as $\mathbf{H}_\infty(X) := -\log \max_x \Pr[X = x]$, and measures how X can be predicted by the best (unbounded) predictor. The conditional average min-entropy [DORS08] of X given a random variable Z (over a set $\mathcal{Z}$) possibly dependent on X, is defined as $\widetilde{\mathbf{H}}_\infty(X|Z) := -\log \mathbf{E}_{z \leftarrow Z}[2^{-\mathbf{H}_\infty(X|Z=z)}]$. Following [ADW09], we sometimes rephrase the notion of conditional min-entropy in terms of predictors A that are given some information Z

[1] Throughout we alternatively use the terms random variables and the distributions they represent abusing notations.

(presumably correlated with X), so $\widetilde{\mathbf{H}}_\infty(X|Z) = -\log(\max_{\mathsf{A}} \Pr[\mathsf{A}(Z) = X])$. The above notion of conditional min-entropy can be generalized to the case of interactive predictors A, which participate in some randomized experiment $\mathcal{E}$. An experiment is modeled as interaction between A and a challenger oracle $\mathcal{E}(\cdot)$ which can be randomized, stateful and interactive.

Definition 2.2.1 ([ADW09]). *The conditional min-entropy of a random variable X, conditioned on the experiment $\mathcal{E}$ is $\widetilde{\mathbf{H}}_\infty(X|\mathcal{E}) = -\log(\max_{\mathsf{A}} \Pr\left[A^{\mathcal{E}(\cdot)}() = X\right])$. In the special case that $\mathcal{E}$ is a non-interactive experiment which simply outputs a random variable Z, then $\widetilde{\mathbf{H}}_\infty(X|Z)$ can be written to denote $\widetilde{\mathbf{H}}_\infty(X|\mathcal{E})$ abusing the notation.*

We will rely on the following basic lemma from [DORS08] in Chapter 3.

Lemma 2.2.2 (Lemma 2.2 of [DORS08]). *For all random variables X, Z and Λ over sets $\mathcal{X}$, $\mathcal{Z}$ and $\{0,1\}^\zeta$ we have*

$$\widetilde{\mathbf{H}}_\infty(X|Z,\Lambda) \geq \widetilde{\mathbf{H}}_\infty(X|Z) - \zeta$$

We recall the following lemma from [BR94] which will be useful in the proofs presented in Chapter 5.

Lemma 2.2.3 (Lemma 2.3 of [BR94]). *Let $t \geq 4$ be an even integer. Suppose $X_1, \ldots, X_n$ are t-wise independent random variables taking values in $[0,1]$. Let $X := \sum_{i=1}^n X_i$ and define $\mu := \mathbf{E}[X]$ be the expectation of the sum. Then, for any $A > 0$, $\Pr[|X - \mu| \geq A] \leq K_t \left(\frac{t\mu + t^2}{A^2}\right)^{t/2}$ where $K_t \leq 8$.*

Lemma 2.2.4 (Lemma A.1 of [BR94]). *Let $t \geq 2$ be an even integer. Suppose $X_1, X_2, \ldots, X_n$ are independent random variables taking values in $[0,1]$. Let $X := \sum_{i=1}^n X_i$ and $\mu := \mathbf{E}[X]$ be the expectation of the sum. Then, $\mathbf{E}[(X-\mu)^t] \leq K_t \left(t\mu + t^2\right)^{t/2}$ where $K_t \leq 8$.*

2.3 Signature Schemes

A signature scheme is a triple of algorithms $\mathcal{SIG} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ such that: (1) KGen takes the security parameter λ as input and outputs a key pair (pk, sk); (2) Sign takes as input a message m and the secret key sk, and outputs a signature σ; (3) Vrfy takes as input a message-signature pair (m, σ) together with the public key pk and outputs a decision bit (indicating whether (m, σ) is a valid signature with respect to pk).

We require that for all messages m and for all keys $(pk, sk) \leftarrow \mathsf{KGen}(1^\lambda)$, algorithm $\mathsf{Vrfy}(pk, m, \mathsf{Sign}(sk, m))$ outputs 1 with all but negligible probability. A signature scheme $\mathcal{SIG}$ is existentially unforgeable against chosen message attacks (EUF-CMA), if for all PPT adversaries A there exists a negligible function $\delta : \mathbb{N} \to [0,1]$ such that $\Pr[\mathsf{A} \text{ wins}] \leq \delta(\lambda)$ in the following game:

1. The challenger samples $(pk, sk) \leftarrow \mathsf{KGen}(1^\lambda)$ and gives pk to A.
2. The adversary is given oracle access to $\mathsf{Sign}(sk, \cdot)$.
3. Eventually A outputs a forgery $(m^\star, \sigma^\star)$ and *wins* if $\mathsf{Vrfy}(pk, (m^\star, \sigma^\star)) = 1$ and $m^\star$ was not asked to the signing oracle before.

Chapter 3

Bounded-Tamper Resilience

In this chapter we propose a new model of memory-only tampering (and leakage) called *bounded leakage and tampering* (BLT) and construct specific public-key schemes like ID-scheme and CCA-encryption. An extended abstract, titled "Bounded Tamper Resilience: How to go beyond the Algebraic Barrier", of the results discussed in this chapter is published in the proceedings of the 19th Annual International Conference on the Theory and Application of Cryptology and Information Security– Asiacrypt 2013 [DFMV13].

3.1 Introduction

Let us start with an illustrative example of memory-tampering attack: consider a digital signature scheme with signing algorithm Sign with public/secret key pair (pk, sk). The tampering adversary obtains pk and can replace sk with $f(sk)$ where f is some adversarial tampering function. Then, the adversary gets access to an oracle $\mathsf{Sign}(f(sk), \cdot)$, i.e., to a signing oracle running with the tampered key $f(sk)$. As usual the adversary wins the game by outputting a valid forgery with respect to the original public key pk. Notice that, f may be the identity function, in which case we get the standard

As mentioned in Chapter 1, Bellare and Kohno [BK03] pioneered the formal security analysis of cryptographic schemes in the presence of related key attacks. In their setting an adversary tampers *continuously* with the key by applying functions f chosen from a set of *admissible* tampering functions $\mathcal{F}$. In the signature example from above, each signing query for message m would be accompanied with a tampering function $f \in \mathcal{F}$ and the adversary obtains $\mathsf{Sign}(f(sk), m)$. Clearly, a result in the RKA setting is stronger if the class of admissible functions $\mathcal{F}$ is larger, and hence several recent works have focused on further broadening $\mathcal{F}$. The prior state of the art (see discussion in Section 1.2.2) mostly considered certain algebraic relations of the key, e.g., $\mathcal{F}$ is the set of all *affine functions* or all polynomials of bounded degree. A natural question that arises from these works is if we can further broaden the class of tampering functions — possibly showing security for *arbitrary* relations. In this work, we study this question and show that under certain assumptions security against arbitrary key relations can be achieved.

Is tamper resilience against arbitrary attacks possible? Unfortunately, the answer to the above question in its most general form is negative. As shown by Gennaro *et al.* [GLM+04], it is *impossible* to protect any cryptographic scheme against arbitrary key relations. In particular, there is an attack that allows to recover the secret key of most stateless cryptographic primitives after only a few number of tampering queries.[1] To prevent this attack the authors proposed to use a *self-destruct* mechanism. That is, before each execution of the cryptographic scheme the key is checked for its validity. In case the key was changed the device self-destructs. In practice, such self-destruct can for instance be implemented by overwriting the secret key with the all-zero string,

[1] The impossibility result of [GLM+04] leaves certain loopholes, which however seem very hard to exploit.

or by switching to a special mode in which the device outputs $\perp$.[2] In this work, we consider an alternative setting to avoid the impossibility results of [GLM+04], and assume that an adversary can only carry out a bounded number of (say t) tampering queries. To explain our setting consider again the example of a digital signature scheme. In our model, we give the adversary access to t tampered signing oracles $\mathsf{Sign}(f_i(sk), \cdot)$, where f_i can be an arbitrary adaptively chosen tampering function. Notice that of course each of these oracles can be queried a polynomial number of times, while t is typically linear in the security parameter.

Is security against bounded tampering useful? Besides from being a natural and non-trivial security notion, we believe that our adversarial model of *arbitrary*, *bounded* tampering is useful for a number of reasons:

1. It is a natural alternative to continuous restricted tampering: our security notion of *bounded*, *arbitrary* tampering is orthogonal to the traditional setting of RKA security where the adversary can tamper *continuously* but is *restricted* to certain classes of attacks. Most previous work in the RKA setting considers algebraic key relations that are tied to the scheme's algebra and may not reflect attacks in practice. For instance, it is not clear that heating up the device or shooting with a laser on the memory can be described by, e.g., an affine function — a class that is usually considered in the literature. We also notice that physical tampering may completely destroy the device, or may be detected by hardware countermeasures, and hence our model of bounded but arbitrary tampering may be sufficient in such settings.

2. It allows to analyze the security of cryptoschemes already used "in the wild": as outlined above a common countermeasure to protect against arbitrary tampering is to implement a key validity check and self-destruct (or output a special failure symbol) in case such check fails. Unfortunately, most cryptographic implementations do not come with such a built-in procedure to check the validity of the key; furthermore, such a self-destruct feature might not always be desirable, for instance in settings where faults are not adversarial, but due to some characteristic of the environment where the device is used (e.g., the temperature). Our notion of bounded tamper resilience allows to make formal security statements about algorithms running directly in implementations without self-destruct, so that neither the construction, nor the implementation needs to be specially engineered.

3. It can be a useful as a building-block: even if the restriction of bounded tamper resilience may be too strong in some settings, it can be useful to achieve results in the stronger continuous tampering setting (we provide some preliminary results on this in Section 3.5).

   Notice that this is similar to the setting of leakage resilient cryptography which also started mainly with "bounded leakage" that later turned out to be very useful to get results in the continuous leakage setting.

We believe that due to the above points the bounded tampering model is an interesting alternative to self-destruct mechanism in order to avoid known impossibility results for arbitrary tampering attacks.

Tampering Model	ID Schemes		IND-CCA PKE
	Σ-Protocols	Okamoto	BHHO
Secret Key	✓	✓	✓
Public Parameters	n.a.	✓	n.a.
Continuous Tampering *i*Floppy	✓	✓	✓
Key Length	$\log \lvert\mathcal{X}\rvert$	$\ell \log p$	$\ell \log p$
Tampering Queries	$\lfloor \log \lvert\mathcal{X}\rvert / \log \lvert\mathcal{Y}\rvert \rfloor - 2$	$\ell - 2$	$\ell - 3$

Table 3.1: An overview of our results for bounded leakage and tamper resilience. All parameters $|\mathcal{X}|$, $|\mathcal{Y}|$ ℓ, p and n are a function of the security parameter λ. For the case of Σ-protocol, the set $\mathcal{X}$ is the set of all possible witnesses and the set $\mathcal{Y}$ is the set of all possible statements for the language; we actually achieve a slightly worse bound depending on the conditional average min-entropy of the witness given the statement (cf. Section 3.3).

3.1.1 Our Contribution

We initiate a general study of schemes resilient to both *bounded* tampering and leakage attacks. We call this model the *bounded leakage and tampering model (BLT)* model. While our general techniques use ideas from the leakage realm, we emphasize that bounded leakage resilience does *not* imply bounded tamper resilience. In fact, it is easy to find contrived schemes that are leakage resilient but completely break for a single tampering query. At a more technical level, we observe that a trivial strategy using leakage to simulate, e.g., faulty signatures, has to fail as the adversary can get any polynomial number of faulty signatures — which clearly cannot be simulated with bounded leakage only. Nevertheless, as we show in this work, we are able to identify certain classes of crypto-schemes for which a small amount of leakage is sufficient to simulate faulty outputs. We discuss this in more detail below.

Our concrete schemes are proven secure under standard assumptions (DL, factoring or DDH) and are efficient and simple. Moreover, we show that our schemes can easily be extended to the continual setting by putting an additional simple assumption on the hardware. We elaborate more on our main contributions in the following paragraphs (see also Table 3.1 for an overview of our results). Importantly, all our results allow arbitrary key tampering.

Identification schemes. It is well known that the Generalized Okamoto identification scheme [Oka92] provides security against bounded leakage from the secret key [ADW09, KV09]. In Section 3.3, we show that additionally it provides strong security against tampering attacks. While in general the tampered view may contain a polynomial number of faulty transcripts that may potentially reveal a large amount of information about the secret key, we can show that fortunately this is not the case for the Generalized Okamaoto scheme. More concretely, our analysis shows that by leaking the public keys corresponding to the modified secret keys allows, for each tampering query, to simulate *any number* of "faulty transcripts" (under the modified keys) by running the *honest-verifier* zero-knowledge simulator. Since the public key is significantly shorter than the secret key, BLT security of the Generalized Okamoto scheme is implied by its leakage resilience.

Our results on the Okamoto identification can be further generalized to a large class of identification schemes (and signature schemes based on the Fiat-Shamir heuristic), namely to all Σ-protocols

[2]We notice that the self-destruct has to be permanent as otherwise the attack of [GLM+04] may still apply.

where the secret key is significantly longer than the public key. In particular, we can instantiate our result with the generalized Guillou-Quisquater ID scheme [GQ88], and its variant based on factoring [FF02], yielding tamper resilient identification based on factoring. We give more details in Section 3.3.

Interestingly, for Okamoto identification security still holds in a stronger model where the adversary is allowed to tamper not only with the secret key of the prover, but also with the description of the public parameters (i.e., the generator g of a group $\mathbb{G}$ of prime order p). The only restrictions are: (i) tampering with the public parameters is independent from tampering with the secret key (which is somewhat similar to popular split-state tampering model, though here one part is public) and (ii) the tampering with public parameters must map to its domain. We also show that the latter restrictions are necessary, by presenting explicit attacks when the adversary can tamper jointly with the secret key and the public parameters or he can tamper the public parameters to some particular range.

Public key encryption. We show how to construct IND-CCA secure public key encryption (PKE) in the BLT model. To this end, we first introduce a weaker CCA-like security notion, where an adversary is given access to a *restricted* (faulty) decryption oracle. Instead of decrypting adversarial chosen ciphertexts such an oracle accepts inputs (m,r), encrypts the message m using randomness r under the original public key, and returns the decryption using the faulty secret key. This notion already provides a basic level of tamper resilience for public key encryption schemes. Consider for instance a setting where the adversary can tamper with the decryption key, but has no control over the ciphertexts that are sent to the decryption oracle, e.g., the ciphertexts are sent over a secure authenticated channel.

Our notion allows the adversary to tamper adaptively with the secret key; intuitively this allows him to learn faulty decryptions of ciphertexts for which he already knows the corresponding plaintext (under the original public key) and the randomness. We show how to instantiate our basic tamper security notion under DDH, by proving that the BHHO cryptosystem [BHHO08] already satisfies it. The proof uses similar ideas as in the proof of the Okamoto identification scheme. In particular our analysis shows that by leaking a *single* group element per tampering query, one can answer *any number* of (restricted) decryption queries; hence restricted IND-CCA BLT security of BHHO follows from its leakage resilience (which was proven in [NS09]).

We then show how to transform the above weaker CCA-like notion to full-fledged CCA security in the BLT model. To this end, we follow the classical paradigm to transform IND-CPA security into IND-CCA security by adding an argument of "plaintext knowledge" π to the ciphertext. Our transformation requires a public *tamper-proof* common reference string similar to earlier work [KKS11]. Intuitively, this works because the argument π enforces the adversary to submit to the faulty decryption oracle only ciphertexts for which he knows the corresponding plaintext (and the randomness used to encrypt it). The pairs (m,r) can then be extracted from the argument π, allowing to simulate arbitrary decryption queries with only access to the restricted decryption oracle.

Updating the key in the *i*Floppy model. As mentioned earlier, if the key is not updated BLT security is the best we can hope for when we consider arbitrary tampering. To go beyond the bound of $|sk|$ tampering queries we may regularly update the secret key with fresh randomness, which renders information that the adversary has learned about earlier keys useless. The effectiveness of key updates in the context of tampering attacks has first been used in the important work of Kalai

et al. [KKS11]. We follow this idea but consider an additional hardware assumption that allows for much simpler and more efficient key updates. More concretely, we propose the *i*Floppy model which is a variant of the floppy model proposed by Alwen *et al.* [ADW09] and subsequently studied in depth by Agrawal *et al.* [ADVW13]. In the floppy model a user of a crypto-device possesses a so-called *floppy* — a secure hardware token — that stores an update key.[3] The floppy is *leakage and tamper proof* and the update key that it holds is solely used to refresh the actual secret key kept on the crypto-device. One may think of the floppy as a particularly secure device that the user keeps at home, while the crypto-device, e.g., a smart-card, runs the actual cryptographic task and is used out in the wild prone to leakage and tampering attacks. We consider a variant called the *i*Floppy model (here "*i*" stands for individual). While in the floppy model of [ADVW13, ADW09] all users can potentially possess an identical hardware token, in the *i*Floppy model we require that each user has an individual floppy storing some secret key related data. We note that from a practical point of view the *i*Floppy model is incomparable to the original floppy model. It may be more cumbersome to produce personalized hardware tokens, but on the other hand, in practice one would not want to distribute hardware tokens that all contain the same global update key as this constitutes a single point of failure.

We show in the *i*Floppy model a simple compiler that "boosts" any ID scheme with BLT security into a scheme with *continuous* leakage and tamper resilience (CLT security). Similarly, we show how to extend IND-CCA BLT security to the CLT setting for the BHHO cryptosystem (borrowing ideas from [ADVW13]). We emphasize that while the *i*Floppy model puts additional requirements on the way users must behave in order to guarantee security, it greatly simplifies cryptographic schemes, and allows us to base security on standard assumptions. Our results in the *i*Floppy model are described in Section 3.5 (Section 3.5.1 for ID schemes, and Section 3.5.2 for PKE schemes).

Comparison with the CLT model of Kalai et al. [KKS11] Kalai *et al.* [KKS11] provide the first feasibility results in the so-called *continuous leakage and tampering* model (CLT). Their constructions achieve strong security requirements where the adversary can arbitrarily tamper continuously with the state. This is achieved by updating the secret key after each usage. While the tampering adversary considered in [KKS11] is clearly stronger (continuous as opposed to bounded tampering), the proposed schemes are "inefficient", and rely on "non-standard" assumptions. Moreover, the approach of key updates requires a stateful device and large amounts of randomness which is costly in practice. The main focus of this work, are simple standard cryptosystems that neither require randomness for key updates nor need to keep state.

3.2 Preliminaries

Below we provide some chapter-specific definitions.

3.2.1 Hard Relations

We start with recalling definitions of language and relation. A *decision problem* related to a language $\mathcal{L} \subseteq \{0,1\}^*$ requires to determine if a given string y is in $\mathcal{L}$ or not. We can associate to any $\mathcal{NP}$-language $\mathcal{L}$ a polynomial-time recognizable relation $\mathfrak{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ defining $\mathcal{L}$ itself,

[3] Notice that "floppy" is just terminology and we use it for consistency with earlier works.

i.e. $\mathcal{L} = \{y : \exists x \text{ s.t. } (y,x) \in \mathfrak{R}\}$ for $|x| \leq poly(|y|)$. The string x is called a *witness* for membership of $y \in \mathcal{L}$.

Let $\mathfrak{R}$ be a relation for some $\mathcal{NP}$-language $\mathcal{L}$. We assume the existence of a probabilistic polynomial time algorithm Setup, called the setup algorithm, which on input 1^λ outputs the description of public parameters pp for the relation $\mathfrak{R}$. Furthermore, we say that the *representation problem* is hard for $\mathfrak{R}$ if for all PPT adversaries A there exists a negligible function $\delta : \mathbb{N} \to [0,1]$ such that

$$\Pr\left[x^\star \neq x; (y,x),(y,x^\star) \in \mathfrak{R} : (y,x,x^\star) \leftarrow \mathsf{A}(pp); pp \leftarrow \mathsf{Setup}(1^\lambda)\right] \leq \delta(\lambda).$$

Representation problem based on discrete log. Let Setup be a group generation algorithm that upon input 1^λ outputs $(\mathbb{G}, g, p)$, where $\mathbb{G}$ is a group of prime order p with generator g. The Discrete Log assumption states that for all PPT adversaries A there exists a negligible function $\delta : \mathbb{N} \to [0,1]$ such that

$$\Pr\left[y = g^x : \ x \leftarrow \mathsf{A}(\mathbb{G}, g, p, y), y \leftarrow \mathbb{G}, (\mathbb{G}, g, p) \leftarrow \mathsf{Setup}(1^\lambda)\right] \leq \delta(\lambda).$$

Let $\ell \in \mathbb{N}$ be a function of the security parameter. Given a vector $\boldsymbol{\alpha} \in \mathbb{Z}_p^\ell$, define $g^{\boldsymbol{\alpha}} = (g_1, \ldots, g_\ell)$ and let $\mathbf{x} = (x_1, \ldots, x_\ell) \leftarrow \mathbb{Z}_p^\ell$. Define $y = \prod_{i=1}^\ell g_i^{x_i}$; the vector $\mathbf{x}$ is called a *representation* of y. We let $\mathfrak{R}_{\mathsf{DL}}$ be the relation corresponding to the representation problem, i.e. $(y, \mathbf{x}) \in \mathfrak{R}_{\mathsf{DL}}$ if and only if $\mathbf{x}$ is a representation of y with respect to $(\mathbb{G}, g, g^{\boldsymbol{\alpha}})$. We say that the ℓ-representation problem is hard in $\mathbb{G}$ if for all PPT adversaries A there exists a negligible function $\delta : \mathbb{N} \to [0,1]$ such that

$$\Pr\left[\mathbf{x}^\star \neq \mathbf{x}; (y,\mathbf{x}),(y,\mathbf{x}^\star) \in \mathfrak{R}_{\mathsf{DL}} : (y,\mathbf{x},\mathbf{x}^\star) \leftarrow \mathsf{A}(\mathbb{G}, g, g^{\boldsymbol{\alpha}}, p); (\mathbb{G}, g, p) \leftarrow \mathsf{Setup}(1^\lambda); \boldsymbol{\alpha} \leftarrow \mathbb{Z}_p^\ell\right] \leq \delta(\lambda).$$

The ℓ-representation problem is equivalent to the Discrete Log problem [ADW09, Lemma 4.1].

Representation problem based on RSA. Let Setup be a group generation algorithm that upon input 1^λ outputs (N, e, d), where $N = p \cdot q$ such that p and q are two primes and also $ed \equiv 1 \bmod ((p-1)(q-1))$. The RSA assumption states that for all PPT adversaries A there exists a negligible function $\delta : \mathbb{N} \to [0,1]$ such that

$$\Pr\left[y = x^e \bmod N : \ x \leftarrow \mathsf{A}(N, e, y), y \leftarrow \mathbb{Z}_N^*, (N, e, d) \leftarrow \mathsf{Setup}(1^\lambda)\right] \leq \delta(\lambda).$$

Let $\ell \in \mathbb{N}$ be a function of the security parameter. Given a vector $\boldsymbol{\alpha} \in \mathbb{Z}_e^\ell$, define $g^{\boldsymbol{\alpha}} = (g_1, \ldots, g_\ell)$ and let $\mathbf{x} = (x_1, \ldots, x_\ell) \leftarrow \mathbb{Z}_e^\ell$ and $\rho \leftarrow \mathbb{Z}_N^*$. Define $y = \prod_{i=1}^\ell g_i^{x_i} \cdot \rho^e \bmod N$; the pair $(\mathbf{x}, \rho)$ is a *representation* of y with respect to $(N, e, g, g^{\boldsymbol{\alpha}})$. We let $\mathfrak{R}_{\mathsf{RSA}}$ be the relation corresponding to the representation problem, i.e. $(y, (\mathbf{x}, \rho)) \in \mathfrak{R}_{\mathsf{RSA}}$ if and only if $(\mathbf{x}, \rho)$ is a representation of y with respect to $(N, e, g, g^{\boldsymbol{\alpha}})$. We say that the ℓ-representation problem is hard in $\mathbb{Z}_N$ if for all PPT adversaries A there exists a negligible function $\delta : \mathbb{N} \to [0,1]$ such that

$$\Pr\left[\begin{array}{c}(\mathbf{x}^\star, \rho^\star) \neq (\mathbf{x}, \rho); (y, (\mathbf{x}, \rho)), (y, (\mathbf{x}^\star, \rho^\star)) \in \mathfrak{R}_{\mathsf{RSA}} \\ : (y, (\mathbf{x}, \rho), (\mathbf{x}^\star, \rho^\star)) \leftarrow \mathsf{A}(N, e, g, g^{\boldsymbol{\alpha}}); (N, e, d) \leftarrow \mathsf{Setup}(1^\lambda); g \leftarrow \mathbb{Z}_N^*; \boldsymbol{\alpha} \leftarrow \mathbb{Z}_e^\ell\end{array}\right] \quad \leq \quad \delta(\lambda).$$

The ℓ-representation problem in $\mathbb{Z}_N$ is equivalent to the RSA problem (see [Oka92, FF02]).

Decisional Diffie Hellman. Let Setup be a group generation algorithm that upon input 1^λ outputs $(\mathbb{G}, g, p)$, where $\mathbb{G}$ is a group of prime order p with generator g. The Decisional Diffie Hellman (DDH) assumption states that for all PPT adversaries A there exists a negligible function $\delta : \mathbb{N} \to [0,1]$ such that

$$\Big|\Pr\left[\mathsf{A}(g, g^x, g^y, g^{xy}) = 1 : \; x, y \leftarrow \mathbb{Z}_p, (\mathbb{G}, g, p) \leftarrow \mathsf{Setup}(1^\lambda)\right] - \Pr\left[\mathsf{A}(g, g^x, g^y, g^z) = 1 : \; x, y, z \leftarrow \mathbb{Z}_p, (\mathbb{G}, g, p) \leftarrow \mathsf{Setup}(1^\lambda)\right]\Big| \leq \delta(\lambda).$$

3.2.2 Σ-protocols

Σ-protocols [Cra96] are a special class of interactive proof systems for membership in a language $\mathcal{L}$, where a prover $\mathsf{P} = (\mathsf{P}_0, \mathsf{P}_1)$ wants to convince a verifier $\mathsf{V} = (\mathsf{V}_0, \mathsf{V}_1)$ (both modelled as PPT algorithms) that it possesses a witness to the fact that a given element y is in some language $\mathcal{L}$. Denote with x the witness corresponding to y, and let pp be public parameters. The protocol proceeds as follows: (1) The prover computes $a \leftarrow \mathsf{P}_0(pp)$ and sends it to the verifier; (2) The verifier chooses challenge $c \leftarrow \mathsf{V}_0(pp, y)$[4], uniformly at random from some challenge space $\mathcal{D}_{\mathsf{ch}}$, and sends it to the prover; (3) The prover answers with $z \leftarrow \mathsf{P}_1(pp, (a, c, x))$; (4) The verifier outputs a result $\mathsf{V}_1(pp, y, (a, c, z)) \in \{\texttt{accept}, \texttt{reject}\}$. We call this a *public-coin* three round interactive proof system. A formal definition of Σ-protocols can be found below.

Definition 3.2.1 (Σ-protocol)**.** *A Σ-protocol* (P, V) *for a relation* $\mathfrak{R}$ *is a three round public-coin interactive proof system with the following properties.*

Completeness. *Whenever* P *and* V *follow the protocol on common input* y*, public parameters* pp *and private input* x *to* P *such that* $(y, x) \in \mathfrak{R}$*, the verifier* V *accepts with all but negligible probability.*

Special soundness. *From any pair of accepting conversations on public input* y*, namely* (a, c, z)*,* (a, c', z') *such that* $c \neq c'$*, one can efficiently compute* x *such that* $(y, x) \in \mathfrak{R}$*.*

Perfect Honest Verifier Zero Knowledge (HVZK). *There exists a PPT simulator* M*, which on input* y *and a random* c *outputs an accepting conversation of the form* (a, c, z)*, with exactly the same probability distribution as conversations between the honest* P*,* V *on input* y*.*

Note that Definition 3.2.1 requires *perfect* HVZK, whereas in general one could ask for a weaker requirement, namely that the HVZK property holds only computationally.

3.2.3 True Simulation Extractability

We recall the notion of true-simulation extractable (tSE) NIZKs [DHLAW10b]. This notion is similar to the notion of simulation-sound extractable NIZKs [Gro06], with the difference that the adversary has oracle access to simulated proofs only of true statements (and not of arbitrary ones).

Let $\mathfrak{R}$ be an NP relation on pairs (y, x) with corresponding language $\mathcal{L} = \{y : \exists\, x \text{ s.t. } (y, x) \in \mathfrak{R}\}$. A tSE NIZK proof system for $\mathfrak{R}$ is a triple of algorithm $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ such that: (1) Algorithm Gen takes as input 1^λ and generates a common reference string (CRS) Ω, a trapdoor tk and an extraction key ek; (2) Algorithm Prove^Ω takes as input a pair (y, x) and produces an argument

[4]Note that here we abuse notation as we use c to denote challenge, not codeword like other chapters

π which proves that $(y,x) \in \mathfrak{R}$; (3) Algorithm Verify^{Ω} takes as input a pair (y,π) and checks the correctness of the argument π with respect to the public input y. Moreover, the following properties are satisfied:

Completeness. For all pairs $(y,x) \in \mathfrak{R}$, if $(\Omega, tk, ek) \leftarrow \mathsf{Gen}(1^\lambda)$ and $\pi \leftarrow \mathsf{Prove}^{\Omega}(y,x)$ then we have $\mathsf{Verify}^{\Omega}(y,\pi) = 1$.

Composable non-interactive zero knowledge. There exists a PPT simulator S such that, for any PPT adversary A, there exists a negligible function $\delta : \mathbb{N} \to [0,1]$ such that $|\Pr[\mathsf{A} \text{ wins}] - \frac{1}{2}| \leq \delta(\lambda)$ in the following game:

1. The challenger samples $(\Omega, tk, ek) \leftarrow \mathsf{Gen}(1^\lambda)$ and gives Ω to A.
2. A chooses $(y,x) \in \mathfrak{R}$ and gives these to the challenger.
3. The challenger samples $\pi_0 \leftarrow \mathsf{Prove}^{\Omega}(y,x)$, $\pi_1 \leftarrow \mathsf{S}(y, tk)$, $b \in \{0,1\}$ and gives π_b to A.
4. A outputs a bit b' and wins iff $b' = b$.

Strong true simulation extractability. Define a simulation oracle $\mathsf{S}'_{tk}(\cdot,\cdot)$ that takes as input a pair (y,x), checks if $(y,x) \in \mathfrak{R}$ and then it either outputs a simulated argument $\pi \leftarrow \mathsf{S}(y, tk)$ (ignoring x) in case the check succeeds or it outputs $\perp$ otherwise. There exists a PPT algorithm $\mathsf{Ext}(y, \pi, ek)$ such that, for all PPT adversaries A, there exists a negligible function $\delta : \mathbb{N} \to [0,1]$ such that $\Pr[\mathsf{A} \text{ wins}] \leq \delta(\lambda)$ in the following game:

1. The challenger samples $(\Omega, tk, ek) \leftarrow \mathsf{Gen}(1^\lambda)$ and gives Ω to A.
2. $\mathsf{A}^{\mathsf{S}'_{tk}(\cdot)}$ can adaptively access the simulation oracle $\mathsf{S}'_{tk}(\cdot,\cdot)$.
3. Eventually A outputs a pair $(y^\star, \pi^\star)$.
4. The challenger runs $x^\star \leftarrow \mathsf{Ext}(y^\star, \pi^\star, ek)$.
5. A wins if: (a) $(y^\star, \pi^\star) \neq (y,\pi)$ for all pairs (y,π) returned by the simulation oracle; (b) $\mathsf{Verify}^{\Omega}(y^\star, \pi^\star) = 1$; (c) $(y^\star, x^\star) \notin \mathfrak{R}$.

In case A is given only one query to $\mathsf{S}'_{tk}(\cdot)$, we speak of *one-time* strong tSE.

3.3 ID Schemes with BLT Security

In an identification scheme a prover tries to convince a verifier of its identity (corresponding to its public key pk). Formally, an identification scheme is a tuple of algorithms $\mathcal{ID} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{P}, \mathsf{V})$ defined as follows:

$pp \leftarrow \mathsf{Setup}(1^\lambda)$: Algorithm Setup takes the security parameter as input and outputs public parameters pp. The set of all public parameters is denoted by $\mathcal{PP}$.

$(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$: Algorithm Gen outputs the public key and the secret key corresponding to the prover's identity. The set of all possible secret keys is denoted by $\mathcal{SK}$.

(P, V): We let $(\mathsf{P}(pp, sk) \leftrightarrows \mathsf{V}(pp, pk))$ denote the interaction between prover P (holding sk) and verifier V (holding pk) on common input pp. Such interaction outputs a result in $\{\texttt{accept}, \texttt{reject}\}$, where `accept` means P's identity is considered as valid.

Definition 3.3.1. *Let $\zeta = \zeta(\lambda)$ and $t = t(\lambda)$ be parameters, and let $\mathcal{F}$ be some set of functions such that $f \in \mathcal{F}$ has a type $f : \mathcal{SK} \times \mathcal{PP} \to \mathcal{SK} \times \mathcal{PP}$. We say that $\mathcal{ID}$ is bounded ζ-leakage and t-tamper secure (in short (ζ,t)-BLT secure) against impersonation attacks with respect to $\mathcal{F}$ if the following properties are satisfied.*

(i) Correctness. *For all $pp \leftarrow \mathsf{Setup}(1^\lambda)$ and $(pk,sk) \leftarrow \mathsf{Gen}(1^\lambda)$ we have that $(\mathsf{P}(pp,sk) \leftrightarrows \mathsf{V}(pp,pk))$ outputs* `accept`.

(ii) Security. *For all PPT adversaries A, there exists a negligible function $\delta : \mathbb{N} \to [0,1]$, such that $\Pr[\mathsf{A}$ wins$] \leq \delta(\lambda)$ in the following game:*

1. *The challenger runs $pp \leftarrow \mathsf{Setup}(1^\lambda)$ and $(pk,sk) \leftarrow \mathsf{Gen}(1^\lambda)$, and gives (pp,pk) to A.*
2. *The adversary is given oracle access to $\mathsf{P}(pp,sk)$ that outputs polynomially many proof transcripts with respect to secret key sk.*
3. *The adversary may adaptively ask t tampering queries. During the i-th query, A chooses a function $f_i \in \mathcal{F}$ and gets oracle access to $\mathsf{P}(\tilde{pp}_i, \tilde{sk}_i)$, where $(\tilde{sk}_i, \tilde{pp}_i) = f_i(sk,pp)$. The adversary can interact with the oracle $\mathsf{P}(\tilde{pp}_i, \tilde{sk}_i)$ a polynomially number of times, where the prover uses the tampered secret key $\tilde{sk}_i$ and the public parameter $\tilde{pp}_i$.*
4. *The adversary may adaptively ask leakage queries. In the j-th query, A chooses a function $L_j : \{0,1\}^* \to \{0,1\}^{\zeta_j}$ and receives back the output of the function applied to sk.*
5. *The adversary loses access to all other oracles and interacts with an honest verifier V (holding pk). We say that A* wins *if $(\mathsf{A}(pp,pk) \leftrightarrows \mathsf{V}(pp,pk))$ outputs* `accept` *and $\sum_j \zeta_j \leq \zeta$.*

Notice that in the above definition the leakage is from the original secret key sk. This is without loss of generality as our tampering functions are modeled as deterministic circuits.

In a slightly more general setting, one could allow A to leak on the original secret key also in the last phase where it has to convince the verifier. In the terminology of [ADW09] this is reminiscent of so-called *anytime leakage* attacks. Our results can be generalized to this setting, however we stick to Definition 3.3.1 for simplicity.

The rest of this section is organized as follows. In Section 3.3.1 we prove that a large class of Σ-protocols are secure in the BLT model, where the tampering function is allowed to modify the secret state of the prover but not the public parameters. In Section 3.3.2 we look at a concrete instantiation based on the Okamoto ID scheme, and prove that this construction is secure in a stronger model where the tampering function can modify both the secret state of the prover and the public parameters (but independently). Finally, in Section 3.3.3 we illustrate that the latter assumption is necessary, as otherwise the Okamoto ID scheme can be broken by (albeit contrived) attacks.

3.3.1 Σ-protocols are Tamper Resilient

It is well known that Σ-protocols (see Section 3.2.2) are a natural tool to design ID schemes. The construction is depicted in Figure 3.1. We restrict our analysis to Σ-protocols for so-called *complete* relations $\mathfrak{R}$ such that for each possible witness $x \in \mathcal{X}$, there is always a corresponding statement $y \in \mathcal{Y}$ such that $(y,x) \in \mathfrak{R}$. As discussed later, the relations considered to instantiate our result satisfy this property.

Consider now the class of tampering functions $\mathcal{F}_{\mathsf{sk}} \subset \mathcal{F}$ such that $f \in \mathcal{F}_{\mathsf{sk}}$ has the following form: $f = (f^{sk}, ID^{pp})$ where $f^{sk} : \mathcal{SK} \to \mathcal{SK}$ is an arbitrary polynomial time computable function and $ID^{pp} : \mathcal{PP} \to \mathcal{PP}$ is the identity function. This models tampering with the secret state of P without changing the public parameters (these must be hard-wired into the prover's code). The proof of the following theorem uses ideas of [ADW09], but is carefully adjusted to incorporate tampering attacks.

ID Scheme from Σ-Protocol

Let $((\mathsf{P}_0, \mathsf{P}_1), (\mathsf{V}_0, \mathsf{V}_1))$ be a Σ-protocol for a relation $\mathfrak{R}$.

$\mathsf{Setup}(1^\lambda)$**:** Sample public parameters $pp \leftarrow \mathcal{PP}$ for the underlying relation $\mathfrak{R}$.

$\mathsf{Gen}(1^\lambda)$**:** Output $(y, x) \in \mathfrak{R}$, where $sk := x \in \mathcal{X}$ and $pk := y \in \mathcal{Y}$ and $|x|$ is polynomially bounded by $|y|$.

$(\mathsf{P}(pp, sk) \leftrightarrows \mathsf{V}(pp, pk))$**:** Parse $sk := x$, and $pk := y$; the protocol works as follows.

1. The prover sends $a \leftarrow \mathsf{P}_0(pp)$ to the verifier.
2. The verifier chooses a random challenge $c \leftarrow \mathsf{V}_0(pp, y)$, with $c \in \mathcal{D}_{\mathsf{ch}}$, and sends it to the prover.
3. The prover computes the answer $z \leftarrow \mathsf{P}_1(pp, (a, c, x))$.
4. The verifier accepts iff $\mathsf{V}_1(pp, y, (a, c, z))$ outputs `accept`.

Figure 3.1: ID scheme based on Σ-protocol for relation $\mathfrak{R}$

Theorem 3.3.2. *Let $\lambda \in \mathbb{N}$ be the security parameter and let (P, V) be a Σ-protocol, for a* complete *relation $\mathfrak{R}$, with challenge space $\mathcal{D}_{\mathsf{ch}}$ of size $O(\lambda^{\log \lambda})$, such that the representation problem is hard for $\mathfrak{R}$ (cf. Section 3.2.1). Assume that conditioned on the distribution of the public input $y \in \mathcal{Y}$, the witness $x \in \mathcal{X}$ has average min entropy at least β, i.e., $\tilde{\mathbf{H}}_\infty(X|Y) \geq \beta$. Then, the ID scheme of Figure 3.1 is $(\zeta(\lambda), t(\lambda))$-BLT secure against impersonation attacks with respect to $\mathcal{F}_{\mathsf{sk}}$, where*

$$\zeta \leq \beta - t \log |\mathcal{Y}| - \lambda \qquad \text{and} \qquad t \leq \left\lfloor \frac{\beta}{\log |\mathcal{Y}|} \right\rfloor - 1.$$

Proof. Assume that there exists a polynomial $\mathsf{poly}(\cdot)$ and an adversary A that succeeds in the BLT experiment (cf. Definition 3.3.1) with probability at least $\delta(\lambda) := 1/\mathsf{poly}(\lambda)$, for infinitely many $\lambda \in \mathbb{N}$. Then, we construct an adversary B (using A as a subroutine) such that:

$$\Pr\left[x^\star \neq x; (y, x), (y, x^\star) \in \mathfrak{R} : \ (y, x, x^\star) \leftarrow \mathsf{B}(pp); pp \leftarrow \mathsf{Setup}(1^\lambda)\right] \geq \delta^2 - |\mathcal{D}_{\mathsf{ch}}|^{-1} - 2^{-\lambda}.$$

Since $|\mathcal{D}_{\mathsf{ch}}|$ is super-polynomial in λ, this contradicts the assumption that the representation problem is hard for $\mathfrak{R}$ (cf. Section 3.2.1).

Adversary B works as follows. It first samples $(y, x) \leftarrow \mathsf{Gen}(1^\lambda)$, then it uses these values to simulate the entire experiment for A. This includes answers to the leakage queries, and access to the oracles $\mathsf{P}(pp, x)$, $\mathsf{P}(pp, \tilde{x}_i)$ where $\tilde{x}_i = f_i(x) = f_i(sk) = \tilde{sk}_i$ for all $i \in [t]$. During the impersonation stage, B chooses a random challenge $c \in \mathcal{D}_{\mathsf{ch}}$ which results in a transcript (a, c, z). At this point B rewinds A to the point after it chose a, and selects a different challenge $c' \in \mathcal{D}_{\mathsf{ch}}$ resulting in a transcript (a, c', z'). Whenever the two transcripts are accepting and $c' \neq c$, the special soundness property ensures that adversary B has extracted successfully some value $x^\star$ such that $(y, x^\star) \in \mathfrak{R}$.

Let us call the event described above E_1, and the event $x = x^\star$ is denoted by E_2. Clearly,

$$\begin{aligned}\Pr[\mathsf{B}\text{ succeeds}] &= \Pr\left[x^\star \neq x; (y,x),(y,x^\star) \in \mathfrak{R} : \ (y,x,x^\star) \leftarrow \mathsf{B}(pp); pp \leftarrow \mathsf{Setup}(1^\lambda)\right] \\ &= \Pr[E_1 \wedge \neg E_2]\,.\end{aligned} \tag{3.1}$$

Claim 3.3.3. *The probability of event E_1 is $\Pr[E_1] \geq \delta^2 - |\mathcal{D}_{\mathsf{ch}}|^{-1}$.*

Proof. The proof is identical to the proof of [ADW09, Claim 4.1]. We repeat it here for completeness.

Denote with V the random variable corresponding to the view of A in one execution of the BLT game up to the challenge phase; this includes the public values (pp, y), the coins of A, the leakage, and the transcripts obtained via the oracles $\mathsf{P}(pp,x)$, $\mathsf{P}(pp,\tilde{x}_1),\ldots,\mathsf{P}(pp,\tilde{x}_t)$. Notice that B is essentially playing the role of the challenger for A (as it knows a correctly distributed pair (y,x)), but at the end of the execution it rewinds A after it already sent the value a in the challenge phase, and samples a new challenge $c' \leftarrow \mathcal{D}_{\mathsf{ch}}$ hoping that $c' \neq c$ (where $c \leftarrow \mathcal{D}_{\mathsf{ch}}$ is the challenge sampled in the first run of A). Hence, the probability space of the event E_1 includes the randomness of the BLT game, the coins of A, and the randomness used to sample $c' \in \mathcal{D}_{\mathsf{ch}}$.

Let W be an indicator random variable, such that $W = 1$ when A wins in one execution of the BLT game (and $W = 0$ otherwise). By definition, $\mathbf{E}[W] := \delta$. Notice that $\Pr[E_1|V=v] \geq \Pr[W^2 = 1|V=v] - |\mathcal{D}_{\mathsf{ch}}|^{-1}$, since the probability that $c = c'$ is at most $|\mathcal{D}_{\mathsf{ch}}|^{-1}$ and this probability is independent of the fact that the two conversations (a,c,z) and (a,c',z') are accepting. Therefore,

$$\begin{aligned}\Pr[E_1] = \sum_v \Pr[E_1|V=v]\Pr[V=v] &\geq \sum_v \Pr\left[W^2=1|V=v\right]\Pr[V=v] - |\mathcal{D}_{\mathsf{ch}}|^{-1} \\ &= \mathbf{E}[W^2] - |\mathcal{D}_{\mathsf{ch}}|^{-1} \geq (\mathbf{E}[W])^2 - |\mathcal{D}_{\mathsf{ch}}|^{-1} = \delta^2 - |\mathcal{D}_{\mathsf{ch}}|^{-1}\end{aligned} \tag{3.2}$$

where the first inequality follows by Jensen's inequality. This concludes the proof of the claim. □

Claim 3.3.4. *The probability of event E_2 is $\Pr[E_2] \leq 2^{-\lambda}$.*

Proof. We prove the claim holds even in case the adversary is unbounded. Consider an experiment $\mathcal{E}_0$ which is similar to the experiment of Definition 3.3.1, except that now the adversary does not get access to the leakage oracle. Consider an adversary A trying to predict the value of x given the view in a run of $\mathcal{E}_0$; such view contains polynomially many transcripts (for the initial secret key and for each of the tampered secret keys) together with the original public input y and the public parameters pp (which are tamper-free), i.e., $\mathsf{view}_{\mathsf{A}}^{\mathcal{E}_0} = \{\boldsymbol{\Psi}, \boldsymbol{\Psi}_1, \ldots, \boldsymbol{\Psi}_t\} \cup \{y, pp\}$. The vector $\boldsymbol{\Psi}$ and each of the vectors $\boldsymbol{\Psi}_i$ contains polynomially many transcripts of the form (a,c,z), corresponding (respectively) to the original key and to the i-th tampered secret key.

We now move to experiment $\mathcal{E}_1$, which is the same as $\mathcal{E}_0$ with the modification that we add to A's view, for each tampering query, the public value $\tilde{y}_i \in \mathcal{Y}$ corresponding to the tampered witness $\tilde{x}_i = f_i(x) \in \mathcal{X}$; note that such value always exists, by our assumption that the relation $\mathfrak{R}$ is complete. Hence, $\mathsf{view}_{\mathsf{A}}^{\mathcal{E}_1} = \mathsf{view}_{\mathsf{A}}^{\mathcal{E}_0} \cup \{(\tilde{y}_1, \ldots, \tilde{y}_t)\}$. Clearly,

$$\widetilde{\mathbf{H}}_\infty(X|\mathcal{E}_0) \geq \widetilde{\mathbf{H}}_\infty(X|\mathcal{E}_1). \tag{3.3}$$

Next, we consider experiment $\mathcal{E}_2$ where A is given only the values $(\tilde{y}_1, \ldots, \tilde{y}_t)$, i.e., $\mathsf{view}_{\mathsf{A}}^{\mathcal{E}_2} = \{\tilde{y}_1 \ldots, \tilde{y}_t\} \cup \{y, pp\}$. We claim that conditioning on $\mathcal{E}_1$ or on $\mathcal{E}_2$ has the same effect on the min-entropy of X. This is because the values $\{\boldsymbol{\Psi}, \boldsymbol{\Psi}_i\}_{i\in[t]}$ can be computed as a deterministic function

of $(y, \tilde{y}_1, \ldots, \tilde{y}_t)$ as follows: For a randomly chosen challenge c run the HVZK simulator M upon input $(pp, \tilde{y}_i, c)$ and append the output (a, c, z) to $\mathbf{\Psi}_i$[5]. (The same can be done to simulate $\mathbf{\Psi}$, by running $\mathsf{M}(pp, y, c)$.) It follows from perfect HVZK that this generates an identical distribution to that of experiment $\mathcal{E}_1$ and thus

$$\widetilde{\mathbf{H}}_\infty(X|\mathcal{E}_1) = \widetilde{\mathbf{H}}_\infty(X|\mathcal{E}_2). \tag{3.4}$$

Since the public parameters are tamper-free and are chosen independently of X, we can remove them from the view and write

$$\widetilde{\mathbf{H}}_\infty(X|\mathcal{E}_2) = \widetilde{\mathbf{H}}_\infty(X|\tilde{Y}_1, \ldots, \tilde{Y}_t, Y) \geq \widetilde{\mathbf{H}}_\infty(X|Y) - |(\tilde{Y}_1, \ldots, \tilde{Y}_t)| \geq \beta - t \log |\mathcal{Y}|, \tag{3.5}$$

where we used Lemma 2.2.2 together with the fact that the joint distribution $(\tilde{Y}_1, \ldots, \tilde{Y}_t)$ can take at most $(|\mathcal{Y}|)^t$ values, and our assumption on the conditional min-entropy of X given Y.

