Protecting Distributed Primitives against Leakage: Equivocal Secret Sharing and More

Carmit Hazay
Bar-Ilan University
carmit.hazay@biu.ac.il

Muthuramakrishnan Venkitasubramaniam
Georgetown University
vmuthu@gmail.com

Mor Weiss
Bar-Ilan University
mor.weiss@biu.ac.il

Abstract

Leakage-resilient cryptography aims to protect cryptographic primitives from so-called "side channel attacks" that exploit their physical implementation to learn their input or secret state. Starting from the works of Ishai, Sahai and Wagner (CRYPTO'03) and Micali and Reyzin (TCC'04), most works on leakage-resilient cryptography either focus on protecting general computations, such as circuits or multiparty computation protocols, or on specific non-interactive primitives such as storage, encryption and signatures. This work focuses on leakage-resilience for the middle ground, namely for *distributed and interactive* cryptographic primitives.

Our main technical contribution is designing the *first* secret-sharing scheme that is *equivocal*, resists *adaptive* probing of a *constant fraction* of bits from each share, while incurring only a *constant* blowup in share size. Equivocation is a strong leakage-resilience guarantee, recently introduced by Hazay et al. (ITC'21). Our construction is obtained via a general compiler which we introduce, that transforms *any* secret-sharing scheme into an equivocal scheme against adaptive leakage. An attractive feature of our compiler is that it respects additive reconstruction, namely, if the original scheme has additive reconstruction, then the transformed scheme has linear reconstruction.

We extend our compiler to a general paradigm for protecting distributed primitives against leakage, and show its applicability to various primitives, including secret sharing, verifiable secret sharing, function secret sharing, distributed encryption and signatures, and distributed zero-knowledge proofs. For each of these primitives, our paradigm transforms *any* construction of the primitive into a scheme that resists adaptive party corruptions, as well as adaptive probing leakage of a constant fraction of bits in each share when the share is stored in memory (but not when it is used in computations). Moreover, the transformation incurs only a constant blowup in the share size, and respects additive reconstruction – an important feature for several of these primitives, such as function secret sharing and distributed encryption.

Contents

1 Introduction

Starting with the works of Kocher et al. [Koc96, KJJ99], and more recently in works on Meltdown and Spectre attacks [LSG+18, KHF+19]), it has been demonstrated time and again that cryptographic primitives are susceptible to side-channel attacks.

The goal of leakage-resilient cryptography is to protect against such attacks. The focus of this work is on constructing lightweight leakage-resilient variants of central distributed and interactive cryptographic primitives. We first survey the main results in the field.

There has been a long and successful line of works on protecting either *general computations* (initiated by Ishai, Sahai and Wagner [ISW03], and Micali and Reyzin [MR04]), or *specific non-interactive primitives* such as storage (e.g., in [Dzi06, CLW06, AGV09]), signatures [BSW11], and encryption schemes [DKL09, AGV09, NS09, DGK+, BG10, BS11]. (We refer the interested reader to the excellent survey by Kalai and Reyzin [KR19] for a discussion of works in the field.)

Previous works on leakage-resilience. Works protecting general computations from leakage constrain the permissible leakage in various ways, either restricting the complexity class from which leakage functions can be chosen [ISW03, FRR+10, Rot12, MV13, Mil14, ADF16], or assuming leakage functions operate on disjoint sets of wires of the circuit [GR10, JV10, GR12, DF12, DLZ15, DDN15, GIM+16] (the so called "Only Computation Leaks" – or OCL – model). Due to their generality, these works are usually complex, rely on additional computational assumptions (such as public-key cryptography [GR10, JV10, GIM+16] or even indistinguishability obfuscation [GIM+16]), or incur high (polynomial) overheads. There are also works on leakage-resilient Multi-Party Computation (MPC) protocols [BGJK12], as well as works on interactive protocols (such as MPC [BCH12, BGJ+13, BDL14] and zero-knowledge proofs [GJS11, BCH12]) that achieve a weaker leakage-resilience guarantee known as *leakage tolerance*, which allows the *ideal-world adversary* to obtain leakage on the inputs of the honest parties.