Consider now the full experiment described in Definition 3.3.1 and call it $\mathcal{E}_3$. Note that this experiment is similar to the experiment $\mathcal{E}_0$, with the only addition that here A has also access to the leakage oracle. Hence, we have $\mathsf{view}_{\mathsf{A}}^{\mathcal{E}_3} = \mathsf{view}_{\mathsf{A}}^{\mathcal{E}_0} \cup \mathsf{view}_{\mathsf{A}}^{\mathsf{leak}}$. Denote with $\Lambda \in \{0,1\}^\zeta$ the random variable corresponding to $\mathsf{view}_{\mathsf{A}}^{\mathsf{leak}}$. Using Lemma 2.2.2 and combining Eq. 3.3–3.5 we get

$$\widetilde{\mathbf{H}}_\infty(X|\mathcal{E}_3) = \widetilde{\mathbf{H}}_\infty(X|\mathcal{E}_0, \Lambda) \geq \widetilde{\mathbf{H}}_\infty(X|\mathcal{E}_0) - \zeta \geq \beta - t \log |\mathcal{Y}| - \zeta \geq \lambda,$$

where the last inequality comes from the value of ζ in the theorem statement. We can thus bound the probability of E_2 as $\Pr[E_2] \leq 2^{-\widetilde{\mathbf{H}}_\infty(X|\mathcal{E}_3)} \leq 2^{-\lambda}$. The claim follows. □

Combining Claim 3.3.3 and Claim 3.3.4 together with Eq. 3.1 yields

$$\Pr[\mathsf{B} \text{ succeeds}] = \Pr[E_1 \wedge \neg E_2] \geq \Pr[E_1] - \Pr[E_2] \geq \delta^2 - |\mathcal{D}_{\mathsf{ch}}|^{-1} - 2^{-\lambda},$$

which contradicts our assumption on the hardness of the representation problem for $\mathfrak{R}$. This finishes the proof. □

Instantiations. Below, we discuss a number of concrete instantiations for Theorem 3.3.2 based on standard hardness assumptions:

- *Generalized Okamoto.* This instantiation is described in detail in Section 3.3.2, where we additionally show that the generalized Okamoto ID scheme [Oka92] remains secure also in case the public parameters (and not only the secret key) are subject to tampering.

- *Generalized Guillou-Quisquater.* Consider the relation $\mathfrak{R}_{\mathsf{RSA}}$ of Section 3.2.1. The relation is easily seen to be complete. Hardness of the ℓ-representation problem for $\mathfrak{R}_{\mathsf{RSA}}$ follows from the RSA assumption, and was shown in [Oka92]. A suitable Σ-protocol is described in [GQ88]. A variant based on factoring can be obtained following Fischlin and Fischlin [FF02].

[5]Notice that, we assume the adversary to act like a standard passive adversary while interacting with the prover in the first phase of the security game, except it can tamper with the witness of the prover and thereby can obtain transcripts corresponding to the tampered witness. Therefore, it is possible for the adversary to run the HVZK simulator.

3.3.2 Concrete Instantiation with more Tampering

We extend the power of the adversary by allowing him to tamper not only with the witness, but also with the public parameters (used by the prover to generate the transcripts). However the tampering has to be independent on the two components. This is reminiscent of the popular split-state model (considered for instance in [LL12]), with the key difference that in our case the secret state does not need to be split into two parts.

We model this through the following class of tampering functions $\mathcal{F}_{\mathsf{pub-split}}$[6]: We say that $f \in \mathcal{F}_{\mathsf{pub-split}}$ if we can write $f = (f^{sk}, f^{pp})$ where $f^{sk} : \mathcal{SK} \to \mathcal{SK}$ and $f^{pp} : \mathcal{PP} \to \mathcal{PP}$ are arbitrary polynomial time computable functions. Recall that the input/output domains of f^{sk}, f^{pp} are identical, hence the size of the witness and the public parameters cannot be changed. As we show in the next section, this restriction is necessary. Note also that $\mathcal{F}_{\mathsf{sk}} \subseteq \mathcal{F}_{\mathsf{pub-split}} \subseteq \mathcal{F}$.

Generalized Okamoto. Consider the generalized version of the Okamoto ID scheme [Oka92], depicted in Figure 3.2. The underlying hard relation here is the relation $\mathfrak{R}_{\mathsf{DL}}$ and the representation problem for $\mathfrak{R}_{\mathsf{DL}}$ is the ℓ-representation problem in a group $\mathbb{G}$ (cf. Section 3.2.1). As proven in [ADW09], this problem is equivalent to the Discrete Log problem in $\mathbb{G}$.

We first argue that the protocol is BLT-secure against impersonation attacks with respect to $\mathcal{F}_{\mathsf{sk}}$. This follows immediately from Theorem 3.3.2 as the relation $\mathfrak{R}_{\mathsf{DL}}$ is complete, and the protocol of Figure 3.2 is a Σ-protocol which satisfies perfect HVZK; moreover $|\mathcal{Y}| = |\mathcal{D}_{\mathsf{ch}}| = p$ and the size of prime p is super-polynomial in λ to ensure hardness of the Discrete Log problem. Observing that the secret key $\mathbf{x}$, conditioned on the public key y, is uniform in a subspace of dimension $\ell - 1$, i.e., $\mathbf{H}_\infty(X|Y) \geq (\ell-1)\log p = \beta$, we obtain parameters $\zeta \leq (\ell - 1 - t)\log(p) - \lambda$ and $t \leq \ell - 2$.

Generalized Okamoto ID Scheme

Let $\ell = \ell(\lambda)$ be some function of the security parameter. Consider the following identification scheme.

Setup: Choose a group $\mathbb{G}$ of prime order p with generator g and a vector $\boldsymbol{\alpha} \leftarrow \mathbb{Z}_p^\ell$, and output $pp = (\mathbb{G}, g, g^{\boldsymbol{\alpha}}, p)$ where $g^{\boldsymbol{\alpha}} = (g_1, \ldots, g_\ell)$.

$\mathsf{Gen}(1^\lambda)$: Select a vector $\mathbf{x} := (x_1, \ldots, x_\ell) \leftarrow \mathbb{Z}_p^\ell$ and set $pk := y = \prod_{i=1}^\ell g_i^{x_i}$ and $sk := \mathbf{x}$.

$(\mathsf{P}(pp, sk) \leftrightarrows \mathsf{V}(pp, pk))$: Parse $pp = (\mathbb{G}, g, g^{\boldsymbol{\alpha}}, p)$, $sk := (x_1, \ldots, x_\ell)$, and $pk := y$; the protocol works as follows.

1. The prover chooses a random vector $\mathbf{r} := (r_1, \ldots, r_\ell) \leftarrow \mathbb{Z}_p^\ell$ and sends $a = \prod_{i=1}^\ell g_i^{r_i}$.
2. The verifier chooses a random challenge $c \leftarrow \mathbb{Z}_p$ and sends it to the prover.
3. The prover computes the answer $\mathbf{z} = (r_1 + cx_1, \ldots, r_\ell + cx_\ell)$.
4. The verifier accepts if and only if $\prod_{i=1}^\ell g_i^{z_i} = a \cdot y^c$.

Figure 3.2: Generalized Okamoto Identification Scheme

Next, we show that the generalized Okamoto ID scheme is actually secure for $\mathcal{F}_{\mathsf{pub-split}}$ (with the same parameters).

Theorem 3.3.5. *Let $\lambda \in \mathbb{N}$ be the security parameter and assume the Discrete Log problem is hard in $\mathbb{G}$. Then, the generalized Okamoto ID scheme is $(\zeta(\lambda), t(\lambda))$-BLT secure against impersonation*

[6] Since in this model one part is public we denote this class with a different notation, $\mathcal{F}_{\mathsf{pub-split}}$ than the standard $\mathcal{F}_{\mathsf{split}}$ which is used in the later chapters. It is easy to see that the class $\mathcal{F}_{\mathsf{pub-split}}$ is actually stronger than $\mathcal{F}_{\mathsf{split}}$.

attacks with respect to $\mathcal{F}_{\mathsf{pub-split}}$*, where*

$$\zeta \leq (\ell - 1 - t)\log(p) - \lambda \qquad \textit{and} \qquad t \leq \ell - 2.$$

Proof (Sketch). The proof follows closely the proof of Theorem 3.3.2, with the key difference that we also have to take care of tampering with respect to $pp = (\mathbb{G}, g, g^{\boldsymbol{\alpha}}, p)$. We sketch how this can be done below.

Given an adversary A winning the BLT security experiment with non-negligible advantage $\delta(\lambda)$, consider the same reduction B outlined in the proof of Theorem 3.3.2, attacking the ℓ-representation problem in $\mathbb{G}$. Notice that the reduction can, as before, perfectly simulate the environment for A as it knows honestly generated parameters (pp, pk, sk). In particular, Claim 3.3.3 still holds here with $|\mathcal{D}_{\mathsf{ch}}| = p$.

It remains to prove Claim 3.3.4. To do so, we modify the view of the adversary in the proof of Theorem 3.3.2 such that it contains also the tampered public parameters $\tilde{pp}_i$ for all $i \in [t]$. In particular, the elements $(a, c, \mathbf{z})$ contained in each vector $\boldsymbol{\Psi}_i$ in the view of experiment $\mathcal{E}_0$ are now sampled from $\mathsf{P}(\tilde{pp}_i, \tilde{\mathbf{x}}_i)$, where $\tilde{\mathbf{x}}_i = f_i^{sk}(\mathbf{x})$ and $\tilde{pp}_i = f_i^{pp}(pp)$ for all $i \in [t]$. We then modify $\mathcal{E}_1$ and $\mathcal{E}_2$ by additionally including the values of the tampered public parameters $\{\tilde{pp}_i\}_{i\in[t]}$.

We claim that $\widetilde{\mathbf{H}}_\infty(X|\mathcal{E}_1) = \widetilde{\mathbf{H}}_\infty(X|\mathcal{E}_2)$, in particular the view of A in $\mathcal{E}_1$ can be simulated given only $\{\tilde{pk}_i, \tilde{pp}_i\}_{i\in[t]}$. This follows from the fact that the generalized Okamoto ID scheme maintains the completeness and perfect HVZK properties even when the transcripts are computed using tampered public parameters $\tilde{pp} = (\tilde{\mathbb{G}}, \tilde{g}, \tilde{g}_1, \ldots, \tilde{g}_\ell, \tilde{p})$. (Whereas of course in this case the protocol is not sound.) The HVZK simulator $\mathsf{M}(\tilde{pp}, \tilde{y}, c)$ works as follows: Choose $z_1, \ldots, z_\ell$ at random in $\mathbb{Z}_{\tilde{p}}$ and if $\tilde{y} \neq 0 \mod \tilde{p}$, then compute $a = (\prod_{i=1}^{\ell} \tilde{g}_i^{z_i})/\tilde{y}^c \bmod \tilde{p}$. In case $\tilde{y} = 0 \mod \tilde{p}$, then just set $a = 0$.[7] For any $(\tilde{\mathbf{x}}, \tilde{pp}) = (f^{sk}(\mathbf{x}), f^{pp}(pp))$, the distributions $\mathsf{M}(\tilde{pp}, \tilde{y}, c)$ and $(\mathsf{P}(\tilde{pp}, \tilde{\mathbf{x}}) \leftrightarrows \mathsf{V}(\tilde{pp}, \tilde{y}))$ are both uniformly random over all values $(a, c, \mathbf{z} = (z_1, \ldots, z_\ell))$ such that $\prod_{i=1}^{\ell} \tilde{g}_i^{z_i} = a\tilde{y}^c \bmod \tilde{p}$.

Therefore the simulation perfectly matches the honest conversation. This proves Eq. 3.4. Now Eq. 3.5 follows from the fact that the tampering functions f^{pp} cannot depend on sk. The rest of the proof remains the same. □

3.3.3 Some Attacks

We show that for the Okamoto scheme it is hard to hope for BLT security beyond the class of tampering functions $\mathcal{F}_{\mathsf{pub-split}}$. We illustrate this by concrete attacks which work in case one tries to extend the power of the adversary in two different ways: (1) Allowing A to tamper jointly with the witness and the public parameters; (2) Allowing A to tamper independently with the witness and with the public parameters, but to increase their size.

Tampering jointly with the public parameters. Consider the class of functions $\mathcal{F}$ introduced in Definition 3.3.1.

Claim 3.3.6. *The generalized Okamoto ID scheme is* not *BLT-secure against impersonation attacks with respect to arbitrary* $\mathcal{F}$*.*

Proof. The attack uses a single tampering query. Define the tampering function $f(\mathbf{x}, pp) = (\tilde{\mathbf{x}}, \tilde{pp})$ to be as follows:

[7] Note that $\tilde{y} = 0 \bmod \tilde{p}$ implies that for at least one of the generators g_i's we get $\tilde{g}_i = 0 \bmod \tilde{p}$, so that $a = \prod_{i=1}^{\ell} \tilde{g}_i^{r_i} = 0 \bmod \tilde{p}$.

- The witness is unchanged, i.e., $\mathbf{x} = \tilde{\mathbf{x}}$.
- The value $\tilde{p}$ is some prime of size $|\tilde{p}| \approx |p|$ such that the Discrete Log problem is easy in the corresponding group $\tilde{\mathbb{G}}$. (This can be done efficiently by choosing $\tilde{p}-1$ to be the product of small prime (power) factors [PH78].)
- Let $\tilde{g}$ be a generator of $\tilde{\mathbb{G}}$ (which exists since $\tilde{p}$ is a prime) and define the new generators as $\tilde{g}_i = \tilde{g}^{x_i} \bmod \tilde{p}$.

Consider now a transcript $(a, c, \mathbf{z})$ produced by a run of $\mathsf{P}(\tilde{pp}, \mathbf{x})$. We have $a = \tilde{g}^{\sum_{i=1}^{\ell} x_i r_i} \bmod \tilde{p}$ for random $r_i \in \mathbb{Z}_{\tilde{p}}$. By computing the Discrete Log of a in base $\tilde{g}$ (which is easy by our choice of $\tilde{\mathbb{G}}$), we get one equation $\sum_{i=1}^{\ell} x_i r_i = \log_{\tilde{g}}(a) \bmod \tilde{p}$. Asking for polynomially many transcripts, yields ℓ linearly independent equations (with overwhelming probability) and thus allows to solve for $(x_1, \ldots, x_\ell)$. (Note here that with high probability $x_i \bmod p = x_i \bmod \tilde{p}$ since $|p| \approx |\tilde{p}|$.) □

Tampering by "inflating" the prime p. Consider the following class of tampering functions $\mathcal{F}^*_{\mathsf{pub-split}} \supseteq \mathcal{F}_{\mathsf{pub-split}}$: We say that $f \in \mathcal{F}^*_{\mathsf{pub-split}}$ if $f = (f^{sk}, f^{pp})$, where $f^{sk} : \mathcal{SK} \to \{0,1\}^*$ and $f^{pp} : \mathcal{PP} \to \{0,1\}^*$.

Claim 3.3.7. *The generalized Okamoto ID scheme is* not *BLT-secure against impersonation attacks with respect to* $\mathcal{F}^*_{\mathsf{pub-split}}$.

Proof. The attack uses a single tampering query. Consider the following tampering function $f = (f^{sk}, f^{pp}) \in \mathcal{F}^*_{\mathsf{pub-split}}$:

- Choose $\tilde{p}$ to be a prime of size $|\tilde{p}| = \Omega(\ell|p|)$, such that the Discrete Log problem is easy in $\tilde{\mathbb{G}}$. (This can be done as in the proof of Claim 3.3.6.)
- Choose a generator $\tilde{g}$ of $\tilde{\mathbb{G}}$; define $\tilde{g}_1 = \tilde{g}$ and $\tilde{g}_j = 1$ for all $j = 2, \ldots, \ell$.
- Define the witness to be $\tilde{\mathbf{x}}$ such that $\tilde{x}_1 = x_1 || \ldots || x_\ell$ and $\tilde{x}_j = 0$ for all $j = 2, \ldots, \ell$.

Given a single transcript $(a, c, \mathbf{z})$ the adversary learns $a = \tilde{g}^{r_1}$ for some $r_1 \in \mathbb{Z}_{\tilde{p}}$. Since the Discrete Log is easy in this group, A can find r_1. Now the knowledge of c and $z_1 = r_1 + c\tilde{x}_1$, allows to recover $\tilde{x}_1 = (x_1, \ldots, x_\ell)$. □

3.3.4 BLT-Secure Signatures (in Random Oracle Model)

It is well known that every Σ-protocol can be turned into a signature scheme via the Fiat-Shamir heuristic [FS86]. By applying the Fiat-Shamir transformation to the protocol of Figure 3.1, we get efficient BLT-secure signatures in the random oracle model.

3.4 IND-CCA PKE with BLT Security

We start by defining IND-CCA public key encryption (PKE) with BLT security. A PKE scheme is a tuple of algorithms $\mathcal{PKE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ defined as follows. (1) Algorithm Setup takes as input the security parameter and outputs the description of public parameters pp; the set of all public parameters is denoted by $\mathcal{PP}$. (2) Algorithm KGen takes as input the security parameter and outputs a public/secret key pair (pk, sk); the set of all secret keys is denoted by $\mathcal{SK}$ and the

set of all public keys by $\mathcal{PK}$. (3) The randomized algorithm $\mathsf{Encrypt}$ takes as input the public key pk, a message $m \in \mathcal{M}$ and randomness $r \in \mathcal{R}$ and outputs a ciphertext $c = \mathsf{Encrypt}(pk, m; r)$[8]; the set of all ciphertexts is denoted by $\mathcal{C}$. (4) The deterministic algorithm $\mathsf{Decrypt}$ takes as input the secret key sk and a ciphertext $c \in \mathcal{C}$ and outputs $m = \mathsf{Decrypt}(sk, c)$ which is either equal to some message $m \in \mathcal{M}$ or to an error symbol $\perp$.

Definition 3.4.1. *Let $\zeta = \zeta(\lambda)$ and $t = t(\lambda)$ be parameters, and let $\mathcal{F}_{\mathsf{sk}}$ be some set of functions such that $f \in \mathcal{F}_{\mathsf{sk}}$ has a type $f : \mathcal{SK} \to \mathcal{SK}$. We say that $\mathcal{PKE}$ is IND-CCA $(\zeta(\lambda), t(\lambda))$-BLT secure with respect to $\mathcal{F}_{\mathsf{sk}}$ if the following properties are satisfied.*

(i) Correctness. *For all $pp \leftarrow \mathsf{Setup}(1^\lambda)$, $(pk, sk) \leftarrow \mathsf{KGen}(1^\lambda)$ we have that $\Pr[\mathsf{Decrypt}(sk, \mathsf{Encrypt}(pk, m)) = m] = 1$ (where the randomness is taken over the internal coin tosses of algorithm $\mathsf{Encrypt}$).*

(ii) Security. *For all PPT adversaries A, there exists a negligible function $\delta(\lambda) : \mathbb{N} \to [0, 1]$, such that $\Pr[\mathsf{A} \text{ wins}] \leq \frac{1}{2} + \delta(\lambda)$ in the following game:*

1. *The challenger runs $pp \leftarrow \mathsf{Setup}(1^\lambda)$, $(pk, sk) \leftarrow \mathsf{KGen}(1^\lambda)$ and gives (pp, pk) to A.*
2. *The adversary is given oracle access to $\mathsf{Decrypt}(sk, \cdot)$. This oracle outputs polynomially many decryptions of ciphertexts using secret key sk.*
3. *The adversary may adaptively ask t tampering queries. During the ith query, A chooses a function $f_i \in \mathcal{F}_{\mathsf{sk}}$ and gets oracle access to $\mathsf{Decrypt}(\tilde{sk}_i, \cdot)$, where $\tilde{sk}_i = f_i(sk)$. This oracle outputs polynomially many decryptions of ciphertexts using secret key $\tilde{sk}_i$.*
4. *The adversary may adaptively ask polynomially many leakage queries. In the jth query, A chooses a function $L_j : \{0,1\}^* \to \{0,1\}^{\zeta_j}$ and receives back the output of the function applied to sk.*
5. *The adversary outputs two messages of the same length $m_0, m_1 \in \mathcal{M}$ and the challenger computes $c_b \leftarrow \mathsf{Encrypt}(pk, m_b)$ where b is a uniformly random bit.*
6. *The adversary keeps access to $\mathsf{Decrypt}(sk, \cdot)$ and outputs a bit b'. We say A wins if $b = b'$, $\sum_j \zeta_j \leq \zeta$ and c_b has not been queried for to the decryption oracle.*

In case $t = 0$ we get, as a special case, the notion of semantic security against a-posteriori chosen-ciphertext $\zeta(\lambda)$-key-leakage attacks from [NS09]. Notice that A is not allowed to tamper with the secret key after seeing the challenge ciphertext. As we show in Section 3.4.4, this restriction is necessary because otherwise A could overwrite the secret key depending on the plaintext encrypted in c_b, and thus gain some advantage in guessing the value of b by asking additional decryption queries.

We build an IND-CCA BLT-secure PKE scheme in two steps. In Section 3.4.1 we define a weaker notion which we call *restricted* IND-CCA BLT security. In Section 3.4.2 we show a general transformation from restricted IND-CCA BLT security to full-fledged IND-CCA BLT security relying on tSE NIZK proofs [DHLAW10a] in the common reference string (CRS) model. The CRS is supposed to be tamper-free and must be hard-wired into the code of the encryption algorithm; however tampering and leakage can depend adaptively on the CRS and the public parameters. Finally, in Section 3.4.3, we prove that a variant of the BHHO encryption scheme [NS09] satisfies our notion of restricted IND-CCA BLT security.

[8] Here again we abuse notation by using c to denote ciphertexts.

3.4.1 Restricted IND-CCA BLT Security

The main idea of our new security notion is as follows. Instead of giving A full access to a tampering oracle (as in Definition 3.4.1) we restrict his power by allowing him to see the output of the (tampered) decryption oracle only for ciphertexts c for which A already knows both the corresponding plaintext m and the randomness r used to generate c (via the real public key). Essentially this restricts A to submit to the tampering oracle only "well-formed" ciphertexts.

Definition 3.4.2. *Let $\zeta = \zeta(\lambda)$ and $t = t(\lambda)$ be parameters, and let $\mathcal{F}_{\mathsf{sk}}$ be some set of functions such that $f \in \mathcal{F}_{\mathsf{sk}}$ has a type $f : \mathcal{SK} \to \mathcal{SK}$. We say that $\mathcal{PKE}$ is* restricted *IND-CCA $(\zeta(\lambda), t(\lambda))$-BLT secure with respect to $\mathcal{F}_{\mathsf{sk}}$ if it satisfies property (i) of Definition 3.4.1 and property (ii) is modified as follows:*

(ii) Security. *For all PPT adversaries A, there exists a negligible function $\delta(\lambda) : \mathbb{N} \to [0,1]$, such that $\Pr[\mathsf{A}$ wins$] \leq \frac{1}{2} + \delta(\lambda)$ in the following game:*

1. *The challenger runs $pp \leftarrow \mathsf{Setup}(1^\lambda)$, $(pk, sk) \leftarrow \mathsf{KGen}(1^\lambda)$ and gives (pp, pk) to A.*
2. *The adversary may adaptively ask t tampering queries. During the ith query, A chooses a function $f_i \in \mathcal{F}_{\mathsf{sk}}$ and gets oracle access to $\mathsf{Decrypt}^*(\tilde{sk}_i, \cdot, \cdot)$, where $\tilde{sk}_i = f_i(sk)$. This oracle answers polynomially many queries of the following form: Upon input a pair $(m, r) \in \mathcal{M} \times \mathcal{R}$, compute $c \leftarrow \mathsf{Encrypt}(pk, m; r)$ and output a plaintext $\tilde{m} = \mathsf{Decrypt}(\tilde{sk}_i, c)$ using the current tampered key.*
3. *The adversary may adaptively ask leakage queries. In the jth query, A chooses a function $L_j : \{0,1\}^* \to \{0,1\}^{\zeta_j}$ and receives back the output of the function applied to sk.*
4. *The adversary outputs two messages of the same length $m_0, m_1 \in \mathcal{M}$ and the challenger computes $c_b \leftarrow \mathsf{Encrypt}(pk, m_b)$ where b is a uniformly random bit.*
5. *The adversary loses access to all oracles and outputs a bit b'. We say that A wins if $b = b'$ and $\sum_j \zeta_j \leq \zeta$.*

We note that, by setting $t = 0$, we recover the original notion of semantic security under ζ-key-leakage attacks for public key encryption, as defined in [NS09].

3.4.2 A General Transformation

We compile an arbitrary restricted IND-CCA BLT-secure encryption scheme into a full-fledged IND-CCA BLT-secure one by appending to the ciphertext c an argument of "plaintext knowledge" π computed through a (one-time, strong) tSE NIZK argument system (cf. Section 3.2.3). The same construction has been already used by Dodis *et al.* [DHLAW10a] to go from IND-CPA security to IND-CCA security in the context of memory leakage.

The intuition why the transformation works is fairly simple: The argument π enforces the adversary to submit to the tampered decryption oracle only ciphertexts for which he knows the corresponding plaintext (and the randomness used to encrypt it). In the security proof the pair (m, r) can indeed be extracted from such argument, allowing to reduce IND-CCA BLT security to restricted IND-CCA BLT security.

Theorem 3.4.3. *Let $\lambda \in \mathbb{N}$ be the security parameter. Assume that $\mathcal{PKE}$ is a* restricted *IND-CCA $(\zeta(\lambda), t(\lambda))$-BLT secure encryption scheme and that $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ is a one-time strong tSE NIZK argument system for relation $\mathfrak{R}_{\mathsf{PKE}}$. Then, the encryption scheme $\mathcal{PKE}'$ of Figure 3.3 is IND-CCA $(\zeta(\lambda), t(\lambda))$-BLT secure.*

From Restricted to Full-Fledged IND-CCA BLT Security

Let $\mathcal{PKE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a PKE scheme and $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ be a tSE NIZK argument system for the relation:

$$\mathfrak{R}_{\mathsf{PKE}} = \{(pk, c), (m, r) : \ c = \mathsf{Encrypt}(pk, m; r)\}.$$

Define the following PKE scheme $\mathcal{PKE}' = (\mathsf{Setup}', \mathsf{KGen}', \mathsf{Encrypt}', \mathsf{Decrypt}')$.

Setup': Sample $pp \leftarrow \mathsf{Setup}(1^\lambda)$ and $(\Omega, tk, ek) \leftarrow \mathsf{Gen}(1^\lambda)$ and let $pp' = (pp, \Omega)$.

KGen': Run $(pk, sk) \leftarrow \mathsf{KGen}(1^\lambda)$ and set $pk' := pk$ and $sk' := sk$.

$\mathsf{Encrypt}'$: Sample $r \leftarrow \mathcal{R}$ and compute $c \leftarrow \mathsf{Encrypt}(pk, m; r)$. Output (c, π), where $\pi \leftarrow \mathsf{Prove}^{\Omega}((pk, c), (m, r))$.

$\mathsf{Decrypt}'$: Check that $\mathsf{Verify}^{\Omega}((pk, c), \pi) = 1$. If not output $\perp$; otherwise, output $m = \mathsf{Decrypt}(sk, c)$.

Figure 3.3: How to transform a restricted IND-CCA BLT-secure PKE into an IND-CCA BLT-secure PKE

Proof. We prove the theorem by a series of games. All games are a variant of the IND-CCA BLT game and in all games the adversary gets correctly generated public parameters (pp, Ω, pk). Leakage and tampering queries are answered using the corresponding secret key sk. The games will differ only in the way the challenge ciphertext is computed or in the way the decryption oracles work.

Game $\mathcal{G}_1$. This is the IND-CCA BLT game of Definition 3.4.1 for the scheme $\mathcal{PKE}'$. Note in particular that all decryption oracles expect to receive as input a ciphertext of the form (c, π) and proceed to verify the proof π before decrypting the ciphertext (and output $\perp$ if such verification fails). The challenge ciphertext is a pair (c_b, π_b) such that $c_b = \mathsf{Encrypt}(pk, m_b; r)$ and $\pi_b \leftarrow \mathsf{Prove}^{\Omega}((pk, c_b), (m_b, r))$, where $m_b \in \{m_0, m_1\}$ for a uniformly random bit b. Our goal is to upper bound $|\Pr[\mathsf{A} \text{ wins in } \mathcal{G}_1] - 1/2|$.

Game $\mathcal{G}_2$. In this game we change the way the challenge ciphertext is computed by replacing the argument π_b with a simulated argument $\pi_b \leftarrow \mathsf{S}((pk, c_b), tk)$. It follows from the composable NIZK property of the argument system that $\mathcal{G}_1$ and $\mathcal{G}_2$ are computationally close. In particular, there exists a negligible function $\delta_1(\lambda)$ such that $|\Pr[\mathsf{A} \text{ wins in } \mathcal{G}_1] - \Pr[\mathsf{A} \text{ wins in } \mathcal{G}_2]| \leq \delta_1(\lambda)$.

Game $\mathcal{G}_3$. We change the way decryption queries are handled. Queries (c, π) to $\mathsf{Decrypt}(sk, \cdot)$ (such that π accepts) are answered by running the extractor Ext on π, yielding $(m, r) \leftarrow \mathsf{Ext}((pk, c), \pi, ek)$, and returning m.

Queries (c, π) to $\mathsf{Decrypt}(\tilde{sk}_i, \cdot)$ (such that π accepts) are answered as follows. We first extract $(m, r) \leftarrow \mathsf{Ext}((pk, c), \pi, ek)$ as above. Then, instead of returning m, we recompute $c = \mathsf{Encrypt}(pk, m; r)$ and return $\tilde{m} = \mathsf{Decrypt}(\tilde{sk}_i, c)$.

It follows from one-time strong tSE that $\mathcal{G}_2$ and $\mathcal{G}_3$ are computationally close. The reason for this is that A gets to see only a single simulated proof for a true statement (i.e., the pair (pk, c_b)) and thus cannot produce a pair $(c, \pi) \neq (c_b, \pi_b)$ such that the proof π accepts and Ext fails to extract the corresponding plaintext m. In particular, there exists a negligible function $\delta_2(\lambda)$ such that $|\Pr[\mathsf{A} \text{ wins in } \mathcal{G}_2] - \Pr[\mathsf{A} \text{ wins in } \mathcal{G}_3]| \leq \delta_2(\lambda)$.

Game $\mathcal{G}_4$. In the last game we replace the ciphertext c_b in the challenge with an encryption of $0^{|m_b|}$, whereas we still compute the proof as $\pi_b \leftarrow \mathsf{S}((pk, c_b), tk)$.

We claim that $\mathcal{G}_3$ and $\mathcal{G}_4$ are computationally close. This follows from restricted IND-CCA BLT-security of $\mathcal{PKE}$. Assume there exists a distinguisher D between $\mathcal{G}_3$ and $\mathcal{G}_4$. We build an adversary B breaking restricted IND-CCA BLT security for $\mathcal{PKE}$. The adversary B uses D as a black-box as follows.

Reduction B^{D}:

1. Receive (pp, pk) from the challenger, sample $(\Omega, tk, ek) \leftarrow \mathsf{Gen}(1^\lambda)$ and give $pp' = (pp, \Omega)$ and $pk' = pk$ to A.
2. Upon input a normal decryption query (c, π) from A, run the extractor to compute $(m, r) \leftarrow \mathsf{Ext}((pk, c), \pi, ek)$ and return m.
3. Upon input a tampering query $f_i \in \mathcal{F}_{\mathsf{sk}}$, forward f_i to the tampering oracle for $\mathcal{PKE}$. To answer a query (c, π), run the extractor to compute $(m, r) \leftarrow \mathsf{Ext}((pk, c), \pi, ek)$. Submit (m, r) to oracle $\mathsf{Decrypt}^*(\tilde{sk}_i, \cdot, \cdot)$ and receive the answer $\tilde{m}$. Return $\tilde{m}$ to A.
4. Upon input a leakage query L_j, forward L_j to the leakage oracle for $\mathcal{PKE}$.
5. When A outputs $m_0, m_1 \in \mathcal{M}$, sample a random bit b' and output $(m_{b'}, 0^{|m_{b'}|})$. Let c_b be the corresponding challenge ciphertext. Compute $\pi_b \leftarrow \mathsf{S}((pk, c_b), tk)$ and forward (c_b, π_b) to A. Continue to answer normal decryption queries (c, π) from A as above.
6. Output whatever D does.

Notice that the reduction perfectly simulates the environment for A; in particular c_b is either the encryption of randomly chosen message among (m_0, m_1) (as in $\mathcal{G}_3$) or an encryption of zero (as in $\mathcal{G}_4$). Since $\mathcal{PKE}$ is restricted IND-CCA (ζ, t)-BLT secure, it must be $|\Pr[\mathsf{A} \text{ wins in } \mathcal{G}_3] - \Pr[\mathsf{A} \text{ wins in } \mathcal{G}_4]| \leq \delta_3(\lambda)$ for a negligible function $\delta_3 : \mathbb{N} \to [0, 1]$.

As clearly $\Pr[\mathsf{A} \text{ wins in } \mathcal{G}_4] = 1/2$, we have obtained:

$$\begin{aligned}
|\Pr[\mathsf{A} \text{ wins in } \mathcal{G}_1] - 1/2| &= |\Pr[\mathsf{A} \text{ wins in } \mathcal{G}_1] - \Pr[\mathsf{A} \text{ wins in } \mathcal{G}_4]| \\
&\leq |\Pr[\mathsf{A} \text{ wins in } \mathcal{G}_1] - \Pr[\mathsf{A} \text{ wins in } \mathcal{G}_2]| + |\Pr[\mathsf{A} \text{ wins in } \mathcal{G}_2] \\
&\qquad - \Pr[\mathsf{A} \text{ wins in } \mathcal{G}_3]| + |\Pr[\mathsf{A} \text{ wins in } \mathcal{G}_3] - \Pr[\mathsf{A} \text{ wins in } \mathcal{G}_4]| \\
&\leq \delta_1(\lambda) + \delta_2(\lambda) + \delta_3(\lambda) = \mathsf{negl}(\lambda).
\end{aligned}$$

This concludes the proof. □

3.4.3 Instantiation from BHHO

We show that the variant of the encryption scheme introduced by Boneh *et al.* [BHHO08] used in [NS09] is restricted IND-CCA BLT-secure. The proof relies on the observation that one can simulate polynomially many decryption queries for a given tampered key by only leaking a bounded amount of information from the secret key. Hence, security follows from leakage resilience of BHHO (which was already proven in [NS09]).

The BHHO PKE scheme works as follows: (1) Algorithm Setup chooses a group $\mathbb{G}$ of prime order p with generator g and let $pp = (\mathbb{G}, g, p)$; (2) Algorithm KGen samples random vectors $\mathbf{x}, \boldsymbol{\alpha} \in \mathbb{Z}_p^\ell$, computes $g^{\boldsymbol{\alpha}} = (g_1, \ldots, g_\ell)$ and let $sk := \mathbf{x} = (x_1, \ldots, x_\ell)$ and $pk := (h, g^{\boldsymbol{\alpha}})$ where $h = \prod_{i=1}^{\ell} g_i^{x_i}$; (3) Algorithm $\mathsf{Encrypt}$ takes as input pk and a message $m \in \mathcal{M}$, samples a random $r \in \mathbb{Z}_p$ and returns $c = \mathsf{Encrypt}(pk, m; r) = (g_1^r, \ldots, g_\ell^r, h^r \cdot m)$; (4) Algorithm $\mathsf{Decrypt}$ parses $c = (g^{\mathbf{c}_0}, c_1)$ and outputs $m = c_1 \cdot g^{-\langle \mathbf{c}_0, \mathbf{x} \rangle}$, where $\langle \mathbf{c}_0, \mathbf{x} \rangle$ denotes the inner product of $\mathbf{c}_0$ and $\mathbf{x}$.

Proposition 3.4.1. *Let $k \in \mathbb{N}$ be the security parameter and assume that the DDH assumption holds in $\mathbb{G}$ (cf. Section 3.2.1). Then, the BHHO encryption scheme is* restricted *IND-CCA $(\zeta(\lambda), t(\lambda))$-BLT secure, where*

$$\zeta \leq (\ell - 2 - t)\log p - \omega(\log \lambda) \qquad \textit{and} \qquad t \leq \ell - 3.$$

Proof. Naor and Segev [NS09, Section 5.2] showed that BHHO is restricted IND-CCA $(\zeta', 0)$-BLT secure up to $\zeta' \leq (\ell-2)\log p - \omega(\log \lambda)$.[9] Assume there exists an adversary A which breaks restricted IND-CCA (ζ, t)-BLT security with probability $\delta(\lambda) = 1/p(\lambda)$, for some polynomial $p(\cdot)$ and infinitely many values of $\lambda \in \mathbb{N}$. We build an adversary B which breaks restricted IND-CCA $(\zeta', 0)$-BLT security of the encryption scheme, with the same advantage, yielding a contradiction.

Adversary B uses A as a black-box and is described below.

Reduction B^{A}:

1. Receive (pp, pk) from the challenger and forward these values to A.
2. Whenever A asks for a leakage query, submit this query to the leakage oracle and return the answer to A.
3. Upon input a tampering query $f_i \in \mathcal{F}_{\mathsf{sk}}$, submit a leakage query in order to retrieve the value $\tilde{h}_i = \prod_{j=1}^{\ell} g_j^{-\tilde{x}_{j,i}}$, where $\tilde{\mathbf{x}}_i = f_i(\mathbf{x}) = (\tilde{x}_{1,i}, \ldots, \tilde{x}_{\ell,i})$. When A asks for a decryption query (m, r), return $\tilde{m} = (h^r \cdot m) \cdot \tilde{h}_i^r$.
4. Whenever A outputs $m_0, m_1 \in \mathcal{M}$, forward m_0, m_1 to the challenger. Let c_b be the corresponding challenge ciphertext; forward c_b to A.
5. Output whatever A does.

Note that for each of A's tampering queries B has to leak one element in $\mathbb{Z}_p$. Using the value of ζ' from above, this gives $\zeta = \zeta' - t\log p = (\ell - 2 - t)\log p - \omega(\log \lambda)$. Moreover, B produces the right distribution since

$$\tilde{m} = (h^r \cdot m) \cdot \tilde{h}_i^r = c_1 \cdot \left(\prod_{j=1}^{\ell} g_j^{-\tilde{x}_{j,i}} \right)^r = c_1 \cdot \prod_{j=1}^{\ell} g_j^{-r \cdot \tilde{x}_{j,i}} = c_1 \cdot g^{-\sum_{j=1}^{\ell} r\alpha_j \cdot \tilde{x}_{j,i}} = c_1 \cdot g^{\langle \mathbf{c}_0, \tilde{\mathbf{x}}_i \rangle},$$

where $(g^{\mathbf{c}_0}, c_1) = ((g^{r\alpha_1}, \ldots, g^{r\alpha_\ell}), h^r \cdot m)$ is an encryption of m using randomness r and public key h. This simulates perfectly the answer of oracle $\mathsf{Decrypt}^*(\tilde{sk}_i, \cdot, \cdot)$. Hence, B has the same advantage as A which concludes the proof. □

3.4.4 Impossibility of "Post-Challenge" IND-CCA BLT Security

Previous definitions of related-key security for IND-CCA PKE allow the adversary to issue tampering queries even after seeing the challenge ciphertext [BPT12]. The reason why the schemes of [BPT12] can achieve this stronger flavour is that the class of tampering functions is too limited to cause any harm. In fact, as we argue below, when the tampering function can be an arbitrary polynomial time function (as is the case in our schemes), no PKE scheme can be secure if such "post-challenge" tampering queries are allowed.

Proposition 3.4.2. *No one-bit PKE scheme can be "post-challenge" IND-CCA $(0,1)$-BLT secure.*

[9] Recall that for $t = 0$ no decryption query is allowed, and thus restricted IND-CCA $(\zeta', 0)$-BLT security collapses to the notion of semantic security against ζ'–key-leakage attacks from [NS09].

Proof. We build a polynomial time adversary A breaking IND-CCA BLT security. A will ask a single tampering query after seeing the challenge ciphertext c_b (corresponding to $m_b \in \{0,1\}$) and then make a single decryption query to the tampered decryption oracle, to learn the bit b with probability negligibly close to 1. Given the public key pk and challenge ciphertext c_b, adversary A proceeds as follows:

1. Sample $m^* \in \{0,1\}$ uniformly at random and compute $c^* \leftarrow \mathsf{Encrypt}(pk, m^*)$.
2. Define the following tampering query $f_{c_b,c^*,m^*}(sk)$:
   - Run $m_b = \mathsf{Decrypt}(sk, c_b)$. In case $m_b = 0$, let $\tilde{sk} = sk$.
   - In case $m_b = 1$, sample $(pk^*, sk^*) \leftarrow \mathsf{KGen}(1^\lambda)$ until $\mathsf{Decrypt}(sk^*, c^*) \neq m^*$. When this happens, let $\tilde{sk} = sk^*$.
3. Query the decryption oracle $\mathsf{Decrypt}(\tilde{sk}, \cdot)$ with c^*. In case the answer from the oracle is m^* output 0 and otherwise output 1.

For the analysis, assume first that A runs in polynomial time. In this case it is easy to see that the attack is successful with overwhelming probability. In fact, $c^* \neq c_b$ with overwhelming probability and the answer from the tampered decryption oracle clearly allows to recover b.

We claim that A runs in expected polynomial time. This is because if one tries to decrypt c^* using an independent freshly generated secret key sk^*, the resulting plaintext will be uncorrelated, up to a small bias, to the plaintext m^*, for otherwise the underlying PKE scheme would not even be IND-CPA secure. (Recall that c^* is an encryption of m^* under the original public key pk.) This shows that $\Pr[\mathsf{Decrypt}(sk^*, c^*) \neq m^*] \approx 1/2$ and thus the loop ends on average after 2 attempts.

If one insists on the tampering function being polynomial time (and not expected polynomial time) we can just put an upped bound on the number of pairs (pk^*, sk^*) that the function can sample in the loop. This comes at the expense of a negligible error probability. □

3.5 Updating the Key in the *i*Floppy Model

We complement the results from the previous two sections by showing how to obtain security against an unbounded number of tampering queries in the floppy model of [ADW09, ADVW13]. Recall that in this model we assume the existence of an external tamper-free and leakage-free storage (the floppy), which is needed to refresh the secret key on the tamperable device. An important difference between the floppy model considered in this paper and the model of [ADVW13] is that in our case the floppy can contain "user-specific" information, whereas in [ADVW13] it contains a *unique* master key which in principle could be equal for all users. To stress this difference, we refer to our model as the *i*Floppy model.

Clearly, the assumption of a unique master key makes production easier but it is also a single point of failure in the system since in case the content of the floppy is published (e.g., by a malicious user) the entire system needs to be re-initialized.[10] A solution for this is to assume that each floppy contains a different master key as is the case in the *i*Floppy model, resulting in a trade-off between security and production cost and convenience.

[10] We note that in the schemes of [ADVW13] making the content of the floppy public does not constitute a total breach of security; however the security proof completely breaks down, leaving no security guarantee for the schemes at hand.

For simplicity, we consider a model with polynomially many updates where, between each update, the adversary is allowed to leak and tamper only once. However, the schemes in this section can be proven secure in the stronger model where between two key updates the attacker is allowed to leak adaptively ζ bits from the current secret key and tamper with it for some bounded number of times.

3.5.1 ID Schemes in the *i*Floppy Model

An identification scheme $\mathcal{ID} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{P}, \mathsf{V}, \mathsf{Refresh})$ in the *i*Floppy model is defined as follows. (1) Algorithm Setup is defined as in a standard ID scheme. (2) Algorithm Gen outputs an update key uk together with an initial public/secret key pair (pk, sk). (3) Algorithms P and V are defined as in a standard ID scheme. (4) Algorithm $\mathsf{Refresh}$ takes as input the update key uk and outputs a new key sk' for the same public key pk.

Definition 3.5.1. *Let $\zeta = \zeta(\lambda)$ be a parameter, and let $\mathcal{F}_{\mathsf{sk}}$ be some set of functions such that $f \in \mathcal{F}_{\mathsf{sk}}$ has a type $f : \mathcal{SK} \to \mathcal{SK}$. We say that $\mathcal{ID}$ is $(\zeta(\lambda), 1)$-CLT secure against impersonation attacks with respect to $\mathcal{F}_{\mathsf{sk}}$ in the* i*Floppy model, if the following properties are satisfied.*

(i) Correctness. *For all $pp \leftarrow \mathsf{Setup}(1^\lambda)$, $(pk, sk, uk) \leftarrow \mathsf{Gen}(1^\lambda)$ we have that:*

$$(\mathsf{P}(pp, sk) \leftrightarrows \mathsf{V}(pp, pk)) = (\mathsf{P}(pp, \mathsf{Refresh}(uk)) \leftrightarrows \mathsf{V}(pp, pk)) = \texttt{accept}.$$

(ii) Security. *For all PPT adversaries A, there exists a negligible function $\delta : \mathbb{N} \to [0,1]$, such that $\Pr[\mathsf{A}$ wins$] \leq \delta(\lambda)$ in the following game:*

1. *The challenger runs $pp \leftarrow \mathsf{Setup}(1^\lambda)$ and $(pk, sk, uk) \leftarrow \mathsf{Gen}(1^\lambda)$, and gives (pp, pk) to A; let $sk_1 = sk$.*
2. *The adversary is given oracle access to $\mathsf{P}(pp, sk_1)$.*
3. *The adversary may adaptively ask leakage and tampering queries. During the ith query:*
   (a) *A specifies a function $L_i : \{0,1\}^* \to \{0,1\}^\zeta$ and receives back $L_i(sk_i)$.*
   (b) *A specifies a function $f_i : \mathcal{SK} \to \mathcal{SK}$ and is given oracle access to $\mathsf{P}(pp, \tilde{sk}_i)$, where $\tilde{sk}_i = f_i(sk_i)$.*
   (c) *The challenger updates the secret key, $sk_{i+1} \leftarrow \mathsf{Refresh}(uk)$.*
4. *The adversary loses access to all oracles and interacts with an honest verifier V (holding public key pk). We say that A* wins *if $(\mathsf{A}(pp, pk) \leftrightarrows \mathsf{V}(pp, pk))$ outputs $\texttt{accept}$.*

Remark 3.5.2. *One could also consider a more general definition where between two key updates A is allowed to ask multiple leakage queries with output size ζ_j, as long as $\sum_j \zeta_j \leq \zeta$. Similarly, we could allow A to tamper in each round for t times with the secret key sk_i. The constructions in this section can be proven secure in this extended setting, but we stick to Definition 3.5.1 for simplicity.*

A general compiler from bounded to continuous attack. We now describe a compiler to boost any (ζ, t)-BLT secure ID scheme (P, V), to a (ζ, t)-CLT secure ID scheme $(\mathsf{P}', \mathsf{V}')$. The compiler is based upon a standard (not necessarily leakage or tamper resilient) signature scheme $\mathcal{SIG}$, and is described in Figure 3.4.

The basic idea is as follows. We generate the key pair (mpk, msk) using the key generation algorithm of the underlying signature scheme. We store msk in the floppy and publish mpk as

iFloppy ID Compiler

Given as input an ID scheme $\mathcal{ID} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{P}, \mathsf{V})$ and a signature scheme $\mathcal{SIG} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ output an ID scheme $\mathcal{ID}' = (\mathsf{Setup}', \mathsf{Gen}', \mathsf{P}', \mathsf{V}', \mathsf{Refresh})$, specified below.

Setup'**:** Run $pp \leftarrow \mathsf{Setup}(1^\lambda)$ and publish pp.

Gen'**:** Run the key generation algorithm of the underlying signature scheme, obtaining $(mpk, msk) \leftarrow \mathsf{KGen}(1^\lambda)$. Also, run $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$. The value mpk is the actual public key, whereas we refer to the values (pk, sk) as the *temporary* keys. Compute and publish a helper value $\mathsf{help} \leftarrow \mathsf{Sign}(msk, pk)$. The prover P' holds $((pp, pk, \mathsf{help}), sk)$, the verifier V' holds mpk. The master key msk is the update key, which is stored in the floppy.

$\mathsf{P}'((pp, pk, \mathsf{help}), sk) \leftrightarrows \mathsf{V}'(pp, mpk)$**:** The prover P' first sends the pair (pk, help) to V'. The verifier verifies the signature, i.e. it checks that $\mathsf{Vrfy}(mpk, (pk, \mathsf{help}))$ outputs `accept`. If the verification was successful, they run $(\mathsf{P}(pp, sk) \leftrightarrows \mathsf{V}(pp, pk))$ and V' accepts if and only if the interaction leads to `accept`.

$\mathsf{Refresh}$**:** Sample a *fresh* pair $(pk', sk') \leftarrow \mathsf{Gen}(1^\lambda)$ and update the helper value as in $\mathsf{help}' \leftarrow \mathsf{Sign}(msk, pk')$. The prover now holds $((pp, pk', \mathsf{help}'), sk')$.

Figure 3.4: Boosting BLT security to CLT security for ID schemes

P's identity. We also sample a key pair (pk, sk) for $\mathcal{ID}$ (which we call the *temporary* keys) and we provide the prover with a value help which is a signature of pk under the master secret key msk. Whenever P wants to prove its identity, it first sends the temporary pk together with the helper value and V verifies this signature using mpk.[11] If the verification succeeds, P and V run an execution of $\mathcal{ID}$ where P proves it knows the secret key sk corresponding to pk. At the end of each authentication the prover updates its pair of temporary keys using the floppy, using the update key msk to sign the new public key pk' that is freshly generated. We prove the following result.

Theorem 3.5.3. *If $\mathcal{SIG}$ is EUF-CMA and $\mathcal{ID}$ is $(\zeta, 1)$-BLT secure against impersonation attacks with respect to $\mathcal{F}_{\mathsf{sk}}$, then the scheme $\mathcal{ID}'$ output by the compiler of Figure 3.4 is $(\zeta, 1)$-CLT secure against impersonation attacks with respect to $\mathcal{F}_{\mathsf{sk}}$ in the* i*Floppy model.*

Proof. We show that if there exists a PPT adversary A who wins the CLT security game against $\mathcal{ID}'$ with non-negligible probability, then we can build either of two reductions B or C violating BLT security of $\mathcal{ID}$ or EUF-CMA of $\mathcal{SIG}$ (respectively) with non-negligible probability. Let us assume that $\Pr[\mathsf{A} \text{ wins}] \geq \delta(\lambda)$, where $\delta(\lambda) = 1/\mathsf{poly}(\lambda)$ for some polynomial $\mathsf{poly}(\cdot)$ and infinitely many $\lambda \in \mathbb{N}$. The CLT experiment for $\mathcal{ID}'$ is specified below:

CLT Experiment:

1. The challenger runs $pp \leftarrow \mathsf{Setup}'(1^\lambda)$ and $(mpk, msk) \leftarrow \mathsf{KGen}(1^\lambda)$, and gives (pp, mpk) to A.
2. For each $i = 1, \ldots, q(\lambda)$ (where $q(\lambda)$ is some polynomial in the security parameter), the challenger does the following:
   - During round i sample $(pk_i, sk_i) \leftarrow \mathsf{Gen}(1^\lambda)$ and compute $\mathsf{help}_i \leftarrow \mathsf{Sign}(msk, pk_i)$.
   - Give A oracle access to $\mathsf{P}'((pp, pk_i, \mathsf{help}_i), sk_i))$.

[11] Alternatively P can send (pk, help) together with the first message of the identification scheme, in order to keep the same round complexity as in $\mathcal{ID}$.

- Answer the leakage and tampering query from A using key sk_i. The leakage query consists of a function $L_i : \{0,1\}^* \to \{0,1\}^{\zeta}$; the tampering query consists of a tampering function $f_i : \mathcal{SK} \to \mathcal{SK}$.

3. During the impersonation stage, the challenger (playing now the role of the verifier V') receives the pair $(pk^\star, \mathsf{help}^\star)$ from A; if $\mathsf{Vrfy}(mpk, (pk^\star, \mathsf{help}^\star))$ outputs 0, the challenger outputs `reject`. Otherwise, it runs $(\mathsf{A}(pp, pk^\star) \leftrightarrows \mathsf{V}(pp, pk^\star))$ and outputs whatever V does.

Let FRESH be the following event: The event becomes true if the pair $(pk^\star, \mathsf{help}^\star)$ used by A during the impersonation stage of the above experiment is equal to one of the pairs A has seen during the learning phase (i.e., one of the pairs (pk_i, help_i)). We have

$$\Pr[\mathsf{A} \text{ wins}] = \Pr[\mathsf{A} \text{ wins} \wedge \text{FRESH}] + \Pr\left[\mathsf{A} \text{ wins} \wedge \overline{\text{FRESH}}\right], \tag{3.6}$$

where all probabilities are taken over the randomness space of the CLT experiment and over the randomness of A. We now describe a reduction B (using A as a black-box) which breaks BLT security of $\mathcal{ID}$.

Reduction B^{A}:

1. Receive $pp \leftarrow \mathsf{Setup}(1^\lambda)$ from the challenger. Sample $(mpk, msk) \leftarrow \mathsf{KGen}(1^\lambda)$ and forward (pp, mpk) to A.
2. Choose an index $j \leftarrow [q]$ uniformly at random.
3. For all $i = 1, \ldots, q$, simulate the learning stage of A as follows.
   (a) During all rounds i such that $i \neq j$:
      - Sample $(pk_i, sk_i) \leftarrow \mathsf{Gen}(1^\lambda)$ and compute $\mathsf{help}_i \leftarrow \mathsf{Sign}(msk, pk_i)$. Give A oracle access to $\mathsf{P}'((pp, \mathsf{help}_i, pk_i), sk_i)$.
      - Simulate A's leakage and tampering query by using key sk_i.

   (b) During round j:
      - Receive the public key $\overline{pk}$ from the challenger and use this key as the jth temporary public key. Compute $\overline{\mathsf{help}} \leftarrow \mathsf{Sign}(msk, \overline{pk})$.
      - Simulate oracle $\mathsf{P}'((pp, \overline{\mathsf{help}}, \overline{pk}), \overline{sk})$ by forwarding $(\overline{pk}, \overline{\mathsf{help}})$ to A and using the target oracle $\mathsf{P}(pp, \overline{sk})$.
      - Simulate leakage query L_j and tampering query f_j by submitting the same functions to the target oracle.
4. Simulate the impersonation stage for A as follows:
   (a) Receive $(pk^\star, \mathsf{help}^\star)$ from A. If $pk^\star \neq \overline{pk}$ (i.e., B's guess is wrong) abort the execution. Otherwise, run $\mathsf{Vrfy}(mpk, (pk^\star, \mathsf{help}^\star))$ and output `reject` if verification fails.
   (b) Run $(\mathsf{A}(pp, pk^\star) \leftrightarrows \mathsf{V}(pp, pk^\star))$ and use the messages from A in the impersonation stage, to answer the challenge from the target oracle.

Note that B's simulation is perfect, since it simulates all rounds using honestly generated keys whereas round j is simulated using the target oracle which allows for one tampering query and ζ bits of leakage from $\overline{sk}$. Denote with GUESS the event that B guesses the index j correctly. Since

B wins whenever A is successful and $\overline{\text{FRESH}}$ occurs, and moreover event GUESS is independent of all other events, we get

$$\begin{aligned}\Pr[\mathsf{B}\text{ wins}] &= \Pr[\mathsf{B}\text{ wins} \wedge \text{GUESS}] + \Pr\left[\mathsf{B}\text{ wins} \wedge \overline{\text{GUESS}}\right] \\ &\geq \Pr[\mathsf{B}\text{ wins} \wedge \text{GUESS}] = \frac{1}{q(\lambda)} \Pr\left[\mathsf{A}\text{ wins} \wedge \overline{\text{FRESH}}\right].\end{aligned} \tag{3.7}$$

We now describe a second reduction C (using A as a black-box), breaking existential unforgeability of $\mathcal{SIG}$.

Reduction C^{A}:

1. Run $pp \leftarrow \mathsf{Setup}(1^\lambda)$, receive the public key mpk from the challenger and forward (pp, mpk) to A. Denote with msk the secret key corresponding to mpk (which of course is not known to C).
2. For all $i = 1, \ldots, q$, simulate the learning stage of A as follows:
   (a) Sample $(pk_i, sk_i) \leftarrow \mathsf{Gen}(1^\lambda)$. Forward pk_i to the target signing oracle and receive back the corresponding signature $\mathsf{help}_i \leftarrow \mathsf{Sign}(msk, pk_i)$. Simulate oracle access to $\mathsf{P}'((pp, \mathsf{help}_i, pk_i), sk_i)$ using knowledge of key sk_i.
   (b) Simulate the leakage and tampering query using knowledge of key sk_i.
3. During the impersonation stage:
   (a) Receive $(pk^\star, \mathsf{help}^\star)$ (which is a message-signature pair) from A and verify the signature with public key mpk. If verification fails, output some random guess and abort. (In that case A loses and C can only win with negligible probability.)
   (b) Otherwise, run $(\mathsf{A}(pp, pk^\star) \leftrightarrows \mathsf{V}(pp, pk^\star))$ and return to A whatever V does.
   (c) Output forgery $(m^\star = pk^\star, \sigma^\star = \mathsf{help}^\star)$.

Whenever FRESH occurs, the pair $(pk^\star, \mathsf{help}^\star)$ returned by A is such that this $pk^\star$ is different from all the pk_i's it has seen during the learning phase. In this case, whenever A wins, the forgery $(m^\star, \sigma^\star)$ output by C is a valid forgery. Hence,

$$\Pr[\mathsf{C}\text{ wins}] \geq \Pr[\mathsf{A}\text{ wins} \wedge \text{FRESH}]. \tag{3.8}$$

Combining Eq. 3.6-3.8, we obtain:

$$q(\lambda) \cdot \Pr[\mathsf{B}\text{ wins}] + \Pr[\mathsf{C}\text{ wins}] \geq \Pr\left[\mathsf{A}\text{ wins} \wedge \overline{\text{FRESH}}\right] + \Pr[\mathsf{A}\text{ wins} \wedge \text{FRESH}] = \Pr[\mathsf{A}\text{ wins}] \geq \delta(\lambda).$$

Hence either $\Pr[\mathsf{B}\text{ wins}] \geq \delta/(2q)$ or $\Pr[\mathsf{C}\text{ wins}] \geq \delta/2$, which are both non-negligible. □

Remark 3.5.4. *Assuming factoring or DL is hard, we can instantiate Theorem 3.5.3 with our schemes from Section 3.3 resulting into tamper resilient identification schemes in the* i*Floppy model under polynomial many tampering and leakage attacks.*

3.5.2 PKE Schemes in the *i*Floppy Model

A PKE scheme $\mathcal{PKE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Refresh})$ in the *i*Floppy model is defined as follows. (1) Algorithm Setup is defined as in a standard PKE scheme. (2) Algorithm KGen outputs an update key uk together with an initial public/secret key pair (pk, sk). (3) Algorithm $\mathsf{Encrypt}$ and $\mathsf{Decrypt}$ are defined as in a standard PKE scheme. (4) Algorithm $\mathsf{Refresh}$ takes as input the update key uk and outputs a new key sk' for the same public key pk.

Definition 3.5.5. *Let $\zeta = \zeta(\lambda)$ be a parameter, and let $\mathcal{F}_{\mathsf{sk}}$ be some set of functions such that $f \in \mathcal{F}_{\mathsf{sk}}$ has a type $f : \mathcal{SK} \to \mathcal{SK}$. We say that $\mathcal{PKE}$ is IND-CCA $(\zeta(\lambda), 1)$-CLT secure with respect to $\mathcal{F}_{\mathsf{sk}}$ in the* i*Floppy model, if the following properties are satisfied.*

(i) Correctness. *For all $pp \leftarrow \mathsf{Setup}(1^\lambda)$, $(pk, sk, uk) \leftarrow \mathsf{Gen}(1^\lambda)$ we have that:*

$$\Pr\left[\mathsf{Decrypt}(\mathsf{Refresh}(uk), \mathsf{Encrypt}(pk, m)) = m\right] = 1.$$

(ii) Security. *For all PPT adversaries A, there exists a negligible function $\delta : \mathbb{N} \to [0,1]$, such that $\Pr[\mathsf{A}$ wins$] \leq 1/2 + \delta(\lambda)$ in the following game:*

1. *The challenger runs $pp \leftarrow \mathsf{Setup}(1^\lambda)$ and $(pk, sk, uk) \leftarrow \mathsf{Gen}(1^\lambda)$, and gives (pp, pk) to A; let $sk_1 = sk$.*
2. *The adversary is given oracle access to $\mathsf{Decrypt}(sk_1, \cdot)$.*
3. *The adversary may adaptively ask leakage and tampering queries. During the ith query:*
   (a) *A specifies a function $L_i : \{0,1\}^* \to \{0,1\}^\zeta$ and receives back $L_i(sk_i)$.*
   (b) *A specifies a function $f_i : \mathcal{SK} \to \mathcal{SK}$ and is given oracle access to $\mathsf{Decrypt}(\tilde{sk}_i, \cdot)$, where $\tilde{sk}_i = f_i(sk_i)$.*
   (c) *The challenger updates the secret key, $sk_{i+1} \leftarrow \mathsf{Refresh}(uk)$.*
4. *The adversary outputs two messages of the same length $m_0, m_1 \in \mathcal{M}$ and the challenger computes $c_b \leftarrow \mathsf{Encrypt}(pk, m_b)$ where b is a uniformly random bit.*
5. *The adversary outputs a bit b' and wins if $b = b'$.*

The same considerations of Remark 3.5.2 hold here.

Construction from BHHO. As noted in [ADVW13], the BHHO PKE scheme (cf. Section 3.4.3) allows for a very simple update mechanism. When we plug this encryption scheme in the construction of Figure 3.3, we obtain the following scheme. (1) Algorithm Setup chooses a group $\mathbb{G}$ of prime order p with generator g, runs $(\Omega, tk, ek) \leftarrow \mathsf{Gen}(1^\lambda)$ and lets $pp = (\mathbb{G}, g, p, \Omega)$. (2) Algorithm KGen samples random vectors $\boldsymbol{\alpha}, \mathbf{x} \in \mathbb{Z}_p^\ell$ and sets $uk = (\boldsymbol{\alpha}, \mathbf{x})$; furthermore it chooses $sk = \mathbf{x}_1 = \mathbf{x} + \boldsymbol{\beta}$ (where $\boldsymbol{\beta} \leftarrow \ker(\boldsymbol{\alpha})$) and lets $pk = (h, g^{\boldsymbol{\alpha}})$ for $h = g^{\langle \boldsymbol{\alpha}, \mathbf{x} \rangle}$. (3) Algorithm $\mathsf{Encrypt}$ takes as input pk and a message $m \in \mathcal{M}$, samples a random $r \in \mathbb{Z}_p$ and returns $c = (g^{r\boldsymbol{\alpha}}, h^r \cdot m)$ together with a proof $\pi \leftarrow \mathsf{Prove}^\Omega((pk, c), (m, r))$ for $((pk, c), (m, r)) \in \mathfrak{R}_{\mathsf{PKE}}$ (cf. Figure 3.3). (4) Algorithm $\mathsf{Decrypt}$ parses $c = (g^{\mathbf{c}_0}, c_1)$, runs $\mathsf{Verify}^\Omega((pk, c), \pi)$ and outputs $m = c_1 \cdot g^{-\langle \mathbf{c}_0, \mathbf{x}_1 \rangle}$ in case the verification succeeds and $\perp$ otherwise. (5) Algorithm $\mathsf{Refresh}$ samples $\boldsymbol{\beta}_i \leftarrow \ker(\boldsymbol{\alpha})$ and outputs $\mathbf{x}_i = \mathbf{x} + \boldsymbol{\beta}_i$.

The theorem below shows that the above scheme is IND-CCA CLT-secure in the *i*Floppy model. One would expect that a proof of this fact is simple, since the keys after each update are completely fresh and independent (given the public key) and thus security should follow from BLT security of

the underlying scheme. However, it is easy to see that such a proof strategy does not work directly (at least in a black-box way).[12] Unfortunately this requires us to make the proof from scratch. Since the proof relies on ideas already introduced in this paper or borrowed from [ADVW13], we give only a sketch here.

Theorem 3.5.6. *Let $\lambda \in \mathbb{N}$ be the security parameter. Assume that the DDH assumption holds in $\mathbb{G}$. Then, the PKE scheme described above is IND-CCA $(\zeta(\lambda),1)$-CLT secure with respect to $\mathcal{F}_{\mathsf{sk}}$ in the* iFloppy *model, where $\zeta \leq (\ell-3)\log p - \omega(\log\lambda)$.*

Proof (sketch). We define a series of games (starting with the original IND-CCA CLT game) and prove that they are all close to each other.

Game $\mathcal{G}_1$. This is the IND-CCA CLT game. In particular the challenge ciphertext is a pair of the form $(c^* = (g^{r\boldsymbol{\alpha}}, h^r \cdot m_b), \pi^*)$ where $\pi^* \leftarrow \mathsf{Prove}^{\Omega}((pk,c^*),(m_b,r))$, for $m_b \in \{m_0,m_1\}$ and $b \leftarrow \{0,1\}$. Our goal is to bound $|\Pr[\mathsf{A} \text{ wins in } \mathcal{G}_1] - 1/2|$.

Game $\mathcal{G}_2$. In this game we change the way the challenge ciphertext is computed by replacing the argument π^* with a simulated argument $\pi^* \leftarrow \mathsf{S}((pk,c^*),tk)$. It follows from the composable NIZK property of the argument system that $\mathcal{G}_1$ and $\mathcal{G}_2$ are computationally close.

Game $\mathcal{G}_3$. In this game we change the way decryption queries are handled. Queries (c,π) to $\mathsf{Decrypt}(\mathbf{x}_i,\cdot)$ (such that π accepts) are answered by running the extractor Ext on π, yielding $(m,r) \leftarrow \mathsf{Ext}((pk,c),\pi,ek)$, and returning m. Queries (c,π) to $\mathsf{Decrypt}(\tilde{\mathbf{x}}_i,\cdot)$ (such that π accepts) are answered as follows. We first extract $(m,r) \leftarrow \mathsf{Ext}((pk,c),\pi,ek)$ as above. Then, instead of returning m, we recompute $c = \mathsf{Encrypt}(pk,m;r)$ and return $\tilde{m} = \mathsf{Decrypt}(\tilde{\mathbf{x}}_i,c)$.

As argued in the proof of Theorem 3.4.3, $\mathcal{G}_2$ and $\mathcal{G}_3$ are computationally close by the one-time strong tSE property of the argument system.

Game $\mathcal{G}_4$. In this game we change the way the secret keys are refreshed. The challenger first chooses a random $(\ell-2)$-dimensional subspace $S \subset \ker(\boldsymbol{\alpha})$ and samples the new keys $\mathbf{x}_i$ from the affine subspace $\mathbf{x}+S$. We prove that $\mathcal{G}_3$ and $\mathcal{G}_4$ are statistically close by a hybrid argument. Assume there are $q = poly(\lambda)$ updates and define for each $i = 0,\ldots,q$ the following hybrid distribution:

Game $\mathcal{G}_{3,i}$. Sample at the beginning a random $(\ell-2)$-dimensional subspace $S \subset \ker(\boldsymbol{\alpha})$ and modify the refreshing of the key as follows.

- For every $1 < j \leq q-i$, let $\mathbf{x}_j = \mathbf{x} + \boldsymbol{\beta}_j$ where $\boldsymbol{\beta}_j \leftarrow \ker(\boldsymbol{\alpha})$.
- For every $q-i < j \leq q$, let $\mathbf{x}_j = \mathbf{x} + \mathbf{s}_j$ where $\mathbf{s}_j \leftarrow S$.

Note that $\mathcal{G}_3 = \mathcal{G}_{3,0}$ and $\mathcal{G}_4 = \mathcal{G}_{3,q}$. As argued in [ADVW13, Theorem 13] it follows from the affine version of the subspace hiding lemma (see [ADVW13, Corollary 8]) that as long as the leakage is bounded an adversary cannot distinguish leakage on $\boldsymbol{\beta}_i \leftarrow \ker(\boldsymbol{\alpha})$ from leakage on $\mathbf{s}_i \leftarrow S$ (and this holds even if $\boldsymbol{\alpha}$ is public and known at the beginning of the experiment and S becomes known after the leakage occurs). We do loose an additional factor $\log p$ in the leakage bound here, due to the fact that we use one additional leakage query to leak the group element $\tilde{h}_i$ needed to simulate the tampered decryption oracle $\mathsf{Decrypt}(\tilde{\mathbf{x}}_i,\cdot)$ (as we do

[12] We stress that in the PKE case we cannot apply the same trick as for the compiler of Figure 3.4, since that would require to make the scheme interactive.

in the proof of Proposition 3.4.1). This yields the bound $\zeta \leq (\ell-3)\log p - \omega(\log\lambda)$ on the tolerated leakage.

Game $\mathcal{G}_5$. In this game we compute the component c^* of the challenge ciphertext (c^*, π^*) as

$$c^* = (g^{\mathbf{c}_0} = g^{r\boldsymbol{\alpha}}, c_1 = g^{\langle \mathbf{c}_0, \mathbf{x}\rangle} \cdot m_b). \tag{3.9}$$

This is only a syntactical change since $g^{\langle \mathbf{c}_0, \mathbf{x}\rangle} \cdot m_b = (g^{\langle \boldsymbol{\alpha}, \mathbf{x}\rangle})^r \cdot m_b = h^r \cdot m_b$.

Game $\mathcal{G}_6$. In this game the challenger chooses $\boldsymbol{\alpha}$, $\mathbf{x}$ as before and in addition samples a vector $\mathbf{c}_0 \leftarrow \mathbb{Z}_p^\ell$ and sets S to be the $(\ell-2)$-dimensional subspace $S = \ker(\boldsymbol{\alpha}, \mathbf{c}_0)$. The secret keys $\mathbf{x}_i$ are chosen as in the previous game from S. The component c^* of the challenge ciphertext (c^*, π^*) is computed as in Eq. 3.9 using the above vector $\mathbf{c}_0$.

As shown in [ADVW13, Theorem 13], $\mathcal{G}_5$ and $\mathcal{G}_6$ are computationally close by the extended rank-hiding assumption (which is equivalent to DDH).

Game $\mathcal{G}_7$. In this game we change again the way the keys are refreshed, namely each key $\mathbf{x}_i$ is sampled from the full original $(\ell-1)$-dimensional space $\mathbf{x} + \ker(\boldsymbol{\alpha})$. As before, the last two games are close by the affine subspace hiding lemma.

Game $\mathcal{G}_8$. In the last game we change the way the challenge ciphertext is chosen. Namely, we choose a random $v \leftarrow \mathbb{Z}_p$ and let $c^* = (g^{\mathbf{c}_0}, g^v)$. Game $\mathcal{G}_8$ and $\mathcal{G}_7$ are statistically close since $\mathcal{G}_7$ does not reveal anything about $\mathbf{x}$ beyond $\langle \boldsymbol{\alpha}, \mathbf{x}\rangle$ from the public key, and thus $\langle \mathbf{c}_0, \mathbf{x}\rangle$ are statistically close to uniform.

Note that the second element is now independent of the message. Hence, the probability that A wins in $\mathcal{G}_8$ is $1/2$ concluding the proof.

□

Chapter 4

Introduction to Non-malleable Codes

In this chapter we provide formal definitions of non-malleable codes and briefly describe the generic compiler of [DPW10] which turns any functionality into a "hardened" functionality resilience to memory tampering attack using non-malleable codes. The definitions are mostly from the pioneering paper on non-malleable codes by Dziembowski et al. [DPW10] and our work [FMVW14]. We conclude the chapter with a brief overview of the results on non-malleable codes.

Non-malleable codes were introduced by Dziembowski, Pietrzak and Wichs in [DPW10] as a natural relaxation of the well-established notions of error correction and error detection. Intuitively, a code $(\mathsf{Enc}, \mathsf{Dec})$ is non-malleable w.r.t. a family of tampering functions $\mathcal{F}$ if the message contained in a codeword modified via a function $f \in \mathcal{F}$ is either the original message, or a completely "unrelated value". Illustrating more, we consider a three-step experiment Tamper_x^f for some input x and some function $f \in \mathcal{F}$ as follows:

1. Compute an encoding $c \leftarrow \mathsf{Enc}(x)$ using the encoding procedure.
2. Apply the tampering function $f \in \mathcal{F}$ to obtain the tampered codeword $c' = f(c)$.
3. If $c' = c$ then return the special symbol $\mathsf{same}^\star$; otherwise, return $\mathsf{Dec}(c')$.

We say that the coding scheme $(\mathsf{Enc}, \mathsf{Dec})$ is ϵ-non-malleable w.r.t. some family $\mathcal{F}$ if, for every function $f \in \mathcal{F}$ and every messages x, the experiment Tamper_x^f reveals almost no information about x, meaning that it is ϵ-close to some "universal" distribution D_f which depends only on f but not on x. The encoding/decoding functions are public (although encoding may be randomized) and do not contain any secret keys. This makes the notion of non-malleable codes somewhat different from (but conceptually related to) well-studied notions of non-malleability in cryptography introduced by the seminal work of Dolev, Dwork and Naor [DDN91].

Relation to Error Correction/Detection. Notice that non-malleability is a weaker guarantee than error correction/detection; the latter notions ensure that any change in the codeword can be corrected or at least detected by the decoding procedure, whereas the former does allow the message to be modified, but only to an unrelated value. However, in contrast to error correction/detection, efficient non-malleable codes can exist for much richer classes of tampering functions $\mathcal{F}$ thereby enabling it useful for tamper-resilience. For example, error correction/detection is already impossible to achieve for the simple class $\mathcal{F}_{\mathsf{const}}$ which, for every constant c^*, includes a "constant" function f_{c^*} that maps all inputs to c^*. There is always some function in $\mathcal{F}_{\mathsf{const}}$ that maps everything to a *valid* codeword c^*. In contrast, it is trivial to construct codes that are non-malleable w.r.t $\mathcal{F}_{\mathsf{const}}$, as the output of a constant function is clearly independent of its input.

Limitations & Possibility. In Sec. 1.3.2 we have already mentioned that it is impossible to have efficient non-malleable codes against all efficient functions $\mathcal{F}_{\mathsf{eff}}$. Similarly, one can easily observe

that it is *impossible* to have codes that are non-malleable for *all* possible tampering functions[1] denoted by $\mathcal{F}_{\mathsf{all}}$. For any coding scheme $(\mathsf{Enc}, \mathsf{Dec})$, there exists a tampering function $f_{\mathsf{bad}}(c)$, which is hard-coded with the decoding and encoding algorithms, and that recovers $x = \mathsf{Dec}(c)$, creates x' by (e.g.,) flipping the first bit of x, and outputs a valid encoding c' of x'. However, in [DPW10], the authors showed an important positive result: non-malleable codes exist for any function family of size *strictly* less that 2^{2^n}. Given this possibility the natural goal is: *How can we restrict the family such that it is rich as well as does not include* f_{bad} ?

Below we present formal definitions of non-malleable codes and its variants.

4.1 Definitions of Non-malleable Codes

We start with a syntactic definition of *coding scheme*. In general we denote the input length by k and the output length by n.

Definition 4.1.1 (Coding Scheme)**.** *A (k,n)-coding scheme consists of two functions: a possibly* randomized *encoding function* $\mathsf{Enc} : \{0,1\}^k \to \{0,1\}^n$*, and* deterministic *decoding function* $\mathsf{Dec} : \{0,1\}^n \to \{0,1\}^k \cup \{\perp\}$ *such that, for each* $x \in \{0,1\}^k$*,* $\Pr[\mathsf{Dec}(\mathsf{Enc}(x)) = x] = 1$.

We now define non-malleability w.r.t. some family $\mathcal{F}$ of tampering functions.

Definition 4.1.2 (Non-Malleability [DPW10])**.** *Let* $(\mathsf{Enc}, \mathsf{Dec})$ *be a (k,n)-coding scheme and $\mathcal{F}$ be a family of functions $f : \{0,1\}^n \to \{0,1\}^n$. We say that the scheme is* $(\mathcal{F}, \varepsilon)$-non-malleable *if for any $x \in \{0,1\}^k$ and any $f \in \mathcal{F}$, there exists an efficiently sampleable distribution D_f over $\{\mathsf{same}^\star, \perp\} \cup \{0,1\}^n$ such that we have $\mathsf{Real}^f_x \approx_\varepsilon \mathsf{Ideal}^f_x$ where*

$$\mathsf{Real}^f_x \stackrel{def}{=} \left\{ \begin{array}{c} c \leftarrow \mathsf{Enc}(x), c' := f(c), x' = \mathsf{Dec}(c') \\ \textit{Output } x' \textit{ otherwise.} \end{array} \right\}. \tag{4.1}$$

and

$$\mathsf{Ideal}^f_x \stackrel{def}{=} \left\{ \begin{array}{c} \widetilde{x} \leftarrow D_f \\ \textit{Output } x \textit{ if } \widetilde{x} = \mathsf{same}^\star, \textit{ and } \widetilde{x} \textit{ otherwise.} \end{array} \right\}. \tag{4.2}$$

The work of [DPW10] also defines a *stronger* version of non-malleability apart from the above version. The main difference is that, in the above version, the tampered codeword $c' \neq c$ may still encode the original message x whereas the strong version ensures that any change to the codeword completely destroys the original message.

Definition 4.1.3 (Strong Non-Malleability [DPW10])**.** *Let* $(\mathsf{Enc}, \mathsf{Dec})$ *be a (k,n)-coding scheme and $\mathcal{F}$ be a family of functions $f : \{0,1\}^n \to \{0,1\}^n$. We say that the scheme is* $(\mathcal{F}, \varepsilon)$-strong non-malleable *if for any $x_0, x_1 \in \{0,1\}^k$ and any $f \in \mathcal{F}$, we have $\mathsf{Tamper}^f_{x_0} \approx_\varepsilon \mathsf{Tamper}^f_{x_1}$ where*

$$\mathsf{Tamper}^f_x \stackrel{def}{=} \left\{ \begin{array}{c} c \leftarrow \mathsf{Enc}(x), c' := f(c), x' = \mathsf{Dec}(c') \\ \textit{Output } \mathsf{same}^\star \textit{ if } c' = c, \textit{ and } x' \textit{ otherwise.} \end{array} \right\}. \tag{4.3}$$

We now add an additional strengthening which we call *super* non-malleability [FMVW14]. For *super* non-malleable security (defined below), if the tampering manages to modify c to c' such that $c' \neq c$ and $\mathsf{Dec}(c') \neq \perp$, then we will even give the attacker the tampered codeword c' in *full* rather than just giving $x' = \mathsf{Dec}(c')$. Our results in the subsequent chapters mainly consider this definition.

[1] An easy counting shows that there are $2^{n \cdot 2^n}$ n-bit to n-bit functions in total i.e. $|\mathcal{F}_{\mathsf{all}}| = 2^{n \cdot 2^n}$

Definition 4.1.4 (Super Non-Malleability)**.** *Let* $(\mathsf{Enc}, \mathsf{Dec})$ *be a* (k,n)*-coding scheme and* $\mathcal{F}$ *be a family of functions* $f : \{0,1\}^n \to \{0,1\}^n$*. We say that the scheme is* $(\mathcal{F}, \varepsilon)$-super non-malleable *if for any* $x_0, x_1 \in \{0,1\}^k$ *and any* $f \in \mathcal{F}$*, we have* $\mathsf{Tamper}^f_{x_0} \approx_\varepsilon \mathsf{Tamper}^f_{x_1}$ *where:*

$$\mathsf{Tamper}^f_x \stackrel{def}{=} \left\{ \begin{array}{c} c \leftarrow \mathsf{Enc}(x), c' := f(c) \\ \textit{Output } \mathsf{same}^\star \textit{ if } c' = c, \textit{ output } \perp \textit{ if } \mathsf{Dec}(c') = \perp, \textit{ and else output } c'. \end{array} \right\}. \tag{4.4}$$

4.2 Tamper-resilience using Non-Malleable Codes

In the same work [DPW10], Dziembowski at al. showed how non-malleable codes can be used to build a generic (memory only) tamper-resilient compiler. Consider a reactive functionality G (can be equivalently thought of as a circuit $\mathcal{C}$ like in Section 1.2) with secret state st that can be executed on input x. Given a non-malleable code $\mathsf{Code} = (\mathsf{Enc}, \mathsf{Dec})$ w.r.t. some function family $\mathcal{F}$, the generic compiler of [DPW10] transforms the functionality $\mathsf{G}(\mathsf{st}, \cdot)$ into a hardened-functionality $\mathsf{G}^h(\widetilde{\mathsf{st}}, \cdot)$ as follows: Initially, at a pre-processing phase, $\widetilde{\mathsf{st}}$ is set to $\widetilde{\mathsf{st}} \leftarrow \mathsf{Enc}(\mathsf{st})$. Then in the on-line phase each time G^h is executed on input x, the transformed functionality reads the encoded state $\widetilde{\mathsf{st}}$ from memory, decodes it to obtain $\mathsf{st} = \mathsf{Dec}(\widetilde{\mathsf{st}})$ and runs the original functionality $y \leftarrow \mathsf{G}(\mathsf{st}, x)$ outputting y. Finally, G^h re-encodes the key $\widetilde{\mathsf{st}} \leftarrow \mathsf{Enc}(\mathsf{st})$ and restores it in the secret memory. The adversary can now issue tampering query $f \in \mathcal{F}$ to the hardened functionality. A tampering query replaces the current secret state $\widetilde{\mathsf{st}}$ with a tampered state $\widetilde{\mathsf{st}}' = f(\widetilde{\mathsf{st}})$, and the functionality G^h continues its computation using $\widetilde{\mathsf{st}}'$ as the secret state. Notice that in case of $\mathsf{Dec}(\widetilde{\mathsf{st}}') = \perp$ ($\perp$ denotes detection of tampering) the functionality G^h self-destructs, e.g. by setting the memory to a dummy value, to protect the device. Intuitively, non-malleable codes guarantee that any "undetected" (does not result in $\perp$) tampering would be "unrelated" to the original secret and hence is equivalent to *overwriting*. Notice that, the transformed functionality $\mathsf{G}^h(\widetilde{\mathsf{st}}, \cdot)$ is secure against tampering attacks carried out by any function belonging to the same family $\mathcal{F}$ against which the encoding scheme Code is non-malleable. Naturally, richer the $\mathcal{F}$, stronger is our model of tampering. We refer to [DPW10] for more detailed descriptions with formal definitions, security model etc.

4.3 A brief survey on Non-malleable Codes

Since it is impossible to construct non-malleable codes against all possible functions, the main goal is to restrict the function family $\mathcal{F}$. Moreover, the existence result [DPW10], which says that there exists non-malleable codes for any $\mathcal{F}$ of size strictly lesser than 2^{2^n} provides great optimism. Now, the question is *how to meaningfully restrict a family ?*[2]

Granular and split-state tampering One way of such restriction has been already shown in [DPW10]. They showed an efficient construction that protects against bit-wise independent tampering $\mathcal{F}_{\mathsf{bit}}$ (i.e., the adversary can tamper with each bit of the codeword independently of every other bit). Later, Cheraghchi and Guruswami [CG14b] improved that by giving a construction with optimal rate (input message to codeword ratio) and better efficiency for the same family. Choi *et al.* [CKM11] considered an extended tampering family, where the tampering functions can

[2]The instant thought is that, somehow the restriction should enable no tampering function to decode.

be applied to a small (logarithmic in the security parameter) number of blocks independently. Intuitively, this direction assumes *granular* restriction, where the codewords have multiple blocks that can be tampered independently.

Perhaps the least granular and most general such model is the so-called *split-state* model which was first introduced in the context of leakage resilience [DP08, DDV10], where one assumes that the memory is split into two parts that leak independently. Naturally in tampering-context each codeword c consists of two parts, say c_0 and c_1, and the adversary can tamper each of them *arbitrarily but independently*. Starting with the random oracle construction of [DPW10], a few other efficient constructions of non-malleable split-state codes have been proposed, both in the computational setting [LL12, FMNV14] and in the information theoretic setting [DKO13, ADL14, CG14b, CZ14, ADKO15a, ADKO15b]. Very recently, a combination of two beautiful results by Chattopadhyay et al. [CZ14] and Aggarwal et al. [ADKO15a] provided the best state-of-art currently in this direction by showing a construction which attains optimal (constant) rate. In [FMNV14] we put forth a stronger definition of non-malleability called *continuous non-malleable code* and provide construction in split-state model. In Chapter 6 we discuss this work in detail.

Global tampering. Though the split-state model has been extensively studied, there has been only a few attempts to construct non-malleability when the entire codeword is subject to tampering, so-called *global tampering*. In [FMVW14] we took a first dig at this question. Since, due to the trivial impossibility it is not possible to construct non-malleable codes against all efficient functions $\mathcal{F}_{\mathsf{eff}}$, we achieved the "next best thing": for any known bound s we construct a (Monte-Carlo construction of) non-malleable codes secure against global tampering. We discuss this in detail in the next chapter. A concurrent and independent work by Cheraghchi and Guruswami [CG14a] also achieved a similar result. Later our result is improved by Jafargholi and Wichs [JW15]. More recently Agrawal et al. [AGM+15a] provided explicit (in contrast to our Monte-Carlo scheme) construction against bit-level permutation.

New applications of non-malleable codes. Above discussion provides an idea of the huge development took place in the constructions and foundations of non-malleable codes recently. Unfortunately, in spite of such great progress in theory, the applications were limited to memory-tampering. Very recently, a few works tried to find more applications of non-malleable codes apart from the traditional memory-tampering. Two concurrent works by Faust et al. [FMNV15] and Dachman-Soled et al. [DLSZ15] applied non-malleable codes to protect RAM architecture against tampering, thereby initiated a new direction of protecting computations against tampering. The aforementioned work by Agrawal et al. [AGM+15a] applied their non-malleable codes resistant to permutation to construct string-commitments from bit-commitments. Coretti et al. [CMTV14] achieved (some form of) CCA-secure encryption using non-malleable codes. More recently Chandran et al. [CGM+15] constructed non-malleable commitments and another work by Coretti at al. [CDTV15] constructed non-malleable encryptions from non-malleable codes, thus connected the traditional non-malleability notions [DDN91] with non-malleable codes.

Chapter 5

Efficient Non-Malleable Codes, Key-derivations and their applications in Tamper-resilience

In this chapter we provide a construction of "efficient" non-malleable codes, secure against any $\mathcal{F}$ whose size is bounded by a priory known value (which can be any polynomial function of the security parameter λ). Our construction is information theoretic and achieves optimal rate. Moreover, we introduce a related concept of non-malleable key-derivation which can also be used to protect against memory-tampering. An extended abstract, titled "Efficient Non-Malleable Codes and Key-Derivation for Poly-Size Tampering Circuits", of the results discussed in this chapter is published in the proceedings of the 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques – Eurocrypt 2014.

5.1 Introduction

Recall from Chapter 1 that, it is not possible to to construct *efficient* non-malleable codes for all *efficient* tampering functions, because there is always a function f_{bad} which can decode and re-encode and thereby makes it hopeless to achieve non-malleability.

Works [LL12, CKM11, DKO13, ADL14, CG14b] prior to this one constructed non-malleable codes for several rich and interesting function families. In all of them, $\mathcal{F}$ is restricted through their *granularity* rather than their computational *complexity*. In particular, their approach considered that the codeword is split into several (possibly just 2) components, each of which can only be tampered independently of the others. Intuitively, the tampering function only has a "granular" rather than "global" view of the codeword. However, in this work we stick to the global model investigating the following question:

How can we construct efficient non-malleable codes without assuming granular restrictions ?

5.1.1 Our Contribution

In this work, we answered the above question positively. As we saw, we cannot have a single efficient code that is non-malleable for all efficient tampering functions. However, we show the following positive result, which we view as the "next best thing":

Main Result: For any polynomial bound $s = s(n)$ in the codeword size n, and any tampering family $\mathcal{F}$ of size $|\mathcal{F}| \leq 2^s$, there is an efficient code of complexity $\mathsf{poly}(s, \log(1/\varepsilon))$ which is ε-non-malleable w.r.t. $\mathcal{F}$. In particular, $\mathcal{F}$ can be the family of all circuits of size at most s.

The code is secure in the information theoretic setting, and achieves *optimal rate* (the ratio between message and codeword size) arbitrarily close to 1. It has a *simple* construction relying only on limited (say, t-wise) independent hashing.

The CRS Model. In more detail, if we fix some family $\mathcal{F}$ of tampering functions (e.g., circuits of bounded size), our result gives us a *family* of efficient codes, such that, with overwhelming probability, a random member of the family is non-malleable w.r.t. $\mathcal{F}$. Each code in the family is indexed by some hash function h from a t-wise independent family of hash functions $\mathcal{H}$. This result already shows the *existence* of efficient non-malleable codes with some small *non-uniform advice* to indicate a "good" hash function h.

However, we can also efficiently sample a random member of the code family by sampling a random hash function h. Therefore, we find it most appealing to think of this result as providing a uniformly efficient *construction* of non-malleable codes in the "common reference string (CRS)" model, where a random public string consisting of the hash function h is selected once and fixes the non-malleable code. We emphasize that, although the family $\mathcal{F}$ (e.g., circuits of bounded size) is fixed prior to the choice of the CRS, the attacker can choose the tampering function $f \in \mathcal{F}$ (e.g., a particular small circuit) adaptively depending on the choice of h.

We argue that it is unlikely that we can completely de-randomize our construction and come up with a fixed uniformly-efficient code which is non-malleable for all circuits of size (say) $s = O(n^2)$. In particular, this would require a circuit lower bound, showing that the function f_{bad} (described above) *cannot* be computed by a circuit of size $O(n^2)$.

Non-Malleable Key-Derivation. As an additional contribution, we introduce a new primitive called *non-malleable key derivation*. Intuitively, a function $h : \{0,1\}^n \to \{0,1\}^k$ is a non-malleable key derivation for tampering family $\mathcal{F}$ if it guarantees that for any tampering function $f \in \mathcal{F}$, if we sample uniform randomness $x \leftarrow \{0,1\}^n$, the "derived key" $y = h(x)$ is statistically close to uniform even given $y' = h(f(x))$ derived from "tampered" randomness $f(x) \neq x$. Our positive results for non-malleable key derivation are analogous to those for non-malleable codes. One difference is that the rate k/n is now at most $1/2$ rather than 1, and we show that this is optimal.

While we believe that non-malleable key derivation is an interesting notion on its own (e.g., it can be viewed as a dual version of non-malleable extractors [DW09]), we also show it has useful applications for tamper resilience. For instance, consider some cryptographic scheme G using a uniform key in $y \leftarrow \{0,1\}^k$. To protect G against tampering attacks, we can store a bigger key $x \leftarrow \{0,1\}^n$ on the device and temporarily derive $y = h(x)$ each time we want to execute G.

In Sec. 5.6, we show that this approach protects any cryptographic scheme with a *uniform key* against one-time tampering attacks. The main advantage of using a non-malleable key-derivation rather than non-malleable codes is that the key x stored in memory is simply a uniformly random string with no particular structure (in contrast, the codeword in a non-malleable code requires structure).

In Section 5.5, we also show how to use non-malleable key derivation to build a tamper-resilient stream cipher. Our construction is based on a PRG $\mathsf{prg} : \{0,1\}^k \to \{0,1\}^{n+v}$ and a non-malleable key derivation function $h : \{0,1\}^n \to \{0,1\}^k$. For an initial key $s_0 \leftarrow \{0,1\}^n$, sampled uniformly at random, the output of the stream cipher at each round $i \in [q]$ is $(s_i, x_i) := \mathsf{prg}(h(s_{i-1}))$.

5.1.2 Our Techniques

Non-Malleable Codes. Our construction of non-malleable codes is incredibly simple and relies on t-wise independent hashing, where t is proportional to $s = \log |\mathcal{F}|$. In particular, if h_1, h_2 are two such hash functions, we encode a message x into a codeword $c = (r, z, \sigma)$ where r is randomness, $z = x \oplus h_1(r)$ and $\sigma = h_2(r, z)$. The security analysis, on the other hand, requires two independently

interesting components. Firstly, we rely on the notion of leakage-resilient encodings, proposed by Davì, Dziembowski and Venturi [DDV10]. This is a method to encode a secret in such a way that a limited form of leakage on the encoding does not reveal anything about the secret. One of our contributions is to significantly improve the parameters of the construction from [DDV10] by using a fresh and more careful analysis, which gives us such schemes with an essentially optimal rate. Secondly, we analyze a simpler/weaker notion of *bounded* malleability[1], which intuitively guarantees that an adversary seeing the decoding of a tampered codeword can learn only a bounded amount of information on the encoded value. This notion of bounded malleability is significantly simpler to analyze than full non-malleability. Finally, we show how to carefully combine leakage-resilient encodings with bounded non-malleability to get our full construction of non-malleable codes. On a very high (and not entirely precise) level, we can think of h_1 above as providing "leakage resilience" and h_2 as providing "bounded non-malleability".

Non-Malleable Key-Derivation. Our construction of non-malleable key-derivation functions is even simpler: a random t-wise independent hash function h already satisfies the definition, where t is proportional to $s = \log|\mathcal{F}|$. The analysis is again subtle and relies on a careful probabilistic method argument.

5.1.3 Related Works

Most of the related works are provided in Sec. 4.3 of Chapter 4. Here we briefly elaborate a few of them for direct connections with the results of this chapter.

The work of [DPW10] gives an *existential* (inefficient) construction of non-malleable codes for even doubly-exponential sized function families. More precisely, for any constant $0 < \alpha < 1$ and any family $\mathcal{F}$ of functions of size $|\mathcal{F}| \leq 2^{2^{\alpha n}}$ in the codeword size n, there exists an inefficient non-malleable code w.r.t. $\mathcal{F}$; indeed a completely random function gives such a code with high probability. The code is clearly not efficient, and this should be expected for such a broad result: the families $\mathcal{F}$ can include all circuits of size (e.g.,) $s(n) = 2^{n/2}$, which means that the efficiency of the code must exceed $O(2^{n/2})$. Unfortunately, there is no direct way to "scale down" the result in [DPW10] so as to get an efficient construction for singly-exponential-size families. (One can view our work as providing such "scaled down" result.) Moreover, the analysis only yielded a rate of at most $(1-\alpha)/3 < 1/3$, and it was previously not known if rate 1 codes exist even for small function families.

Concurrent and Independent Work. In a concurrent and independent work, Cheraghchi and Guruswami [CG14a] give two related results. Firstly, they improve the probabilistic method construction of [DPW10] and show that, for families $\mathcal{F}$ of size $|\mathcal{F}| \leq 2^{2^{\alpha n}}$, there exist (inherently inefficient) non-malleable codes with rate $1-\alpha$, which they also show to be optimal. This gives the first characterization of the rate of non-malleable codes. Secondly, similar to our results, they discuss a "scaled-down" version that gives non-malleable codes with some interesting level of "efficiency" for singly-exponential tampering families $\mathcal{F}$ of size $|\mathcal{F}| \leq 2^{s(n)}$ for a polynomial $s(n)$. Unfortunately, the construction of [CG14a] is not "efficient" in the usual cryptographic sense. To get error-probability ε, the encoding and decoding procedures require complexity $\mathsf{poly}(1/\varepsilon)$ (or, more precisely, $O(1/\varepsilon^6)$). If we set ε to be negligible, as usually desired in cryptography, then

[1] We stress that the notion of bounded malleability has no connection with bounded tampering model discussed in Chapter 3

the encoding/decoding procedures would require super-polynomial time. In contrast, our encoding/decoding procedures have efficiency $\mathsf{poly}(\log(1/\varepsilon))$ and therefore we can set ε to be negligible while maintaining polynomial-time encoding/decoding.

5.2 Improved Leakage-Resilient Codes

We will rely on leakage-resilience as an important tool in our analysis. The following notion of leakage-resilient codes was defined by [DDV10]. Informally, a code is leakage resilience w.r.t some leakage family $\mathcal{G}$ if, for any $g \in \mathcal{G}$, "leaking" $g(c)$ for a codeword c does not reveal anything about the encoded value.

Definition 5.2.1 (Leakage-Resilient Codes [DDV10])**.** *Let* $(\mathsf{LREnc}, \mathsf{LRDec})$ *be a* (k,n)*-coding scheme. For a function family* $\mathcal{G}$*, we say that* $(\mathsf{LREnc}, \mathsf{LRDec})$ *is* $(\mathcal{G},\varepsilon)$*-leakage-resilient, if for any* $g \in \mathcal{G}$ *and any* $x \in \{0,1\}^k$ *we have* $\mathbf{SD}(g(\mathsf{LREnc}(x)), g(U_n)) \leq \varepsilon$.

The work of [DDV10] gave a probabilistic method construction showing that such codes exist and can be efficient when the size of the leakage family $|\mathcal{G}|$ is singly-exponential. However, the rate k/n was at most some small constant ($< \frac{1}{4}$), even when the family size $|\mathcal{G}|$ and the leakage size ζ are small. Here, we take the construction of [DDV10] and give an improved analysis with improved parameters, showing that the rate can approach 1. In particular, the additive overhead of the code is very close to the leakage-amount ζ, which is optimal. Our result and analysis are also related to the "high-moment crooked leftover hash lemma" of [RSV13], although our construction is somewhat different, relying only on high-independence hash-functions rather than permutations.

Construction. Let $\mathcal{H}$ be a t-wise independent function family consisting of functions $h : \{0,1\}^v \to \{0,1\}^k$. For any $h \in \mathcal{H}$ we define the $(k, n = k+v)$-coding scheme $(\mathsf{LREnc}_h, \mathsf{LRDec}_h)$ where: (1) $\mathsf{LREnc}_h(x) := (r, h(r) \oplus x)$ for $r \leftarrow \{0,1\}^v$; (2) $\mathsf{LRDec}_h((r,z)) := z \oplus h(r)$.

Theorem 5.2.2. *Fix any function family* $\mathcal{G}$ *consisting of functions* $g : \{0,1\}^n \to \{0,1\}^\zeta$*. With probability* $1-\rho$ *over the choice of a random* $h \leftarrow \mathcal{H}$*, the coding scheme* $(\mathsf{LREnc}_h, \mathsf{LRDec}_h)$ *is* $(\mathcal{G},\varepsilon)$*-leakage-resilient as long as:*

$$t \geq \log|\mathcal{G}| + \zeta + k + \log(1/\rho) + 3 \quad \textit{and} \quad v \geq \zeta + 2\log(1/\varepsilon) + \log(t) + 3.$$

Proof. Fix a function family $\mathcal{G}$. Now, taking probabilities (only) over the choice of h, let BAD be the event that $(\mathsf{LREnc}, \mathsf{LRDec})$ is not an $(\mathcal{G},\varepsilon)$-leakage-resilient code. Then,

$$\begin{aligned}
\Pr[\text{BAD}] &= \Pr_{h \leftarrow \mathcal{H}}[\exists\, g \in \mathcal{G}, x \in \{0,1\}^k \quad : \quad \mathbf{SD}(g(\mathsf{LREnc}_h(x)), g(U_n)) > \varepsilon] \\
&\leq \sum_{g \in \mathcal{G}} \sum_{x \in \{0,1\}^k} \Pr_{h \leftarrow \mathcal{H}} \left[\sum_{\alpha \in \{0,1\}^\zeta} \left| \Pr_{r \leftarrow \{0,1\}^v}[g(r, h(r) \oplus x) = \alpha] - \Pr[g(U_n) = \alpha] \right| > 2\varepsilon \right] \quad (5.1)
\end{aligned}$$

where Eq. (5.1) follows by taking a union bound over all $g \in \mathcal{G}$ and $x \in \{0,1\}^k$.