On the other extreme, there are leakage-resilient constructions of storage [Dzi06, CLW06, AGV09, DKL09, BKKV10, DHLW10]), signatures [BSW11], and encryption schemes [DKL09, AGV09, NS09, DGK+, BG10, BS11]. While focusing on these primitives admits simpler and more efficient constructions than in the general case, these constructions are naturally more limited since they offer solutions for specific (and non-interactive) tasks only, and do not readily extend to the distributed setting in which multiple parties wish to jointly compute on their inputs in a leakage-resilient manner.

There are only a handful of works on the middle ground, namely on protecting *distributed (and interactive) primitives* from leakage, despite the centrality of such primitives in designing cryptographic protocols.[1] Specifically, Boyle et al. [BGK11] use randomness extractors to construct coin-tossing and verifiable secret sharing secure against a single leakage query (i.e., non-adaptive leakage), and [GJS11, BCH12] design leakage-tolerant zero-knowledge proofs (as noted above, leakage tolerance is a weaker notion than leakage-resilience).

Leakage-resilient secret sharing. Unlike other distributed primitives, the leakage-resilient notion of *secret sharing* – an important cryptographic primitive with various applications in different areas of cryptography – has received extensive attention from the community in recent years [DP07, DDV10, GK18a, GK18b, BDIR18, SV19, BS19, KMS19, ADN+19, CGG+20, LCG+20, MNP+21, MPSW21, TX21, CKOS21]. Focusing on secret sharing, which only protects *information*

[1] We note that one could protect such primitives from leakage using works protecting general computations, or by employing generic leakage-resilient MPC protocols. However, as discussed above these are complex, and either incur high costs or require additional computational assumptions.

(as opposed to protecting *computation*, as in the works on general leakage-resilient computation discussed above) allows to obtain simpler and more efficient constructions.

Works on Leakage-Resilient Secret Sharing Schemes (LR-SSSs) can be roughly divided into two categories. The first category, initiated by [BDIR18] (though some results appeared already in earlier works, e.g., [BIVW16]), analyzes the leakage-resilience properties of *existing* secret-sharing schemes. They show that linear secret sharing schemes[2] resist *non-adaptive local* leakage, namely the adversary can make a single leakage query, which returns few leakage bits computed on each share independently.

Proving leakage-resilience of existing primitives has several advantages. First, applications can still exploit specific design features or additional properties, such as additivity or linearity of reconstruction (which are very useful, as discussed in Section 1.4 below). Moreover, classic cryptographic primitives are usually more efficient than special-purpose leakage-resilient schemes. However, restricting oneself to existing primitives has a major disadvantage: we are restricted by the leakage-resilience properties of a scheme, which was *not designed with the goal of protecting against leakage*. In some cases, it is unknown whether leakage resilience can be enhanced (e.g., to allow *adaptive* leakage in which the adversary makes several leakage queries, each depending on the answers to previous queries). There are also inherent limitations to the leakage-resilience of some schemes, in terms of the leakage bound and reconstruction threshold (see Section 1.5). For example, Shamir's secret sharing scheme over fields of characteristic 2 does not resist leakage of even *a single bit* from each share [GW16].

These limitations of existing secret sharing schemes have motivated the second category of LR-SSSs which focuses on designing *new* schemes – thus offering the possibility to overcome the inherent limitations of existing schemes. The goal of this line of works is to resist leakage from a wide range of leakage functions and arbitrary thresholds [ADN+19, SV19, KMS19, CGG+20, LCG+20, CKOS21]. The advantage of designing new schemes is clear: we can (potentially) protect against a wide range of leakage classes, and in particular ones against which classic secret sharing schemes are insecure, such as adaptive leakage queries [KMS19, CGG+20, LCG+20, CKOS21], or global leakage (for restricted classes of permissible leakage functions) [BIVW16, LCG+20]. Unfortunately, the stronger leakage-resilience guarantee does not come without a price: these constructions are typically complex [KMS19, CGG+20, LCG+20, CKOS21]; some works are considerably less efficient than standard (non-LR) schemes [GK18a, GK18b, CGG+20]; and most (if not all) works do not maintain the design benefits of classic constructions (such as linear reconstruction) and thus cannot be used in many applications of standard secret sharing [DP07, DDV10, GK18a, GK18b, ADN+19, SV19, BS19, KMS19, CGG+20, LCG+20, CKOS21]. Some of these works describe a general compiler that transforms any secret sharing schem into a leakage-resilient one [SV19, ADN+19, CGG+20], while others describe specially-tailored constructions [GK18a, KMS19, LCG+20, CKOS21].