Fix any $g \in \mathcal{G}, x \in \{0,1\}^k$. For any $\alpha \in \{0,1\}^\zeta, r \in \{0,1\}^v$, define $p_\alpha := \Pr[g(U_n) = \alpha]$ and $p_{r,\alpha} := \Pr[g(r, U_k) = \alpha]$. Let $\tilde{p}_\alpha := \max\{\, p_\alpha \,,\, 2^{-\zeta} \,\}$. Note that

$$\sum_{\alpha \in \{0,1\}^\zeta} \tilde{p}_\alpha \leq \sum_\alpha p_\alpha + \sum_\alpha 2^{-\zeta} \leq 2. \quad (5.2)$$

Define the random variable $Y_{r,\alpha}$ such that $Y_{r,\alpha}=1$ if $g(r,h(r)\oplus x)=\alpha$, where the randomness is over the choice of $h\leftarrow\mathcal{H}$. Then $\Pr[Y_{r,\alpha}=1]=p_{r,\alpha}$ and, for a fixed α, the random variables $\{Y_{r,\alpha}\}_{r\in\{0,1\}^v}$ are t-wise independent. Moreover, $\mathbf{E}[\sum_{r\in\{0,1\}^v}Y_{r,\alpha}]=2^v p_\alpha$. Therefore, we have:

$$\begin{aligned}
&\Pr_{h\leftarrow\mathcal{H}}\left[\sum_{\alpha\in\{0,1\}^\zeta}\left|\Pr_{r\leftarrow\{0,1\}^v}[g(r,h(r)\oplus x)=\alpha]-\Pr[g(U_n)=\alpha]\right|>2\varepsilon\right]\\
\le\ &\Pr_{h\leftarrow\mathcal{H}}\left[\exists\alpha\in\{0,1\}^\zeta\quad:\quad\left|\Pr_{r\leftarrow\{0,1\}^v}[g(r,h(r)\oplus x)=\alpha]-p_\alpha\right|>\varepsilon\cdot\tilde{p}_\alpha\right] &(5.3)\\
\le\ &\sum_{\alpha\in\{0,1\}^\zeta}\Pr_{h\leftarrow\mathcal{H}}\left[\left|\sum_{r\in\{0,1\}^v}Y_{r,\alpha}-2^v p_\alpha\right|>2^v\varepsilon\cdot\tilde{p}_\alpha\right]\\
\le\ &\sum_{\alpha\in\{0,1\}^\zeta}K_t\left(\frac{t2^v p_\alpha+t^2}{(2^v\varepsilon\tilde{p}_\alpha)^2}\right)^{t/2} &(5.4)\\
\le\ &\sum_{\alpha\in\{0,1\}^\zeta}K_t\left(\frac{2t2^v\cdot\tilde{p}_\alpha}{(2^v\varepsilon\tilde{p}_\alpha)^2}\right)^{t/2}\le 2^\zeta K_t\left(\frac{2t}{2^{v-\zeta}\varepsilon^2}\right)^{t/2}, &(5.5)
\end{aligned}$$

where (5.3) follows from (5.2), and (5.4) follows from Lemma 2.2.3, with $K_t\le 8$. Finally, (5.5) follow from the fact that the theorem's parameters ensure: $2^v\cdot\tilde{p}_\alpha\ge 2^{v-\zeta}\ge t$.

Combining the above with (5.1), we get: $\Pr[\text{BAD}]\le|\mathcal{G}|\cdot 2^{\zeta+k}\cdot 8\left(\frac{2t}{2^{v-\zeta}\varepsilon^2}\right)^{t/2}$. Therefore to satisfy $\Pr[\text{BAD}]\le\rho$ we can set:

$$v\ge\zeta+2\log(1/\varepsilon)+\log(t)+3\quad\text{and}\quad t\ge\log|\mathcal{G}|+\zeta+k+\log(1/\rho)+3.$$

□

5.3 Our Non-Malleable Codes

We now construct a non-malleable code for any family $\mathcal{F}$ of sufficiently small size. We will rely on leakage-resilience as an integral part of the analysis.

Construction. Let $\mathcal{H}_1$ be a family of hash functions $h_1:\{0,1\}^{v_1}\to\{0,1\}^k$, and $\mathcal{H}_2$ be a family of hash functions $h_2:\{0,1\}^{k+v_1}\to\{0,1\}^{v_2}$ such that $\mathcal{H}_1$ and $\mathcal{H}_2$ are both t-wise independent. For any $(h_1,h_2)\in\mathcal{H}_1\times\mathcal{H}_2$, define $\mathsf{Enc}_{h_1,h_2}(x)=(r,z,\sigma)$ where $r\leftarrow\{0,1\}^{v_1}$ is random, $z:=x\oplus h_1(r)$ and $\sigma:=h_2(r,z)$. The codewords are of size $n:=|(r,z,\sigma)|=k+v_1+v_2$. Correspondingly define $\mathsf{Dec}((r,z,\sigma))$ which first checks $\sigma\stackrel{?}{=}h_2(r,z)$ and if this fails, outputs $\perp$, else outputs $z\oplus h_1(r)$. Notice that, we can think of (r,z) as being a leakage-resilient encoding of x; i.e., $(r,z)=\mathsf{LREnc}_{h_1}(x;r)$.

Theorem 5.3.1. *For any function family $\mathcal{F}$, the above construction $(\mathsf{Enc}_{h_1,h_2},\mathsf{Dec}_{h_1,h_2})$ is an $(\mathcal{F},\varepsilon)$-super non-malleable code with probability $1-\rho$ over the choice of h_1,h_2 as long as:*

$$\begin{aligned}
t\ge t^*\quad &\textit{for some}\quad t^*=O(\log|\mathcal{F}|+n+\log(1/\rho))\\
v_1>v_1^*\quad &\textit{for some}\quad v_1^*=3\log(1/\varepsilon)+3\log(t^*)+O(1)\\
v_2>v_1+3.&
\end{aligned}$$

For example, in the above theorem, if we set $\rho = \varepsilon = 2^{-\lambda}$ for "security parameter" λ, and $|\mathcal{F}| = 2^{s(n)}$ for some polynomial $s(n) = n^{O(1)} \geq n \geq \lambda$, then we can set $t = O(s(n))$ and the message length $k := n - (v_1 + v_2) = n - O(\lambda + \log n)$. Therefore the rate of the code k/n is $1 - O(\lambda + \log n)/n$ which approaches 1 as n grows relative to λ.

5.3.1 Proof of Theorem 5.3.1

Useful Notions. For a coding scheme $(\mathsf{Enc}, \mathsf{Dec})$, we say that $c \in \{0,1\}^n$ is *valid* if $\mathsf{Dec}(c) \neq \perp$. For any function $f : \{0,1\}^n \to \{0,1\}^n$, we say that $c' \in \{0,1\}^n$ is δ-heavy for f if $\Pr[f(\mathsf{Enc}(U_k)) = c'] \geq \delta$. Define

$$H_g(\delta) = \{c' \in \{0,1\}^n \ : \ c' \text{ is } \delta\text{-heavy for } f\}.$$

Notice that $|H_f(\delta)| \leq 1/\delta$.

Definition 5.3.2 (Bounded-malleable)**.** *We say that a coding scheme* $(\mathsf{Enc}, \mathsf{Dec})$ *is* $(\mathcal{F}, \delta, \tau)$-bounded-malleable *if for all* $f \in \mathcal{F}, x \in \{0,1\}^k$ *we have*

$$\Pr[c' \neq c \wedge c' \text{ is valid } \wedge c' \notin H_f(\delta) \mid c \leftarrow \mathsf{Enc}(x), c' = f(c)] \leq \tau,$$

where the probability is over the randomness of the encoding.

Intuition. The above definition says the following. Take any message $x \in \{0,1\}^k$, tampering function $f \in \mathcal{F}$ and do the following: choose $c \leftarrow \mathsf{Enc}(x)$, set $c' = f(c)$, and output: (1) $\mathsf{same}^\star$ if $c' = c$, (2) $\perp$ if c' is not valid, (3) c' otherwise. Then, with probability $1 - \tau$ the output of the above experiment takes on one of the values: $\{\mathsf{same}^\star, \perp\} \cup H_f(\delta)$. Therefore, the output of the above tampering experiment only leaks a bounded amount of information about c; in particular it leaks at most $\zeta = \lceil \log(1/\delta + 2) \rceil$ bits. Furthermore the "leakage" on c is independent of the choice of the code, up to knowing which codewords are valid and which are δ-heavy. In particular, in our construction, the "leakage" only depends on the choice of h_2 but *not* on the choice of h_1. This will allow us to then rely on the fact that $\mathsf{LREnc}_{h_1}(x; r) = (r, h_1(r) \oplus x)$ is a leakage-resilient encoding of x to argue that the output of the above experiment is the same for x as for a uniformly random value. We formalize this intuition below.

From Bounded-Malleable to Non-Malleable. For any "tampering function" family $\mathcal{F}$ consisting of functions $f : \{0,1\}^n \to \{0,1\}^n$, any $\delta > 0$, and any $h_2 \in \mathcal{H}_2$ we define the "leakage function" family $\mathcal{G} = \mathcal{G}(\mathcal{F}, h_2, \delta)$ which consists of the functions $g_f : \{0,1\}^{k+v_1} \to H_f(\delta) \cup \{\mathsf{same}^\star, \perp\}$ for each $f \in \mathcal{F}$. The functions are defined as follows:

- $g_f(c_1)$: Compute $\sigma = h_2(c_1)$. Let $c := (c_1, \sigma), c' = f(c)$. If c' is not valid output $\perp$. Else if $c' = c$ output $\mathsf{same}^\star$. Else if $c' \in H_f(\delta)$ output c'. Lastly, if none of the above cases holds, output $\perp$.

Notice that the notion of "δ-heavy" and the set $H_f(\delta)$ are completely specified by h_2 and do not depend on h_1. This is because the distribution $\mathsf{Enc}_{h_1,h_2}(U_k)$ is equivalent to $(U_{k+v_1}, h_2(U_{k+v_1}))$ and therefore c' is δ-heavy if and only if $\Pr[f(U_{k+v_1}, h_2(U_{k+v_1})) = c'] \geq \delta$. Therefore the family $\mathcal{G} = \mathcal{G}(\mathcal{F}, h_2, \delta)$ is fully specified by $\mathcal{F}, h_2, \delta$. Also notice that $|\mathcal{G}| = |\mathcal{F}|$ and that the output length of the functions g_f is given by $\zeta = \lceil \log(|H_f(\delta)| + 2) \rceil \leq \lceil \log(1/\delta + 2) \rceil$.

Lemma 5.3.3. *Let $\mathcal{F}$ be any function family and let $\delta > 0$. Fix any h_1, h_2 such that $(\mathsf{Enc}_{h_1,h_2}, \mathsf{Dec}_{h_1,h_2})$ is $(\mathcal{F}, \delta, \varepsilon/4)$-bounded-malleable and $(\mathsf{LREnc}_{h_1}, \mathsf{LRDec}_{h_1})$ is $(\mathcal{G}(\mathcal{F}, h_2, \delta), \varepsilon/4)$-leakage-resilient, where the family $\mathcal{G} = \mathcal{G}(\mathcal{F}, h_2, \delta)$, with size $|\mathcal{G}| = |\mathcal{F}|$, is defined above, and the leakage amount is $\zeta = \lceil \log(1/\delta + 2) \rceil$. Then $(\mathsf{Enc}_{h_1,h_2}, \mathsf{Dec}_{h_1,h_2})$ is $(\mathcal{F}, \varepsilon)$-non-malleable.*

Proof. For any $x_0, x_1 \in \{0,1\}^k$ and any $f \in \mathcal{F}$:

$$
\begin{aligned}
\mathsf{Tamper}^f_{x_0} \quad = \quad & \left\{ \begin{array}{c} c \leftarrow \mathsf{Enc}_{h_1,h_2}(x_0), \tilde{c} := f(c) \\ \text{Output} \ : \mathsf{same}^\star \text{ if } \tilde{c} = c, \perp \text{ if } \mathsf{Dec}_{h_1,h_2}(\tilde{c}) = \perp, \tilde{c} \text{ otherwise.} \end{array} \right\} \\
\overset{\text{stat}}{\approx}_{\varepsilon/4} \quad & \left\{ \begin{array}{c} c_1 \leftarrow \mathsf{LREnc}_{h_1}(x_0) \\ \text{Output} \ : g_f(c_1) \end{array} \right\} && (5.6) \\
\overset{\text{stat}}{\approx}_{\varepsilon/4} \quad & \left\{ \begin{array}{c} c_1 \leftarrow \mathsf{LREnc}_{h_1}(U_k) \\ \text{Output} \ : g_f(c_1) \end{array} \right\} && (5.7) \\
\overset{\text{stat}}{\approx}_{\varepsilon/4} \quad & \left\{ \begin{array}{c} c_1 \leftarrow \mathsf{LREnc}_{h_1}(x_1) \\ \text{Output} \ : g_f(c_1) \end{array} \right\} && (5.8) \\
\overset{\text{stat}}{\approx}_{\varepsilon/4} \quad & \left\{ \begin{array}{c} c \leftarrow \mathsf{Enc}_{h_1,h_2}(x_1), \tilde{c} := f(c) \\ \text{Output} \ : \mathsf{same}^\star \text{ if } \tilde{c} = c, \perp \text{ if } \mathsf{Dec}_{h_1,h_2}(\tilde{c}) = \perp, \tilde{c} \text{ otherwise.} \end{array} \right\} && (5.9) \\
= \mathsf{Tamper}^f_{x_1}
\end{aligned}
$$

Eq. (5.6) and Eq. (5.9) follows as $(\mathsf{Enc}_{h_1,h_2}, \mathsf{Dec}_{h_1,h_2})$ is an $(\mathcal{F}, \delta, \varepsilon/4)$-bounded-malleable code, and Eq. (5.7) and Eq. (5.8) follow as the code $(\mathsf{LREnc}_{h_1}, \mathsf{LRDec}h_1)$ is $(\mathcal{G}(\mathcal{F}, \delta), \varepsilon/4)$-leakage-resilient. □

We can use Theorem 5.2.2 to show that $(\mathsf{LREnc}_{h_1}, \mathsf{LRDec}_{h_1})$ is $(\mathcal{G}(\mathcal{F}, h_2, \delta), \varepsilon/4)$-leakage-resilient with overwhelming probability. Therefore, it remains to show that our construction is $(\mathcal{F}, \delta, \tau)$-bounded-malleable which is given by the following lemma.

Lemma 5.3.4. *For any function family $\mathcal{F}$, any $\delta > 0$, the code $(\mathsf{Enc}_{h_1,h_2}, \mathsf{Dec}_{h_1,h_2})$ is $(\mathcal{F}, \delta, \tau)$-bounded-malleable with probability $1 - \psi$ over the choice of h_1, h_2 as long as:*

$$
\begin{aligned}
\tau &\geq 2(\log|\mathcal{F}| + k + \log(1/\psi) + 2)\delta \\
t &\geq \log|\mathcal{F}| + n + k + \log(1/\psi) + 5 \\
v_1 &\geq 2\log(1/\delta) + \log(t) + 4 \quad \textit{and} \quad v_2 \geq v_1 + 3.
\end{aligned}
$$

Analysis of Bounded-Malleable Codes. Before providing the detail proof we give some intuitions. We now show that the code $(\mathsf{Enc}_{h_1,h_2}, \mathsf{Dec}_{h_1,h_2})$ is bounded-malleable with overwhelming probability. As a very high-level intuition, if a tampering function f can often map valid codewords to other valid codewords (and many different ones), then it must guess the output of h_2 on many different inputs. If the family $\mathcal{F}$ is small enough, it is highly improbable that it would contain some such f. For more detailed intuition, we show that the following two properties hold for any message x and any function f with overwhelming probability: (i) there is at most some "small" set of q valid codewords c' that we can hit by tampering some encoding of x via f; (ii) for each such codeword c' which is not δ-heavy, the probability of landing in c' after tampering an encoding of x cannot be higher than 2δ. This shows that the total probability of tampering an encoding of x and landing in a valid codeword which is not δ-heavy is at most $2q\delta$, which is small. Property (i) roughly follows by showing that f would need to "predict" the output of h_2 on q different inputs,

and property (ii) follows by using "leakage resilience" of h_1 to argue that we cannot distinguish an encoding of x from an encoding of a random message, for which the probability of landing in c' is at most δ.

Next we provide the detailed proof.

Proof of Lemma 5.3.4 Set $q := \lceil \log|\mathcal{F}| + k + \log(1/\psi) + 1 \rceil$. For any $f \in \mathcal{F}, x \in \{0,1\}^k$ define the events $E_1^{f,x}$ and $E_2^{f,x}$ over the random choice of h_1, h_2 as follows:

1. $E_1^{f,x}$ occurs if there exist at least q distinct values $c'_1, \ldots, c'_q \in \{0,1\}^n$ such that each c'_i is valid and $c'_i = f(c_i)$ for some $c_i \neq c'_i$ which encodes the message x (i.e., $c_i = \mathsf{Enc}_{h_1,h_2}(x; r_i)$ for some r_i).

2. $E_2^{f,x}$ occurs if there exists some $c' \in \{0,1\}^n \setminus H_f(\delta)$ such that $\Pr_{r \leftarrow \{0,1\}^{v_1}}[f(\mathsf{Enc}_{h_1,h_2}(x;r)) = c'] \geq 2\delta$.

Let $E_1 = \bigvee_{f,x} E_1^{f,x}$, $E_2 = \bigvee_{f,x} E_2^{f,x}$ and $\textsc{Bad} = E_1 \vee E_2$. Assume (h_1, h_2) are any hash functions for which the event $\textsc{Bad}$ does *not* occur. Then, for every $f \in \mathcal{F}, x \in \{0,1\}^k$:

$$\begin{aligned} & \Pr[f(C) \neq C \wedge f(C) \text{ is valid } \wedge f(C) \notin H_f(\delta)] \\ = & \sum_{c' \colon c' \text{valid and } c' \notin H_f(\delta)} \Pr[f(C) = c' \wedge C \neq c'] < 2q\delta \leq \tau, \end{aligned} \tag{5.10}$$

where $C = \mathsf{Enc}_{h_1,h_2}(x; U_{v_1})$ is a random variable. Eq. (5.10) holds since (i) given that E_1 does not occur, there are fewer than q values c' that are valid and for which $\Pr[f(C) = c' \wedge C \neq c'] > 0$, and (ii) given that E_2 does not occur, for any $c' \notin H_f(\delta)$, we also have $\Pr[f(C) = c' \wedge C \neq c'] \leq \Pr[f(C) = c'] < 2\delta$.

Therefore, if the event $\textsc{Bad}$ does not occur, then the code is $(\mathcal{F}, \delta, \tau)$-bounded-malleable. This means:

$$\Pr_{h_1,h_2}[(\mathsf{Enc}_{h_1,h_2}, \mathsf{Dec}_{h_1,h_2}) \quad \text{is not} \quad (\mathcal{F}, \delta, \tau)\text{-bounded-malleable}] \leq \Pr[\textsc{Bad}] \leq \Pr[E_1] + \Pr[E_2].$$

So it suffices to show that $\Pr[E_1]$ and $\Pr[E_2]$ are both bounded by $\psi/2$, which we do next.

Claim 5.3.5. $\Pr[E_1] \leq \psi/2$.

Proof. Fix some message $x \in \{0,1\}^k$ and some function $f \in \mathcal{F}$. Assume that the event $E_1^{f,x}$ occurs for some choice of hash functions (h_1, h_2). Then there must exist some values $\{r_1, \ldots, r_q\}$ such that: if we define $c_i := \mathsf{Enc}(x; r_i), c'_i := f(c_i)$ then $c'_i \neq c_i$, c'_i is valid, and $|\{c'_1, \ldots, c'_q\}| = q$. The last condition also implies $|\{c_1, \ldots, c_q\}| = q$. However, it is possible that $c_i = c'_j$ for some $i \neq j$. We claim that we can find a subset of at least $s := \lceil q/3 \rceil$ of the indices such that the $2s$ values $\{c_{a_1}, \ldots, c_{a_s}, c'_{a_1}, \ldots, c'_{a_s}\}$ are all distinct. To do so, notice that if we want to keep some index i corresponding to values c_i, c'_i, we need to take out at most two indices j, k in case $c'_j = c_i$ or $c_k = c'_i$.[2] To summarize, if $E_1^{f,x}$ occurs, then (by re-indexing) there is some set $R = \{r_1, \ldots, r_s\} \subseteq \{0,1\}^{v_1}$ of size $|R| = s$ satisfying the following two conditions:

[2] In other words, if we take any set of tuples $\{(c_i, c'_i)\}$ such that all the left components are distinct $c_i \neq c_j$ and all the right components are distinct $c'_i \neq c'_j$, but there may be common values $c_i = c'_j$, then there is a subset of at least $1/3$ of the tuples such that all left and right components in this subset are mutually distinct.

(i) If we define $c_i := \mathsf{Enc}(x; r_i)$, $c'_i \neq c_i$ and c'_i is valid meaning that $c'_i = (r'_i, z'_i, \sigma'_i)$ where $\sigma' = h_2(r', z')$.

(ii) $|\{c_1, \ldots, c_s, c'_1, \ldots, c'_s\}| = 2s$.

Therefore we have:

$$\begin{aligned}
\Pr[E_1^{f,x}] &\leq \Pr_{h_1,h_2}[\exists R \subseteq \{0,1\}^{v_1}, |R| = s, R \text{ satisfies } (i) \text{ and } (ii)] \leq \sum_R \Pr_{h_1,h_2}[R \text{ satisfies } (i) \text{ and } (ii)] \\
&\leq \sum_{R=\{r_1,\ldots,r_s\}} \max_{h_1,\sigma_1,\ldots,\sigma_s} \Pr_{h_2}\left[\forall i\ ,\ c'_i \text{ valid} \;\middle|\; \begin{array}{c} c_i := (r_i, z_i = h_1(r_i) \oplus x, \sigma_i), c'_i := f(c_i), c'_i \neq c_i \\ |\{c_1, \ldots, c_s, c'_1, \ldots, c'_s\}| = 2s \end{array}\right] \\
&\leq \binom{2^{v_1}}{s} 2^{-sv_2} \leq \left(\frac{e2^{v_1}}{s}\right)^s 2^{-sv_2} \leq 2^{s(v_1-v_2)} \leq 2^{q(v_1-v_2)/3} \leq 2^{-q}, \qquad (5.11)
\end{aligned}$$

where Eq. (5.11) follows from the fact that, even if we condition on any choice of the hash function h_1 which fixes $z_i = h_1(r_i) \oplus x$, and any choice of the s values $\sigma_i = h_2(r_i, z_i)$, which fixes $c_i := (r_i, z_i = h_1(r_i) \oplus x, \sigma_i), c'_i := f(c_i)$ such that $c'_i \neq c_i$ and $|\{c_1, \ldots, c_s, c'_1, \ldots, c'_s\}| = 2s$, then the probability that $h_2(r'_i, z'_i) = \sigma'_i$ for all $i \in [s]$ is at most 2^{-sv_2}. Here we use the fact that $\mathcal{H}_2$ is t-wise independent where $t \geq q \geq 2s$. Now, we calculate

$$\Pr[E_1] \leq \sum_{f \in \mathcal{F}} \sum_{x \in \{0,1\}^k} \Pr[E_1^{f,x}] \leq |\mathcal{F}| 2^{k-q} \leq \psi/2,$$

where the last inequality follows from the assumption that $q = \lceil \log |\mathcal{F}| + k + \log(1/\psi) + 1 \rceil$. □

Claim 5.3.6. $\Pr[E_2] \leq \psi/2$.

Proof. For this proof, we will rely on the leakage resilience property of the code $(\mathsf{LREnc}_{h_1}, \mathsf{LRDec}_{h_1})$ as shown in Theorem 5.2.2. First, let us write:

$$\begin{aligned}
\Pr[E_2] &= \Pr_{h_1,h_2}\left[\exists (f, x, c') \in \mathcal{F} \times \{0,1\}^k \times \{0,1\}^n \setminus H_f(\delta) \ :\ \Pr[f(\mathsf{Enc}_{h_1,h_2}(x; U_{v_1})) = c'] \geq 2\delta\right] \\
&\leq \Pr_{h_1,h_2}\Big[\exists (f, x, c') \in \mathcal{F} \times \{0,1\}^k \times \{0,1\}^n \setminus H_f(\delta) \ : \qquad (5.12) \\
&\qquad\qquad \left|\Pr[f(\mathsf{Enc}_{h_1,h_2}(x; U_{v_1})) = c'] - \Pr[f(\mathsf{Enc}_{h_1,h_2}(U_k; U_{v_1})) = c']\right| \geq \delta\Big]
\end{aligned}$$

since, for any $c' \notin H_f(\delta)$, we have $\Pr[f(\mathsf{Enc}_{h_1,h_2}(U_k; U_{v_1})) = c'] < \delta$ by definition. Notice that we can write $\mathsf{Enc}_{h_1,h_2}(x; r) = (c_1, c_2)$ where $c_1 = \mathsf{LREnc}_{h_1}(x; r)$, $c_2 = h_2(c_1)$. We will now rely on the leakage resilience of the code $(\mathsf{LREnc}_{h_1}, \mathsf{LRDec}_{h_1})$ to bound the above probability by $\psi/2$. In fact, we show that the above holds even if we take the probability over h_1 only, for a worst-case choice of h_2.

Let us fix some choice of h_2 and define the family $\mathcal{G} = \mathcal{G}(h_2)$ of leakage functions $\mathcal{G} = \{g_{f,c'} : \{0,1\}^{k+v_1} \to \{0,1\} \mid f \in \mathcal{F}, c' \in \{0,1\}^n\}$ with output size $\zeta = 1$ bits as follows:

- $g_{f,c'}(c_1)$: Set $c = (c_1, c_2 = h_2(c_1))$. If $f(c) = c'$ output 1, else output 0.

Notice that the size of the family $\mathcal{G}$ is $2^n|\mathcal{F}|$ and the family does not depend on the choice of h_1. Therefore, continuing from inequality (5.12), we get:

$$\begin{aligned}
\Pr[E_2] &\leq \Pr_{h_1,h_2}\Big[\exists(f,x,c') \in \mathcal{F}\times\{0,1\}^k\times\{0,1\}^n \setminus H_f(\delta) \ : \\
&\qquad\qquad \big|\Pr[f(\mathsf{Enc}_{h_1,h_2}(x;U_{v_1}))=c'] - \Pr[f(\mathsf{Enc}_{h_1,h_2}(U_k;U_{v_1}))=c']\big| \geq \delta\Big] \\
&\leq \max_{h_2}\Pr_{h_1}\left[\exists(g_{f,c'},x)\in\mathcal{G}(h_2)\times\{0,1\}^k \ : \ \left|\begin{array}{c}\Pr[g_{f,c'}(\mathsf{LREnc}_{h_1}(x;U_{v_1}))=1]\\ -\Pr[g_{f,c'}(\mathsf{LREnc}_{h_1}(U_k;U_{v_1}))=1]\end{array}\right| \geq \delta\right] \\
&= \max_{h_2}\Pr_{h_1}\left[\exists(g_{f,c'},x)\in\mathcal{G}(h_2)\times\{0,1\}^k \ : \ \left|\begin{array}{c}\Pr[g_{f,c'}(\mathsf{LREnc}_{h_1}(x;U_{v_1}))=1]\\ -\Pr[g_{f,c'}(U_{k+v_1})=1]\end{array}\right| \geq \delta\right] \\
&\leq \max_{h_2}\Pr_{h_1}[\ (\mathsf{LREnc}_{h_1},\mathsf{LRDec}_{h_1}) \text{ is not } (\mathcal{G}(h_2),\delta)\text{-Leakage-Resilient }] \leq \psi/2,
\end{aligned}$$

where the last inequality follows from Theorem 5.2.2 by the choice of parameters. □

Putting it All Together. Lemma 5.3.3 tells us that for any $\delta > 0$ and any function family $\mathcal{F}$:

$$\begin{aligned}
&\Pr[(\mathsf{Enc}_{h_1,h_2},\mathsf{Dec}_{h_1,h_2}) \text{ is not } (\mathcal{F},\varepsilon)\text{-super-non-malleable}] \\
&\leq \Pr[(\mathsf{Enc}_{h_1,h_2},\mathsf{Dec}_{h_1,h_2}) \text{ is not } (\mathcal{F},\delta,\varepsilon/4)\text{-bounded-malleable}] \qquad (5.13)\\
&\quad + \Pr[(\mathsf{LREnc}_{h_1},\mathsf{LRDec}_{h_1}) \text{ is not } (\mathcal{G}(\mathcal{F},h_2,\delta),\varepsilon/4)\text{-leakage-resilient}], \qquad (5.14)
\end{aligned}$$

where $\mathcal{G} = \mathcal{G}(\mathcal{F},h_2,\delta)$ is of size $|\mathcal{G}| = |\mathcal{F}|$ and consists of function with output size $\zeta = \lceil\log(1/\delta+2)\rceil$.

Let us set $\delta := (\varepsilon/8)(\log|\mathcal{F}|+k+\log(1/\rho)+3)^{-1}$. This ensures that the first requirement of Lemma 5.3.4 is satisfied with $\tau = \varepsilon/4$. We choose $t^* = O(\log|\mathcal{F}|+n+\log(1/\rho))$ such that $\log(1/\delta) \leq \log(1/\varepsilon)+\log(t^*)+O(1)$. Notice that the leakage amount of $\mathcal{G}$ is $\zeta = \lceil\log(1/\delta+2)\rceil \leq \log(1/\varepsilon)+\log(t^*)+O(1)$. With v_1, v_2 as in Theorem 5.3.1, we satisfy the remaining requirements of Lemma 5.3.4 (bounded-malleable codes) and Theorem 5.2.2 (leakage-resilient codes) to ensure that the probabilities (5.13), (5.14) are both bounded by $\rho/2$, which proves Theorem 5.3.1.

5.4 Non-malleable Key-derivation

In this section we introduce a new primitive, which we name non-malleable key derivation. Intuitively a function h is a non-malleable key derivation function if $h(x)$ is close to uniform even given the output of h applied to a related input $f(x)$, as long as $f(x) \neq x$.

Definition 5.4.1 (Non-malleable Key-derivation Function)**.** *Let $\mathcal{F}$ be any family of functions $f : \{0,1\}^n \to \{0,1\}^n$. We say that a function $h : \{0,1\}^n \to \{0,1\}^k$ is an $(\mathcal{F},\varepsilon)$-*non-malleable key derivation function *if for every $f \in \mathcal{F}$ we have $\mathbf{SD}(\mathsf{Real}_h(f);\ \mathsf{Sim}_h(f)) \leq \varepsilon$ where $\mathsf{Real}_h(f)$ and $\mathsf{Sim}_h(f)$ denote the output distributions of the corresponding experiments described in Fig. 5.1.*

Note that the above definition can be interpreted as a dual version of the definition of non-malleable extractors [DW09].[3] The theorem below states that by sampling a function h from a set

[3]The duality comes from the fact that the output of a non-malleable extractor is close to uniform even given a certain number of outputs computed with related seeds (whereas for non-malleable key derivation the seed is unchanged but the input can be altered).

$\mathcal{H}$ of $2t$-wise independent hash functions, we obtain a non-malleable key derivation function with overwhelming probability.

Experiment $\mathsf{Real}_h(f)$ vs. $\mathsf{Sim}_h(f)$

Experiment $\mathsf{Real}_h(f)$:
Sample $x \leftarrow U_n$.
If $f(x) = x$:
Output $(h(x), \mathsf{same}^\star)$.
Else
Output $(h(x), h(f(x)))$.

Experiment $\mathsf{Sim}_h(f)$:
Sample $x \leftarrow U_n$; $y \leftarrow U_k$
If $f(x) = x$:
Output $(y, \mathsf{same}^\star)$.
Else
Output $(y, h(f(x)))$.

Figure 5.1: Experiments defining a non-malleable key derivation function h

Theorem 5.4.2. *Let $\mathcal{H}$ be a $2t$-wise independent function family consisting of functions $h : \{0,1\}^n \to \{0,1\}^k$ and let $\mathcal{F}$ be some function family as above. Then with probability $1-\rho$ over the choice of a random $h \leftarrow \mathcal{H}$, the function h is an $(\mathcal{F}, \varepsilon)$-non-malleable key-derivation function as long as:*

$$n \geq 2k + \log(1/\varepsilon) + \log(t) + 3 \quad \textit{and} \quad t > \log(|\mathcal{F}|) + 2k + \log(1/\rho) + 5.$$

Proof. For any $h \in \mathcal{H}$ and $f \in \mathcal{F}$, define a function $h_f : \{0,1\}^n \to \{0,1\}^k \cup \mathsf{same}^\star$ such that if $f(x) = x$ then $h_f(x) = \mathsf{same}^\star$ otherwise $h_f(x) = h(f(x))$. Fix a function family $\mathcal{F}$. Now, taking probabilities (only) over the choice of h, let BAD be the event that h is not an $(\mathcal{F}, \varepsilon)$-non-malleable-key-derivation function. Then:

$$\begin{aligned}
\Pr[\text{BAD}] &= \Pr_{h \leftarrow \mathcal{H}}\Big[\exists f \in \mathcal{F} \;:\; \mathbf{SD}(\, \mathsf{Real}_h(f) \,,\, \mathsf{Sim}_h(f) \,) > \varepsilon\Big] \\
&= \Pr_{h \leftarrow \mathcal{H}}\Big[\exists f \in \mathcal{F} \;:\; \mathbf{SD}(\, (h(X), h_f(X)) \,,\, (U_k, h_f(X)) \,) > \varepsilon\Big] \\
&\leq \sum_{f \in \mathcal{F}} \Pr_{h \leftarrow \mathcal{H}}\Big[\sum_{y \in \{0,1\}^k} \sum_{y' \in \{0,1\}^k \cup \mathsf{same}^\star} \Big| \begin{array}{r} \Pr[h(X) = y \wedge h_f(X) = y'] \\ -\Pr[U_k = y \wedge h_f(X) = y'] \end{array} \Big| > 2\varepsilon\Big] \\
&\leq \sum_{f \in \mathcal{F}} \Pr_{h \leftarrow \mathcal{H}}\Big[\exists\, y \in \{0,1\}^k, y' \in \{0,1\}^k \cup \mathsf{same}^\star \;:\; \Big| \begin{array}{r} \Pr[h(X) = y \wedge h_f(X) = y'] \\ -\Pr[U_k = y \wedge h_f(X) = y'] \end{array} \Big| > 2^{-2k}\varepsilon\Big] \\
&\leq \sum_{f \in \mathcal{F}} \sum_{y \in \{0,1\}^k} \sum_{y' \in \{0,1\}^k \cup \mathsf{same}^\star} \Pr_{h \leftarrow \mathcal{H}}\Big[\Big| \begin{array}{r} \Pr[h(X) = y \wedge h_f(X) = y'] \\ -2^{-k}\Pr[h_f(X) = y'] \end{array} \Big| > 2^{-2k}\varepsilon\Big] \qquad (5.15)
\end{aligned}$$

Fix f, y, y'. For every $x \in \{0,1\}^n$, define a random variable C_x over the choice of $h \leftarrow \mathcal{H}$, such that

$$C_x = \begin{cases} 1 - 2^{-k} & \text{if } h(x) = y \wedge h_f(x) = y' \\ -2^{-k} & \text{if } h(x) \neq y \wedge h_f(x) = y' \\ 0 & \text{otherwise.} \end{cases}$$

Notice that each C_x is 0 on expectation. However, the random variables C_x are not even pairwise independent.[4] In Section 5.4.1, we prove the following lemma about the variables C_x.

[4] For example if $f(x) = f(x')$ and $C_x = 0$ then $C_{x'} = 0$ as well.

Lemma 5.4.3. *There exists a partitioning of $\{0,1\}^n$ into four disjoint subsets $\{A_j\}_{j=1}^4$, such that for any $A > 0$ and for all $j = 1,\ldots,4$:*

$$\Pr\left[\left|\sum_{x\in A_j} C_x\right| > A\right] < K_t\left(\frac{t}{A}\right)^t,$$

where $K_t \leq 8$.

Continuing from Eq. (5.15), we get:

$$\begin{aligned}
&\Pr_{h\leftarrow\mathcal{H}}\left[\left|\Pr[h(X) = y \wedge h_f(X) = y'] - 2^{-k}\Pr[h_f(X) = y']\right| > 2^{-2k}\varepsilon\right] \\
&= \Pr_{h\leftarrow\mathcal{H}}\left[\left|\sum_{x\in\{0,1\}^n} C_x\right| > 2^{n-2k}\varepsilon\right] \qquad (5.16)\\
&\leq \sum_{j=1}^{4}\Pr_{h\leftarrow\mathcal{H}}\left[\left|\sum_{x\in A_j} C_x\right| > 2^{n-2k-2}\varepsilon\right] < 4K_t\left(\frac{t}{2^{n-2k-2}\varepsilon}\right)^t. \qquad (5.17)
\end{aligned}$$

Eq. (5.16) follows from the definitions of the variables C_x and Eq. (5.17) follows by applying Lemma 5.4.3 to the sum. Combining Eq. (5.15) and Eq. (5.17), we get $\Pr[\text{BAD}] < |\mathcal{F}|2^{2k}\left[4K_t\left(\frac{t}{2^{n-2k-2}\varepsilon}\right)^t\right]$. In particular, it holds that $\Pr[\text{BAD}] \leq \rho$ as long as:

$$n \geq 2k + \log(1/\varepsilon) + \log(t) + 3 \quad \text{and} \quad t > \log(|\mathcal{F}|) + 2k + \log(1/\rho) + 5.$$

□

5.4.1 Proof of Lemma 5.4.3

It will be convenient to represent the tampering function f as a graph. In particular, we define a directed graph $G = (V, E)$ with vertices $V = \{0,1\}^n$ and edges $E = \{(x, x') \;:\; x' = f(x)\}$. Each vertex has out-degree 1, and the graph may contain self-loops. We choose a random $h \leftarrow \mathcal{H}$, and label each vertex x with a value $h(x)$. Each random variable C_x depends only on the labels of the vertices x and $f(x)$. As we mentioned, the random variables C_x are not independent. However, we show how to partition the variables into 4 sets such that, within each set, the variables are either independent or "as good as independent".

First we partition the values $x \in \{0,1\}^n$ into two sets S_f (self-loop) and $\overline{S}_f$ (no self-loop) such that $x \in S_f$ if and only if $f(x) = x$. We prove the following lemma.

Lemma 5.4.4. *Let $t > 2$ be an even integer. Consider the set of random variables $\{C_x\}_{x\in S_f}$. Then for any $A > 0$,*

$$\Pr\left[\left|\sum_{x\in S_f} C_x\right| > A\right] < K_t\left(\frac{t}{A}\right)^t.$$

where $K_t \leq 8$.

Proof. First consider the case when $y' \neq \mathsf{same}^\star$. In this case, for all $x \in S_f$, we have that $C_x = 0$ for any choice of h and thus the statement is verified.

On the other hand when $y' = \mathsf{same}^\star$ the variables $\{C_x\}_{x\in S_f}$ are t-wise independent. Hence, the statement follows directly from Lemma 2.2.3. □

Now, consider the subgraph $G(\overline{S}_f)$ induced by $\overline{S}_f \subseteq V$. In $\overline{S}_f$, there is no self-loop and each vertex has at most one outgoing edge.[5] We prove the following lemma about $G(\overline{S}_f)$.

Lemma 5.4.5. *The graph $G(\overline{S}_f)$ is 3-colorable.*

Proof. We prove the lemma by constructing an algorithm to color $G(\overline{S}_f)$ with 3 colors (say R, B and W). We recall that, by definition, in graph $G(\overline{S}_f)$ each vertex has at most one outgoing edge. Without loss of generality we assume that each vertex in the graph has exactly one outgoing edge. It is easy to see that this is the worst-case for graph coloring, as more edges might enforce to use more colors. This fact we exploit heavily in this proof. The algorithm is described as follows:

1. Pick up a vertex randomly and color it by some arbitrary color, say R.
2. Follow the unique outgoing edge to color the next vertex by a different color, say B.
3. Continue to color vertices alternately following unique outgoing edges successively, with the following check at each step. To color a vertex check if its successor vertex is uncolored. If it is, then color the vertex with R or B, whichever appropriate. Otherwise, there may be a situation such that both the predecessor and the successor of a vertex are colored with different colors, which enforces to color that vertex with the third color, namely W. After that, pick up another uncolored vertex randomly and repeat from the beginning of step 3. If no such vertex is left then stop.

Note that the algorithm always terminates; this is because whenever we encounter a loop we choose a new vertex from the set of uncolored vertices, and there are only finitely many vertices.

To prove the correctness, we observe the following invariant is maintained throughout. According to the algorithm, once we are done with coloring some vertex v we move to color its unique uncolored successor v'. Now we claim that all the other predecessors v'' of v' which are different from v must be uncolored. To see this, we assume by contradiction that there is some v'' different from v which is colored. Now, since v' is the unique successor of v'', following the algorithm we must have colored v' immediately after we color v''. Therefore, assuming v'' is colored leads to the fact that v' is already colored which is a contradiction. So, the only neighbor of v' which might be colored is its unique successor $\hat{v}$. Note that this may enforce us to color v' differently from both its predecessor v and its unique successor $\hat{v}$ with the third color, which we are allowed to do. We conclude that the algorithm imposes a proper 3-coloring on $G(\overline{S}_f)$. □

By the above lemma we can partition the set $\overline{S}_f \subseteq \{0,1\}^n$ into three disjoint subsets A_1, A_2, A_3 (i.e., the three colors) such that for $j \in \{1,2,3\}$, and all $x, x' \in A_j$, $f(x) \neq x'$. We consider the set of variables $\{C_x\}_{x \in A_j}$. Intuitively, the above partitioning removed some "bad" dependence between variables C_x and $C_{f(x)}$ by seeparating them into different subsets. However, even within any set A_j, the variables are not t-wise independent. For example if $f(x) = f(x')$ then C_x and $C_{x'}$ may be in the same set A_j but are correlated. Nevertheless, we prove that the correlation within each set goes in the "right" direction and allows us to bound the sum. In particular, we prove the following lemma about the set of variables $\{C_x\}_{x \in A_j}$; note that Lemma 5.4.4 and Lemma 5.4.6 imply Lemma 5.4.3 by letting $S_f = A_4$.

[5] Note that it might happen that some outgoing edge lands in S_f. In that case the induced subgraph $G(\overline{S}_f)$ would exclude that edge.

Lemma 5.4.6. *Let $t > 2$ be an even integer. Consider the set of random variables $\{C_x\}_{x \in A_j}$ for some $j \in \{1,2,3\}$. Denote their sum by $\Sigma_j = \sum_{x \in A_j} C_x$. Then for any $A > 0$ and for all $j \in \{1,2,3\}$,*

$$\Pr[\Sigma_j > A] < K_t \left(\frac{t}{A}\right)^t,$$

where $K_t \leq 8$.

Proof. Fix some $j \in \{1,2,3\}$. First consider the case when $y' = \mathsf{same}^\star$. In this case, for all $x \in A_j$, we have that $C_x = 0$ for any choice of h. Thus, $\Pr[\Sigma_j > A] = 0$ and the statement of the lemma is verified. For the remaining of this proof we will assume that $y' \neq \mathsf{same}^\star$.

Let $|A_j| = m$ for some $m \in [2^n]$, and denote the variables $\{C_x\}_{x \in A_j}$ by $\{C_{x_1}, \ldots, C_{x_m}\}$. For each variable C_{x_i} ($i \in [m]$) we can define the corresponding conditional random variable $C_{x_i} | (h(f(x_i)) = y')$ as follows:

$$\widetilde{C}_i := C_{x_i} | (h(f(x_i)) = y') = \begin{cases} 1-p & \text{with probability } \Pr_h[h(x_i) = y] = p \\ -p & \text{with probability } \Pr_h[h(x_i) \neq y] = 1-p \end{cases}$$

where $p = 2^{-k}$. Note that the variables $\{\widetilde{C}_i\}_{i=1}^m$ satisfy the following properties: (i) They are t-wise independent; (ii) Each $\widetilde{C}_i$ is 0 on expectation and hence $\mathbf{E}[\widetilde{C}] = 0$, where $\widetilde{C} = \sum_{i=1}^m \widetilde{C}_i$. We can thus apply Lemma 2.2.4 to the variables $\widetilde{C}_1, \widetilde{C}_2, \cdots, \widetilde{C}_m$ with $\mu = \mathbf{E}[\widetilde{C}] = 0$ to get the following:

$$\mathbf{E}[\widetilde{C}^t] \leq K_t \cdot t^t, \tag{5.18}$$

where $K_t \leq 8$.

The next claim shows that $\mathbf{E}[\Sigma_j^t] < \mathbf{E}[\widetilde{C}^t]$. Note that from Eq. (5.18) and Claim 5.4.7 we get that for all $j \in \{1,2,3\}$, $\mathbf{E}[\Sigma_j^t] < K_t \cdot t^t$; applying Markov's inequality we obtain that for any $A > 0$, $\Pr[\Sigma_j > A] < K_t \cdot \left(\frac{t}{A}\right)^t$. This concludes the proof of Lemma 5.4.6.

Claim 5.4.7. $\mathbf{E}[\Sigma_j^t] < \mathbf{E}[\widetilde{C}^t]$.

Proof. For any m variables $Y_1, Y_2, \ldots, Y_m$, we have that $(Y_1 + Y_2 + \ldots + Y_m)^t$ is a polynomial of degree t. Therefore, to prove the above claim, using linearity of expectation, it is sufficient to show that the expectation of each monomial in the right hand side of the inequality is individually greater than the expectation of each monomial in the left hand side. From both sides, we take a term of the form[6] $\mathbf{E}[\prod_{i=1}^l Y_{a_i}^{e_i}]$ where $e_i, l \in [t]$, $a_i \in [m]$ and $\sum_{i=1}^l e_i = t$.

Note that the variables $C_{x_1}, C_{x_2}, \ldots, C_{x_m}$ are not independent. However, since within a set A_j there is no pair x, x' such that $f(x) = x'$, the only possibility of dependence among the variables arises from the event: $f(x) = f(x')$. We can further partition the variables in the product $\prod_{i=1}^l C_{x_{a_i}}^{e_i}$ into sub-products $\Pi_b = \prod_{i \in S_b} C_{x_{a_i}}^{e_i}$ for disjoint subsets $S_b \subseteq [l]$ where $b \in \{1, 2, \ldots, l'\}$ and $1 \leq l' \leq l$, such that, in each sub-product Π_b, for all $i \in S_b$, $f(x_{a_i}) = x'_b$ (for some $x'_b \in A_{j'} \cup S_f$, with $j' \neq j$). Now, since (i) by definition, dependent variables are within the same sub-product and (ii) the hash function h is $2t$-wise independent, the sub-products $\{\Pi_b\}_{b=1}^{l'}$ are mutually independent.

Next, we compute the expectation $\mathbf{E}[\Pi_b]$:

[6] We ignore the multiplicative constant as it is the same on both the sides.

$$\mathbf{E}[\Pi_b] = \mathbf{E}[\Pi_b | h(x'_b) = y'] \Pr[h(x'_b) = y'] + \mathbf{E}[\Pi_b | h(x'_b) \neq y'] \Pr[h(x'_b) \neq y']$$
$$= \mathbf{E}\left[\prod_{i \in S_b} C_{x_{a_i}}^{e_i} | h(x'_b) = y'\right] \Pr[h(x'_b) = y'] = p \cdot \mathbf{E}\left[\prod_{i \in S_b} \widetilde{C}_{a_i}^{e_i}\right]. \quad (5.19)$$

The second equality of Eq. (5.19) follows from the fact that $C_x = 0$ whenever $h(f(x)) \neq y'$; the third equality follows from the definition of $\widetilde{C}_{a_i}$. We now compute the expectation of the product $\prod_{i=1}^{l} C_{x_{a_i}}^{e_i}$, as follows:

$$\mathbf{E}\left[\prod_{i=1}^{l} C_{x_{a_i}}^{e_i}\right] = \prod_{b=1}^{l'} \mathbf{E}[\Pi_b] = \prod_{b=1}^{l'} \left(p \cdot \mathbf{E}\left[\prod_{i \in S_b} \widetilde{C}_{a_i}^{e_i}\right]\right) = p^{l'} \mathbf{E}\left[\prod_{i=1}^{l} (\widetilde{C}_{a_i}^{e_i})\right] < \mathbf{E}\left[\prod_{i=1}^{l} (\widetilde{C}_{a_i}^{e_i})\right]. \quad (5.20)$$

The first equality of Eq. (5.20) follows from the fact that the sub-products Π_b are mutually independent, the second equality follows from Eq. (5.19) and the third equality from the t-wise independence of the variables $\widetilde{C}_{a_i}$. This concludes the proof of the claim. □

□

Optimal Rate of Non-Malleable Key-Derivation. We can define the rate of a key derivation function $h : \{0,1\}^n \to \{0,1\}^k$ as the ratio k/n. Notice that our construction achieves rate arbitrary close to $1/2$. We claim that this is *optimal* for non-malleable key derivation. To see this, consider a tampering function $f : \{0,1\}^n \to \{0,1\}^n$ which is a permutation and never identity: $f(x) \neq x$. In this case the joint distribution $(h(X), h(f(X)))$ is ε-close to $(U_k, h(f(X)))$ which is ε-close to the distribution (U_k, U'_k) consisting of $2k$ random bits. Since all of the randomness in $(h(X), h(f(X)))$ comes from X, this means that X must contain at least $2k$ bits of randomness, meaning that $n > 2k$.

5.5 A Tamper-resilient Stream Cipher using NMKD

In this section we provide a construction of tamper-resilient stream cipher using non-malleable key-derivation. In contrast to the usual application of non-malleable codes into tamper-resilience, this result however follows a different approach (Approach-1 as described in Chapter 1) as here we construct a *specific* tamper-resilient scheme rather than building a *generic* compiler.

Throughout this section, we write $\mathbf{CD}^{\mathsf{A}}(X_1; X_2) = |\Pr[\mathsf{A}(X_1) = 1] - \Pr[\mathsf{A}(X_2) = 1]|$ for the advantage of a PPT adversary A in distinguishing two random variables X_1 and X_2 (defined over some space $\mathcal{X}$).

A stream cipher SC takes as input an initial key $s_0 \in \{0,1\}^n$ and is executed in rounds. For $i \in [q]$, it computes in the i-th round $(s_i, x_i) = \mathsf{SC}(s_{i-1})$ where $x_i \in \{0,1\}^v$ is given to the adversary. We write $\mathsf{SC}^i(s_0) = (s_i, x_i)$ to denote the i-th output of SC, when run for i rounds with initial key s_0. We also write (s_i, x_i) for the corresponding random variables, generated by sampling $s_0 \leftarrow \{0,1\}^n$ and running the stream cipher on this key.

The standard security requirement says that even given $x_1, \ldots, x_{i-1}$, the adversary cannot distinguish between the next block x_i and a uniform random sample $u \leftarrow U_v$. We strengthen this security requirement and allow the adversary to tamper additionally with the secret state of SC. Of course, if an adversary can apply an arbitrary function f_i to the state, he may just overwrite it with a known key, which clearly contradicts the pseudorandomness of x_{i+1}. Notice that such

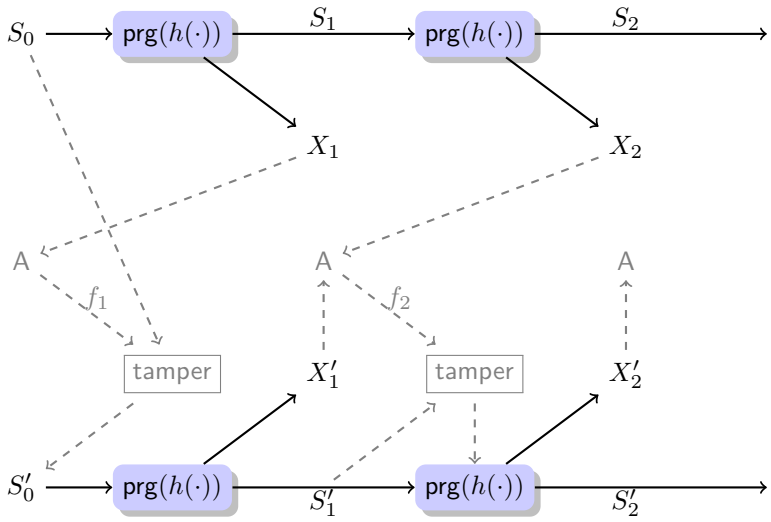

Figure 5.2: Construction of a tamper-resilient stream cipher. The regular evaluation is shown in black (at the top line), the attack related part is shown in gray with dashed lines, and the corresponding tampered evaluation is shown in black (at the bottom line). Recall that the choice of the tampering function is adaptive from the a-priori fixed set $\mathcal{F}$ that is tolerated by the underlying non-malleable key-derivation.

an "overwriting" makes the cipher useless and does not help the adversary to, e.g., decrypt ciphertexts that were encrypted with $\mathsf{SC}(s_0)$. To model tamper resilience of SC we consider an adversary A that obtains both the correctly evaluated outputs $\mathbf{x} = (x_1, \ldots, x_i)$ of $\mathsf{SC}(S_0)$ and the faulty outputs $\mathbf{x}' = (x'_1, \ldots, x'_i)$ where $(s'_{i+1}, x'_{i+1}) = \mathsf{SC}(f_{i+1}(s'_i))$. We say that SC is secure against tampering attacks if even given the outputs $(\mathbf{x}, \mathbf{x}')$ the random variable x_{i+1} is computationally close to uniform.

More formally, consider the following experiment (running with a PPT adversary A and a family of functions $\mathcal{F}$) and denote its output by $\mathsf{out}_{\mathsf{A},\mathcal{F}}(q)$:

1. Sample $s_0 \leftarrow U_n$ and let $s'_0 := s_0$.
2. The adversary repeats the following for each $i \in [q]$:
   (a) *Compute untampered output:* Compute $(x_i, s_i) = \mathsf{SC}(s_{i-1})$ and give x_i to A.
   (b) *Compute tampered output:* Receive $f_i \in \mathcal{F}$ from A. Compute $(x'_i, s'_i) = \mathsf{SC}(f_i(s'_{i-1}))$ and give x'_i to A.
3. The output of the experiment is defined as $\mathbf{x} = (x_1, \ldots, x_q)$ and $\mathbf{x}' = (x'_1, \ldots, x'_q)$.

Definition 5.5.1. *Let* $\mathsf{SC} : \{0,1\}^n \to \{0,1\}^v \times \{0,1\}^n$. *We say that* SC *is continuous* $(\mathcal{F}, \varepsilon)$*-tamper-resilient if for all PPT adversaries* A

$$\mathbf{CD}^{\mathsf{A}}((x_{q+1}, \mathsf{out}_{\mathsf{A},\mathcal{F}}(q)); (U_v, \mathsf{out}_{\mathsf{A},\mathcal{F}}(q))) \leq \varepsilon,$$

where $\mathsf{out}_{\mathsf{A},\mathcal{F}}(q)$ *is defined as above,* $x_{q+1} := \mathsf{SC}^q(s_0)$, *and* SC^q *denotes* q *executions of the stream cipher.*

The construction. Recall that a pseudorandom generator (PRG) is a function $\mathsf{prg}:\{0,1\}^{k_1}\to\{0,1\}^{k_2}$; we say that prg is ε-secure if for all PPT adversaries A we have $\mathbf{CD}^{\mathsf{A}}(\mathsf{prg}(U_{k_1});U_{k_2})\leq\varepsilon$.

Consider the following construction of a stream cipher $\mathsf{SC}^{h,\mathsf{prg}}$, based on a PRG $\mathsf{prg}:\{0,1\}^{k}\to\{0,1\}^{n+v}$ and a non-malleable key-derivation function $h:\{0,1\}^{n}\to\{0,1\}^{k}$ (see also Figure 5.2). At the beginning a description of the function h and of the pseudorandom generator prg are output as public parameters. Then a key $s_0\leftarrow\{0,1\}^n$ is sampled uniformly at random and for each $i\in[q]$ we define the output of the stream cipher at round i as $(s_i,x_i):=\mathsf{prg}(h(s_{i-1}))$.

Theorem 5.5.2. *Assume that* prg *is an* $\varepsilon_{\mathsf{prg}}$*-secure pseudorandom generator and that* h *is an* $(\mathcal{F},\varepsilon)$*-non-malleable key-derivation function. Then the stream cipher* $\mathsf{SC}^{h,\mathsf{prg}}$ *defined above is continuous* $(\mathcal{F},\varepsilon')$*-tamper-resilient, where* $\varepsilon'\leq(2q+1)\varepsilon+2q\varepsilon_{\mathsf{prg}}$.

Proof. Consider the distribution $D_{f,h}$ over $\{0,1\}^k\cup\mathsf{same}^\star$, which samples $s\leftarrow U_n$ and outputs $\mathsf{same}^\star$ if $f(s)=s$ and $h(f(s))$ otherwise.

Before giving the formal proof, let us discuss some intuition. We show the desired computational indistinguishability through several intermediate hybrid games. Notice that the interaction of the challenger and the adversary in the definition, can be viewed as if there were two chains of values: (i) an "untampered" chain, similar to the standard stream cipher game (c.f. Step 2a in the definition); and (ii) a "tampered chain", where the adversary can tamper with each input of the function h (c.f. Step 2b in the definition). In the i-th hybrid we replace the i-th output of h in the untampered chain, by a uniform random value. In the tampered chain, if the adversary is yet to tamper, then we replace the output of h with a random sample from the distribution $D_{f_i,h}$. In case $D_{f_i,h}$ returns some value which is not $\mathsf{same}^\star$, i.e. the adversary tampered, then we stop sampling further and continue to simulate the output from that point on using the last sampled value. Non-malleability of the key-derivation function h guarantees that, once the adversary tampers, the modified value reveals almost no information about the "extracted" key. Notice that, if before entering the i-th round the adversary had already tampered, then there is no difference between the i-th and $(i+1)$-th hybrids in the tampered chain. We now proceed with the formal proof.

Define the following hybrid games for all $i\in\{0,\ldots,q\}$ and all $b\in\{0,1\}$.

$\mathsf{Game}_i(b)$:

1. Sample $s_0\leftarrow U_n$ and let $s'_0:=s_0$. Set initial mode to *normal mode*.
2. For all $j=1,\ldots,i$ do the following.
   (a) *Compute untampered output:* Sample $\kappa_j\leftarrow U_k$. Compute $(x_j,s_j)\leftarrow\mathsf{prg}(\kappa_j)$ and give x_j to A.
   (b) *Compute tampered output:* Receive $f_j\in\mathcal{F}$ from A. Depending on the current mode behave in one of the following ways:
   - *Normal Mode*: Sample $\tilde{\kappa_j}\leftarrow D_{f_j,h}$. If $\tilde{\kappa_j}=\mathsf{same}^\star$, then set $(x'_j,s'_j):=(x_j,s_j)$ and give x'_j to A. Else, if $\tilde{\kappa_j}\neq\mathsf{same}^\star$, then set the current mode to *overwritten mode*, set $(x'_j,s'_j):=\mathsf{prg}(\tilde{\kappa_j})$ and give x'_j to A.
   - *Overwritten Mode*: Compute $(x'_j,s'_j)\leftarrow\mathsf{prg}(h(f_j(s'_{j-1})))$, give x'_j to A.
3. For all $j=i+1,\ldots,q$ do the following.
   (a) *Compute untampered output:* Compute $(x_j,s_j)\leftarrow\mathsf{prg}(h(s_{j-1}))$ and give x_j to A.
   (b) *Compute tampered output:* Receive $f_j\in\mathcal{F}$ from A. Compute $(x'_j,s'_j)\leftarrow\mathsf{prg}(h(f_j(s'_{j-1})))$ and give x'_j to A.

4. *Challenge Phase.* If $b=0$, then set $(x_{q+1}, s_{q+1}) \leftarrow \mathsf{prg}(h(s_q))$; otherwise, sample $x_{q+1} \leftarrow U_v$. The output of the experiment is defined as $\mathsf{Game}_i(b) = (x_{q+1}, \mathbf{x}, \mathbf{x}')$ where $\mathbf{x} = (x_1, \ldots, x_q)$ and $\mathbf{x}' = (x'_1, \ldots, x'_q)$.

Notice that the output distribution of $\mathsf{Game}_0(b)$ is either equal to $(x_{q+1}, \mathsf{out}_{\mathsf{A},\mathcal{F}}(q))$ (in case $b=0$) or to $(u \leftarrow U_v, \mathsf{out}_{\mathsf{A},\mathcal{F}}(q))$ (in case $b=1$); thus our goal is to bound $\mathbf{CD}^{\mathsf{A}}(\mathsf{Game}_0(0); \mathsf{Game}_0(1))$. By using the triangle inequality, we write:

$$\begin{aligned}
\mathbf{CD}^{\mathsf{A}}(\mathsf{Game}_0(0); \mathsf{Game}_0(1)) &\leq \mathbf{CD}^{\mathsf{A}}(\mathsf{Game}_0(0); \mathsf{Game}_1(0)) + \mathbf{CD}^{\mathsf{A}}(\mathsf{Game}_1(0); \mathsf{Game}_q(0)) \\
&\quad + \mathbf{CD}^{\mathsf{A}}(\mathsf{Game}_q(0); \mathsf{Game}_q(1)) + \mathbf{CD}^{\mathsf{A}}(\mathsf{Game}_q(1); \mathsf{Game}_1(1)) \\
&\quad + \mathbf{CD}^{\mathsf{A}}(\mathsf{Game}_1(1); \mathsf{Game}_0(1)) \\
&\leq 3\varepsilon + 2\varepsilon_{\mathsf{prg}} + \sum_{b\in\{0,1\}} \sum_{i=1}^{q-1} \mathbf{CD}^{\mathsf{A}}(\mathsf{Game}_i(b); \mathsf{Game}_{i+1}(b)) \qquad (5.21) \\
&\leq (2q+1)\varepsilon + 2q\varepsilon_{\mathsf{prg}}. \qquad (5.22)
\end{aligned}$$

Eq. (5.21) follows by applying Claim 5.5.3 and Claim 5.5.5 whereas Eq. (5.22) follows from Claim 5.5.4. This concludes the proof of the theorem except the proofs of the claims which are given below.

Claim 5.5.3. $\mathbf{CD}^{\mathsf{A}}(\mathsf{Game}_q(0); (\mathsf{Game}_q(1)) \leq \varepsilon + 2\varepsilon_{\mathsf{prg}}$.

Proof. Consider the following modified games.

Game'_q: First run $\mathsf{Game}_q(0)$ until Step 3. In Step 4 sample $s_q \leftarrow U_n$ and compute $(x_{q+1}, s_{q+1}) \leftarrow \mathsf{prg}(h(s_q))$. Output $\mathsf{Game}'_q = (x_{q+1}, \mathbf{x}, \mathbf{x}')$.

Game''_q: First run $\mathsf{Game}_q(1)$ until Step 3. In Step 4 sample $\kappa_{q+1} \leftarrow U_k$ and compute $(x_{q+1}, s_{q+1}) \leftarrow \mathsf{prg}(\kappa_{q+1})$. Output $\mathsf{out}''_q = (x_{q+1}, \mathbf{x}, \mathbf{x}')$.

We notice that the only difference between $\mathsf{Game}_q(0)$ and Game'_q is in the challenge phase: in the former s_q is computed as the output of $\mathsf{prg}(\kappa_{q-1})$ for some uniformly random $\kappa_{q-1} \leftarrow U_k$, whereas in the latter s_q is uniformly random. Clearly, $\mathbf{CD}^{\mathsf{A}}(\mathsf{Game}_q(0); \mathsf{Game}'_q) \leq \varepsilon_{\mathsf{prg}}$. By a similar argument, $\mathbf{CD}^{\mathsf{A}}(\mathsf{Game}''_q; \mathsf{Game}_q(1)) \leq \varepsilon_{\mathsf{prg}}$.

Let us now compare the output distribution of Game'_q and Game''_q. Again, the only difference is in the challenge phase: in the former the input of prg is computed as $h(s_q)$ for some uniformly random $s_q \leftarrow U_n$, whereas in the latter the input of prg is sampled uniformly as $\kappa_{q+1} \leftarrow U_k$. Now the fact that h is non-malleable clearly implies that the output of h on a random input is close to uniform.[7] This shows that $\mathbf{CD}^{\mathsf{A}}(\mathsf{Game}'_q; \mathsf{Game}''_q) \leq \varepsilon$.

Combining the above arguments, we conclude

$$\begin{aligned}
\mathbf{CD}^{\mathsf{A}}(\mathsf{Game}_q(0); \mathsf{Game}_q(1)) &\leq \mathbf{CD}^{\mathsf{A}}(\mathsf{Game}_q(0); \mathsf{Game}'_q) \\
&\quad + \mathbf{CD}^{\mathsf{A}}(\mathsf{Game}'_q; \mathsf{Game}''_q) + \mathbf{CD}^{\mathsf{A}}(\mathsf{Game}''_q; \mathsf{Game}_q(1)) \\
&\leq \varepsilon + 2\varepsilon_{\mathsf{prg}},
\end{aligned}$$

as desired. □

[7] In this step we only require that the output of h is close to uniform, which is obviously a weaker property than non-malleability.

Claim 5.5.4. *For all $b \in \{0,1\}$ and all $i \in [q-1]$,* $\mathbf{CD}^{\mathsf{A}}(\mathsf{Game}_i(b); \mathsf{Game}_{i+1}(b)) \leq \varepsilon + \varepsilon_{\mathsf{prg}}$.

Proof. Fix some $b \in \{0,1\}$ and $i \in [q-1]$. Consider the following modified game.

$\mathsf{Game}'_i(b)$: First run $\mathsf{Game}_i(b)$ until Step 2. In Step 3 first sample $s_i \leftarrow U_n$ and then do the following for all $j = i+1, \ldots, q$:

(a) *Compute untampered output:* Compute $(x_j, s_j) \leftarrow \mathsf{prg}(h(s_{j-1}))$ and give x_j to A.

(b) *Compute tampered output:* Receive $f_j \in \mathcal{F}$ from A. In case the execution is still in normal mode, set $s'_i := s_i$. In the next step, irrespective of the mode, compute $(x'_j, s'_j) \leftarrow \mathsf{prg}(h(f_j(s'_{j-1})))$ and give x'_j to A.

Step 4 is defined exactly as in $\mathsf{Game}_i(b)$.

We notice that the only difference between $\mathsf{Game}_i(b)$ and $\mathsf{Game}'_i(b)$ is in Step 3: in the i-th round the last output of prg (i.e., s_i) is replaced by a uniform random value in the latter. Note that in case the execution has already entered the overwritten mode, this change does not affect the tampered output; on the other hand, in case the execution is still in normal mode, then in the tampered output the i-th output of prg is replaced by the above uniform value s_i. Thus, clearly, $\mathbf{CD}^{\mathsf{A}}(\mathsf{Game}'_i(b); \mathsf{Game}_i(b)) \leq \varepsilon_{\mathsf{prg}}$.

Let us now compare the output distribution of $\mathsf{Game}'_i(b)$ and $\mathsf{Game}_{i+1}(b)$. Again the only differences are as follows:

- Regarding the untampered output, the $(i+1)$-th output of h is replaced by a uniform random value κ_{i+1} in $\mathsf{Game}_{i+1}(b)$ whereas in $\mathsf{Game}'_i(b)$ it is computed as $h(s_i)$ for some uniform s_i.
- Regarding the tampered output, there is no difference in case the execution has already entered the overwritten mode. In case the execution is still normal mode, then $\mathsf{Game}_{i+1}(b)$ samples $\tilde{\kappa}_{i+1} \leftarrow D_{f_{i+1},h}$. If $\tilde{\kappa}_{i+1}$ is not $\mathsf{same}^\star$, then the output will be computed by running prg on $\tilde{\kappa}_{i+1}$; else, the output is just copied from the untampered output.

We observe that $\mathsf{Game}'_i(b)$ can be computed as a deterministic function of $\mathsf{Real}_h(f_{i+1})$ and $\mathsf{Game}_{i+1}(b)$ as a deterministic function of $\mathsf{Sim}_h(f_{i+1})$ (c.f. Figure 5.1). This shows that $\mathbf{CD}^{\mathsf{A}}(\mathsf{Game}'_i(b); \mathsf{Game}_{i+1}(b)) \leq \varepsilon$.

Combining the above arguments, we conclude that for any $b \in \{0,1\}$ and all $i \in [q-1]$

$$\begin{aligned}\mathbf{CD}^{\mathsf{A}}(\mathsf{Game}_i(b); \mathsf{Game}_{i+1}(b)) &\leq \mathbf{CD}^{\mathsf{A}}(\mathsf{Game}_i(b); \mathsf{Game}'_i(b)) + \mathbf{CD}^{\mathsf{A}}(\mathsf{Game}'_i(b); \mathsf{Game}_{i+1}(b)) \\ &\leq \varepsilon + \varepsilon_{\mathsf{prg}},\end{aligned}$$

as desired. □

Claim 5.5.5. *For all $b \in \{0,1\}$,* $\mathbf{CD}^{\mathsf{A}}(\mathsf{Game}_0(b); \mathsf{Game}_1(b)) \leq \varepsilon$.

Proof. The statement can be shown in a similar way as in the proof of Claim 5.5.4, with the only difference that in this case the first input to h is a uniformly random value s_0 (rather that some "pseudorandom" value), and thus the modified game $\mathsf{Game}'_0(b)$ collapses to $\mathsf{Game}_0(b)$. □

□

5.6 One-time Tamper-resilience via NMKD

Similar to non-malleable codes we show that a non-malleable key-derivation function can be use to protect any stateless functionality against (memory-only) tampering attacks. We consider two main differences with respect to the setting considered in [DPW10]: (i) the original functionality is stateless and works with a uniformly chosen state; (ii) the attacker can only tamper once.

A stateless functionality $\mathsf{G}(\mathsf{st},\cdot)$ consists of a public (possibly randomized) function $\mathsf{G}:\{0,1\}^k \times \{0,1\}^u \to \{0,1\}^v$ and a secret initial state $\mathsf{st} \in \{0,1\}^k$. Whenever the state st is chosen uniformly at random from $\{0,1\}^k$, we say that the functionality is *regular*. The main idea is to transform $\mathsf{G}(\mathsf{st},\cdot)$ into some "hardened" functionality $\mathsf{G}(s,\cdot)$ via a non-malleable key-derivation function h.

Definition 5.6.1 (Hardened functionality)**.** *Let $h:\{0,1\}^n \to \{0,1\}^k$ be a function. Let $\mathsf{G}:\{0,1\}^k \times \{0,1\}^u \to \{0,1\}^v$ be any* stateless, regular *functionality with k-bit state. We define the hardened functionality $\mathsf{G}^h:\{0,1\}^n \times \{0,1\}^u \to \{0,1\}^v$ to be the functionality that takes as input $(s,x) \in \{0,1\}^n \times \{0,1\}^u$ and outputs $y \leftarrow \mathsf{G}(h(s),x)$.*

Security of G^h is defined via the comparison of a real and an ideal experiment. The real experiment features an adversary A interacting with G^h; the adversary is allowed to honestly run the functionality on any chosen input, but also to tamper with the original state and interact with the modified functionality. The ideal experiment features a simulator S; the simulator is given black-box access to the original functionality G and to the adversary A, but is *not* allowed any tampering query. The two experiments are formally described below.

Experiment $\mathsf{Real}^{\mathsf{G}^h(s,\cdot)}_{\mathsf{A},\mathcal{F}}$**.** A value $s \leftarrow \{0,1\}^n$ is chosen uniformly at random. Then A can issue the following commands (in any order):

- $\langle \texttt{Eval}, x\rangle$: In response to an evaluation query, return $y \leftarrow \mathsf{G}(h(s),x)$. This command can be run a polynomial number of times.
- $\langle \texttt{Tamper}, f\rangle$: Upon input $f:\{0,1\}^n \to \{0,1\}^n$, with $f \in \mathcal{F}$, replace s by $f(s)$. This command can be run a single time.

The output of the experiment is defined as

$$\mathsf{Real}^{\mathsf{G}^h(s,\cdot)}_{\mathsf{A},F} = ((x_1,y_1),(x_2,y_2),\dots)).$$

Experiment $\mathsf{Ideal}^{\mathsf{G}(\mathsf{st},\cdot)}_{\mathsf{S}}$**.** A value $\mathsf{st} \leftarrow \{0,1\}^k$ is chosen uniformly at random. The simulator is given black-box access to the functionality $\mathsf{G}(\mathsf{st},\cdot)$ and the adversary A. The output of the experiment is defined as

$$\mathsf{Ideal}^{\mathsf{G}(\mathsf{st},\cdot)}_{\mathsf{S}} = ((x_1,y_1),(x_2,y_2),\dots),$$

where (x_j,y_j) are the input/output tuples simulated by S.

Definition 5.6.2 (One-time tamper simulatability)**.** *Let $h:\{0,1\}^n \to \{0,1\}^k$ be a function and consider a* stateless, regular *functionality G. Denote with G^h the hardened functionality corresponding to G. We say that h is one-time $(\mathcal{F},\varepsilon)$-tamper simulatable for G, if for all PPT adversaries A there exists a PPT simulator S such that for any initial state st,*

$$\mathsf{Real}^{\mathsf{G}^h(s,\cdot)}_{\mathsf{A},\mathcal{F}} \approx \mathsf{Ideal}^{\mathsf{G}(\mathsf{st},\cdot)}_{\mathsf{S}},$$

where $\approx$ refers either to statistical or computational indistinguishability.

The theorem below states that whenever h is a non-malleable key-derivation function, then it is also one-time tamper simulatable.

Theorem 5.6.3. *Consider a stateless, regular functionality* G*. Let* h *be an* $(\mathcal{F},\varepsilon)$*-non-malleable key-derivation function, then* h *is one-time* $(\mathcal{F},\varepsilon)$*-tamper simulatable for* G*.*

Proof. Define the following distribution $D_{f,h}$ over $\{0,1\}^k \cup \mathsf{same}^\star$ (which is efficiently samplable given black-box access to functions f and h): Sample $s \leftarrow \{0,1\}^n$ and define $\tilde{\mathsf{st}} = \mathsf{same}^\star$ if $f(s) = s$ and $\tilde{\mathsf{st}} = h(f(s))$ otherwise. The simulator S, having black-box access to the adversary A and the original functionality $\mathsf{G}(\mathsf{st},\cdot)$, will use $D_{f,h}$ to answer A's tampering query.

At the beginning, S runs A and then it works in one of two modes defined below (starting with the "normal mode"):

1. *Normal mode.* While A continues issuing queries, answer as follows:
   - Upon input $\langle \mathtt{Eval}, x\rangle$, forward x to $\mathsf{G}(\mathsf{st},\cdot)$ and return the output y back to A.
   - Upon input $\langle \mathtt{Tamper}, f\rangle$, sample $\tilde{\mathsf{st}} \leftarrow D_{f,h}$. Hence,
     - In case $\tilde{\mathsf{st}} = \mathsf{same}^\star$, stay in the current mode.
     - In case $\tilde{\mathsf{st}} \neq \mathsf{same}^\star$, go to the "overwritten mode" defined below with state $\tilde{\mathsf{st}}$.
2. *Overwritten mode.* Given state $\tilde{\mathsf{st}}$ simulate all further evaluation queries by running $\mathsf{G}(\tilde{\mathsf{st}},\cdot)$.
3. Output whatever A does.

We argue that $\mathsf{Real}_{\mathsf{A},\mathcal{F}}^{\mathsf{G}^h(s,\cdot)} \approx \mathsf{Ideal}_{\mathsf{S}}^{\mathsf{G}(\mathsf{st},\cdot)}$. By non-malleability of the key-derivation function h, we know that for all functions $f \in \mathcal{F}$ it holds that:

$$\mathsf{Real}_h(f) := \left\{ \begin{array}{c} s \leftarrow \{0,1\}^n \\ \text{If } f(s) = s, \ \text{ Output: } (h(s), \mathsf{same}^\star) \\ \text{Else Output: } (h(s), h(f(s))) \end{array} \right\} \approx_\varepsilon \left\{ \begin{array}{c} \tilde{\mathsf{st}} \leftarrow D_{f,h}, \mathsf{st} \leftarrow \{0,1\}^k \\ \text{Output: } (\mathsf{st}, \tilde{\mathsf{st}}) \end{array} \right\} := \mathsf{Sim}_h(f).$$

To argue indistinguishability of the simulation, we observe that the real experiment and the ideal experiment are identical after the simulator enters in overwritten mode. To conclude the proof, we show that they are also indistinguishable for each round up to and including the round in which the simulator enters in overwritten mode. Without loss of generality, assume that A applies the tampering function f at round q, for some $q \in \mathbb{N}$. Denote with $\overline{\mathsf{Real}}_{\mathsf{A},\mathcal{F}}^{\mathsf{G}^h(s,\cdot)}$ the random variable corresponding to the output of the real experiment until round q. The random variable $\overline{\mathsf{Ideal}}_{\mathsf{S}}^{\mathsf{G}(\mathsf{st},\cdot)}$ is defined analogously.

Consider now the following function g, taking as input a pair of values $\mathsf{st}^* \in \{0,1\}^k$ and $\tilde{\mathsf{st}} \in (\{0,1\}^k \cup \mathsf{same}^\star)$, together with any sequence of inputs $x_1,\ldots,x_q \in \{0,1\}^u$: Output $y_i = \mathsf{G}(\mathsf{st}^*, x_i)$ for all $i \leq q-1$ and $y_q = \mathsf{G}(\mathsf{st}^*, x_q)$ if $\tilde{\mathsf{st}} = \mathsf{same}^\star$ and $y_q = \mathsf{G}(\tilde{\mathsf{st}}, x_q)$ otherwise. It is not hard to see that

$$\overline{\mathsf{Real}}_{\mathsf{A},\mathcal{F}}^{\mathsf{G}^h(s,\cdot)} = g(\mathsf{Real}_h(f)) \quad \text{and} \quad \overline{\mathsf{Ideal}}_{\mathsf{S}}^{\mathsf{G}(\mathsf{st},\cdot)} = g(\mathsf{Sim}_h(f)).$$

Thus,

$$\mathbf{SD}(\overline{\mathsf{Real}}_{\mathsf{A},\mathcal{F}}^{\mathsf{G}^h(s,\cdot)}; \overline{\mathsf{Ideal}}_{\mathsf{S}}^{\mathsf{G}(\mathsf{st},\cdot)}) = \mathbf{SD}(g(\mathsf{Real}_h(f)); g(\mathsf{Sim}_h(f))) \leq \mathbf{SD}(\mathsf{Real}_h(f); \mathsf{Sim}_h(f)) \leq \varepsilon,$$

concluding the proof. □

Chapter 6

Tamper-resilience using Continuous Non-Malleable Codes

In this chapter we extend the standard definition of non-malleable codes and propose a new definition called *continuous non-malleable codes*. We provide a construction which satisfies this definition against split-state tampering class $\mathcal{F}_{\mathsf{split}}$. Since we also show that information theoretically it is impossible to construct such code, we have to restrict ourself in computationally bounded scenario. An extended, titled "Continuous Non-malleable Codes", of the results discussed in this chapter is published in the proceedings of the 11th IACR Theory of Cryptography Conference – TCC 2014. [FMNV14].

6.1 Introduction

The standard notion of non-malleability considers a one-shot game: the adversary is allowed to tamper a single time with the codeword and obtains the decoded output. In this work we introduce so-called *continuous non-malleable codes*, where non-malleability is guaranteed even if the adversary *continuously* applies functions from a set $\mathcal{F}$ to the codeword. We show that our new security notion is not only a natural extension of the standard one-shot notion, but moreover allows to protect against tampering attacks in important settings where earlier constructions fall short to achieve security.

Continuous non-malleable codes. Recall from Chapter 4 that, a non-malleable code Code consists of a pair of algorithms $(\mathsf{Enc}, \mathsf{Dec})$ that satisfy the correctness property $\mathsf{Dec}(\mathsf{Enc}(x)) = x$. Also recall the tampering experiment $\mathsf{Tamper}_{f,x}$ defined for a function $f \in \mathcal{F}$ and an input message $x \in \mathcal{X}$:

1. Compute an encoding $c \leftarrow \mathsf{Enc}(x)$ using the encoding procedure.
2. Apply the tampering function $f \in \mathcal{F}$ to obtain the tampered codeword $c' = f(c)$.
3. If $c' = c$ then return the special symbol $\mathsf{same}^\star$; otherwise, return $\mathsf{Dec}(c')$. Notice that $\mathsf{Dec}(c')$ may return the special symbol $\perp$ in case the tampered codeword c' is detected invalid.

A coding scheme Code is said to be (one-shot) non-malleable with respect to functions in $\mathcal{F}$ and message space $\mathcal{X}$, if for every $f \in \mathcal{F}$ and any two messages $x, y \in \mathcal{X}$ the distributions $\mathsf{Tamper}_{f,x}$ and $\mathsf{Tamper}_{f,y}$ are indistinguishable. For formal definition, we refer to Chapter 4.

To define continuous non-malleable codes, we do not fix a single tampering function f a-priori.[1] Instead, we let the adversary repeat step 2 and step 3 from the above game a polynomial number of times, where in the i-th iteration the adversary can adaptively choose a tampering function $f_i \in \mathcal{F}$. We emphasize that this change of the tampering game allows the adversary to tamper

[1] Our main definition in this chapter is actually slight stronger than what we discuss next. Basically our definition achieves super non-malleability as presented in Chapter 4, albeit we restrict ourselves to split-state model in this chapter.

continuously with the initial encoding c. As mentioned in Chapter 3, Gennaro *et al.* [GLM+04] observed that such a strong security notion is impossible to achieve without further assumptions. Therefore, in Chapter 3 we by-pass this by bounding the number of times a secret can be tampered. However, in this chapter we take the different route: we rely on a *self-destruct* mechanism as used in earlier works on non-malleable codes. More precisely, when in step 3 the game detects an invalid codeword and returns $\perp$ for the first time, then the device self-destructs. This can, for instance, be implemented using a single public untamperable bit.

Recall from Section 4.2 that the generic tamper-resilient compiler guarantees continuous tamper resilience even if the underlying non-malleable code is secure only against one-shot tampering. This security "boost" is achieved by re-encoding the secret state/key after each execution of the primitive G^h. As one-shot non-malleability suffices in the above cryptographic application, one may ask *why do we need a continuous non-malleable code ?* We emphasize that besides being a natural extension of the standard non-malleability notion, our new notion has several important applications that we discuss in the next two paragraphs.

Tamper resilience without erasures. The transformation sketched in Section 4.2 necessarily requires that after each execution the entire content of the memory is erased. While such perfect erasures may be feasible in some settings, they are rather problematic in the presence of tampering. To illustrate this issue consider a setting where besides the encoding of a key, the memory also contains other non-encoded data. In the tampering setting, we cannot restrict the erasure to just the part that stores the encoding of the key as a tampering adversary may copy the encoding to some different part of the memory. A simple solution to this problem is to erase the entire memory, but such an approach is not possible in most cases: for instance, think of the memory as being the hard-disk of your computer that besides the encoding of a key stores other important files that you don't want to be erased. Notice that this situation is quite different from the leakage setting, where we also require perfect erasures to achieve continuous leakage resilience. In the leakage setting, however, the adversary cannot "mess around" with the state of the memory by, e.g., copying an encoding of a secret key to some free space, which makes erasures significantly easier to implement.

One option to prevent the adversary from keeping permanent copies is to encode the entire state of the memory. While this may be feasible in settings where the contents of the memory only stores a key but nothing else, such an approach has the following drawbacks.

1. *It is unnatural:* In many cases secret data, e.g., a cryptographic key, is stored together with non-confidential data. Each time we want to read some small part of the memory, e.g., the key, we need to decode and re-encode the entire state, including also the non-confidential data.

2. *It is inefficient:* Decoding and re-encoding the entire state of the memory for each access introduces additional overhead and would result in significantly inefficient solutions.

3. *It does not work in general:* Consider a setting where we want to compute with non-malleable codes in a tamper resilient way (similar in spirit to tamper resilient circuits). Clearly, in this setting the memory will store many independent encodings of different secrets that cannot be erased. Continuous non-malleable codes are hence a first natural step towards non-malleable computation. In fact, a recent work [FMNV15] uses continuous non-malleable codes to construct a tamper-resilient RAM-like architecture.

Using our new notion of continuous non-malleable codes we can avoid the above issues and achieve continuous tamper resilience without using *erasures* and without relying on inefficient solutions that encode the *entire* state.

Stateless tamper resilient transformations. The generic compiler sketched in Chapter 4 which achieves tamper-resilience from one-shot non-malleability necessarily requires the state to be re-encoded using fresh randomness. This not only reduces the efficiency of the proposed construction, but moreover makes the transformation stateful. Using continuous non-malleable codes we get continuous tamper resilience for free, eliminating the need to refresh the encoding after each usage. This is in particular useful when the underlying primitive that we want to protect is stateless itself. Think, for instance, of any standard block-cipher construction that typically keeps the same key. Using continuous non-malleable codes the tamper resilient implementation of such stateless primitives does not need to keep any secret state. We will discuss more about this later in Section 6.6.

6.1.1 Our Contribution

In this work, we propose the first construction of *continuous non-malleable codes* in the split-state model.

Recall from Chapter 4 that, in the split-state tampering model, the codeword consists of two halves c_0 and c_1 that are stored on two different parts of the memory. The adversary is assumed to tamper with both parts independently, but otherwise can apply any efficiently computable tampering function. That is, the adversary picks two arbitrary f_0 and f_1 and replaces the state (c_0, c_1) with the tampered state $(f_0(c_0), f_1(c_1))$. In a computationally bounded scenario naturally the functions must be poly-time computable. Similar to the earlier work of Liu and Lysyanskaya [LL12] our construction assumes a public *untamperable* common reference string (CRS in short). Notice that this is a rather mild assumption as the CRS can be hard-wired into the functionality and is independent of any secret data. More related works on split-state non-malleable codes can be found in Section 4.3.

Continuous non-malleability of existing constructions. We start by analyzing if existing split-state constructions are secure under continuous tampering. The first construction of (one-shot) split-state non-malleable codes in the standard model was given by Liu and Lysyanskaya [LL12]. At a high-level the construction encrypts the input x with a leakage resilient encryption scheme and generates a non-interactive zero-knowledge proof of knowledge showing that (a) the public/secret keys of the PKE are valid, and (b) the ciphertext is an encryption of x under the public key. Then, c_0 is set to the secret key while c_1 holds the corresponding public key, the ciphertext and the above described NIZK proof.

Unfortunately, it is rather easy to break the non-malleable code of Liu and Lysyanskaya in the continuous setting. Recall that our security notion of continuous non-malleable codes allows the adversary to interact in the following game. First, we sample a codeword $(c_0, c_1) \leftarrow \mathsf{Enc}(x)$ and then repeat the following process a polynomial number of times:

1. The adversary submits two polynomial-time computable functions (f_0, f_1) resulting in a tampered state $(c'_0, c'_1) = (f_0(c_0), f_1(c_1))$.

2. We consider three different cases: (1) if $(c'_0, c'_1) = (c_0, c_1)$ then return $\mathsf{same}^\star$; (2) otherwise compute $x' = \mathsf{Dec}(c'_0, c'_1)$ and return x' if $x' \neq \perp$; (3) if $x' = \perp$ self-destruct and terminate the experiment.

The main observation that enables the attack against the scheme of [LL12] is as follows: for a fixed (but adversarially chosen) part c'_0 it is easy to come up with two corresponding parts c'_1 and c''_1 such that both (c'_0, c'_1) and (c'_0, c''_1) form a valid codeword that *does not* lead to a self-destruct. Suppose further that $\mathsf{Dec}(c'_0, c'_1) \neq \mathsf{Dec}(c'_0, c''_1)$, then under continuous tampering the adversary may permanently replace the original encoding c_0 with c'_0, while depending on whether the i-th bit of c_1 is 0 or 1 either replace c_1 by c'_1 or c''_1. This allows to recover the entire c_1 by just $|c_1|$ tampering attacks. Once c_1 is known to the adversary it is easy to tamper with (c_0, c_1) in a way that depends on $\mathsf{Dec}(c_0, c_1)$.

Somewhat surprisingly, our attack can be generalized to break *any* non-malleable code that is secure in the information theoretic setting. Hence, also the recent breakthrough results on information theoretic non-malleability [CZ14, ADKO15a] fail to provide security under continuous attacks.

Uniqueness. The attack above exploits that for a fixed known part c'_0 it is easy to come-up with two valid parts c'_1, c''_1. For the encoding of [LL12] this is indeed easy to find. If the secret key c'_0 is known it is easy to come-up with two valid parts c'_1, c''_1: just encrypt two arbitrary messages $x_0 \neq x_1$ and generate the corresponding proofs. The above weakness motivates a new property that non-malleable codes shall satisfy in order to achieve security against continuous non-malleability. We call this property *uniqueness*, which informally guarantees that for any (adversarially chosen) valid encoding (c'_0, c'_1) it is computationally hard to come up with $c''_b \neq c'_b$ such that (c'_b, c''_{1-b}) forms a valid encoding. Clearly the uniqueness property prevents the above described attack, and hence is a crucial requirement for continuous non-malleability.

A new construction. In light of the above discussion, we need to build a non-malleable code that achieves our uniqueness property. Our construction uses as building blocks a leakage resilient storage (LRS) scheme [DDV10, DF11] for the split-state model (one may view this as a generalization of the leakage resilient PKE used in [LL12]), a collision-resistant hash function and (similar to [LL12]) an extractable NIZK. At a high-level we use the LRS to encode the secret message, hash the resulting shares using the hash function and generate a NIZK proof of knowledge that indeed the resulting hash values are correctly computed from the shares. While intuitively the collision resistance of the hash function guarantees the uniqueness property, a careful analysis is required to show that our code is continuously non-malleable. We refer the reader to Section 6.4 for the details of our construction and to Section 6.4.1 for an outline of the proof. The detailed proof is provided in Sec. 6.5.

Efficient Tamper resilient compilers without erasures. We can use our new construction of continuous non-malleable codes to protect arbitrary computation against continuous tampering attacks. In contrast to earlier works our construction does not need to re-encode the secret state after each usage, which besides being more efficient avoids the use of erasures. As discussed above, erasures are problematic in the tampering setting as one would essentially need to encode the entire state (possibly including large non-confidential data).

Additionally, our transformation does not need to keep any secret state. Hence, if our transformation is used for stateless primitives, then the resulting scheme remains stateless. This solves a problem left open by Dziembowski, Pietrzak and Wichs [DPW10]. Notice that while we do not need to keep any secret state, the transformed functionality requires one single bit to switch to self-destruction mode. This bit can be *public* but must be *untamperable*, and can for instance be implemented through one-time writable memory. As shown in the work of Gennaro et al. [GLM+04] continuous tamper resilience is impossible to achieve without such a mechanism for self-destruction.

Of course, our construction can also be used for stateful primitives, in which case our functionality will re-encode the new state during execution. Note that in this setting, as data is never erased, an adversary can always reset the functionality to a previous valid state. To avoid this, our transformation additionally uses an *untamperable public* counter[2] that helps us to detect whenever the functionality is reset to a previous state, leading to a self-destruct. We notice that such an untamperable counter is necessary, as otherwise there is no way to protect against the above resetting attack.

Adding leakage. As a last contribution, we show that our code is also secure against bounded leakage attacks. This is similar to the works of [LL12, DKO13, ADKO15b] who also consider bounded leakage resilience of their encoding scheme. We then show that bounded leakage resilience is also inherited by functionalities that are protected by our transformation. Notice that without perfect erasures bounded leakage resilience is the best we can achieve, as there is no hope for security if an encoding that is produced at some point in time is gradually revealed to the adversary.

6.1.2 Related Work

We skip the related works about the previous constructions of non-malleable codes here as those has been already discussed in Section 5.1.3. However, since our compiler is similar to that achieved by Gennaro et al. [GLM+04] we present a comparison with their work below.

Relation to [GLM+04]. Genarro *et al.* [GLM+04] proposed a generic method that allows to protect arbitrary computation against continuous tampering attacks, without requiring erasures. At a high-level their scheme works as follows: given the secret state st that we want to protect, we compute a signature σ of st and store (σ, st) in memory. Each time prior to using st the functionality verifies σ using the corresponding hard-wired public key, and self-destructs in case this check fails. While the above approach achieves some form of continuous tamper resilience, non-malleable codes are beneficial because of the the following reasons:

1. The above construction yields a weaker security guarantee. In fact, an adversary may easily learn a logarithmic number of bits of the secret state by tampering with the memory.

2. To update the secret state we need to sign the new state using the secret key of the signature scheme. As this secret key is not available, the transformation can only be used to protect stateless primitives.

A more detailed comparison between this approach and the generic non-malleable code-based approach is given in [DPW10].

[2]Note that a counter uses very small (logarithmic in the security parameter) number of bits.

6.2 Preliminaries

Recall that we denote the security parameter by λ. Throughout this chapter we use the following (chapter-specific) notation: Oracle $\mathcal{O}^{\zeta}(s)$ is parametrized by a value s and takes as input functions L and outputs $\mathsf{L}(s)$, returning a total of at most ζ bits.

6.2.1 Non-Interactive Zero Knowledge

We start with the formal description of (robust) NIZKs. These NIZKs has minor differences with the tSE-NIZKs discussed in Chapter 3 as here the adversary is *not* restricted to access simulated proofs for the true statements. However we use slightly different notations in this chapter for ease of presentation.

Given an **NP**-relation, let $\mathcal{L} = \{x : \exists\ w \text{ such that } \mathfrak{R}(x,w) = 1\}$ be the corresponding language. A robust non-interactive zero knowledge (NIZK) proof system for $\mathcal{L}$, is a tuple of algorithms $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{S} = (\mathsf{S}_1, \mathsf{S}_2), \mathsf{Ext})$ such that the following properties hold [SCO+01].

Completeness. For all $x \in \mathcal{L}$ of length k and all w such that $\mathfrak{R}(x,w) = 1$, for all $\Omega \leftarrow \mathsf{Gen}(1^{\lambda})$ we have that $\mathsf{Verify}(\Omega, x, \mathsf{Prove}(\Omega, x, w)) = \mathtt{accept}$

Multi-theorem zero knowledge. For all PPT adversaries A, we have $\mathsf{Real}(\lambda) \approx \mathsf{Simu}(\lambda)$, where $\mathsf{Real}(\lambda)$ and $\mathsf{Simu}(\lambda)$ are distributions defined via the following experiment:

$$\mathsf{Real}(\lambda) = \left\{\Omega \leftarrow \mathsf{Gen}(1^{\lambda}); out \leftarrow \mathsf{A}^{\mathsf{Prove}(\Omega,\cdot,\cdot)}(\Omega); \text{Output: } out.\right\}$$
$$\mathsf{Simu}(\lambda) = \left\{(\Omega, tk) \leftarrow \mathsf{S}_1(1^{\lambda}); out \leftarrow \mathsf{A}^{\mathsf{S}_2(\Omega,\cdot,tk)}(\Omega); \text{Output: } out.\right\}.$$

Extractability. There exists a PPT algorithm Ext such that, for all PPT adversaries A, we have

$$\Pr\left[\begin{array}{c}(\Omega, tk, ek) \leftarrow \mathsf{S}_1(1^{\lambda}); (x,\pi) \leftarrow \mathsf{A}^{\mathsf{S}_2(\Omega,\cdot,tk)}(\Omega); w \leftarrow \mathsf{Ext}(\Omega, (x,\pi), ek);\\ \mathfrak{R}(x,w) \neq 1 \wedge (x,\pi) \notin \mathcal{Q} \wedge \mathsf{Verify}(\Omega, x, \pi) = \mathtt{accept}\end{array}\right] \leq \mathsf{negl}(\lambda),$$

where the list $\mathcal{Q}$ contains the successful pairs (x_i, π_i) that A has queried to S_2.

We also require that the proof system supports labels, so that the Prove, Verify, S and Ext algorithms now also take a public label Ψ as input, and the completeness, zero knowledge and extractability properties are updated accordingly. This can be easily achieved by appending the label Ψ to the statement x.

Similarly to [LL12], we also assume that different statements have different proofs, i.e., if $\mathsf{Verify}(\Omega, x, \pi) = \mathtt{accept}$ we have that $\mathsf{Verify}(\Omega, x', \pi) = \mathtt{reject}$ for all $x' \neq x$. The last property can be achieved by appending the statement to its proof.

6.2.2 Leakage Resilient Storage

We recall the definition of leakage resilient storage from [DDV10, DF11]. The definition is very similar to the definition of leakage-resilient code provided in Chapter 5, however adapted to the the split-state setting and also with some additional requirement (thus slightly stronger). Again we use different notations as it is more suitable for this chapter.

A leakage resilient storage scheme $(\mathsf{LRS}, \mathsf{LRS}^{-1})$ is a pair of algorithms defined as follows. (1) Algorithm LRS takes as input a secret x and outputs an encoding (s_0, s_1) of $x \in \{0,1\}^k$ for some $k \in \mathsf{poly}(\lambda)$. (2) Algorithm LRS^{-1} takes as input shares (s_0, s_1) and outputs a message x'.

Definition 6.2.1 (LRS)**.** *We call* $(\mathsf{LRS}, \mathsf{LRS}^{-1})$ *an* ζ*-leakage resilient storage scheme (*ζ*-LRS) if for all* $\theta \in \{0,1\}$*, all secrets* x, y *and all adversaries* A *it holds that*

$$\left\{\mathsf{Leakage}_{\mathsf{A},x,\theta}(\lambda)\right\}_{\lambda\in\mathbb{N}} \approx_s \left\{\mathsf{Leakage}_{\mathsf{A},y,\theta}(\lambda)\right\}_{\lambda\in\mathbb{N}},$$

where

$$\mathsf{Leakage}_{\mathsf{A},x,\theta}(\lambda) = \left\{ \ (s_0,s_1) \leftarrow \mathsf{LRS}(x); out_{\mathsf{A}} \leftarrow \mathsf{A}^{\mathcal{O}^{\zeta}(s_0,\cdot),\mathcal{O}^{\zeta}(s_1,\cdot)}; \textit{Output: } (s_\theta, out_{\mathsf{A}}). \ \right\}.$$

We remark that, as mentioned earlier, the above definition is slightly stronger than the standard definition of LRS, as here the adversary is allowed to see one of the two shares after he is done with leakage queries. A careful analysis of the proof, however, shows that the LRS scheme of [DF11, Lemma 22] satisfies the above generalized notion in case the extractor used in the construction is a *strong* extractor [CG88].