The literature on LR-SSSs can be viewed as studying two extremes: either focusing on *existing* schemes and analyzing their leakage-resilience guarantees, or alternatively *pushing leakage-resilience "to the limit"*, mostly by designing new specialized (and often also complex and significantly less efficient) schemes.[3] But what can we obtain by only *slightly modifying* existing schemes? In particular,

Can we obtain LR-SSSs that surpass the leakage-resilience of classic schemes while maintaining their structural features?

More generally, we study the leakage-resilience of distributed cryptographic primitives such

[2] Roughly, in a linear secret sharing over a finite field each share is a linear combination of the secret and random field elements.

[3] As noted above, the works of [SV19, ADN+19, CGG+20] construct general compilers. However, [SV19, ADN+19] resists only non-adaptive leakage and the construction of [CGG+20] has low rate.

as Verifiable Secret Sharing (VSS), Function Secret Sharing (FSS), Threshold Encryption (TES), threshold signatures, and distributed Zero-Knowledge (dZK) proofs, asking the following:

Do there exist simple leakage-resilient variants of distributed cryptographic primitives such as VSS, FSS, TES, threshold signatures, and dZK?

We focus on a setting in which the adversary can maliciously and adaptively corrupt parties (i.e., the adversary does not need to commit ahead of time to the set of corrupted parties). Similar to the case of secret sharing, our goal is to devise *simple* schemes that *preserve the main design features* of existing (non-LR) schemes. To the best of our knowledge, there are no leakage-resilient constructions of FSS, TES, distributed signatures, and dZK proofs, even disregarding the desiderata of simple schemes. Existing leakage-resilient VSS schemes [BGK11] are only secure against static corruptions (i.e., when the set of corrupted parties is determined at the onset of the computation), and employ randomness extractors that do not preserve linearity, which – as explained below – is one of our main goals in this work. (See Section 1.5 for a detailed discussion of the VSS scheme of [BGK11], and a comparison with our results.)

Considering leakage-resilience in the more general context of distributed primitives also helps in identifying the important structural features of existing schemes which we should strive to preserve. For example, applications of FSS require linear sharing (namely, that the reconstruction procedure is linear), and applications of TES in MPC require that parties hold additive shares of a secret key.[4] Thus, in this work we focus on designing leakage-resilient secret sharing schemes – and more generally, distributed primitives – with *linear reconstruction*.

Our leakage model: adaptive probing-resilience for data at rest. We focus on protecting cryptographic primitives against *local probing leakage* (as discussed in Section 1.5, similar models have been considered in the past). We provide information-theoretic security against adversaries that can *adaptively* probe the shares, as well as corrupt *any* unauthorized set (i.e., a subset of parties such that their collective shares reveal nothing about the shared secret).[5] Our focus on probing resilience stems from our goal of preserving linear reconstruction, which implies we cannot hope to protect against leakage classes that are significantly more general than probing.

For distributed primitives beyond secret sharing, we focus on protecting *data at rest*. That is, we protect the shares from leakage throughout the lifetime of the process, *except* when they are directly computed on. Our leakage model is related to several leakage models (in particular, that of [BDIR18]), and can be seen as a combination of the leakage model of [ISW03] that restricts the function class from which leakage functions can be chosen, and the "Only Computation Leaks" (OCL) model of [MR04] which assumes that different memory components leak separately. (As noted in Section 1.1 below, our constructions actually achieve a stronger notion known as OCL+ [BCG+11].)

Our focus on protecting data at rest is motivated by many application scenarios in which following a short sharing phase – in which one party locally generates and distributes shares to the other parties – the shares are stored in memory for extended time periods, during which the adversary can adaptively leak on them. Since leakage may accumulate over time, and the adversary might be able to infiltrate the devices of certain parties (in effect making them – passively or actively – corrupted), it is important to incorporate party corruptions into the security model. We expect the

[4] For example, in the TES based on Diffie-Helman, parties hold public-key shares of the form $g^{\mathrm{SK}_1}, \ldots, g^{\mathrm{SK}_m}$, where g is a public group generator, and $\mathrm{SK}_1, \ldots, \mathrm{SK}_m$ are the secret-key shares. In this case, the public key is $g^{\sum_i \mathrm{SK}_i}$ and the secret key $\mathrm{SK} = \sum_i \mathrm{SK}_i$ is unknown to any party. See Section 5 for further details.