6.3 Continuous Non-Malleability

In this section we present our main definition of continuous non-malleable codes. Since we restrict our focus only on split-state functions, here we take a different approach than Chapter 4 to define the notion which will help the presentation. In particular we start with adapting the definition of a coding scheme presented in Chapter 4 (Def. 4.1.1) to split-state and with a common reference string (CRS).

Below we denote the input length by k and the output length by n.

Definition 6.3.1 (Split-state Encoding Scheme in the CRS Model)**.** *A split-state encoding scheme in the common reference string (CRS) model is a tuple of algorithms* $\mathsf{Code} = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ *specified below.*

- Init *takes as input the security parameter and outputs a CRS* $\Omega \leftarrow \mathsf{Init}(1^\lambda)$*.*
- Enc *takes as input some message* $x \in \{0,1\}^k$ *and the CRS* Ω *and outputs a codeword consisting of two parts* (c_0, c_1) *such that* $c_0, c_1 \in \{0,1\}^n$*.*
- Dec *takes as input a codeword* $(c_0, c_1) \in \{0,1\}^{2n}$ *and the CRS and outputs either a message* $x' \in \{0,1\}^k$ *or a special symbol* $\perp$*.*

Consider the following oracle $\mathcal{O}_{\mathsf{cnm}}((c_0,c_1))$, which is parametrized by an encoding (c_0,c_1) and takes as input functions $f_0, f_1 : \{0,1\}^n \to \{0,1\}^n$.

$\underline{\mathcal{O}_{\mathsf{cnm}}((c_0,c_1),(f_0,f_1))}$:
$(c_0',c_1') = (f_0(c_0), f_1(c_1))$
If $(c_0',c_1') = (c_0,c_1)$ return $\mathsf{same}^\star$
If $\mathsf{Dec}(\Omega,(c_0',c_1')) = \perp$, return $\perp$ and "self-destruct"
Else return (c_0',c_1').

By "self-destruct" we mean that once $\mathsf{Dec}(\Omega,(c_0',c_1'))$ outputs $\perp$, the oracle will answer $\perp$ to any further query.

Now we present our main definition of this chapter that is of continuous non-malleable codes. Actually our definition is a continuous adaptation of so-called super non-malleable code presented earlier in Chapter 4 for the function family $\mathcal{F}_{\mathsf{split}}$ and in computationally bounded setting.

Definition 6.3.2 (Continuous Non-Malleability)**.** *Let* $\mathsf{Code} = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ *be a split-state encoding scheme in the CRS model. We say that* Code *is* q*-continuously non-malleable* ζ*-leakage resilient* ((ζ, q)*-CNMLR for short), if for all messages* $x, y \in \{0,1\}^k$ *and all PPT adversaries* A *it holds that*

$$\left\{\mathsf{Tamper}^{\mathrm{cnmlr}}_{\mathsf{A},x}(\lambda)\right\}_{\lambda\in\mathbb{N}} \approx_c \left\{\mathsf{Tamper}^{\mathrm{cnmlr}}_{\mathsf{A},y}(\lambda)\right\}_{\lambda\in\mathbb{N}}$$

where

$$\mathsf{Tamper}^{\mathrm{cnmlr}}_{\mathsf{A},x}(\lambda) = \left\{ \begin{array}{c} \Omega \leftarrow \mathsf{Init}(1^\lambda); (c_0, c_1) \leftarrow \mathsf{Enc}(\Omega, x); \\ out_{\mathsf{A}} \leftarrow \mathsf{A}^{\mathcal{O}^\zeta(c_0), \mathcal{O}^\zeta(c_1), \mathcal{O}_{\mathsf{cnm}}((c_0,c_1))}; \mathit{Output:}\ out_{\mathsf{A}} \end{array} \right\}$$

and A *asks a total of* q *queries to* $\mathcal{O}_{\mathsf{cnm}}$*.*

Without loss of generality we assume that the variable out_A consists of all the bits leaked from c_0 and c_1 (in a vector $\mathbf{\Lambda}$) and all the outcomes from oracle $\mathcal{O}_{\mathsf{cnm}}(c_0, c_1)$ (in a vector $\mathbf{\Theta}$); we write this as $out_A = (\mathbf{\Lambda}, \mathbf{\Theta})$ where $|\mathbf{\Lambda}| \leq 2\zeta$ and $\mathbf{\Theta}$ has exactly q elements.

Intuitively, the above definition captures a setting where a fully adaptive adversary A tries to break non-malleability by tampering several times with a target encoding, obtaining each time some leakage from the decoding process. The only restriction is that whenever a tampering attempt decodes to $\perp$, the system "self-destructs". Another important thing to note is that in the above definition the adversary gets to tamper with the same codeword repeatedly. This captures the setting motivated by so-called "copy" attack where the adversary can store multiple copies of the same codeword and then tamper one at a time adaptively. This type of tampering is called persistent tampering in the literature.

We do not present a generic definition of continuous non-malleable codes which simply extends the definitions presented in Sec. 4.1[3]. A subsequent work [JW15] by Jafargholi and Wichs formally presents the generic definitions and also discusses variants of that. We briefly discuss their work in Chapter 7.

6.3.1 Uniqueness

As we argue below, constructions that satisfy our new Definition 6.3.2 have to meet the following *uniqueness* requirement. Informally this means that for any (possibly adversarially chosen) side of an encoding c_b it is computationally hard to find two corresponding sides c'_{1-b} and c''_{1-b} such that both (c_b, c'_{1-b}) and (c_b, c''_{1-b}) form a valid encoding.

Definition 6.3.3 (Uniqueness)**.** *Let* $\mathsf{Code} = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ *be a split-state encoding in the CRS model. We say that* Code *satisfies* uniqueness *if for all PPT adversaries* A *and for all* $b \in \{0,1\}$ *we have:*

$$\mathbb{P}\left[\begin{array}{c} \Omega \leftarrow \mathsf{Init}(1^\lambda); (c_b, c'_{1-b}, c''_{1-b}) \leftarrow \mathsf{A}(\Omega); c'_{1-b} \neq c''_{1-b}; \\ \mathsf{Dec}(\Omega, (c_b, c'_{1-b})) \neq \perp; \mathsf{Dec}(\Omega, (c_b, c''_{1-b})) \neq \perp \end{array} \right] \leq \mathsf{negl}(\lambda).$$

The following attack shows that the uniqueness property is necessary to achieve Definition 6.3.2.

Lemma 6.3.4. *Let* Code *be* $(0, \mathsf{poly}(\lambda))$*-CNMLR. Then* Code *must satisfy uniqueness.*

Proof. For the sake of contradiction, assume that we can efficiently find a triple (c_0, c'_1, c''_1) such that (c_0, c'_1) and (c_0, c''_1) are both valid and $c'_1 \neq c''_1$. For a target encoding (τ_0, τ_1), we describe an efficient algorithm recovering τ_1 with overwhelming probability, by asking $n = \mathsf{poly}(\lambda)$ queries to $\mathcal{O}_{\mathsf{cnm}}((\tau_0, \tau_1), \cdot)$.

[3] Each definition of non-malleable codes presented in Sec. 4.1 can be extended to have a continuous version.

For all $i \in [n]$ repeat the following:

Prepare the i-th tampering function as follows:

- $f_0^{(i)}(\tau_0)$: Replace τ_0 by c_0;
- $f_1^{(i)}(\tau_1)$: If $\tau_1[i] = 0$ replace τ_1 by c_1'; otherwise replace it by c_1''.

Submit $(f_0^{(i)}, f_1^{(i)})$ to the tampering oracle $\mathcal{O}_{\mathsf{cnm}}((\tau_0, \tau_1), \cdot)$ and obtain (τ_0', τ_1').

If $(\tau_0', \tau_1') = (c_0, c_1')$, set $z[i] \leftarrow 0$.

Otherwise, if $(\tau_0', \tau_1') = (c_0, c_1'')$, set $z[i] \leftarrow 1$.

Output z as the guess for τ_1.

The above algorithm clearly succeeds with overwhelming probability, whenever $c_1' \neq \tau_1 \neq c_1''$.[4]

Once τ_1 is known, we ask one additional query $(f_0^{(n+1)}, f_1^{(n+1)})$ to $\mathcal{O}_{\mathsf{cnm}}((\tau_0, \tau_1), \cdot)$, as follows:

– $f_0^{(n+1)}(\tau_0)$ hard-wires τ_1 and computes $y \leftarrow \mathsf{Dec}(\Omega, (\tau_0, \tau_1))$; if the first bit of y is 0 then f_0 behaves like the identity function, otherwise it overwrites τ_0 with some invalid codeword (say 0^n).

– $f_1^{(n+1)}(\tau_0)$ is the identity function.

The above clearly allows to learn one bit of the message in the target encoding and hence contradicts the fact that Code is $(0, \mathsf{poly}(\lambda))$-CNMLR. □

Attacking existing schemes. The above procedure can be applied to show that the encoding of [LL12] does not satisfy our notion. Recall that in [LL12] a message x is encoded as $c_0 = (pk, c := \mathsf{Encrypt}(pk, x), \pi)$ and $c_1 = sk$. Here, (pk, sk) is a valid key pair and π is a proof of knowledge of a pair (x, sk) such that c decrypts to x under sk *and* (pk, sk) forms a valid key-pair. Clearly, for some $c_1' = sk'$ it is easy to find two valid corresponding parts $c_0 \neq c_0''$ which violates uniqueness.

We mention two important extensions of the attack from Lemma 6.3.4, leading to even stronger security breaches:

1. In case the valid pair of encodings (c_0, c_1'), (c_0, c_1'') which violates the uniqueness property are such that $\mathsf{Dec}(\Omega, (c_0, c_1')) \neq \mathsf{Dec}(\Omega, (c_0, c_1''))$, one can show that Lemma 6.3.4 still holds in the weaker version of the Definition 6.3.2 in which the experiment does not output tampered encodings but only the corresponding decoded message. Note that this applies in particular to the encoding of [LL12].

2. In case it is possible to find *both* (c_0, c_1', c_1'') and (c_0', c_0'', c_1) violating uniqueness, a simple variant of the attack allows us to recover both halves of the target encoding which is a total breach of security! However, it is not clear for the scheme of [LL12] how to find two valid corresponding parts c_1', c_1'', because given pk' it shall of course be computationally hard to find two corresponding valid secret keys sk', sk''.

The above attack can be easily extended to the information theoretic setting to break the constructions of the non-malleable codes (in split-state) [DKO13, ADL14, ADKO15a, CZ14]. In fact, in the following lemma we show that there does *not* exist any information theoretic secure CNMLR code.

[4] In case $(c_0, c_1') = (\tau_0, \tau_1)$ or $(c_0, c_1'') = (\tau_0, \tau_1)$, then the entire encoding can be recovered even with more ease. In this case, whenever the oracle returns $\mathsf{same}^\star$ we know $\tau_0 = c_0$ and $\tau_1 \in \{c_1', c_1''\}$. In the next step we replace the encoding with (c_0, c_1'); if the oracle returns $\mathsf{same}^\star$ again, then we conclude that $\tau_1 = c_1'$, otherwise we conclude $\tau_1 = c_1''$.

Lemma 6.3.5. *It is impossible to construct information theoretically secure* $(0, \mathsf{poly}(\lambda))$*-CNMLR codes.*

Proof. We prove the lemma by contradiction. Assume that there exists an information theoretically secure $(0, \mathsf{poly}(\lambda))$-CNMLR code with $2n$ bits codewords. By Lemma 6.3.4, the code must satisfy the uniqueness property. In the information theoretic setting this means that, for all codewords $(c_0, c_1) \in \{0,1\}^{2n}$ such that $\mathsf{Dec}(\Omega, (c_0, c_1)) \neq \perp$, the following holds: (i) for all $c_1' \in \{0,1\}^n$ such that $c_1' \neq c_1$, we have $\mathsf{Dec}(\Omega, (c_0, c_1')) = \perp$; (ii) for all $c_0' \in \{0,1\}^n$, such that $c_0' \neq c_0$, we have $\mathsf{Dec}(\Omega, (c_0', c_1)) = \perp$.

Given a target encoding (c_0, c_1) of some secret x, an unbounded A can define the following tampering function f_b (for $b \in \{0,1\}$): Given c_b as input, try all possible $c_{1-b} \in \{0,1\}^n$ until $\mathsf{Dec}(\Omega, (c_0, c_1)) \neq \perp$. By property (i)-(ii) above, we conclude that for all $c_{1-b}' \neq c_{1-b}$, the decoding algorithm $\mathsf{Dec}(\Omega, (c_b, c_{1-b}'))$ outputs $\perp$ with overwhelming probability. Thus, f_b can recover $x = \mathsf{Dec}(\Omega, (c_b, c_{1-b}))$ and if the first bit of the decoded value is 0 leave the target encoding unchanged, otherwise (f_0, f_1) modifies the encoding with an invalid codeword. The above clearly allows to learn one bit of the message in the target encoding, and hence contradicts the fact that the code is $(0, \mathsf{poly}(\lambda))$-CNMLR. □

Note that the attack of Lemma 6.3.5 requires the tampering function to be unbounded. Moreover this attack does not apply in an weaker setting when the tampering is persistent. These facts leave the hope of proving something in the information theoretic setting either by restricting the tampering function to be a polynomial-size circuit or by considering persistent tampering (or both). We leave this direction as an interesting one for further research.

6.4 The Code

Consider the following split-state encoding scheme in the CRS model $(\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$, based on an LRS scheme $(\mathsf{LRS}, \mathsf{LRS}^{-1})$, on a family of collision resistant hash functions $\mathcal{H} = \{H_t : \{0,1\}^{\mathsf{poly}(\lambda)} \to \{0,1\}^{\lambda}\}_{t \in \{0,1\}^{\lambda}}$ and on a robust non-interactive zero knowledge proof system $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ which supports labels, for language $\mathcal{L}_{\mathcal{H},t} = \{h : \exists\ s \text{ such that } h = H_t(s)\}$.

$\mathsf{Init}(1^\lambda)$. Sample $t \leftarrow \{0,1\}^\lambda$ and run $\Omega \leftarrow \mathsf{Gen}(1^\lambda)$.

$\mathsf{Enc}(\Omega, x)$. Let $(s_0, s_1) \leftarrow \mathsf{LRS}(x)$. Compute $h_0 = H_t(s_0)$, $h_1 = H_t(s_1)$ and $\pi_0 \leftarrow \mathsf{Prove}(\Omega, (s_0, h_0), \Psi_1)$, $\pi_1 \leftarrow \mathsf{Prove}(\Omega, (s_1, h_1), \Psi_0)$, where the labels are defined as $\Psi_0 = h_0$, $\Psi_1 = h_1$. (Note that the pre-image of h_b is s_b and the proof π_b is computed for statement h_b using label h_{1-b}.) Output $(c_0, c_1) = ((s_0, h_1, \pi_1, \pi_0), (s_1, h_0, \pi_0, \pi_1))$.

$\mathsf{Dec}(\Omega, (c_0, c_1))$. The decoding parses c_b as $(s_b, h_{1-b}, \pi_{1-b}, \pi_b)$, computes $\Psi_b = H_t(s_b)$ and then proceeds as follows:

(a) *Local check.* If $\mathsf{Verify}(\Omega, (h_0, \Psi_1), \pi_0)$ or $\mathsf{Verify}(\Omega, (h_1, \Psi_0), \pi_1)$ output `reject` in any of the two sides c_0, c_1, return $x' = \perp$.

(b) *Cross check.* If (i) $h_0 \neq H_t(s_0)$ or $h_1 \neq H_t(s_1)$, *or* (ii) the proofs (π_0, π_1) in c_0 are different from the ones in c_1, then return $x' = \perp$.

(c) *Decoding.* Otherwise, return $x' = \mathsf{LRS}^{-1}(s_0, s_1)$.

We start by showing that the above code satisfies the uniqueness property (cf. Definition 6.3.3).

Lemma 6.4.1. *Let* $\mathsf{Code} = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ *be as above. Then, if* $\mathcal{H}$ *is a family of collision resistant hash functions* Code *satisfies uniqueness.*

Proof. We show that Definition 6.3.3 is satisfied for $b = 0$. The proof for $b = 1$ is identical and is therefore omitted.

Assume that there exists a PPT adversary A that, given as input $\Omega \leftarrow \mathsf{Init}(1^\lambda)$, is able to produce (c_0, c_1', c_1'') such that both (c_0, c_1') and (c_0, c_1'') are valid, but $c_1' \neq c_1''$. Let $c_0 = (s_0', h_1', \pi_1', \pi_0')$, $c_1' = (s_1', h_0', \pi_0', \pi_1')$ and $c_1'' = (s_1'', h_0'', \pi_0'', \pi_1'')$.

Since s_0' is the same in both encodings, we must have $h_0' = h_0''$ as the hash function is deterministic. Furthermore, since both (c_0, c_1') and (c_0, c_1'') are valid, the proofs must verify successfully and therefore we must have $\pi_0' = \pi_0''$ and $\pi_1' = \pi_1''$. It follows that $c_1'' = (s_1'', h_0', \pi_0', \pi_1')$, such that $s_1'' \neq s_1'$. Clearly (s_1', s_1'') is a collision for h_1', a contradiction. □

Next, we show that our code achieves continuous non-malleability.

Theorem 6.4.2. *Let* $\mathsf{Code} = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ *be as above. Assume that* $(\mathsf{LRS}, \mathsf{LRS}^{-1})$ *is an* ζ'*-LRS,* $\mathcal{H} = \{H_t : \{0,1\}^{\mathsf{poly}(\lambda)} \to \{0,1\}^\lambda\}_{t \in \{0,1\}^\lambda}$ *is a family of collision resistant hash functions and* $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ *is a robust NIZK proof system for language* $\mathcal{L}_{\mathcal{H},t}$*. Then* Code *is* (ζ, q)*-CNMLR, for any* $q = \mathsf{poly}(\lambda)$ *and* $\zeta' \geq \max\{2\zeta + (\lambda+1)\lceil \log(q) \rceil, 2\lambda + 1\}$.

6.4.1 Outline of the Proof

In order to build some intuition, let us first explain why a few natural attacks do not work. Clearly, the uniqueness property (cf. Lemma 6.4.1) rules out the attack of Lemma 6.3.4. As a first attempt, the adversary could try to modify the proof π_0 to a different proof π_0', by using the fact that c_0 contains the corresponding witness s_0 and the correct label h_1. However, to ensure the validity of (c_0', c_1'), this would require to place π_0' in c_1', which should be hard without knowing a witness (by the robustness of the proof system). Alternatively, one could try to maul the two halves (s_0, s_1) of the LRS scheme, into a pair (s_0', s_1') encoding a related message.[5] This requires, for instance, to change the proof π_0 into π_0' and place π_0' in c_1', which again should be hard without knowing a witness and the correct label.

Let us now try to give a high-level overview of the proof. Given a polynomial time distinguisher D that violates continuous non-malleability of Code, we build another polynomial time distinguisher D' which breaks leakage resilience of $(\mathsf{LRS}, \mathsf{LRS}^{-1})$. Distinguisher D', which can access oracles $\mathcal{O}^\zeta(s_0)$ and $\mathcal{O}^\zeta(s_1)$, has to distinguish whether (s_0, s_1) is the encoding of message x or message y with the help of D's advantage in distinguishing $\mathsf{Tamper}_x^{\mathrm{cnmlr}}$ from $\mathsf{Tamper}_y^{\mathrm{cnmlr}}$. The main difficulty in the reduction is how D' can simulate the answers from the tampering oracle $\mathcal{O}_{\mathsf{cnm}}$ (cf. Definition 6.3.2), without knowing the target encoding (c_0, c_1). This is the main point where our techniques diverge significantly from [LL12] (as in [LL12] the reduction "knows" a complete half of the encoding). In our case, in fact, D' can only access the two halves c_0 and c_1 "inside" the oracles $\mathcal{O}^\zeta(s_0)$ and $\mathcal{O}^\zeta(s_1)$.[6] However, it is not clear how this helps answering tampering queries, as the latter requires access to *both* c_0 and c_1, whereas the reduction can only access c_0 and c_1 *separately*.

[5] When the LRS is implemented using the inner product extractor this is indeed possible, as argued in [DKO13].

[6] Looking ahead, this can be achieved by first leaking the hash values h_0, h_1 of s_0, s_1, simulating the proofs π_0, π_1, and then hard-wiring these values into all leakage queries. For ease of description, in what follows we simply assume that D' can access directly $\mathcal{O}^\zeta(c_0)$ and $\mathcal{O}^\zeta(c_1)$.

To simplify the description, let us assume that D can only issue tampering queries (forget about leakage queries for the moment). Like any standard reduction, D' samples some randomness r and fixes the random tape of D to r. Our novel strategy is to construct a polynomial time algorithm $\mathsf{F}(r)$ that, given access to $\mathcal{O}^{\zeta}(c_0)$, $\mathcal{O}^{\zeta}(c_1)$, outputs the smallest index j^* which indicates the round where $\mathsf{D}(r)$ provokes a self-destruct in $\mathsf{Tamper}_*^{\mathrm{cnmlr}}$. Before explaining how the actual algorithm works, let us explain how D' can complete the reduction using such F. In the beginning it runs $\mathsf{F}(r)$ in order to leak the index j^*. At this point D' is done with leakage queries and asks to get c_0 (i.e., it chooses $\theta = 0$ in Definition 6.2.1). Given c_0, distinguisher D' runs $\mathsf{D}(r)$ (with the *same* random coins r used for F). Hence, for all $1 \leq j < j^*$, upon input the j-th tampering query $(f_0^{(j)}, f_1^{(j)})$, distinguisher D' lets $c_0' = f_0^{(j)}(c_0) = (s_0', h_1', \pi_1', \pi_0')$ and answers as follows:

1. In case $c_0' = c_0$, output $\mathsf{same}^\star$ (call type A queries).
2. In case $c_0' \neq c_0$ and either of the proofs in c_0' does not verify correctly, output $\perp$ (call type B queries).
3. In case $c_0' \neq c_0$ and both the proofs in c_0' verify correctly, check if $\pi_1' = \pi_1$; if 'yes' (in which case there is no hope to extract from π_1') then output $\perp$ (call type C queries).
4. Otherwise, attempt to extract s_1' from π_1', define $c_1' = (s_1', h_0', \pi_0', \pi_1')$ and output (c_0', c_1') (call type D queries).

Note that from round j^* on, all queries can be answered with $\perp$, and this is a correct simulation as $\mathsf{D}(r)$ provokes a self-destruct at round j^* in the real experiment.

In the proof of Theorem 6.4.2, we show that the above strategy is sound: with overwhelming probability over the choice of r the output produced by the above simulation is equal to the output that $\mathsf{D}(r)$ would have seen in the real experiment *until a self-destruct occurs*.[7] Let us give some intuition here. For type A queries, note that when $c_0' = c_0$ we must have $c_1' = c_1$ with overwhelming probability, as otherwise (c_0, c_1, c_1') would violate uniqueness. In case of type B queries, the decoding process in the real experiment would output $\perp$, so D' does a perfect simulation. The case of type C queries is a bit more delicate. In this case we use the facts that (i) in the NIZK proof system we use, different statements must have different proofs and (ii) the hash function is collision resistant, to show that c_0' must be of the form $c_0' = (s_0, h_1, \pi_1, \pi_0')$ and $\pi_0' \neq \pi_1$. A careful analysis shows that the latter contradicts leakage resilience of the underlying LRS scheme. Finally, for type D queries, note that whenever D' extracts a witness from a valid proof $\pi_1' \neq \pi_1$, the witness must be valid with overwhelming probability (as the NIZK is simulation extractable).

Next, let us explain how to construct the algorithm F. Roughly, $\mathsf{F}(r)$ runs $\mathsf{D}(r)$ "inside" the oracles $\mathcal{O}^{\zeta}(c_0)$, $\mathcal{O}^{\zeta}(c_1)$, and simulates the answers for $\mathsf{D}(r)$'s tampering queries using only one side of the target encoding, in the exact same way as outlined in (1)-(4) above. Let $\boldsymbol{\Theta}_b$, for $b \in \{0,1\}$, denote the output simulated by F inside $\mathcal{O}^{\zeta}(c_b)$. To locate the self-destruct index j^*, we rely on the following property: the vectors $\boldsymbol{\Theta}_0$ and $\boldsymbol{\Theta}_1$ contain identical values until coordinate $j^* - 1$, but $\boldsymbol{\Theta}_0[j^*] \neq \boldsymbol{\Theta}_1[j^*]$ with overwhelming probability (over the choice of r). This implies that j^* can be computed as the first coordinate where $\boldsymbol{\Theta}_0$ and $\boldsymbol{\Theta}_1$ are different. Hence, F can obtain the self-destruct index by using its adaptive access to oracles $\mathcal{O}^{\zeta}(c_0)$, $\mathcal{O}^{\zeta}(c_1)$ and apply a standard binary search algorithm to $\boldsymbol{\Theta}_0$, $\boldsymbol{\Theta}_1$. Note that the latter requires at most a logarithmic number of bits of adaptive leakage.

[7] It is crucial that both the real and simulated experiments are run with the same r.

One technical problem is that F, in order to run $\mathsf{D}(r)$ inside say $\mathcal{O}^{\zeta}(c_0)$ and compute $\boldsymbol{\Theta}_0$, has also to answer leakage queries from $\mathsf{D}(r)$. Clearly, all leakages from c_0 can be easily computed, however it is not clear how to simulate leakages from c_1 (as we cannot access $\mathcal{O}^{\zeta}(c_1)$ inside $\mathcal{O}^{\zeta}(c_0)$). Fortunately, the latter issue can be avoided by letting F query $\mathcal{O}^{\zeta}(c_0)$ and $\mathcal{O}^{\zeta}(c_1)$ alternately, and aborting the execution of $\mathsf{D}(r)$ whenever it is not possible to answer a leakage query. It is not hard to show that after at most ζ steps all leakages will be known, and F can run $\mathsf{D}(r)$ inside $\mathcal{O}^{\zeta}(c_0)$ without having access to $\mathcal{O}^{\zeta}(c_1)$. (All this comes at the price of some loss in the leakage bound, but, as we show in the proof, not too much.)

The formal proof is provided next.

6.5 Proof of Theorem 6.4.2

Assume there exists an adversary A, a pair of messages (x, y), some non-negligible function $\varepsilon = \varepsilon(\lambda)$ and a distinguisher D such that

$$\left|\Pr\left[\mathsf{D}(\Omega, \mathsf{Tamper}^{\mathrm{cnmlr}}_{\mathsf{A},x}) = 1\right] - \Pr\left[\mathsf{D}(\Omega, \mathsf{Tamper}^{\mathrm{cnmlr}}_{\mathsf{A},y}) = 1\right]\right| > \varepsilon.$$

We will build a distinguisher D' against the underlying LRS scheme. Distinguisher D' will use D, A, (x, y) to construct an adversary A' and some value $\theta \in \{0,1\}$ such that

$$\left|\Pr\left[\mathsf{D}'(\mathsf{Leakage}_{\mathsf{A}',x,\theta}) = 1\right] - \Pr\left[\mathsf{D}'(\mathsf{Leakage}_{\mathsf{A}',y,\theta}) = 1\right]\right| > \varepsilon'$$

for some non-negligible ε' (a contradiction).

Without loss of generality, we assume that D outputs its guess after a self-destruct takes place. In fact, in case A does not provoke a self-destruct, we can always modify A such that it asks an additional query to $\mathcal{O}_{\mathsf{cnm}}$ generating a self-destruct before outputting its guess; clearly this does not decrease D's advantage. We will construct D' in several steps. As a first step, we describe a mental experiment $\mathsf{Tamper}''_{\mathsf{S}}(b)$ (for a value $b \in \{0,1\}$) where a simulator S (depending on A) simulates the view of experiment $\mathsf{Tamper}_{\mathsf{A}}$ given only access to X_b and a leakage oracle for X_{1-b} (here (X_0, X_1) is the target encoding). We will then explain how to "mimic" S by using A and access to $\mathcal{O}^{\zeta}(X_0)$, $\mathcal{O}^{\zeta}(X_1)$. Finally we show that S can be used by D' to locate the index j^* where a self-destruct happens in $\mathsf{Tamper}_{\mathsf{A}}$. The value of j^* will then be used in the reduction to the security of the leakage-resilient storage scheme. (See also Section 6.4.1 for a more detailed outline of the proof.)

Note that we can assume without loss of generality that D distinguishes the two distributions of experiment $\mathsf{Tamper}_{\mathsf{A}}$ even in case we replace real proofs with simulated proofs. Consider indeed a modified experiment $\mathsf{Tamper}'_{\mathsf{A}}$ which is identical to $\mathsf{Tamper}_{\mathsf{A}}$, but where the Gen algorithm is replaced by $(\Omega, tk, ek) \leftarrow \mathsf{S}_1(1^{\lambda})$ and a proof π is computed by running $\mathsf{S}_2^{\Psi}(\Omega, \cdot, tk)$. If D distinguishes $\mathsf{Tamper}^{\mathrm{cnmlr}}_{\mathsf{A},x}$ and $\mathsf{Tamper}^{\mathrm{cnmlr}}_{\mathsf{A},y}$ but does not not distinguish $\mathsf{Tamper}'_{\mathsf{A},x}$ and $\mathsf{Tamper}'_{\mathsf{A},y}$, then one can use D, A, x and y to break the non-interactive zero knowledge property of the proof system with non-negligible advantage. By a standard argument, this implies:

$$\left|\Pr\left[\mathsf{D}(\Omega, \mathsf{Tamper}'_{\mathsf{A},x}) = 1\right] - \Pr\left[\mathsf{D}(\Omega, \mathsf{Tamper}'_{\mathsf{A},y}) = 1\right]\right| > \varepsilon/2.$$

A mental experiment. Let $q = \mathsf{poly}(\lambda)$ be the number of queries A asks to $\mathcal{O}_{\mathsf{cnm}}((X_0, X_1), \cdot)$; here (X_0, X_1) is the target encoding of some value x (to be specified later). We start by considering a mental experiment where a simulator S (running A) is given one complete side X_b of the encoding

Simulator $\mathsf{S}^{\mathcal{O}^{\zeta}(X_{1-b})}(X_b;r)$

For a value $b \in \{0,1\}$ and a target encoding (X_0, X_1), the algorithm has access to X_b and $\mathcal{O}^{\zeta}(X_{1-b})$, and is defined as follows:

- Define (initially empty) vectors $\boldsymbol{\Theta}_b$, $\boldsymbol{\Lambda}_b$ and initialize $\mathsf{A}(r)$.
- Repeat the following for all $j = 1, \ldots, q$:
  1. Receive $(\mathsf{L}_0^{(j)}, \mathsf{L}_1^{(j)})$ from $\mathsf{A}(r)$ (if any). Compute $\Lambda_b^{(j)} = \mathsf{L}_b(X_b)$ and forward L_{1-b} to $\mathcal{O}^{\zeta}(X_{1-b})$ receiving back a value $\Lambda_{1-b}^{(j)}$. Set $\boldsymbol{\Lambda}_b[j] := (\Lambda_0^{(j)}, \Lambda_1^{(j)})$ and give $(\Lambda_0^{(j)}, \Lambda_1^{(j)})$ to $\mathsf{A}(r)$.
  2. Receive $(f_0^{(j)}, f_1^{(j)})$ from $\mathsf{A}(r)$ and let $X'_b = f_b^{(j)}(X_b) = (s'_b, h'_{1-b}, \pi'_{1-b}, \pi'_b)$.
  3. Set the value $\boldsymbol{\Theta}_b[j]$ as follows:
     (a) (Type A query.) If $X_b = X'_b$ define $\boldsymbol{\Theta}_b[j] := \mathsf{same}^{\star}$ and return $\mathsf{same}^{\star}$ to $\mathsf{A}(r)$.
     (b) (Type B query.) Else if $X_b \neq X'_b$ but the local check on X'_b fails (i.e., if $\mathsf{Verify}^{h'_1}(\Omega, h'_0, \pi'_0)$ or $\mathsf{Verify}^{h'_0}(\Omega, h'_1, \pi'_1)$ output `reject`), define $\boldsymbol{\Theta}_b[j] := \perp_b$ (where $\perp_0$, $\perp_1$ are special symbols such that $\perp_0 \neq \perp_1$) and return $\perp$ to $\mathsf{A}(r)$.
     (c) (Type C query.) Else if $\pi'_{1-b} = \pi_{1-b}$ define $\boldsymbol{\Theta}_b[j] := \perp_b$ and return $\perp$ to $\mathsf{A}(r)$.
     (d) (Type D query.) Else run $s'_{1-b} \leftarrow \mathsf{Ext}^{h'_b}(\Omega, (h'_{1-b}, \pi'_{1-b}), ek)$ and define $\boldsymbol{\Theta}_b[j] := (X'_0, X'_1)$ where $X'_b = (s'_b, h'_{1-b}, \pi'_{1-b}, \pi'_b)$, $X'_{1-b} = (s'_{1-b}, h'_b, \pi'_b, \pi'_{1-b})$; return (X'_0, X'_1) to $\mathsf{A}(r)$.
- Output $out_{\mathsf{S},b} = (\boldsymbol{\Lambda}_b, \boldsymbol{\Theta}_b)$.

Figure 6.1: The simulator S of experiment $\mathsf{Tamper}''_{\mathsf{S}}(b)$

(for some $b \in \{0,1\}$), and has oracle access to $\mathcal{O}^{\zeta}(X_{1-b})$. The simulator outputs vectors $(\boldsymbol{\Lambda}_b, \boldsymbol{\Theta}_b)$ and is described in Figure 6.1.

For a message $x \in \{0,1\}^k$ and a bit $b \in \{0,1\}$, consider the following experiment featuring the above PPT simulator S:

$$\mathsf{Tamper}''_{\mathsf{S},x}(\lambda, b) = \left\{ \begin{array}{c} \Omega \leftarrow \mathsf{Init}(1^{\lambda}); (X_0, X_1) \leftarrow \mathsf{Enc}'(\Omega, x); \\ \mathsf{out}_{\mathsf{S},b} \leftarrow \mathsf{S}^{\mathcal{O}^{\zeta}(X_{1-b})}(X_b); \text{Output: } \mathsf{out}_{\mathsf{S},b} \end{array} \right\}.$$

Algorithm Enc' is identical to Enc, but the NIZKs are replaced with simulated NIZKs (as in experiment $\mathsf{Tamper}'_{\mathsf{A}}$). We will compare an execution of experiment $\mathsf{Tamper}'_{\mathsf{A}}$ with an execution of experiment $\mathsf{Tamper}''_{\mathsf{S}}(b)$, where both A and S run with the same random tape r and attack the same target encoding (X_0, X_1) of x. Denote with

$$\mathsf{view}_{\mathsf{A}} = ((\boldsymbol{\Lambda}[0], \boldsymbol{\Theta}[0]), \ldots, (\boldsymbol{\Lambda}[q], \boldsymbol{\Theta}[q]))$$

the view of adversary $\mathsf{A}(r)$ (i.e., A with random tape r) tampering with encoding (X_0, X_1) in $\mathsf{Tamper}'_{\mathsf{A}}$. We write $\mathsf{view}_{\mathsf{A}}^{(j)}$ for the view until round j.

Similarly, let[8]

$$\mathsf{view}_{\mathsf{S},b} = ((\boldsymbol{\Lambda}_b[0], \boldsymbol{\Theta}_b[0]), \ldots, (\boldsymbol{\Lambda}_b[q], \boldsymbol{\Theta}_b[q]))$$

be the view of produced by $\mathsf{S}(X_b; r)$ (i.e. S running A with random tape r) in $\mathsf{Tamper}''_{\mathsf{S}}(b)$. The first lemma says that, with overwhelming probability over the choice of r and the coin tosses to sample the encoding, the vectors $\mathsf{view}_{\mathsf{A}}$ and $\mathsf{view}_{\mathsf{S}}$ are identical *before a self-destruct happens.*

[8] Note that A can leak at most ζ bits, so some of the values $\boldsymbol{\Lambda}[j]$ and $\boldsymbol{\Lambda}_b[j]$ are empty.

Lemma 6.5.1. *For all x and all PPT adversaries A, let $j^* \in [q]$ be the smallest index such that A generates a self-destruct in $\mathsf{Tamper}'_{\mathsf{A},x}$. Denote with $\mathsf{view}_{\mathsf{A}}$ (resp. $\mathsf{view}_{\mathsf{S},b}$) the view of $\mathsf{A}(r)$ (resp. $\mathsf{S}(r)$) running in experiment $\mathsf{Tamper}'_{\mathsf{A},x}$ (resp. $\mathsf{Tamper}''_{\mathsf{S},x}(b)$) with target encoding (X_0, X_1). Then, for all $b \in \{0,1\}$, we have that $\mathsf{view}_{\mathsf{A}}^{(j^*-1)} = \mathsf{view}_{\mathsf{S},b}^{(j^*-1)}$ with overwhelming probability over the choice of the randomness r and the coin tosses to sample the encoding.*

The second lemma says that the values contained in the vectors $\mathbf{\Theta}_0$ and $\mathbf{\Theta}_1$ produced by $\mathsf{S}(r)$ in a run of $\mathsf{Tamper}''_{\mathsf{S}}(b)$ with encoding (X_0, X_1) are identical until the coordinate corresponding to the self-destruct index (with overwhelming probability).

Lemma 6.5.2. *For all x and for all PPT adversaries A, let $j^* \in [q]$ be the smallest index such that $\mathbf{\Theta}[j^*] = \perp$ in $\mathsf{Tamper}'_{\mathsf{A},x}(\lambda)$. Denote with $\mathsf{view}_{\mathsf{S},b} = (\mathbf{\Lambda}_b, \mathbf{\Theta}_b)$ the view of $\mathsf{S}(r)$ running in experiment $\mathsf{Tamper}''_{\mathsf{S},x}(b)$ with target encoding (X_0, X_1). Then, the following holds with overwhelming probability over the choice of the randomness r and the coin tosses to sample the encoding:*

(i) $\mathbf{\Theta}_0[j] = \mathbf{\Theta}_1[j]$, $\forall 1 \leq j \leq j^* - 1$.

(ii) $\mathbf{\Theta}_0[j^*] \neq \mathbf{\Theta}_1[j^*]$.

Locating the self-destruct point. Consider now the experiment $\mathsf{Tamper}'_{\mathsf{A}}$ and let (X_0, X_1) be the encoding that is computed at the beginning of the experiment. We define an algorithm $\mathsf{F}^{\mathcal{O}^\zeta(X_0),\mathcal{O}^\zeta(X_1)}(r)$ which is given access to oracles $\mathcal{O}^\zeta(X_0), \mathcal{O}^\zeta(X_1)$ and finds the index j^* where $\mathsf{A}(r)$ provokes a self-destruct in $\mathsf{Tamper}'_{\mathsf{A}}$. The main idea will be to compute the vectors $\mathbf{\Theta}_0$ and $\mathbf{\Theta}_1$ inside the leakage oracles (by running the simulator S of Figure 6.1), and then leak the first position where the vectors $\mathbf{\Theta}_0$ and $\mathbf{\Theta}_1$ are different. (By Lemma 6.5.2, this is the right index with overwhelming probability.) Note that the latter can be done with logarithmic amount of adaptive leakage, e.g. by using a standard binary search algorithm, once $\mathbf{\Theta}_0$ and $\mathbf{\Theta}_1$ are *defined*.

A technical difficulty is that we cannot run the simulator (say) on X_0 because S needs to access $\mathcal{O}^\zeta(X_1)$, but we do not have access to such oracle inside the leakage function. To solve this problem, F will attempt to run S alternately on X_0, X_1 and stop every time it encounters a leakage query it cannot answer. (Looking ahead this comes at the price of some loss in the leakage bound but, as we show below, the loss is no more than 2ζ bits.) After a finite number of steps all the leakages will be known and S can be run until the end on both sides. At this point, F can perform the binary search (again using adaptive leakage queries to $\mathcal{O}^\zeta(X_0)$, $\mathcal{O}^\zeta(X_1)$) in order to learn j^*. A formal description of F is given in Figure 6.2. We prove the following property for F.

Lemma 6.5.3. *For all x and for all PPT adversaries A, let $j^* \in [q]$ be the smallest index such that $\mathbf{\Theta}[j^*] = \perp$ in $\mathsf{Tamper}'_{\mathsf{A},x}(\lambda)$. Then, algorithm F outputs j^* with overwhelming probability. Moreover F runs in polynomial time and requires a total of $4\zeta + 2\lambda\lceil\log(q)\rceil$ bits of leakage.*

Proof. The fact that F outputs the correct j^* follows from Lemma 6.5.1 and Lemma 6.5.2. Indeed, assuming that F terminates, we have that the views

$$\mathsf{view}_{\mathsf{S},b}^{(j^*-1)} = (\mathbf{\Lambda}_b, \mathbf{\Theta}_b[1], \ldots, \mathbf{\Theta}_b[j^*-1]) \qquad \text{and} \qquad \mathsf{view}_{\mathsf{A}}^{(j^*-1)} = (\mathbf{\Lambda}, \mathbf{\Theta}[1], \ldots, \mathbf{\Theta}[j^*-1])$$

(cf. step 4a in the description of F), are identical with overwhelming probability (by Lemma 6.5.1). This implies that j^* can be computed as the first entry where $\mathbf{\Theta}_0$ and $\mathbf{\Theta}_1$ are different (by Lemma 6.5.2), which is exactly what the algorithm does (cf. step 5a—5c in the description of F) exploiting the fact that the hash function is collision resistant.

Algorithm $\mathsf{F}^{\mathcal{O}^\zeta(X_0),\mathcal{O}^\zeta(X_1)}(r)$

For a target encoding (X_0, X_1), the algorithm has access to oracles $\mathcal{O}^\zeta(X_0)$, $\mathcal{O}^\zeta(X_1)$ and is defined as follows:

1. **Setup:** Let $\mathcal{H}$ be a set of collision resistant hash functions. Set $j^* := 0$ and $\mathbf{\Lambda}_0, \mathbf{\Lambda}_1 = \emptyset$.
2. **Prepare the leakage to obtain $\mathbf{\Theta}_0$ and $\mathbf{\Theta}_1$:** Define the following leakage query $\mathsf{L}_b(X_b, \mathbf{\Lambda}_0, \mathbf{\Lambda}_1)$, with hard-wired values $(\mathbf{\Lambda}_0, \mathbf{\Lambda}_1)$:
   (a) Attempt to run $\mathsf{S}^{\mathcal{O}^\zeta(X_{1-b})}(X_b; r)$.
   (b) Answer all leakage queries on X_{1-b} by using the next value in $\mathbf{\Lambda}_{1-b}$.
   (c) In case no such value is found, terminate and output ``not done'' together with any value S leaked on X_b which is not already contained in $\mathbf{\Lambda}_b$. Update the vector $\mathbf{\Lambda}_b$ accordingly.
   (d) In case all leakage queries on X_{1-b} can be answered (i.e. S was run successfully), output ``done''.
3. **Loop to obtain $\mathbf{\Theta}_0$ and $\mathbf{\Theta}_1$:** Query alternately $\mathcal{O}^\zeta(X_0)$ with $\mathsf{L}_0(X_0, \mathbf{\Lambda}_0, \mathbf{\Lambda}_1)$ and $\mathcal{O}^\zeta(X_1)$ with $\mathsf{L}_1(X_1, \mathbf{\Lambda}_0, \mathbf{\Lambda}_1)$ until they both output ``done''.
4. **Prepare the leakage to obtain j^*:** Sample a hash function $H_t \leftarrow \mathcal{H}$. Define the following leakage query $\mathsf{L}_b(X_b, i_{\max}, i_{\min})$ depending on two (hard-wired) indexes $i_{\max}, i_{\min} \in [q]$:
   (a) Run $\mathsf{S}^{\mathcal{O}^\zeta(X_{1-b})}(X_b; r)$. (Note that this can be done without having access to $\mathcal{O}^\zeta(X_{1-b})$, as all the leakages on X_{1-b} are already contained in $\mathbf{\Lambda}_{1-b}$). Let $\mathbf{\Theta}_b$ be the vector defined by S.
   (b) Return $H_t(\mathbf{\Theta}_b[i_{\min}], \ldots, \mathbf{\Theta}_b[i_{\max}])$.
5. **Loop for the binary search:** Set $i_{\max} = 1$, $i_{\min} = q$ and repeat the following until $i_{\max} = i_{\min}$:
   (a) Let $i_{\text{mid}} := \left\lfloor \frac{i_{\min} + i_{\max}}{2} \right\rfloor$.
   (b) Forward $\mathsf{L}_0(X_0, i_{\text{mid}}, i_{\min})$ to $\mathcal{O}^\zeta(X_0)$ and $\mathsf{L}_1(X_1, i_{\text{mid}}, i_{\min})$ to $\mathcal{O}^\zeta(X_1)$; denote with Λ_0^*, Λ_1^* the answers from the oracles.
   (c) If $\Lambda_0^* \neq \Lambda_1^*$ set $i_{\max} := i_{\text{mid}} - 1$, otherwise set $i_{\min} = i_{\text{mid}} + 1$.
6. **Output:** Define $j^* := i_{\max} = i_{\min}$; return $(j^*, \mathbf{\Lambda}_0, \mathbf{\Lambda}_1)$.

Figure 6.2: The algorithm F locating the self-destruct index j^*

It remains to show that F runs in polynomial time. This follows by inspection of the description of F in Figure 6.2 as:

1. The loop of step 3 ends after at most 2ζ steps (since S cannot ask more than ζ adaptive leakage queries on each side).
2. The loop of step 5 is a standard binary search algorithm which requires a logarithmic number of steps.

Notice that F requires at most 4ζ bits of leakage (2ζ bits from the oracles $\mathcal{O}^\zeta(X_0)$ and $\mathcal{O}^\zeta(X_1)$ and at most 2ζ more bits for each of the times S returns ``not done'') plus $2\lambda\lceil\log(q)\rceil$ bits for the binary search as the range of the hash function is λ bits.

□

Description of the reduction. We construct D' and A' that for infinitely many λ break the security of the LRS with non-negligible advantage. To this end, they are given access to D and A that win in game $\mathsf{Tamper}'_{\mathsf{A}}$ with non-negligible advantage (recall that in $\mathsf{Tamper}'_{\mathsf{A}}$ we have replaced

the NIZKs with simulated NIZKs). First, A' samples randomness r and runs the sub-routine F of Figure 6.2 with this randomness. By Lemma 6.5.3 the algorithm F requires at most $4\zeta+2\lambda\lceil\log(q)\rceil$ bits of leakage to learn the position j^* of the query where $\mathsf{A}(r)$ provokes the self-destruct. Then, A' runs A as subroutine using the same random coins r that were used to identify the self-destruct position. We emphasize that using the same r is crucial for the reduction to work. A formal description of the reduction follows:

1. **Simulate initial encoding:** Sample uniformly at random a key $t \leftarrow \{0,1\}^\lambda$ for the hash function and run $(\Omega, tk, ek) \leftarrow \mathsf{S}_1(1^\lambda)$ to generate a CRS and the corresponding simulation trapdoor tk. With access to the target leakage oracle $(\mathcal{O}^{\zeta'}(s_0), \mathcal{O}^{\zeta'}(s_1))$ obtain $h_0 = H_t(s_0)$ and $h_1 = H_t(s_1)$; set the labels $\Psi_0 = h_0$ and $\Psi_1 = h_1$. Notice that given (h_b, Ψ_{1-b}) it is easy to simulate π_b, i.e., we can compute $\pi_b \leftarrow \mathsf{S}_2^{\Psi_{1-b}}(\Omega, h_b, tk)$.

2. **Learn the self-destruct point:** Sample random coins $r \leftarrow \{0,1\}^*$ and run $\mathsf{F}^{\mathcal{O}^\zeta(X_0),\mathcal{O}^\zeta(X_1)}(r)$. Note that each leakage query to $\mathcal{O}^\zeta(X_b)$ can be translated to a leakage query to $\mathcal{O}^{\zeta'}(s_b)$ by hard-wiring $(h_{1-b}, \pi_{1-b}, \pi_b)$. Let $(j^*, \mathbf{\Lambda}_0, \mathbf{\Lambda}_1)$ be the output of F.

3. **Simulate tampering queries:** At this point A' is done with the leakage queries and picks $\theta = 0$, asking to reveal s_0. Hence, A' runs $\mathsf{A}(r)$ simulating the answer for its tampering queries as follows:

   (a) For all $j = 1, \ldots, j^*$ given tampering query $(f_0^{(j)}, f_1^{(j)})$ forward to A the value $\mathbf{\Theta}_0$ as computed in step 3a—3d of the description of $\mathsf{S}^{\mathcal{O}^\zeta(X_1)}(X_0; r)$ (cf. Figure 6.1). Notice that to run S no further leakage query is needed, as the answers to $\mathsf{A}(r)$'s leakage queries are already contained in $(\mathbf{\Lambda}_0, \mathbf{\Lambda}_1)$.

   (b) For all $j > j^*$ given tampering query $(f_0^{(j)}, f_1^{(j)})$ forward to A the value $\perp$.

4. **Guess:** Output whatever D does.

For the analysis, note that by Lemma 6.5.3 the reduction runs in polynomial time and requires at most $4\zeta+(2\lambda+1)\lceil\log(q)\rceil$ bits of leakage. Moreover, by Lemma 6.5.1, the simulation of the answers to A's tampering queries is identical to the distribution A would have seen in the real experiment (with overwhelming probability). Since we are assuming that D, A have a non-negligible advantage, we have that D', A' have also a non-negligible advantage, contradicting the assumption that the LRS scheme is secure. This finishes the proof.

6.5.1 Proof of Lemma 6.5.1

Before proving the lemma, we establish some notation that we will use also in the proof of Lemma 6.5.2. Denote with $X_0 = (s_0, h_1, \pi_1, \pi_0)$, $X_1 = (s_1, h_0, \pi_0, \pi_1)$ the target encoding that is sampled at the beginning of the two experiments $\mathsf{Tamper}'_{\mathsf{A}}$ and $\mathsf{Tamper}''_{\mathsf{S}}$. Given a tampering query $(f_0^{(j)}, f_1^{(j)})$ we write $X'_0 = f_0^{(j)}(X_0)$, $X'_1 = f_1^{(j)}(X_1)$ for the tampered codeword in experiment $\mathsf{Tamper}'_{\mathsf{A}}$. For some value $b \in \{0,1\}$, we write $\mathsf{S}(b)$ as a short hand of $\mathsf{S}^{\mathcal{O}^\zeta(X_{1-b})}(X_b)$. Let $X'_{b,b} = f_b^{(j)}(X_b)$ be the tampered half computed by the simulator $\mathsf{S}(b)$ in experiment $\mathsf{Tamper}''_{\mathsf{S}}(b)$. Recall that in case $(f_0^{(j)}, f_1^{(j)})$ is of type D, then according to the description of S (c.f. step 3d in Figure 6.1), then the simulator "extracts" the other half of the codeword which we denote by $X'_{b,1-b}$.

On the probability space. In each experiment there are two sources of randomness: (i) the randomness of the encoding process to generate the target encoding (X_0, X_1) and (ii) the randomness r of the adversary. We have to show that with overwhelming probability over the choice of this randomness the simulator $\mathsf{S}(r)$ produces exactly the same view as $\mathsf{A}(r)$ would have seen in the real experiment. In the proof we will consider some "bad" event over the above randomness space and prove that such event happens with negligible probability to obtain the statement. In what follows all probabilities are taken over the above randomness space.

We emphasize that we cannot directly assume that $(f_0^{(j)}, f_1^{(j)})$ is the same in $\mathsf{Tamper}'_{\mathsf{A}}$ and $\mathsf{Tamper}''_{\mathsf{S}}$, as the tampering functions are chosen adaptively by A depending on the view in the two experiments. However, since in both the experiments A is run with the same random tape r, we have that $(f_0^{(1)}, f_1^{(1)})$ is the same in $\mathsf{Tamper}'_{\mathsf{A}}$ and $\mathsf{Tamper}''_{\mathsf{S}}$ as the choice of the first tampering query depends only on the distribution of the CRS Ω (which is the same in the two worlds).

Below, we state that whenever the simulator S extracts a value $X'_{b,1-b}$, then the resulting encoding is valid with overwhelming probability.

Claim 6.5.4. *Whenever $\mathsf{S}(b)$ sets $\boldsymbol{\Theta}_b[j] = (X'_{b,b}, X'_{b,1-b})$ in response to a tampering query $(f_0^{(j)}, f_1^{(j)})$ for some $1 \le j \le q$, then $\boldsymbol{\Theta}_b[j]$ is a valid encoding with overwhelming probability.*

Proof. We make the proof for $b = 0$; the proof for $b = 1$ is similar. Following the description of $\mathsf{S}(0)$, we observe that the experiment $\mathsf{Tamper}''_{\mathsf{S}}(0)$ sets $\boldsymbol{\Theta}_0[j] = (X'_{0,0}, X'_{0,1})$ only in the step 3d (i.e., when the j-th tampering query is of type D). Let $X'_{0,0} = (s'_0, h'_1, \pi'_1, \pi'_0)$ and $X'_{0,1} = (s'_1, h'_0, \pi'_0, \pi'_1)$, where $s'_1 \leftarrow \mathsf{Ext}^{h'_0}(\Omega, (h'_1, \pi'_1), ek)$.[9] Now, for the sake of contradiction, assume that $(X'_{0,0}, X'_{0,1})$ is invalid. By definition of the decoding algorithm (c.f. Section 6.4), we observe that the only possibility is that the extracted value s'_1 does not match the statement h'_1, i.e. $h'_1 \neq H_t(s'_1)$. By simulation extractability of the NIZK, this can only happen with negligible probability. Therefore we conclude that $(X'_{0,0}, X'_{0,1})$ is a valid codeword with overwhelming probability, as desired. □

We now turn to the proof of Lemma 6.5.1.

Lemma 6.5.1. *For all x and all PPT adversaries A, let $j^* \in [q]$ be the smallest index such that A generates a self-destruct in $\mathsf{Tamper}'_{\mathsf{A},x}$. Denote with $\mathsf{view}_{\mathsf{A}}$ (resp. $\mathsf{view}_{\mathsf{S},b}$) the view of $\mathsf{A}(r)$ (resp. $\mathsf{S}(r)$) running in experiment $\mathsf{Tamper}'_{\mathsf{A},x}$ (resp. $\mathsf{Tamper}''_{\mathsf{S},x}(b)$) with target encoding (X_0, X_1). Then, for all $b \in \{0,1\}$, we have that $\mathsf{view}_{\mathsf{A}}^{(j^*-1)} = \mathsf{view}_{\mathsf{S},b}^{(j^*-1)}$ with overwhelming probability over the choice of the randomness r and the coin tosses to sample the encoding.*

Proof. We make the proof for $b = 0$; the proof for $b = 1$ is similar. Without loss of generality we will always assume that in the j-th round the adversary $\mathsf{A}(r)$ asks a leakage query and a tampering query. The corresponding output of the experiments in this round is denoted by $\mathsf{out}_{\mathsf{A}}^{(j)} = (\boldsymbol{\Lambda}[j], \boldsymbol{\Theta}[j])$ in $\mathsf{Tamper}'_{\mathsf{A}}$ and $\mathsf{out}_{\mathsf{S},0}^{(j)} = (\boldsymbol{\Lambda}_0[j], \boldsymbol{\Theta}_0[j])$ in $\mathsf{Tamper}''_{\mathsf{S}}(0)$. Denote the distribution of the outputs of the two experiments $\mathsf{Tamper}''_{\mathsf{S}}(0)$ and $\mathsf{Tamper}'_{\mathsf{A}}$ until round j as

$$\mathsf{view}_{\mathsf{S},0}^{(j)} = (\boldsymbol{\Lambda}_0[1], \boldsymbol{\Theta}_0[1], \ldots, \boldsymbol{\Lambda}_0[j], \boldsymbol{\Theta}_0[j]) \qquad \text{and} \qquad \mathsf{view}_{\mathsf{A}}^{(j)} = (\boldsymbol{\Lambda}[1], \boldsymbol{\Theta}[1], \ldots, \boldsymbol{\Lambda}[j], \boldsymbol{\Theta}[j]) .$$

[9] Looking at the description of the simulator, the hashes (h'_0, h'_1) and the proofs (π'_0, π'_1) are the same in $X'_{0,0}$ and in $X'_{0,1}$.

We compute the following:

$$\begin{aligned}
&\Pr\left[\mathsf{view}_{\mathsf{A}}^{(j^*-1)} \neq \mathsf{view}_{\mathsf{S},0}^{(j^*-1)}\right] \\
&\leq \Pr\left[\exists j \in [j^*-1] : (\mathsf{out}_{\mathsf{A}}^{(j)} \neq \mathsf{out}_{\mathsf{S},0}^{(j)}) \wedge (\mathsf{view}_{\mathsf{A}}^{(j-1)} = \mathsf{view}_{\mathsf{S},0}^{(j-1)})\right] \\
&\leq \sum_{j=1}^{j^*-1} \Pr\left[(\mathsf{out}_{\mathsf{A}}^{(j)} \neq \mathsf{out}_{\mathsf{S},0}^{(j)}) \wedge (\mathsf{view}_{\mathsf{A}}^{(j-1)} = \mathsf{view}_{\mathsf{S},0}^{(j-1)})\right] \\
&\leq \sum_{j=1}^{j^*-1} \Pr\left[\mathsf{out}_{\mathsf{A}}^{(j)} \neq \mathsf{out}_{\mathsf{S},0}^{(j)} \mid (\mathsf{view}_{\mathsf{A}}^{(j-1)} = \mathsf{view}_{\mathsf{S},0}^{(j-1)})\right] \cdot \Pr\left[(\mathsf{view}_{\mathsf{A}}^{(j-1)} = \mathsf{view}_{\mathsf{S},0}^{(j-1)})\right] \\
&\leq \sum_{j=1}^{j^*-1} \Pr\left[\mathsf{out}_{\mathsf{A}}^{(j)} \neq \mathsf{out}_{\mathsf{S},0}^{(j)} \mid (\mathsf{view}_{\mathsf{A}}^{(j-1)} = \mathsf{view}_{\mathsf{S},0}^{(j-1)})\right] \\
&\leq \mathsf{negl}(\lambda)
\end{aligned}$$

The last inequality follows from Lemma 6.5.5 (which we prove below), and the fact the sum is taken only for polynomially many values since j^* is polynomially bounded. □

Lemma 6.5.5. *For all $j \in [j^*-1]$, we have that*

$$\Pr\left[\mathsf{out}_{\mathsf{A}}^{(j)} \neq \mathsf{out}_{\mathsf{S},0}^{(j)} \mid (\mathsf{view}_{\mathsf{A}}^{(j-1)} = \mathsf{view}_{\mathsf{S},0}^{(j-1)})\right] \leq \mathsf{negl}(\lambda).$$

Proof. Fix any $j \in [j^*-1]$. Let $\textsc{Good}^{(j-1)}$ denote the event that $(\mathsf{view}_{\mathsf{A}}^{(j-1)} = \mathsf{view}_{\mathsf{S},0}^{(j-1)})$. Note that, since we are comparing the variables $\mathsf{out}_{\mathsf{A}}^{(j)}$ and $\mathsf{out}_{\mathsf{S},0}^{(j)}$ conditioned on the event $\textsc{Good}^{(j-1)}$, in both the experiments $\mathsf{Tamper}'_{\mathsf{A}}$ and $\mathsf{Tamper}''_{\mathsf{S}}(0)$ the leakage query $(\mathsf{L}_0^{(j)}, \mathsf{L}_1^{(j)})$ and the tampering query $(f_0^{(j)}, f_1^{(j)})$ issued by adversary $\mathsf{A}(\mathbf{V}; r)$ (where $\mathbf{V}$ is either $\mathsf{view}_{\mathsf{A}}^{(j-1)}$ or $\mathsf{view}_{\mathsf{S},0}^{(j-1)}$) in the j-th round are the same.

Consider the following events in the experiment $\mathsf{Tamper}''_{\mathsf{S}}(0)$.

- $\textsc{Bad}_1^{(j)}$: The event becomes true when $(f_0^{(j)}, f_1^{(j)})$ is of type A *and* we have $f_1^{(j)}(X_1) \neq X_1$.
- $\textsc{Bad}_2^{(j)}$: The event becomes true when $(f_0^{(j)}, f_1^{(j)})$ is of type B.
- $\textsc{Bad}_3^{(j)}$: The event becomes true when $(f_0^{(j)}, f_1^{(j)})$ is of type C.
- $\textsc{Bad}_4^{(j)}$: The event becomes true when $(f_0^{(j)}, f_1^{(j)})$ is of type D, *and* either of the following two facts happens:
  - The encoding $(X'_{0,0}, X'_{0,1})$ is not valid; *or*
  - The encoding $(X'_{0,0}, X'_{0,1})$ is valid, but we have $f_1^{(j)}(X_1) \neq X'_{0,1}$ (recall that $X'_{0,1}$ is the "extracted half").

  Define the union of all those four events by $\textsc{Bad}^{(j)} = \bigvee_{i=1}^{4} \textsc{Bad}_i^{(j)}$.

The next four claims show that the probability of each event $\textsc{Bad}_i^{(j)}$ is negligible, conditioned on the fact that the views until round $j-1$ are equal in the real and the simulated experiment.

Claim 6.5.6. $\Pr\left[\text{BAD}_1^{(j)} | \text{GOOD}^{(j-1)}\right] \leq \mathsf{negl}(\lambda)$.

Proof. Recall the definition of the event $\text{BAD}_1^{(j)}$: The event becomes true when $(f_0^{(j)}, f_1^{(j)})$ is of type A *and* we have $f_1^{(j)}(X_1) \neq X_1$ in $\mathsf{Tamper}''_{\mathsf{S}}(0)$. By definition of type A query (cf. Figure 6.1), we have that $X'_{0,0} = f_0^{(j)}(X_0) = X_0$ and $\mathsf{Tamper}''_{\mathsf{S}}(0)$ would output $\mathsf{same}^\star$ in this round. Now, since we are conditioning on the the event $\text{GOOD}^{(j-1)}$, we get the same tampering query in this round which implies $X'_0 = X'_{0,0} = X_0$ and $X'_1 = f_1^{(j)}(X_1) \neq X_1$. As (X_0, X_1) and (X_0, X'_1) are both valid, (X_0, X_1, X'_1) violates uniqueness (c.f. Lemma 6.4.1) of our encoding scheme. □

Claim 6.5.7. $\Pr\left[\text{BAD}_2^{(j)} | \text{GOOD}^{(j-1)}\right] \leq \mathsf{negl}(\lambda)$.

Proof. Recall the definition of the event $\text{BAD}_2^{(j)}$: The event becomes true when $(f_0^{(j)}, f_1^{(j)})$ is of type B in $\mathsf{Tamper}''_{\mathsf{S}}(0)$. Let $X'_{0,0} = (s'_0, h'_1, \pi'_1, \pi'_0)$ be the value computed in $\mathsf{Tamper}''_{\mathsf{S}}(0)$. From the description of S we observe that in this case $X'_{0,0} \neq X_0$ and at least one of the proofs π'_0, π'_1 does not verify correctly.

Now, conditioning on $\text{GOOD}^{(j-1)}$, we get $X'_0 = X'_{0,0}$ in $\mathsf{Tamper}'_{\mathsf{A}}$. In this case the experiment would output $\perp$, resulting in a self-destruct, which contradicts our assumption that $j < j^*$. □

Claim 6.5.8. $\Pr\left[\text{BAD}_3^{(j)} | \text{GOOD}^{(j-1)}\right] \leq \mathsf{negl}(\lambda)$.

Proof. Recall the definition of the event $\text{BAD}_3^{(j)}$: The event becomes true when $(f_0^{(j)}, f_1^{(j)})$ is of type C, i.e. $X'_{0,0} = (s'_0, h'_1, \pi'_1, \pi'_0) \neq X_0$ and $\pi'_1 = \pi_1$ in $\mathsf{Tamper}''_{\mathsf{S}}(0)$. Again conditioning on $\text{GOOD}^{(j-1)}$, we get that $X'_0 = (s'_0, h'_1, \pi_1, \pi'_0)$ in $\mathsf{Tamper}'_{\mathsf{A}}$. In this experiment consider the other half of the tampered codeword, i.e. $X'_1 = (s'_1, h'_0, \pi'_0, \pi_1)$. Note that the proofs match, as otherwise we would get $\perp$ in $\mathsf{Tamper}'_{\mathsf{A}}$ which contradicts the fact that $j < j^*$.

Note that we consider a NIZK proof system with the following two properties: (i) different statements have different proofs; (ii) it supports public labels which are appended with the statements (c.f. Section 6.2.1). Using these properties we get that $\pi'_1 = \pi_1$ implies $h'_1 = h_1$ and $h'_0 = h_0$. Here we use the fact that π_1 is a proof of statement h_1 with label h_0. In the next step, by collision resistance of the hash function we get $s'_1 = s_1$ and $s'_0 = s_0$, with overwhelming probability.

So far, we have shown that $X'_0 = (s_0, h_1, \pi_1, \pi'_0)$ and $X'_1 = (s_1, h_0, \pi'_0, \pi_1)$ in $\mathsf{Tamper}'_{\mathsf{A}}$ (with overwhleming probability). Since we must have $X'_0 = X'_{0,0} \neq X_0$, we get $\pi'_0 \neq \pi_0$ and hence the proof π'_0 can be extracted. We claim that the above contradicts leakage resilience of the underlying LRS scheme. To see this, we construct an adversary A' against $(\mathsf{LRS}, \mathsf{LRS}^{-1})$ attacking a target encoding (s_0, s_1) given a pair $(f_0^{(j)}, f_1^{(j)})$ which provokes the event $\text{BAD}_3^{(j)}$. Adversary A' works as follows:

- First, A' queries its own leakage oracles $\mathcal{O}^\zeta(s_0), \mathcal{O}^\zeta(s_1)$ to get the hash values $H_t(s_0) = h_0$ and $H_t(s_1) = h_1$. Then it runs $(\Omega, tk, ek) \leftarrow \mathsf{S}_1(1^k)$ and computes the simulated proofs $\pi_0 \leftarrow \mathsf{S}_2^{h_1}(\Omega, h_0, tk)$ and $\pi_1 \leftarrow \mathsf{S}_2^{h_0}(\Omega, h_1, tk)$.

- Next, A' asks one additional leakage query to the oracle $\mathcal{O}^\zeta(s_0)$ with hard-wired values h_1, π_0, π_1, and ek, and hard-wired function $f_1^{(j)}$ as follows: Compute $X'_1 = f_1^{(j)}(X_1) = (s_1, h_0, \pi'_0, \pi_1)$; extract $s_0 \leftarrow \mathsf{Ext}^{h_1}(\Omega, (h_0, \pi'_0), ek)$ and output the first bit of the decoded value $(\mathsf{LRS}^{-1}(s_0, s_1))$.

Note that the above attack requires $2\lambda + 1 \leq \zeta'$ bits of leakage and clearly allows to break $(\mathsf{LRS}, \mathsf{LRS}^{-1})$ with overwhelming probability. This concludes the proof of claim. □

Claim 6.5.9. $\Pr\left[\textsc{Bad}_4^{(j)} | \textsc{Good}^{(j-1)}\right] \leq \mathsf{negl}(\lambda)$.

Proof. Recall the definition of the event $\textsc{Bad}_4^{(j)}$: The event becomes true when in $\mathsf{Tamper}''_{\mathsf{S}}(0)$, $(f_0^{(j)}, f_1^{(j)})$ is of type D *and* either of the following two facts happens:

- The encoding $(X'_{0,0}, X'_{0,1})$ is not valid; *or*
- The encoding $(X'_{0,0}, X'_{0,1})$ is valid, but we have $f_1^{(j)}(X_1) \neq X'_{0,1}$ which is actually the extracted half.

Note that by Claim 6.5.4, the encoding $(X'_{0,0}, X'_{0,1})$ must be valid with overwhelming probability. So, by a union bound, we can consider only the second case.

Conditioning on $\textsc{Good}^{(j-1)}$ implies that in $\mathsf{Tamper}'_{\mathsf{A}}$, $X'_0 = f_0^{(j)}(X_0) = X'_{0,0}$ and $X'_1 = f_1^{(j)}(X_1) \neq X'_{0,1}$. Moreover, (X'_0, X'_1) is a valid encoding pair as we are assuming $j < j^*$. Clearly this violates uniqueness, as in this case $X'_1 \neq X'_{0,1}$, concluding the proof. □

To conclude the proof of Lemma 6.5.5 we observe the following:

$$
\begin{aligned}
&\Pr\left[\mathsf{out}_{\mathsf{A}}^{(j)} \neq \mathsf{out}_{\mathsf{S},0}^{(j)} \mid (\mathsf{view}_{\mathsf{A}}^{(j-1)} = \mathsf{view}_{\mathsf{S},0}^{(j-1)})\right] \\
&= \Pr\left[\mathsf{out}_{\mathsf{A}}^{(j)} \neq \mathsf{out}_{\mathsf{S},0}^{(j)} \mid \textsc{Good}^{(j-1)}\right] \\
&\leq \Pr\left[\mathsf{out}_{\mathsf{A}}^{(j)} \neq \mathsf{out}_{\mathsf{S},0}^{(j)} \mid \overline{\textsc{Bad}^{(j)}} \wedge \textsc{Good}^{(j-1)}\right] + \Pr\left[\textsc{Bad}^{(j)} | \textsc{Good}^{(j-1)}\right] && (6.1) \\
&\leq \Pr\left[\mathsf{out}_{\mathsf{A}}^{(j)} \neq \mathsf{out}_{\mathsf{S},0}^{(j)} \mid \overline{\textsc{Bad}^{(j)}} \wedge \textsc{Good}^{(j-1)}\right] + \sum_{i=1}^{4} \Pr\left[\textsc{Bad}_i^{(j)} | \textsc{Good}^{(j-1)}\right] && (6.2) \\
&\leq \mathsf{negl}(\lambda). && (6.3)
\end{aligned}
$$

Eq. (6.1) follows from Bayes Theorem, Eq. (6.2) follows from the union bound and Eq. (6.3) from Claim 6.5.6—6.5.9 and Claim 6.5.10 (see below).

Claim 6.5.10. $\Pr\left[\mathsf{out}_{\mathsf{A}}^{(j)} \neq \mathsf{out}_{\mathsf{S},0}^{(j)} \mid \overline{\textsc{Bad}^{(j)}} \wedge \textsc{Good}^{(j-1)}\right] \leq \mathsf{negl}(\lambda)$.

Proof. Here we condition on both the events $\textsc{Good}^{(j-1)}$ and $\overline{\textsc{Bad}^{(j)}}$. Conditioning on $\textsc{Good}^{(j-1)}$ implies that $(\mathsf{L}_0^{(j)}, \mathsf{L}_1^{(j)})$ and $(f_0^{(j)}, f_1^{(j)})$ are the same in both $\mathsf{Tamper}'_{\mathsf{A}}$ and $\mathsf{Tamper}''_{\mathsf{S}}(0)$. Given that, we need to analyze the following leakage and tampering components: $\mathsf{out}_{\mathsf{A}}^{(j)} = (\mathbf{\Lambda}[j], \mathbf{\Theta}[j])$ and $\mathsf{out}_{\mathsf{S},0}^{(j)} = (\mathbf{\Lambda}_0[j], \mathbf{\Theta}_0[j])$. It is easy to observe that, S simulates the leakage correctly as it can compute $\mathsf{L}_0^{(j)}(X_0)$ directly and forward $\mathsf{L}_1^{(j)}$ to $\mathcal{O}^{\zeta}(X_1)$. This shows that $\mathbf{\Lambda}[j] = \mathbf{\Lambda}_0[j]$.

Now consider the following cases based on the type of the tampering query $(f_0^{(j)}, f_1^{(j)})$ in experiment $\mathsf{Tamper}''_{\mathsf{S}}(0)$:

- If $(f_0^{(j)}, f_1^{(j)})$ is of type A, then $\mathbf{\Theta}_0[j] = \mathsf{same}^\star$. Since $\overline{\textsc{Bad}^{(j)}} \wedge \textsc{Good}^{(j-1)}$ implies $\overline{\textsc{Bad}_1^{(j)}} \wedge \textsc{Good}^{(j-1)}$, we get $\mathbf{\Theta}[j] = \mathsf{same}^\star$ in the experiment $\mathsf{Tamper}'_{\mathsf{A}}$.
- Since $\overline{\textsc{Bad}^{(j)}} \wedge \textsc{Good}^{(j-1)}$ implies $\overline{\textsc{Bad}_2^{(j)}} \wedge \overline{\textsc{Bad}_3^{(j)}} \wedge \textsc{Good}^{(j-1)}$, the tampering query $(f_0^{(j)}, f_1^{(j)})$ is neither of type B nor of type C.

- If $(f_0^{(j)}, f_1^{(j)})$ is of type D, since $\overline{\text{BAD}^{(j)}} \wedge \text{GOOD}^{(j-1)}$ implies $\overline{\text{BAD}_4^{(j)}} \wedge \text{GOOD}^{(j-1)}$ we get $(X'_0, X'_1) = \boldsymbol{\Theta}[j] = \boldsymbol{\Theta}_0[j] = (X'_{0,0}, X'_{0,1})$ in both $\mathsf{Tamper}'_{\mathsf{A}}$ and $\mathsf{Tamper}''_{\mathsf{S}}(0)$.

We conclude that in every case we get $\boldsymbol{\Theta}[j] = \boldsymbol{\Theta}_0[j]$ conditioned on $\overline{\text{BAD}^{(j)}}$ and $\text{GOOD}^{(j-1)}$ which proves the claim. □

□

6.5.2 Proof of Lemma 6.5.2

Lemma 6.5.2. *For all x and for all PPT adversaries A, let $j^* \in [q]$ be the smallest index such that $\boldsymbol{\Theta}[j^*] = \perp$ in $\mathsf{Tamper}'_{\mathsf{A},x}(\lambda)$. Denote with $\mathsf{view}_{\mathsf{S},b} = (\boldsymbol{\Lambda}_b, \boldsymbol{\Theta}_b)$ the view of $\mathsf{S}(r)$ running in experiment $\mathsf{Tamper}''_{\mathsf{S},x}(b)$ with target encoding (X_0, X_1). Then, the following holds with overwhelming probability over the choice of the randomness r and the coin tosses to sample the encoding:*

(i) $\boldsymbol{\Theta}_0[j] = \boldsymbol{\Theta}_1[j]$, $\forall 1 \le j \le j^* - 1$.

(ii) $\boldsymbol{\Theta}_0[j^*] \neq \boldsymbol{\Theta}_1[j^*]$.

Proof. We compute the probability for which property (i) does not hold and show that this probability is negligible.

$$
\begin{aligned}
&\Pr\left[\exists j \in \{1,\ldots,j^*-1\} \ : \ \boldsymbol{\Theta}_0[j] \neq \boldsymbol{\Theta}_1[j]\right] \\
&\quad\le \sum_{j=1}^{j^*-1} \Pr\left[\boldsymbol{\Theta}_0[j] \neq \boldsymbol{\Theta}_1[j]\right] && (6.4)\\
&\quad\le \sum_{j=1}^{j^*-1} \Big(\Pr\left[\boldsymbol{\Theta}_0[j] \neq \boldsymbol{\Theta}_1[j] \mid \boldsymbol{\Theta}_0[j] = \boldsymbol{\Theta}[j]\right] \cdot \Pr\left[\boldsymbol{\Theta}_0[j] = \boldsymbol{\Theta}[j]\right] \\
&\qquad\qquad + \Pr\left[\boldsymbol{\Theta}_0[j] \neq \boldsymbol{\Theta}_1[j] \mid \boldsymbol{\Theta}_0[j] \neq \boldsymbol{\Theta}[j]\right] \cdot \Pr\left[\boldsymbol{\Theta}_0[j] \neq \boldsymbol{\Theta}[j]\right] \Big) \\
&\quad\le \sum_{j=1}^{j^*-1} \Big(\Pr\left[\boldsymbol{\Theta}_0[j] \neq \boldsymbol{\Theta}_1[j] \mid \boldsymbol{\Theta}_0[j] = \boldsymbol{\Theta}[j]\right] + \Pr\left[\boldsymbol{\Theta}_0[j] \neq \boldsymbol{\Theta}[j]\right] \Big) \\
&\quad\le \sum_{j=1}^{j^*-1} \sum_{b=1}^{2} \Pr\left[(\boldsymbol{\Theta}_b[j] \neq \boldsymbol{\Theta}[j]\right] && (6.5)\\
&\quad\le \mathsf{negl}(\lambda). && (6.6)
\end{aligned}
$$

Eq. (6.4) follow by a union bound. Eq. (6.6) follows from Lemma 6.5.1 and the fact that j^* is polynomially bounded. This concludes the proof of part (i).

Now we prove part (ii) of the lemma. Define the following event BAD', over the probability space of experiment $\mathsf{Tamper}''_{\mathsf{S}}(b)$: The event becomes true when in round j^*, we have $\boldsymbol{\Theta}_0[j^*] = \boldsymbol{\Theta}_1[j^*]$. In order to satisfy this condition, the only possible values $\boldsymbol{\Theta}_b[j^*]$ (for $b \in \{0,1\}$) can take are as follows:

1. $\mathbf{\Theta}_0[j^*] = \mathbf{\Theta}_1[j^*] = \mathsf{same}^\star$. By Lemma 6.5.1, we get that before round j^* the view of $\mathsf{A}(r)$ in $\mathsf{Tamper}'_{\mathsf{A}}$ is equal to the view of $\mathsf{A}(r)$ in $\mathsf{Tamper}''_{\mathsf{S}}(b)$ (with overwhelming probability). This implies that in round j^* the tampering query $(f_0^{(j^*)}, f_1^{(j^*)})$ is the same in both experiments. This gives $\mathbf{\Theta}[j^*] = \mathsf{same}^\star$, which contradicts the fact that j^* is the self-destruct round in $\mathsf{Tamper}'_{\mathsf{A}}$.

2. $\mathbf{\Theta}_0[j^*] = \mathbf{\Theta}_1[j^*] = (X'_0, X'_1)$. Note that (X'_0, X'_1) is valid with overwhelming probability (by Claim 6.5.4). Now due to the same reason as above Lemma 6.5.1 implies that $\mathbf{\Theta}[j^*] = (X'_0, X'_1)$ in $\mathsf{Tamper}'_{\mathsf{A}}$ with overwhelming probability, which is a contradiction to the fact that j^* is the self-destruct round.

Therefore using a union bound we can argue that $\Pr[\textsc{Bad}'] \leq \mathsf{negl}(\lambda)$ which concludes the proof of part (ii).

□

6.6 Application to Tamper Resilient Security

In this section we apply our notion of CNMLR codes to protect arbitrary functionalities against split-state tampering and leakage attacks. We handle the stateless functionalities and stateful functionalities separately and propose two separate compilers.

6.6.1 Stateless Functionality

The main idea is to transform the original functionality $\mathsf{G}(\mathsf{st}, \cdot)$ (where st is the secret state) into some "hardened" functionality $\mathsf{G}^h(\mathsf{st}', \cdot)$ via a CNMLR code Code. Previous transformations aiming to protect stateless functionalities [DPW10, LL12] required to freshly re-encode the state st each time the functionality is invoked. Our approach avoids the re-encoding of the state at each invocation, leading to a stateless transformation. This solves an open question from [DPW10]. Moreover we consider a setting where the encoded state is stored in a memory $(\mathcal{M}_0, \mathcal{M}_1)$ which is much larger than the size needed to store the encoding itself (say $|\mathcal{M}_0| = |\mathcal{M}_1| = \mu$ where μ is polynomial in the length of the encoding). When (perfect) erasures are not possible, this feature allows the adversary to make copies of the initial encoding and tamper continuously with it, and was not considered in previous models.

Let us formally define what it means to harden a stateless functionality.

Definition 6.6.1 (Stateless hardened functionality)**.** *Let* $\mathsf{Code} = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ *be a split-state encoding scheme in the CRS model, with* k *bits messages and* $2n$ *bits codewords. Let* $\mathsf{G} : \{0,1\}^k \times \{0,1\}^u \to \{0,1\}^v$ *be a* stateless *functionality with secret state* $\mathsf{st} \in \{0,1\}^k$*, and let* $\varphi \in \{0,1\}$ *be a public value initially set to zero. We define a* stateless *hardened functionality* $\mathsf{G}^h : \{0,1\}^{2\mu} \times \{0,1\}^u \to \{0,1\}^v$ *with a modified state* $\mathsf{st}' \in \{0,1\}^{2\mu}$ *and* $\mu = \mathsf{poly}(n)$*. The hardened functionality* G^h *is a triple of algorithms* $(\mathsf{Init}, \mathsf{Setup}, \mathsf{Exec})$ *described as follows:*

- $\Omega \leftarrow \mathsf{Init}(1^\lambda)$*: Run the initialization procedure of the coding scheme to sample* $\Omega \leftarrow \mathsf{Init}(1^\lambda)$*.*
- $(\mathcal{M}_0, \mathcal{M}_1) \leftarrow \mathsf{Setup}(\Omega, \mathsf{st})$*: Let* $(X_0, X_1) \leftarrow \mathsf{Enc}(\Omega, \mathsf{st})$*. For* $b \in \{0,1\}$*, store* X_b *in the first* n *bits of* $\mathcal{M}_b$*, i.e.* $\mathcal{M}_b[1 \ldots n] \leftarrow X_b$*. (The remaining bits of* $\mathcal{M}_b$ *are set to* $0^{\mu-n}$*.) Define* $\mathsf{st}' := (\mathcal{M}_0, \mathcal{M}_1)$*.*

- $y \leftarrow \mathsf{Exec}(x)$: *Read the public value* φ. *In case* $\varphi = 1$ *output* $\perp$. *Otherwise, let* $X_b = \mathcal{M}_b[1 \dots n]$ *for* $b \in \{0,1\}$. *Run* $\mathsf{st} \leftarrow \mathsf{Dec}(\Omega, (X_0, X_1))$; *if* $\mathsf{st} = \perp$, *then output* $\perp$ *and set* $\varphi = 1$. *Otherwise output* $y \leftarrow \mathsf{G}(\mathsf{st}, x)$.

Remark 6.6.2 (On φ)**.** *The public value* φ *is just a way how to implement the "self-destruct" feature. An alternative approach would be to let the hardened functionality simply output a dummy value and overwrite* $(\mathcal{M}_0, \mathcal{M}_1)$ *with the all-zero string. As we insist on the hardened functionality being stateless, we use the first approach here.*

Note that we assume that φ *is untamperable. It is easy to see that this is necessary, as an adversary tampering with* φ *could always switch-off the self-destruct feature and apply a variant of the attack from [GLM+04] to recover the secret state.*

Similarly to [DPW10, LL12], security of G^h is defined via the comparison of a real and an ideal experiment. The real experiment features an adversary A interacting with G^h; the adversary is allowed to honestly run the functionality on any chosen input, but also to modify the secret state and retrieve a bounded amount of information from it. The ideal experiment features a simulator S; the simulator is given black-box access to the original functionality G and to the adversary A, but is *not* allowed any tampering or leakage query. The two experiments are formally described below.

Experiment $\mathsf{Real}_{\mathsf{A}}^{\mathsf{G}^h(\mathsf{st}',\cdot)}(\lambda)$**.** First $\Omega \leftarrow \mathsf{Init}(1^\lambda)$ and $(\mathcal{M}_0, \mathcal{M}_1) \leftarrow \mathsf{Setup}(\Omega, \mathsf{st})$ are run and Ω is given to A. Then A can issue the following commands polynomially many times (in any order):

- $\langle \mathtt{Leak}, (\mathrm{L}_j^{(0)}, \mathrm{L}_j^{(1)}) \rangle$: In response to the j-th leakage query, compute $\Lambda_0^{(j)} \leftarrow \mathrm{L}_j^{(0)}(\mathcal{M}_0)$ and $\Lambda_1^{(j)} \leftarrow \mathrm{L}_j^{(1)}(\mathcal{M}_1)$ and output $(\Lambda_0^{(j)}, \Lambda_1^{(j)})$.
- $\langle \mathtt{Tamper}, (f_0^{(j)}, f_1^{(j)}) \rangle$: In response to the j-th tampering query, compute $\mathcal{M}_0' \leftarrow f_0^{(j)}(\mathcal{M}_0)$ and $\mathcal{M}_1' \leftarrow f_1^{(j)}(\mathcal{M}_1)$ and replace $(\mathcal{M}_0, \mathcal{M}_1)$ with $(\mathcal{M}_0', \mathcal{M}_1')$.
- $\langle \mathtt{Eval}, x_j \rangle$: In response to the j-th evaluation query, run $y_j \leftarrow \mathsf{Exec}(x_j)$. In case $y_j = \perp$ output $\perp$ and self-destruct; otherwise output y_j.

The output of the experiment is defined as

$$\mathsf{Real}_{\mathsf{A}}^{\mathsf{G}^h(\mathsf{st}',\cdot)}(\lambda) = (\Omega; ((x_1, y_1), (x_2, y_2), \dots); ((\Lambda_0^{(1)}, \Lambda_1^{(1)}), (\Lambda_0^{(2)}, \Lambda_1^{(2)}), \cdots)).$$

Experiment $\mathsf{Ideal}_{\mathsf{S}}^{\mathsf{G}(\mathsf{st},\cdot)}(\lambda)$**.** The simulator sets up the CRS Ω and is given black-box access to the functionality $\mathsf{G}(\mathsf{st}, \cdot)$ and the adversary A. The output of the experiment is defined as

$$\mathsf{Ideal}_{\mathsf{S}}^{\mathsf{G}(\mathsf{st},\cdot)}(\lambda) = (\Omega; ((x_1, y_1), (x_2, y_2), \dots); ((\Lambda_0^{(1)}, \Lambda_1^{(1)}), (\Lambda_0^{(2)}, \Lambda_1^{(2)}), \cdots)),$$

where $((x_j, y_j), ((\Lambda_0^{(j)}, \Lambda_1^{(j)})))$ are the input/output/leakage tuples simulated by S.

Definition 6.6.3 (Polyspace leak/tamper simulatability)**.** *Let* Code *be a split-state encoding scheme in the CRS model and consider a stateless functionality* G *with corresponding hardened functionality* G^h. *We say that* Code *is polyspace* (ζ, q)*-leak/tamper simulatable for* G, *if the following conditions are satisfied:*

1. *Each memory part* $\mathcal{M}_b$ *(for* $b \in \{0,1\}$*) has size* $\mu = \mathsf{poly}(n)$.

2. *The adversary asks at most* q *tampering queries and leaks a total of at most* ζ *bits from each memory part.*

3. *For all PPT adversaries* A *there exists a PPT simulator* S *such that for any initial state* st,

$$\left\{\mathsf{Real}_{\mathsf{A}}^{\mathsf{G}^h(\mathsf{st}',\cdot)}(\lambda)\right\}_{\lambda\in\mathbb{N}} \approx_c \left\{\mathsf{Ideal}_{\mathsf{S}}^{\mathsf{G}(\mathsf{st},\cdot)}(\lambda)\right\}_{\lambda\in\mathbb{N}}.$$

We show the following result.

Theorem 6.6.4. *Let* G *be a stateless functionality and* $\mathsf{Code} = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ *be any* (ζ, q)*-CNMLR split-state encoding scheme in the CRS model. Then* Code *is polyspace* (ζ, q)*-leak/tamper simulatable for* G.

Proof. We discuss the overall proof approach first. We start with describing a simulator S running in experiment $\mathsf{Ideal}_{\mathsf{S}}^{\mathsf{G}(\mathsf{st},\cdot)}(\lambda)$ which attempts to simulate the view of adversary A running in the experiment $\mathsf{Real}_{\mathsf{A}}^{\mathsf{G}^h(\mathsf{st}',\cdot)}(\lambda)$; the simulator is given black-box access to A (which can issue `Tamper`, `Leak`, and `Eval` queries) and to the functionality $\mathsf{G}(\mathsf{st},\cdot)$ for some state st. To argue that our simulator is "good" we show that if there exists a PPT distinguisher D and a PPT adversary A such that for some state st, D distinguishes the experiments $\mathsf{Ideal}_{\mathsf{S}}^{\mathsf{G}(\mathsf{st},\cdot)}$ and $\mathsf{Real}_{\mathsf{A}}^{\mathsf{G}^h(\mathsf{st}',\cdot)}$ with non-negligible probability, then we can build another distinguisher D' and an adversary A' such that D' can distinguish $\mathsf{Tamper}_{\mathsf{A}',0^\lambda}^{\mathrm{cnmlr}}$ and $\mathsf{Tamper}_{\mathsf{A}',\mathsf{st}}^{\mathrm{cnmlr}}$ with non-negligible probability. In the last step essentially we reduce the CNMLR property of Code to the polyspace leak/tamper simulatability of the code itself.

The simulator starts by sampling the common reference string $\Omega \leftarrow \mathsf{Init}(1^\lambda)$ and the public value $\varphi = 0$. Then it samples a random encoding of 0^λ, namely $(Z_0, Z_1) \leftarrow \mathsf{Enc}(\Omega, 0^\lambda)$ and sets $\mathcal{M}_b[1\ldots,n] \leftarrow Z_b$ for $b \in \{0,1\}$. The remaining bits of $(\mathcal{M}_0, \mathcal{M}_1)$ are set to $0^{\mu-n}$. Hence, S alternates between the following two modes (starting with the normal mode in the first round):

- *Normal Mode.* Given state $(\mathcal{M}_0, \mathcal{M}_1)$, while A continues issuing queries, answer as follows:
  - $\langle \mathtt{Eval}, x_j \rangle$: Upon input the j-th evaluation query invoke $\mathsf{G}(\mathsf{st},\cdot)$ to get $y_j \leftarrow \mathsf{G}(\mathsf{st}, x_j)$ and reply with y_j.
  - $\langle \mathtt{Tamper}, (f_0^{(j)}, f_1^{(j)}) \rangle$: Upon input the j-th tampering query, compute $\mathcal{M}'_b \leftarrow f_b^{(j)}(\mathcal{M}_b)$ for $b \in \{0,1\}$. In case $(\mathcal{M}'_0[1\ldots n], \mathcal{M}'_1[1\ldots n]) = (Z_0, Z_1)$ then continue in the current mode. Otherwise go to the overwritten mode defined below with state $(\mathcal{M}'_0, \mathcal{M}'_1)$.
  - $\langle \mathtt{Leak}, (\mathrm{L}_j^{(0)}, \mathrm{L}_j^{(1)}) \rangle$: Upon input the j-th leakage query, compute $\Lambda_b^{(j)} = \mathrm{L}_j^{(b)}(Z_b)$ for $b \in \{0,1\}$ and reply with $(\Lambda_0^{(j)}, \Lambda_1^{(j)})$.
- *Overwritten Mode.* Given state $(\mathcal{M}'_0, \mathcal{M}'_1)$, while A continues issuing queries, answer as follows:
  - Let $\Xi = (\mathcal{M}'_0, \mathcal{M}'_1)$. Simulate the hardened functionality $\mathsf{G}^h(\Xi,\cdot)$ and answer all `Eval` and `Leak` queries as the real experiment $\mathsf{Real}_{\mathsf{A}}^{\mathsf{G}^h(\Xi,\cdot)}(\lambda)$ would do.

– Upon input the j-th tampering query $(f_0^{(j)}, f_1^{(j)})$, compute $\mathcal{M}''_b \leftarrow f_b^{(j)}(\mathcal{M}'_b)$ for $b \in \{0,1\}$. In case $(\mathcal{M}''_0[1\dots n], \mathcal{M}''_1[1\dots n]) = (Z_0, Z_1)$ then go to the normal mode with state $(\mathcal{M}_0, \mathcal{M}_1) := (\mathcal{M}''_0, \mathcal{M}''_1)$. Otherwise continue in the current mode.

- When A halts and outputs $\mathsf{view}_\mathsf{A} = (\Omega; ((x_1, y_1), (x_2, y_2), \dots); ((\Lambda_0^{(1)}, \Lambda_1^{(1)}), (\Lambda_0^{(2)}, \Lambda_1^{(2)}), \cdots))$, set $\mathsf{view}_\mathsf{S} = \mathsf{view}_\mathsf{A}$ and output view_S as output of $\mathsf{Ideal}_\mathsf{S}^{\mathsf{G}(\mathsf{st},\cdot)}$.

Intuitively, since the coding scheme is non-malleable, the adversary can either keep the encoding unchanged or overwrite it with the encoding of some unrelated message. These two cases are captured in the above modes: The simulator starts in the normal mode and then, whenever the adversary mauls the initial encoding, it switches to the overwritten mode. However, the adversary can use the extra space to keep a copy of the original encoding and place it back at some later point in time. When this happens, the simulator switches back to the normal mode; this switching is important to maintain simulation.

To finish the proof, we have to argue that the output of experiment $\mathsf{Ideal}_\mathsf{S}^{\mathsf{G}(\mathsf{st},\cdot)}$ is computationally indistinguishable from the output of experiment $\mathsf{Real}_\mathsf{A}^{\mathsf{G}^h(\mathsf{st}',\cdot)}$. This is done in the lemma below.

Lemma 6.6.5. *Let* S *be defined as above. Then for all PPT adversaries* A *and all* $\mathsf{st} \in \{0,1\}^k$*, the following holds:*

$$\left\{\mathsf{Real}_\mathsf{A}^{\mathsf{G}^h(\mathsf{st}',\cdot)}(\lambda)\right\}_{\lambda\in\mathbb{N}} \approx_c \left\{\mathsf{Ideal}_\mathsf{S}^{\mathsf{G}(\mathsf{st},\cdot)}(\lambda)\right\}_{\lambda\in\mathbb{N}}.$$

Proof. By contradiction, assume that there exists a PPT distinguisher D, a PPT adversary A and some state $\mathsf{st} \in \{0,1\}^k$ such that:

$$\left|\Pr\left[\mathsf{D}(\mathsf{Ideal}_\mathsf{S}^{\mathsf{G}(\mathsf{st},\cdot)}(\lambda)) = 1\right] - \Pr\left[\mathsf{D}(\mathsf{Real}_\mathsf{A}^{\mathsf{G}^h(\mathsf{st}',\cdot)}(\lambda)) = 1\right]\right| \geq \varepsilon, \tag{6.7}$$

where $\varepsilon(\lambda)$ is some non-negligible function of the security parameter λ.

We build a PPT distinguisher D' and a PPT adversary A' telling apart the experiments $\mathsf{Tamper}_{\mathsf{A}',0^\lambda}^{\mathrm{cnmlr}}(\lambda)$ and $\mathsf{Tamper}_{\mathsf{A}',\mathsf{st}}^{\mathrm{cnmlr}}(\lambda)$; this contradicts our assumption that Code is CNMLR. The distinguisher D' is given the CRS $\Omega \leftarrow \mathsf{Init}(1^\lambda)$ and can access $\mathcal{O}_{\mathsf{cnm}}((X_0, X_1), \cdot)$ (for at most q times) and $\mathcal{O}^\zeta(X_0)$, $\mathcal{O}^\zeta(X_1)$; here (X_0, X_1) is either an encoding of 0^k or an encoding of st. The distinguisher D' keeps a flag SAME (initially set to TRUE) and a flag STOP (initially set to FALSE). After simulating the public values, D' mimics the enviroment for D as follows:

- $\langle \mathtt{Tamper}, (f_0^{(j)}, f_1^{(j)})\rangle$: Upon input tampering functions $(f_0^{(j)}, f_1^{(j)})$, the distinguisher D' uses the oracle $\mathcal{O}_{\mathsf{cnm}}((X_0, X_1), \cdot)$ to answer them.[10] However, it can not simply forward the queries because of the following two reasons:
  - The tampering functions $(f_0^{(j)}, f_1^{(j)})$ maps from μ bits to μ bits, whereas the oracle $\mathcal{O}_{\mathsf{cnm}}((X_0, X_1), \cdot)$ expects tampering functions mapping from n bits to n bits.
  - In both the real and the ideal experiments the tampering functions are applied to the current state (which may be different from the initial state), whereas in experiment $\mathsf{Tamper}_{\mathsf{A}',*}^{\mathrm{cnmlr}}$ the oracle $\mathcal{O}_{\mathsf{cnm}}((X_0, X_1), \cdot)$ always applies $(f_0^{(j)}, f_1^{(j)})$ to the target encoding (X_0, X_1).

[10] Formally D' has to access $\mathcal{O}_{\mathsf{cnm}}(\cdot)$ via A'. For simplicity we assume that D' can access the oracle directly. In fact, A' just acts as an interface between the experiment $\mathsf{Tamper}_{\mathsf{A}',*}^{\mathrm{cnmlr}}$ and D'.

To take into account the above differences, D' modifies $(f_0^{(j)}, f_1^{(j)})$ as follows. Define the functions $f_{\mathsf{in}} : \{0,1\}^n \to \{0,1\}^\mu$ and $f_{\mathsf{out}} : \{0,1\}^\mu \to \{0,1\}^n$ as $f_{\mathsf{in}}(x) = (x||0^{\mu-n})$ and $f_{\mathsf{out}}(x||x') = x$, for any $x \in \{0,1\}^n$ and $x' \in \{0,1\}^{\mu-n}$. The distinguisher D' queries $\mathcal{O}_{\mathsf{cnm}}((X_0, X_1), \cdot)$ with the function pair $(\widetilde{f}_j^{(0)}, \widetilde{f}_j^{(1)})$ where each $\widetilde{f}_j^{(b)}$ is defined as $\widetilde{f}_j^{(b)} := f_{\mathsf{out}} \circ f_b^{(j)} \circ f_b^{(j-1)} \circ \ldots \circ f_b^{(1)} \circ f_{\mathsf{in}}$ for $b \in \{0,1\}$.

In case the oracle returns $\perp$, then D' sets STOP to TRUE. In case the oracle returns $\mathsf{same}^\star$, then D' sets SAME to TRUE. Otherwise, in case the oracle returns an encoding (X_0', X_1'), then D' sets SAME to FALSE.

- $\langle \mathtt{Leak}, (\mathrm{L}_j^{(0)}, \mathrm{L}_j^{(1)}) \rangle$: Upon input leakage functions $(\mathrm{L}_j^{(0)}, \mathrm{L}_j^{(1)})$, the distinguisher D' defines $(\widetilde{\mathrm{L}}_j^{(0)}, \widetilde{\mathrm{L}}_j^{(1)})$ (in a similar way as above), forwards those functions to $\mathcal{O}^\zeta(X_0)$, $\mathcal{O}^\zeta(X_1)$ and sends the answer from the oracles back to D.