[5] For primitives with computational security (such as FSS and TES), we provide the best possible security, namely computational security for party corruptions, and *information-theoretic* for leakage. See Sections 1.4 and 5 for further details on the security definition.

periods of time during which shares are computed on – which involve fast local computations, or fast interactive protocols requiring all parties to be online simultaneously – to be much shorter than the lifetime of the system (indeed, state-of-the-art practical secure multi-party computation protocols last milliseconds [YWZ20]). Therefore, an adversary would not be able to launch a meaningful leakage attack during these short computation phases.

Example: Function Secret Sharing (FSS). We demonstrate our leakage model and its motivation through the example of FSS. FSS consists of a generation phase in which a function f is shared between multiple parties by generating function keys; and a local evaluation phase – whose duration we expect to typically be marginal compared to the lifetime of the system – in which each party can locally compute from its function key a share of the output $f(x)$, for any input x. These output shares form an additive (or, more generally, linear) secret sharing of $f(x)$. During this process, we protect the function keys and output shares from leakage, *except* during the evaluation phase in which the output shares are (locally) computed from the function keys. (We stress that once the evaluation phase terminates, function keys and output shares are again protected from leakage.)

In FSS applications, the function and output shares are often stored in memory for extended time periods, thus naturally necessitating leakage-resilience for data at rest. Indeed, in one FSS application, statistics (such as website traffic statistics [BGI16]) – in the form of output shares – are accumulated over time. In particular, output shares are stored in memory for long periods of time. Another application is "FSS as a service", namely when the function keys are distributed between multiple servers (each server holds a single function key), and users can ask the servers to evaluate the function on inputs of their choice. More specifically, a data-owner generates function keys $k_1, \ldots, k_m$ for a secret function f, and stores k_i on server i. Users can then ask the servers for the value $f(x)$ by sending x to each server i, who locally evaluates its function key k_i on x (using the FSS-evaluation procedure), and sends the output share y_i back to the user, who can then recover $f(x)$ from the output shares $y_1 \ldots, y_m$. Thus, the user learns $f(x)$ but does not learn f, and none of the servers learn f or $f(x)$. In this scenario, function keys may be stored on the servers indefinitely, and output shares y_i might likewise be stored for extended time periods, e.g., for backup purposes, so it is important to protect both from leakage.

Our leakage model is further motivated by other situations in which shares are stored in memory for a long time, e.g., when using Beaver triples [Bea91] that are generated in an offline phase and later consumed during the execution of an MPC protocol. We describe further applications (e.g., for TES) in Sections 1.4 and 5, and note that in some cases (e.g., for certain TES constructions) we are able to protect *the entire process* – including computation – from leakage.

Beyond leakage-resilience. In the context of secret sharing, we also explore a stronger leakage-resilience guarantee known as *equivocation*, recently introduced by [HVW21] and employed towards constructing zero-knowledge proof systems. Roughly, equivocation guarantees that there exists an efficient simulator that can answer adaptive leakage queries, as well as adaptively simulate the views of corrupted parties. Moreover, at some point during the simulation, the simulator is given an *arbitrary* secret, and is then able to generate an entire secret sharing of this secret which is consistent with the leakage and the views of corrupted parties. That is, the leakage can be efficiently "explained" as leakage on the shares of *any* arbitrary secret. More specifically, we consider *semi-honest* adversaries that can *adaptively* obtain the shares of any unauthorized set, as well as adaptively probe a constant fraction of bits from the shares of every other party.