- $\langle \mathtt{Eval}, x_j \rangle$: Upon input an evaluation query for value x_j, the distinguisher D' checks first that STOP equals FALSE. If this is not the case, then D' returns $\perp$ to D. Otherwise, D' checks that SAME equals TRUE. If this is the case, it runs $y_j \leftarrow \mathsf{G}(\mathsf{st}, x_j)$ and gives y_j to D. Else (if SAME equals FALSE), it computes $y_j \leftarrow \mathsf{G}(\mathsf{st}', x_j)$, where st' is the output of $\mathsf{Dec}(\Omega, (X_0', X_1'))$, and gives y_j to D.

Finally, D' outputs whatever D outputs.

For the analysis, first note that D' runs in polynomial time. Furthermore, D' asks exactly q queries to $\mathcal{O}_{\mathsf{cnm}}$ and leaks at most ζ bits from the target encoding (X_0, X_1). It is also easy to see that in case (X_0, X_1) is an encoding of $\mathsf{st} \in \{0,1\}^k$, then D' perfectly simulates the view of adversary D in the experiment $\mathsf{Real}_{\mathsf{A}}^{\mathsf{G}^h(\mathsf{st}', \cdot)}$. On the other hand, in case (X_0, X_1) is an encoding of 0^k, we claim that D' perfectly simulates the view of D in the experiment $\mathsf{Ideal}_{\mathsf{S}}^{\mathsf{G}(\mathsf{st}, \cdot)}$. This is because: (i) Whenever SAME equals TRUE, then D' answers evaluation queries by running G on state st and tampering/leakage queries using a pre-sampled encoding of 0^λ (this corresponds to the normal mode of S); (ii) Whenever SAME equals FALSE, then D' answers evaluation queries by running G on the current tampered state st' which results from applying the tampering functions to a pre-sampled encoding of 0^λ (this corresponds to the overwritten mode of S).

Combining the above argument with Eq. (6.7) we obtain

$$\left| \Pr\left[\mathsf{D}(\mathsf{Tamper}_{\mathsf{A}', 0^\lambda}^{\mathrm{cnmlr}}(\lambda)) = 1\right] - \Pr\left[\mathsf{D}(\mathsf{Tamper}_{\mathsf{A}', \mathsf{st}}^{\mathrm{cnmlr}}(\lambda)) = 1\right] \right| \geq \varepsilon(\lambda),$$

which is a contradiction to the fact that Code is (ζ, q)-CNMLR. □

□

6.6.2 Stateful Functionalities

Finally, we consider the case of primitives that update their state at each execution, i.e. functionalities of the type $(\mathsf{st}_{\mathsf{new}}, y) \leftarrow \mathsf{G}(\mathsf{st}, x)$ (a.k.a. *stateful* functionalities). Note that in this case the hardened functionality re-encodes the new state at each execution.

Note that, since we do *not* assume erasure in our model, an adversary can always 'reset' the functionality to a previous valid state as follows: It could just copy the previous state to some part of the large memory and replace the current encoding by that. To avoid this, our transformation uses an untamperable *public* counter (along with the untamperable self-destruct bit) that helps us

to detect whether the functionality is reset to a previous state, leading to a self-destruct. However such a counter can be implemented, for instance using $\log(\lambda)$ bits. We notice that such a counter is necessary to protect against the above resetting attack. However, we stress that if we do not assume such a counter this "resetting" is the only harm the adversary can make in our model.

Below, we define what it means to harden a stateful functionality.

Definition 6.6.6 (Stateful hardened functionality)**.** *Let* $\mathsf{Code} = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ *be a split-state encoding scheme in the CRS model, with* $k + \log(\lambda)$ *bits messages and* $2n$ *bits codewords. Let* $\mathsf{G} : \{0,1\}^k \times \{0,1\}^u \to \{0,1\}^k \times \{0,1\}^v$ *be a* stateful *functionality with secret state* $\mathsf{st} \in \{0,1\}^k$*,* $\varphi \in \{0,1\}$ *be a public value and let* $\langle\gamma\rangle$ *be a public* $\log(\lambda)$*-bit counter both initially set to zero. We define a* stateful *hardened functionality* $\mathsf{G}^h : \{0,1\}^{2\mu} \times \{0,1\}^u \to \{0,1\}^{2\mu} \times \{0,1\}^v$ *with a modified state* $\mathsf{st}' \in \{0,1\}^{2\mu}$ *and* $\mu = \mathsf{poly}(n)$*. The hardened functionality* G^h *is a triple of algorithms* $(\mathsf{Init}, \mathsf{Setup}, \mathsf{Exec})$ *described as follows:*

- $\Omega \leftarrow \mathsf{Init}(1^\lambda)$*: Run the initialization procedure of the coding scheme to sample* $\Omega \leftarrow \mathsf{Init}(1^\lambda)$*.*
- $(\mathcal{M}_0, \mathcal{M}_1) \leftarrow \mathsf{Setup}(\Omega, \mathsf{st})$*: Let* $(X_0, X_1) \leftarrow \mathsf{Enc}(\Omega, \mathsf{st} || \langle 1 \rangle)$ *and increment* $\langle\gamma\rangle \leftarrow \langle\gamma\rangle + 1$*. For* $b \in \{0,1\}$*, store* X_b *in the first* n *bits of* $\mathcal{M}_b$*, i.e.* $\mathcal{M}_b[1 \dots n] \leftarrow X_b$*.*[11] *(The remaining bits of* $\mathcal{M}_b$ *are set to* $0^{\mu-n}$*.) Define* $\mathsf{st}' := (\mathcal{M}_0, \mathcal{M}_1)$*.*
- $y \leftarrow \mathsf{Exec}(x)$*: Read the public bit* φ*. In case* $\varphi = 1$ *output* $\perp$*. Otherwise recover* $X_b = \mathcal{M}_b[1 \dots n]$ *for* $b \in \{0,1\}$ *and run* $(\mathsf{st}'' || \langle\gamma'\rangle) \leftarrow \mathsf{Dec}(\Omega, (X_0, X_1))$*. Read the public counter* $\langle\gamma\rangle$*. If* $\langle\gamma\rangle \neq \langle\gamma'\rangle$ *or* $\mathsf{st}'' = \perp$*, set* $\varphi = 1$*. Else run* $(\mathsf{st}_{\mathsf{new}}, y) \leftarrow \mathsf{G}(\mathsf{st}'', x)$ *and output* y*. Finally, write* $\mathsf{Enc}(\Omega, \mathsf{st}_{\mathsf{new}} || \langle\gamma + 1\rangle)$ *in* $(\mathcal{M}_0[1, \dots, n], \mathcal{M}_1[1, \dots, n])$ *and increment* $\langle\gamma\rangle \leftarrow \langle\gamma\rangle + 1$*.*

Remark 6.6.7 (On $\langle\gamma\rangle$)**.** *Note that the counter is incremented after each evaluation query, and the current value is encoded together with the new state. We require* $\langle\gamma\rangle$ *to be untamperable. This assumption is necessary, as otherwise an adversary could always use the extra space to keep a copy of a previous valid state and place it back at some later point in time. The above attack allows essentially to reset the functionality to a previous state, and cannot be simulated with black-box access to the original functionality.*

In the case of stateful primitives, the hardened functionality has to re-encode the new state at each execution. Still, as the memory is large, the adversary can use the extra space to tamper continuously with a target encoding of some valid state. Security of a stateful hardened functionality is defined analogously to the stateless case (cf. Definition 6.6.3). We show the following result:

Theorem 6.6.8. *Let* G *be a stateful functionality and* $\mathsf{Code} = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ *be any* (ζ, q)*-CNMLR encoding scheme in the split-state CRS model. Then* Code *is polyspace* (ζ, q)*-leak/tamper simulatable for* G*.*

Proof. Similarly to the proof of Theorem 6.6.4, we describe a simulator S running in experiment $\mathsf{Ideal}_{\mathsf{S}}^{\mathsf{G}(\mathsf{st},\cdot)}(\lambda)$ which simulates the view of adversary A in the experiment $\mathsf{Real}_{\mathsf{A}}^{\mathsf{G}^h(\mathsf{st}',\cdot)}(\lambda)$. The simulator is given black-box access to A (which can issue `Tamper`, `Leak`, and `Eval` queries) and to the reactive functionality $\mathsf{G}(\mathsf{st}, \cdot)$ with initial state st. Simulator S sets $(\mathcal{M}_0, \mathcal{M}_1)$ to $(0^\mu, 0^\mu)$ and keeps a $\log(\lambda)$-bit counter $\langle\gamma\rangle$ (initially set to zero). Then, it samples the common reference string $\Omega \leftarrow \mathsf{Init}(1^\lambda)$, simulates the public values, and proceeds as follows:

[11] Without erasure this can be easily implemented by a stack.

- *Normal Mode.* Given state $(\mathcal{M}_0, \mathcal{M}_1)$, while A continues issuing queries, answer as follows:
  - $\langle \texttt{Eval}, x_j \rangle$: Upon input the j-th evaluation query, forward the input x_j to $\mathsf{G}(\mathsf{st}_{\mathsf{curr}}, \cdot)$ and send the reply y_j back to A. (Here $\mathsf{st}_{\mathsf{curr}}$ is the current state, as the functionally updates the state at each invocation.) Increment the counter $\langle \gamma \rangle$.
  - $\langle \texttt{Tamper}, (f_0^{(j)}, f_1^{(j)}) \rangle$: Upon input the j-th tampering query sample a random encoding of $0^{k+\log(\lambda)}$, namely $(Z_0^{(j)}, Z_1^{(j)}) \leftarrow \mathsf{Enc}(\Omega, 0^{k+\log(\lambda)})$, and set $\mathcal{M}_b[1 \ldots, n] \leftarrow Z_b^{(j)}$ for $b \in \{0,1\}$. Hence, compute $\mathcal{M}'_b \leftarrow f_b^{(j)}(\mathcal{M}_b)$. In case $(\mathcal{M}'_0[1 \ldots n], \mathcal{M}'_1[1 \ldots n]) = (Z_0^{(j)}, Z_1^{(j)})$ then continue in the current mode with state $(\mathcal{M}_0, \mathcal{M}_1) := (\mathcal{M}'_0, \mathcal{M}'_1)$. Otherwise go to the overwritten mode defined below with the new state $(\mathcal{M}'_0, \mathcal{M}'_1)$.
  - $\langle \texttt{Leak}, (\mathrm{L}_j^{(0)}, \mathrm{L}_j^{(1)}) \rangle$: Upon input the j-th leakage query, compute $\Lambda_b^{(j)} = \mathrm{L}_j^{(b)}(Z_b^{(j)})$ for $b \in \{0,1\}$ and reply with $(\Lambda_0^{(j)}, \Lambda_1^{(j)})$.
- *Overwritten Mode.* Given state $(\mathcal{M}'_0, \mathcal{M}'_1)$ and current counter $\langle \gamma \rangle$, while A continues issuing queries, answer as follows:
  - Let $\Xi = (\mathcal{M}'_0, \mathcal{M}'_1)$. Simulate the hardened functionality $\mathsf{G}^h(\Xi, \cdot)$ (with counter set to $\langle \gamma \rangle$ and self-destruct bit $\varphi = 0$) and answer all `Eval`, `Leak` and `Tamper` queries as the real experiment $\mathsf{Real}_{\mathsf{A}}^{\mathsf{G}^h(\Xi,\cdot)}(\lambda)$ would do.
- When A halts and outputs $\mathsf{view}_{\mathsf{A}} = (\Omega; ((x_1, y_1), (x_2, y_2), \ldots); ((\Lambda_0^{(1)}, \Lambda_1^{(1)}), (\Lambda_0^{(2)}, \Lambda_1^{(2)}), \cdots))$, set $\mathsf{view}_{\mathsf{S}} = \mathsf{view}_{\mathsf{A}}$ and output $\mathsf{view}_{\mathsf{S}}$ as output of $\mathsf{Ideal}_{\mathsf{S}}^{\mathsf{G}(\mathsf{st},\cdot)}(\lambda)$.

We show the following lemma about S's simulation, which directly implies the theorem.

Lemma 6.6.9. *Let S be defined as above. Then for all PPT adversaries A and all $\mathsf{st} \in \{0,1\}^k$, the following holds:*

$$\left\{ \mathsf{Real}_{\mathsf{A}}^{\mathsf{G}^h(\mathsf{st}',\cdot)}(\lambda) \right\}_{\lambda \in \mathbb{N}} \approx_c \left\{ \mathsf{Ideal}_{\mathsf{S}}^{\mathsf{G}(\mathsf{st},\cdot)}(\lambda) \right\}_{\lambda \in \mathbb{N}}.$$

Proof. By contradiction, assume that there exists a PPT distinguisher D, a PPT adversary A and some state $\mathsf{st} \in \{0,1\}^k$ such that:

$$\left| \Pr\left[\mathsf{D}(\mathsf{Ideal}_{\mathsf{S}}^{\mathsf{G}(\mathsf{st},\cdot)}(\lambda)) = 1 \right] - \Pr\left[\mathsf{D}(\mathsf{Real}_{\mathsf{A}}^{\mathsf{G}^h(\mathsf{st}',\cdot)}(\lambda)) = 1 \right] \right| \geq \varepsilon, \tag{6.8}$$

where $\varepsilon(\lambda)$ is some non-negligible function of the security parameter λ.

Similar to [LL12], without loss of generality we define a round as a sequence of queries which contains at most one execute query, one tampering query and one leakage query. For all $i = 1, \ldots, q$, consider the following hybrid experiments where in each hybrid the simulator does a modified normal mode and the same overwritten mode:

Experiment $\mathsf{HYB}_{\mathsf{S},\mathsf{st}}^{(i)}(\lambda)$. The simulator S starts with generating the public values and proceeds as follows.

- In the first i round it does exactly the same as in the experiment $\mathsf{Ideal}_{\mathsf{S}}^{\mathsf{G}(\mathsf{st},\cdot)}(\lambda)$.
- Starting from the $i+1$-th round if the simulator is already in the overwritten mode, then continue simulating in the overwritten mode. Otherwise, let $\mathsf{st}_{\mathsf{curr}}$ be the current state of the functionality. Proceed with the following modified normal mode:

– $\langle \texttt{Eval}, x_j \rangle$: Run $(\mathsf{st}_{\mathsf{new}}, y_j) \leftarrow \mathsf{G}(\mathsf{st}_{\mathsf{cur}}, x_j)$, set $\mathsf{st}_{\mathsf{cur}} := \mathsf{st}_{\mathsf{new}}$ and give y_j to A. Increment the counter $\langle \gamma \rangle$.

– $\langle \texttt{Tamper}, (f_0^{(j)}, f_1^{(j)}) \rangle$: Sample a random encoding of $\mathsf{st}_{\mathsf{cur}}$, namely $(Z_0^{(j)}, Z_1^{(j)}) \leftarrow \mathsf{Enc}(\Omega, \mathsf{st}_{\mathsf{cur}} || \langle \gamma \rangle)$, and set $\mathcal{M}_b[1 \ldots, n] \leftarrow Z_b^{(j)}$ for $b \in \{0,1\}$. Hence, compute $\mathcal{M}'_b \leftarrow f_b^{(j)}(\mathcal{M}_b)$. In case $(\mathcal{M}'_0[1 \ldots n], \mathcal{M}'_1[1 \ldots n]) = (Z_0^{(j)}, Z_1^{(j)})$ then continue in the current mode with state $(\mathcal{M}_0, \mathcal{M}_1) := (\mathcal{M}'_0, \mathcal{M}'_1)$. Otherwise go to the overwritten mode with state $(\mathcal{M}'_0, \mathcal{M}'_1)$.

– $\langle \texttt{Leak}, (\mathrm{L}_j^{(0)}, \mathrm{L}_j^{(1)}) \rangle$: Compute $\Lambda_b^{(j)} = \mathrm{L}_j^{(b)}(Z_b^{(j)})$ for $b \in \{0,1\}$ and reply with $(\Lambda_0^{(j)}, \Lambda_1^{(j)})$.

Note that $\mathsf{HYB}_{\mathsf{S},\mathsf{st}}^{(0)}(\lambda)$ is distributed exactly as experiment $\mathsf{Real}_{\mathsf{A}}^{\mathsf{G}^h(\mathsf{st}',\cdot)}(\lambda)$, and $\mathsf{HYB}_{\mathsf{S},\mathsf{st}}^{(q)}(\lambda)$ is distributed exactly as $\mathsf{Ideal}_{\mathsf{S}}^{\mathsf{G}(\mathsf{st},\cdot)}(\lambda)$. Hence, by a standard argument, Eq. 6.8 implies

$$\left| \Pr\left[\mathsf{D}(\mathsf{HYB}_{\mathsf{S},\mathsf{st}}^{(i)}(\lambda)) = 1\right] - \Pr\left[\mathsf{D}(\mathsf{HYB}_{\mathsf{S},\mathsf{st}}^{(i+1)}(\lambda)) = 1\right] \right| \geq \varepsilon/q, \tag{6.9}$$

a non-negligible quantity. We will exploit the advantage of D in distinguishing between $\mathsf{HYB}_{\mathsf{S},\mathsf{st}}^{(i)}(\lambda)$ and $\mathsf{HYB}_{\mathsf{S},\mathsf{st}}^{(i+1)}(\lambda)$, to build another distinguisher D' and an adversary A' breaking the CNMLR property of Code. The distinguisher works similar to the proof of [LL12, Lemma 22] with some technical differences, that we emphasize below.

Given the common reference string Ω, the distinguisher D' initializes a flag STOP to FALSE, lets $\overline{\Omega} = (\Omega, 0, \langle 1 \rangle)$, and runs the simulator S of hybrid $\mathsf{HYB}_{\mathsf{S},\mathsf{st}}^{(i)}$ until round i to obtain the current state $\mathsf{st}_{\mathsf{cur}}$ and the current memory content $(\mathcal{M}'_0, \mathcal{M}'_1)$. (Note that D' can do this as it is given st as advice.) Hence, D' outputs the message pair $(0^{k+\log\lambda}, \mathsf{st}_{\mathsf{cur}} || \langle i \rangle)$ and is given access to oracle $\mathcal{O}_{\mathsf{cnm}}((X_0, X_1), \cdot)$ and $\mathcal{O}^{\zeta}(X_0)$, $\mathcal{O}^{\zeta}(X_1)$, where (X_0, X_1) is either an encoding of $0^{k+\log\lambda}$ or an encoding of $\mathsf{st}_{\mathsf{cur}} || \langle i \rangle$. For all rounds $i+1 \leq j \leq q$, the distinguisher handles A's queries as follows:

- $\langle \texttt{Tamper}, (f_0^{(j)}, f_1^{(j)}) \rangle$: The distinguisher will access the oracle $\mathcal{O}_{\mathsf{cnm}}((X_0, X_1), \cdot)$. Similarly to the proof of Theorem 6.6.4, D' cannot simply forward the pair of functions to the oracle, so it modifies it as follows. Define the functions $f_{\mathsf{in}} : \{0,1\}^n \to \{0,1\}^{\mu}$ and $f_{\mathsf{out}} : \{0,1\}^{\mu} \to \{0,1\}^n$ as $f_{\mathsf{in}}(x) = (x || \mathcal{M}'_b[n+1, \ldots, s])$ and $f_{\mathsf{out}}(x || x') = x$, for any $x \in \{0,1\}^n$ and $x' \in \{0,1\}^{\mu-n}$; here $(\mathcal{M}'_0, \mathcal{M}'_1)$ is the above state obtained at the end of round i. The distinguisher D' queries $\mathcal{O}_{\mathsf{cnm}}((X_0, X_1), \cdot)$ with the function pair $(\widetilde{f}_j^{(0)}, \widetilde{f}_j^{(1)})$ where each $\widetilde{f}_j^{(b)}$ is defined as $\widetilde{f}_j^{(b)} := f_{\mathsf{out}} \circ f_b^{(j)} \circ f_b^{(j-1)} \circ \ldots \circ f_b^{(i+1)} \circ f_{\mathsf{in}}$ for $b \in \{0,1\}$.

  In case the oracle returns $\perp$, then D' sets STOP to TRUE and $\varphi = 1$. In case the oracle returns $\mathsf{same}^{\star}$, then D' does nothing. Otherwise, in case the oracle returns an encoding (X'_0, X'_1), then D' checks that the last $\log\lambda$ bits of $\mathsf{Dec}(\Omega, (X'_0, X'_1))$ equal $\langle j \rangle$; if that is not the case, it sets STOP to TRUE and $\varphi = 1$.

- $\langle \texttt{Leak}, (\mathrm{L}_j^{(0)}, \mathrm{L}_j^{(1)}) \rangle$: Upon input leakage functions $(\mathrm{L}_j^{(0)}, \mathrm{L}_j^{(1)})$, the distinguisher D' defines $(\widetilde{\mathrm{L}}_j^{(0)}, \widetilde{\mathrm{L}}_j^{(1)})$ (in a similar way as above), forwards the functions to $\mathcal{O}^{\zeta}(X_0)$, $\mathcal{O}^{\zeta}(X_1)$ and sends the answer from the oracles back to D.

- $\langle \texttt{Eval}, x_j \rangle$: Upon input an evaluation query for value x_j, the distinguisher D' checks first that STOP equals FALSE. If this is not the case, then D' returns $\perp$ to D. Otherwise, it runs $(\mathsf{st}_{\mathsf{new}}, y_j) \leftarrow \mathsf{G}(\mathsf{st}_{\mathsf{cur}}, x_j)$, sets $s_{\mathsf{cur}} := s_{\mathsf{new}}$, increments $\langle \gamma \rangle$, and gives y_j to D.

Define the following event FAIL: the event becomes true in case the simulation has entered already the overwritten mode at round $i+1$, or the simulation is in normal mode but D never issues `Tamper` or `Leak` commands after round i. Now, if FAIL happens, D' gives-up and outputs a random guess. Otherwise it outputs whatever D does.

It is easy to see that $\mathsf{HYB}^{(i)}_{\mathsf{S},\mathsf{st}}(\lambda)$ and $\mathsf{HYB}^{(i+1)}_{\mathsf{S},\mathsf{st}}(\lambda)$ are statistically close in case the simulation enters the overwritten mode before round i. It follows that there is a non-negligible chance that the simulation is in normal mode at round i and moreover D will either issue a `Leak` or a `Tamper` command in some later round. The above argument implies that: (i) D' correctly simulates either the distribution of $\mathsf{HYB}^{(i)}_{\mathsf{S},\mathsf{st}}(\lambda)$ or $\mathsf{HYB}^{(i+1)}_{\mathsf{S},\mathsf{st}}(\lambda)$ (depending on the value in the target encoding (X_0, X_1)), conditioning on $\overline{\text{FAIL}}$; (ii) $\Pr\left[\overline{\text{FAIL}}\right] \geq \alpha$, where $\alpha = \alpha(\lambda)$ is non-negligible; (iii) D' has advantage $1/2$ conditioning on FAIL. Let $\mathsf{st}^* = \mathsf{st}_{\mathsf{cur}} || \langle i \rangle$. Setting $k' = k + \log \lambda$, we have obtained

$$
\begin{aligned}
&\left|\Pr\left[\mathsf{D}'(\mathsf{Tamper}^{\text{cnmlr}}_{\mathsf{A}',0^{k'}}(\lambda)) = 1\right] - \Pr\left[\mathsf{D}'(\mathsf{Tamper}^{\text{cnmlr}}_{\mathsf{A}',\mathsf{st}^*}(\lambda)) = 1\right]\right| \\
&= \Big|\Pr\left[\text{FAIL}\right]\Pr\left[\mathsf{D}'(\mathsf{Tamper}^{\text{cnmlr}}_{\mathsf{A},0^{\lambda'}}(\lambda)) = 1 | \text{FAIL}\right] + \Pr\left[\overline{\text{FAIL}}\right]\Pr\left[\mathsf{D}'(\mathsf{Tamper}^{\text{cnmlr}}_{\mathsf{A}',0^{\lambda'}}(\lambda)) = 1 | \overline{\text{FAIL}}\right] \\
&\qquad - \Pr\left[\text{FAIL}\right]\Pr\left[\mathsf{D}'(\mathsf{Tamper}^{\text{cnmlr}}_{\mathsf{A}',\mathsf{st}^*}(\lambda)) = 1 | \text{FAIL}\right] - \Pr\left[\overline{\text{FAIL}}\right]\Pr\left[\mathsf{D}'(\mathsf{Tamper}^{\text{cnmlr}}_{\mathsf{A}',\mathsf{st}^*}(\lambda)) = 1 | \overline{\text{FAIL}}\right]\Big| \\
&= \left|\Pr\left[\mathsf{D}(\mathsf{HYB}^{(i)}_{\mathsf{S},\mathsf{st}}(\lambda)) = 1\right] - \Pr\left[\mathsf{D}(\mathsf{HYB}^{(i+1)}_{\mathsf{S},\mathsf{st}}(\lambda)) = 1\right]\right| \cdot \Pr\left[\overline{\text{FAIL}}\right] \\
&\geq \varepsilon/q \cdot \alpha,
\end{aligned}
$$

a non-negligible quantity. This concludes the proof. □

□

Chapter 7

Conclusion: Subsequent and future works

In this thesis we explore different mechanisms to secure cryptographic memory against powerful tampering attacks from a theoretical perspective. En route we presented new results on non-malleable codes which recently evolved independently as an interesting direction. Summarizing, in this dissertation we present three main results:

1. [DFMV13]: How to protect some specific public-key schemes like ID-scheme and encryptions against bounded tampering (and leakage).

2. [FMVW14]: An efficient (and information theoretic) construction of non-malleable codes which protects against poly-size tampering circuits. A new concept called non-malleable key-derivation and its applications in tamper-resilience.

3. [FMNV14]: A new extension of non-malleable codes called continuous non-malleable codes which helps protecting any cryptographic functionality against split-state tampering attacks and moreover removes the requirements of erasures.

A comparative overview is provided in Table 1.1. Below we present a few follow-ups based on these works.

7.1 Subsequent works.

Tamper-resilient von-Neumann Architecture [FMNV15]. Recall that in this dissertation we mainly consider the weaker form of tampering where the adversary can only tamper with the memory but not with the computation. Now, given a significant number of positive results in the weaker model including those described in this thesis, it is a natural question that if these techniques could have any use in protecting computations from tampering.

In [FMNV15] Faust et al. addressed this question and made significant progress towards answering that. They presented a *universal framework* for tamper and leakage resilient computation on a von Neumann Random Access Architecture (or simply called tamper and leakage resilient RAM) which uses continuous non-malleable codes as the main building block. The RAM has one CPU that accesses a storage, called the disk. The disk is subject to leakage and tampering. So is the bus connecting the CPU to the disk. They assumed that the CPU is leakage and tamper-free. For a fixed value of the security parameter, the CPU has *constant size*. Therefore the code of the program to be executed is stored on the disk, i.e., they essentially considered a von Neumann architecture. The most prominent consequence of this is that the code of the program executed will be subject to tampering.

They constructed a compiler for this architecture which transforms any keyed primitive into a RAM program where the key is encoded and stored on the disk along with the program to evaluate the primitive on that key. It is solely based on continuous non-malleable codes in a black-box

way. No further (cryptographic) assumptions are needed. This in particular means that given an information theoretic code[1], the overall construction is information theoretically secure.

Although it is required that the CPU is tamper and leakage proof, its design is independent of the actual primitive being computed and its internal storage is non-persistent, i.e., all secret registers are reset between invocations. Hence, this result can be interpreted as *reducing the problem of shielding arbitrary complex computations to protecting a single, simple yet universal component.*

Another recent result by Dachman-Solde et al. [DLSZ15] constructed a similar tamper-resilient RAM. In that they use a different type of non-malleable codes called *locally decodable* non-malleable codes.

General study of continuous non-malleable codes [JW15]. A recent work by Jafargholi and Wichs [JW15] addressed the problem of continuous non-malleability in more detail. Recall (from Chapter 6) that we restricted our focus only to split-state tampering. In contrast, this work provided a general study of continuous non-malleable codes. In particular, they proposed different variants of the continuous non-malleable codes depending on: (i) whether or not *tampering is persistent*, i.e. each successive attack modifies the codeword that has been modified by previous attacks, or whether tampering is non-persistent and is always applied to the original codeword, (ii) whether *"self-destruct" is allowed* if a tampered codeword is ever detected to be invalid or whether the attacker can continue tampering even after that. They provided a comprehensive study of each variants. They considered global model of tampering and information theoretic setting which is in contrast to our approach in Chapter 6 where we consider split-state tampering and computationally bounded setting. One may observe that our model falls into one *intermediate* category where the tampering is *non-persistent* and *self-destruct* is allowed.

More on efficient non-malleable codes and key-derivations [JW15, QLY+15]. In the same work [JW15] the authors revisited our constructions from Chapter 5 of efficient non-malleable codes against poly-size tampering circuits. Essentially their construction is same as ours, albeit their analysis is much optimized and also provide better modularity. In particular, they put forth a nice abstraction called *tamper-detection* (in contrast to our bounded-malleability in Chapter 5) which can be thought of as a notion similar to well-known error-detecting codes, albeit with some additional properties.

The concept of non-malleable key-derivation discussed in Chapter 5 has been explored in computational setting by Qin et al. [QLY+15] recently. They extended it to a continuous setting and moreover established connections with related-key security. In that, they proposed a framework for for RKA-secure public-key encryptions, identity-based encryptions, signatures and provided new schemes for a class of polynomial tampering functions of bounded degree.

Generic BLT security [DFMV15] In a recent follow up of our results on bounded tamper-resilience (c.f. Chapter 3) Damgård et al. [DFMV15] showed how to transform an arbitrary crypto-scheme into one satisfying a slightly weaker form of BLT security; the number of tampering queries tolerated, however, is significantly smaller than the one achieved by our constructions (given in Chapter 3). In fact, the transformation in [DFMV15] can also be understood as applying a non-malleable key derivation function to the secret-state.

[1] Although we showed in Chapter 6 that information theoretic continuous non-malleable codes can not exist for $\mathcal{F}_{\mathsf{split}}$, it leaves possibilities for such codes against some different family other than $\mathcal{F}_{\mathsf{split}}$

7.2 Future directions

In this dissertation we mainly discussed *theoretical* methods of protecting crypto-devices against tampering attack. Generally theoretical models rely on several assumptions which fails to hold in many practical scenarios. However, security in a "reasonable" theoretical model is still much desirable to establish a profound logical foundation. Obviously, the most natural goal is to minimize (if possible obliterate) the gap between the theoretical model and the practical setting. As discussed earlier, one of the hope to consider weaker model like memory-only tampering was that they could be a first step for full-fledged protection against tampering. The subsequent works discussed above indeed keeps the hope alive. However, there are still plenty of things left to explore in this direction in order to actually achieve a "truly practical" model. A recent breakthrough result [DDF14] in the leakage-resilience which took an important step to unify the theoretical and practical models fuels more optimism for its tampering counter-part.

Moreover, quite amazingly, the notion of non-malleable codes, which initially was proposed mainly as an abstraction to build tamper-resilient compiler, was established as an independent entity substantiated by numerous recent works. A lot of such works involves innovative techniques, new abstractions and rigorous mathematical tools (e.g. additive combinatorics [ADL14, CZ14]). Quite naturally such huge theoretical development calls for finding more crypto-applications of non-malleable codes which has started recently with a number of works [AGM+15a, CMTV14, CDTV14, CGM+15] (see Sec. 4.3 for a brief overview). Hopefully, in near future we will be able to see more applications of non-malleable codes into cryptography.

7.3 Conclusion

Though the primary attempt of this dissertation was to provide new solutions to practical crypto-problem, at the same time it contributes to the development of theoretical cryptography through new results on non-malleable codes. As discussed above, the works presented here already attracted significant research attentions. We believe that the exposition and analysis provided in the dissertation will have more impact in future research.

Bibliography

[ABF+03] Christian Aumüller, Peter Bier, Wieland Fischer, Peter Hofreiter, and J-P Seifert. Fault attacks on rsa with crt: Concrete results and practical countermeasures. In *Cryptographic hardware and embedded systems-CHES 2002*, pages 260–275. Springer, 2003.

[ABP15] Michel Abdalla, Fabrice Benhamouda, and Alain Passelègue. An algebraic framework for pseudorandom functions and applications to related-key security. In Gennaro and Robshaw [GR15], pages 388–409.

[ABPP14] Michel Abdalla, Fabrice Benhamouda, Alain Passelègue, and Kenneth G. Paterson. Related-key security for pseudorandom functions beyond the linear barrier. In Garay and Gennaro [GG14], pages 77–94.

[ACM+14] Per Austrin, Kai-Min Chung, Mohammad Mahmoody, Rafael Pass, and Karn Seth. On the impossibility of cryptography with tamperable randomness. In Garay and Gennaro [GG14], pages 462–479.

[ADKO15a] Divesh Aggarwal, Yevgeniy Dodis, Tomasz Kazana, and Maciej Obremski. Non-malleable reductions and applications. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 459–468, 2015.

[ADKO15b] Divesh Aggarwal, Stefan Dziembowski, Tomasz Kazana, and Maciej Obremski. Leakage-resilient non-malleable codes. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part I*, pages 398–426, 2015.

[ADL14] Divesh Aggarwal, Yevgeniy Dodis, and Shachar Lovett. Non-malleable codes from additive combinatorics. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 774–783. ACM, 2014.

[ADVW13] Shweta Agrawal, Yevgeniy Dodis, Vinod Vaikuntanathan, and Daniel Wichs. On continual leakage of discrete log representations. In *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II*, pages 401–420, 2013.

[ADW09] Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In *CRYPTO*, pages 36–54, 2009.

[AGM+15a] Shashank Agrawal, Divya Gupta, Hemanta K. Maji, Omkant Pandey, and Manoj Prabhakaran. Explicit non-malleable codes against bit-wise tampering and permutations. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, pages 538–557, 2015.

[AGM+15b] Shashank Agrawal, Divya Gupta, Hemanta K. Maji, Omkant Pandey, and Manoj Prabhakaran. A rate-optimizing compiler for non-malleable codes against bit-wise tampering and permutations. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part I*, pages 375–397, 2015.

[AHI11] Benny Applebaum, Danny Harnik, and Yuval Ishai. Semantic security under related-key attacks and applications. In *ICS*, pages 45–60, 2011.

[AK96] Ross Anderson and Markus Kuhn. Tamper resistance: a cautionary note. In *WOEC'96: Proceedings of the 2nd conference on Proceedings of the Second USENIX Workshop on Electronic Commerce*, pages 1–1, Berkeley, CA, USA, 1996. USENIX Association.

[BC10] Mihir Bellare and David Cash. Pseudorandom functions and permutations provably secure against related-key attacks. In *CRYPTO*, pages 666–684, 2010.

[BCM11] Mihir Bellare, David Cash, and Rachel Miller. Cryptography secure against related-key attacks and tampering. In *ASIACRYPT*, pages 486–503, 2011.

[BDL97] Dan Boneh, Richard A DeMillo, and Richard J Lipton. On the importance of checking cryptographic protocols for faults. In *Advances in Cryptology – EUROCRYPT '97*, pages 37–51. Springer, 1997.

[BDL01] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of eliminating errors in cryptographic computations. *J. Cryptology*, 14(2):101–119, 2001.

[BHHO08] Dan Boneh, Shai Halevi, Michael Hamburg, and Rafail Ostrovsky. Circular-secure encryption from decision diffie-hellman. In *CRYPTO*, pages 108–125, 2008.

[BK03] Mihir Bellare and Tadayoshi Kohno. A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications. In *EUROCRYPT*, pages 491–506, 2003.

[BKKV10] Zvika Brakerski, Yael Tauman Kalai, Jonathan Katz, and Vinod Vaikuntanathan. Overcoming the hole in the bucket: Public-key cryptography resilient to continual memory leakage. *FOCS*, 2010.

[Boc10] Ruhr-University Bochum. Side channel cryptanalysis lounge. `http://www.crypto.rub.de/en_sclounge.html`, 2010.

[BPT12] Mihir Bellare, Kenneth G. Paterson, and Susan Thomson. RKA security beyond the linear barrier: IBE, encryption and signatures. In *ASIACRYPT*, pages 331–348, 2012.

[BR94] Mihir Bellare and John Rompel. Randomness-efficient oblivious sampling. In *FOCS*, pages 276–287. IEEE Computer Society, 1994.

[BR13] Rishiraj Bhattacharyya and Arnab Roy. Secure message authentication against related key attack. In *FSE*, 2013.

[BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. pages 513–525, 1997.

[CDTV14] Sandro Coretti, Yevgeniy Dodis, Björn Tackmann, and Daniele Venturi. Self-destruct non-malleability. *IACR Cryptology ePrint Archive*, 2014:866, 2014.

[CDTV15] Sandro Coretti, Yevgeniy Dodis, Björn Tackmann, and Daniele Venturi. Non-malleable encryption: Simpler, shorter, stronger. Cryptology ePrint Archive, Report 2015/772, 2015. http://eprint.iacr.org/.

[CG88] Benny Chor and Oded Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM J. Comput.*, 17(2):230–261, 1988.

[CG14a] Mahdi Cheraghchi and Venkatesan Guruswami. Capacity of non-malleable codes. In *Innovations in Theoretical Computer Science, ITCS*, pages 155–168, 2014.

[CG14b] Mahdi Cheraghchi and Venkatesan Guruswami. Non-malleable coding against bit-wise and split-state tampering. In *TCC*, pages 440–464, 2014.

[CGM+15] Nishanth Chandran, Vipul Goyal, Pratyay Mukherjee, Omkant Pandey, and Jalaj Upadhyay. Block-wise non-malleable codes. *IACR Cryptology ePrint Archive*, 2015:129, 2015.

[CKM11] Seung Geol Choi, Aggelos Kiayias, and Tal Malkin. Bitr: Built-in tamper resilience. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, volume 7073 of *Lecture Notes in Computer Science*, pages 740–758. Springer, 2011.

[CMTV14] Sandro Coretti, Ueli Maurer, Björn Tackmann, and Daniele Venturi. From single-bit to multi-bit public-key encryption via non-malleable codes. *IACR Cryptology ePrint Archive*, 2014:324, 2014. To appear in TCC 2015.

[Cra96] Ronald Cramer. *Modular Design of Secure yet Practical Cryptographic Protocols*. PhD thesis, University of Amsterdam, November 1996.

[CZ14] Eshan Chattopadhyay and David Zuckerman. Non-malleable codes against constant split-state tampering. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 306–315. IEEE, 2014.

[DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, pages 423–440, 2014.

[DDN91] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography (extended abstract). In Cris Koutsougeras and Jeffrey Scott Vitter, editors, *STOC*, pages 542–552. ACM, 1991.

[DDV10] Francesco Davì, Stefan Dziembowski, and Daniele Venturi. Leakage-resilient storage. In Juan A. Garay and Roberto De Prisco, editors, *SCN*, volume 6280 of *LNCS*, pages 121–137. Springer, 2010.

[DF11] Stefan Dziembowski and Sebastian Faust. Leakage-resilient cryptography from the inner-product extractor. In *ASIACRYPT*, pages 702–721, 2011.

[DFMV13] Ivan Damgård, Sebastian Faust, Pratyay Mukherjee, and Daniele Venturi. Bounded tamper resilience: How to go beyond the algebraic barrier. In *ASIACRYPT*, pages 140–160, 2013.

[DFMV15] Ivan Damgård, Sebastian Faust, Pratyay Mukherjee, and Daniele Venturi. The chaining lemma and its application. In *Information Theoretic Security - 8th International Conference, ICITS 2015, Lugano, Switzerland, May 2-5, 2015. Proceedings*, pages 181–196, 2015.

[DHLAW10a] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Cryptography against continuous memory attacks. In *FOCS*, pages 511–520, 2010.

[DHLAW10b] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Efficient public-key cryptography in the presence of key leakage. In *ASIACRYPT*, pages 613–631, 2010.

[DK14] Dana Dachman-Soled and Yael Tauman Kalai. Securing circuits and protocols against 1/poly(k) tampering rate. In *TCC*, pages 540–565, 2014.

[DKL09] Yevgeniy Dodis, Yael Tauman Kalai, and Shachar Lovett. On cryptography with auxiliary input. In *STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 621–630, New York, NY, USA, 2009. ACM.

[DKO13] Stefan Dziembowski, Tomasz Kazana, and Maciej Obremski. Non-malleable codes from two-source extractors. In *CRYPTO*, pages 239–257, 2013.

[DLSZ15] Dana Dachman-Soled, Feng-Hao Liu, Elaine Shi, and Hong-Sheng Zhou. Locally decodable and updatable non-malleable codes and their applications. In Dodis and Nielsen [DN15], pages 427–450.

[DN15] Yevgeniy Dodis and Jesper Buus Nielsen, editors. *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part I*, volume 9014 of *Lecture Notes in Computer Science*. Springer, 2015.

[DORS08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139, 2008.

[DP08] Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302, 2008.

[DPW10] Stefan Dziembowski, Krzysztof Pietrzak, and Daniel Wichs. Non-malleable codes. In *ICS*, pages 434–452, 2010.

[DSK12] Dana Dachman-Soled and Yael Tauman Kalai. Securing circuits against constant-rate tampering. In *CRYPTO*, pages 533–551, 2012.

[DW09] Yevgeniy Dodis and Daniel Wichs. Non-malleable extractors and symmetric key cryptography from weak secrets. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 601–610. ACM, 2009.

[FF02] Marc Fischlin and Roger Fischlin. The representation problem based on factoring. In *CT-RSA*, pages 96–113, 2002.

[FKPR10] Sebastian Faust, Eike Kiltz, Krzysztof Pietrzak, and Guy N. Rothblum. Leakage-resilient signatures. In Daniele Micciancio, editor, *TCC*, volume 5978 of *LNCS*, pages 343–360. Springer, 2010.

[FMNV14] Sebastian Faust, Pratyay Mukherjee, Jesper Buus Nielsen, and Daniele Venturi. Continuous non-malleable codes. In *TCC*, 2014.

[FMNV15] Sebastian Faust, Pratyay Mukherjee, Jesper Buus Nielsen, and Daniele Venturi. A tamper and leakage resilient von neumann architecture. In Jonathan Katz, editor, *Public-Key Cryptography - PKC 2015 - 18th IACR International Conference on Practice and Theory in Public-Key Cryptography, Gaithersburg, MD, USA, March 30 - April 1, 2015, Proceedings*, volume 9020 of *Lecture Notes in Computer Science*, pages 579–603. Springer, 2015.

[FMVW14] Sebastian Faust, Pratyay Mukherjee, Daniele Venturi, and Daniel Wichs. Efficient non-malleable codes and key-derivation for poly-size tampering circuits. In *EUROCRYPT*, pages 111–128, 2014.

[FPV11] Sebastian Faust, Krzysztof Pietrzak, and Daniele Venturi. Tamper-proof circuits: How to trade leakage for tamper-resilience. In *ICALP (1)*, pages 391–402, 2011.

[FRR+10] Sebastian Faust, Tal Rabin, Leonid Reyzin, Eran Tromer, and Vinod Vaikuntanathan. Protecting circuits from leakage: the computationally-bounded and noisy cases. In *EUROCRYPT 2010*, pages 135–156, 2010.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.

[GG14] Juan A. Garay and Rosario Gennaro, editors. *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*. Springer, 2014.

[GIP+14] Daniel Genkin, Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and Eran Tromer. Circuits resilient to additive attacks with applications to secure computation. In *STOC*, pages 495–504, 2014.

[GLM+04] Rosario Gennaro, Anna Lysyanskaya, Tal Malkin, Silvio Micali, and Tal Rabin. Algorithmic tamper-proof (atp) security: Theoretical foundations for security against hardware tampering. In *TCC*, pages 258–277, 2004.

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

[GOR11] Vipul Goyal, Adam O'Neill, and Vanishree Rao. Correlated-input secure hash functions. In *TCC*, pages 182–200, 2011.

[GQ88] Louis C. Guillou and Jean-Jacques Quisquater. A "paradoxical" indentity-based signature scheme resulting from zero-knowledge. In *CRYPTO*, pages 216–231, 1988.

[GR15] Rosario Gennaro and Matthew Robshaw, editors. *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*. Springer, 2015.

[Gro06] Jens Groth. Simulation-sound nizk proofs for a practical language and constant size group signatures. In *ASIACRYPT*, pages 444–459, 2006.

[GST14] Daniel Genkin, Adi Shamir, and Eran Tromer. Rsa key extraction via low-bandwidth acoustic cryptanalysis. In *Advances in Cryptology–CRYPTO 2014*, pages 444–461. Springer, 2014.

[HAS10] N. Homma, T. Aoki, and A. Satoh. Electromagnetic information leakage for side-channel analysis of cryptographic modules. In *Electromagnetic Compatibility (EMC), 2010 IEEE International Symposium on*, pages 97–102, July 2010.

[HTH+06] D Hwang, Kris Tiri, Alireza Hodjat, Bo-Cheng Lai, Shenglin Yang, Patrick Schaumont, and Ingrid Verbauwhede. Aes-based security coprocessor ic in 0.18-um cmos with resistance to differential power analysis side-channel attacks. *IEEE Journal of Solid-State Circuits*, 41(4):781–792, 2006.

[IPSW06] Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and David Wagner. Private circuits II: Keeping secrets in tamperable circuits. In *EUROCRYPT*, pages 308–327, 2006.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, pages 463–481, 2003.

[JW15] Zahra Jafargholi and Daniel Wichs. Tamper detection and continuous non-malleable codes. In Dodis and Nielsen [DN15], pages 451–480.

[Kal]

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptolog – CRYPTO '99*, pages 388–397. Springer, 1999.

[KKS11] Yael Tauman Kalai, Bhavana Kanukurthi, and Amit Sahai. Cryptography with tamperable and leaky memory. In *CRYPTO*, pages 373–390, 2011.

[Koc95] Paul C Kocher. Cryptanalysis of diffie-hellman, rsa, dss, and other crypto-systems using timing attacks. In *Advances in Cryptology – CRYPTO '95*. Springer, 1995.

[Koc96] Paul C Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in Cryptology – CRYPTO '96*, pages 104–113. Springer, 1996.

[KQ07] Chong Hee Kim and Jean-Jacques Quisquater. Fault attacks for crt based rsa: New attacks, new results, and new countermeasures. In *Information Security Theory and Practices. Smart Cards, Mobile and Ubiquitous Computing Systems*, pages 215–228. Springer, 2007.

[KV09] Jonathan Katz and Vinod Vaikuntanathan. Signature schemes with bounded leakage resilience. In *ASIACRYPT*, pages 703–720, 2009.

[L+02] Tom Lash et al. A study of power analysis and the advanced encryption standard. *MS Scholarly Paper, George Mason University*, 2002.

[LL12] Feng-Hao Liu and Anna Lysyanskaya. Tamper and leakage resilience in the split-state model. In *CRYPTO*, pages 517–532, 2012.

[LST12] Jie Li, Wei Wei Shan, and Chao Xuan Tian. Hamming distance model based power analysis for cryptographic algorithms. In *Applied Mechanics and Materials*, volume 121, pages 867–871. Trans Tech Publ, 2012.

[Luc04] Stefan Lucks. Ciphers secure against related-key attacks. In *FSE*, pages 359–370, 2004.

[MR04] Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In Moni Naor, editor, *TCC'04*, volume 2951 of *LNCS*, pages 278–296. Springer, 2004.

[MSS06] Amir Moradi, Mohammad T. Manzuri Shalmani, and Mahmoud Salmasizadeh. A generalized method of differential fault attack against AES cryptosystem. pages 91–100, 2006.

[NS09] Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. In *CRYPTO*, pages 18–35, 2009.

[Oka92] Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In *CRYPTO*, pages 31–53, 1992.

[PH78] Stephen Pohlig and Martin Hellman. An improved algorithm for computing logarithms over and its cryptographic significance. *IEEE Transactions on Information Theory*, 24(1):106–110, 1978.

[Pie09] Krzysztof Pietrzak. A leakage-resilient mode of operation. In *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, pages 462–482, 2009.

[Pie12] Krzysztof Pietrzak. Subspace LWE. In *TCC*, pages 548–563, 2012.

[QLY+15] Baodong Qin, Shengli Liu, Tsz Hon Yuen, Robert H. Deng, and Kefei Chen. Continuous non-malleable key derivation and its application to related-key security. In *Public-Key Cryptography - PKC 2015 - 18th IACR International Conference on Practice and Theory in Public-Key Cryptography, Gaithersburg, MD, USA, March 30 - April 1, 2015, Proceedings*, pages 557–578, 2015.

[RSV13] Ananth Raghunathan, Gil Segev, and Salil P. Vadhan. Deterministic public-key encryption for adaptively chosen plaintext distributions. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 93–110. Springer, 2013.

[SA02] Sergei P. Skorobogatov and Ross J. Anderson. Optical fault induction attacks. pages 2–12, 2002.

[SCO+01] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In *CRYPTO*, pages 566–598, 2001.

[ST04] Adi Shamir and Eran Tromer. Acoustic cryptanalysis. *presentation available from http://www. wisdom. weizmann. ac. il/ tromer*, 2004.

[SYO09] Kazuo Sakiyama, Tatsuya Yagi, and Kazuo Ohta. Fault analysis attack against an AES prototype chip using RSL. pages 429–443, 2009.

[Wee12] Hoeteck Wee. Public key encryption against related key attacks. In *Public Key Cryptography*, pages 262–279, 2012.