Unfortunately, the existing equivocal SSS [HVW21] is a degenerate one-party scheme,[6] so the adversary cannot obtain the share in full. Therefore, we ask:

[6] In such schemes there is a single party and only one share. Secrecy holds under the supported leakage class, but not if the share is revealed in full. This is a degenerate scheme since it is not useful as a secret-sharing mechanism,

1.1 Our Results

We construct the *first multiparty equivocal secret sharing scheme*. Our scheme can withstand *any number* of corrupted parties, as well as leakage of a *constant fraction* of bits from each of the other shares. Our scheme is also *tamper-resilient* against arbitrary tampering of every share that modifies a constant fraction of bits (below the error-correction bound) in the share. We then *extend our technique* to obtain leakage-resilience and tampering-resilience for *other distributed primitives*.

Equivocal secret-sharing. We design a compiler that transforms standard (possibly non-leakage-resilient) SSSs into LR-SSSs with the following features. First, it obtains the stronger guarantee of equivocation. Second, it respects additive reconstruction, namely if the underlying SSS has additive reconstruction, then the resultant LR-SSS will have linear reconstruction. Third, our compiler guarantees leakage-resilience against *any* access structure, in the following sense. Given any SSS for an access structure Acc, the resultant LR-SSS is leakage-resilient against an adversary obtaining the shares of an unauthorized set $T \notin \mathsf{Acc}$, as well as the answers to *adaptive* probing queries that can leak a *constant* fraction of bits from each share. Finally, the resultant scheme is also tampering-resilient in the following sense: the correct secret will be reconstructed, even if an external adversary can arbitrarily modify a constant fraction of bits *in each share*. In fact, we obtain the stronger guarantee that *the original share can be efficiently recovered* from the modified share. (Thus, necessarily, we can only handle tampering of a fraction of bits which is *below* the error-correction bound. See Section 1.5 for a comparison with other tampering-resilient notions.) Our compiler is also conceptually simple: roughly, it protects the shares by encoding them using a linear code with equivocation properties (known as an *RPE* [CDMW08, BDKM16, CDMW18, BDG+18], see Section 1.4).

Our construction is summarized in the following theorem, where (Acc, ℓ)-equivocation means the adversary can *adaptively* obtain the shares of any unauthorized set $T \notin \mathsf{Acc}$, as well as *adaptively* probe ℓ bits from every other share, and τ-tampering-resilience guarantees that the correct secret will be reconstructed even if an external adversary can arbitrarily modify τ bits in *every* share. (See Theorem 4.1 for the formal statement.)

Theorem 1.1 (Equivocal SSS from standard SSS – informal)**.** *Let* SSS *be an* m*-party secret sharing scheme for secrets in* $\mathcal{S}$ *and an access structure* Acc*, with shares of length* N*. Then there exists an* $\ell = \Omega(N)$ *for which there exists an* m*-party secret sharing scheme* SSS' *for secrets in* $\mathcal{S}$ *and access structure* Acc*, with* ℓ*-tampering-resilience, and* (Acc, ℓ)*-equivocation. Moreover, the secret shares have length* $O(N)$*. Furthermore, if* SSS *has additive reconstruction, then* SSS' *has linear reconstruction.*

In Table 1 (Page 17) we compare our equivocal SSS with existing LR-SSSs. Applying Theorem 1.1 to Shamir's secret sharing, we obtain the following (see Corollary 4.2 for the formal statement).

Corollary 1.2 (Equivocal SSS – informal)**.** *There exists an* $\ell \in \mathbb{N}$ *such that for any* $t < m \in \mathbb{N}$ *there exists an* m*-party* (Acc, ℓ)*-equivocal and* ℓ*-tampering-resilient secret sharing scheme, where* Acc *consists of all subsets of* $[m]$ *of size more than* t*. Moreover,* $\ell = \Omega(N)$*, where* N *is the share length.*

Corollary 1.2 demonstrates that by applying our transformation to Shamir's secret sharing scheme, we can circumvent the impossibility result of [GW16] on the leakage-insecurity of Shamir's scheme over fields of characteristic 2, as well as the lower bounds of [BDIR18] and [NS20] (which are discussed in Section 1.5), while still preserving the main features of Shamir's scheme.

but has application to, e.g., construction of zero-knowledge probabilistic proof systems.

Leakage-resilient distributed primitives. We extend our compiler to a general paradigm for securing cryptographic primitives against adaptive probing leakage, and show its applicability to a wide range of primitives in which a secret is distributed between multiple parties. Specifically, we obtain leakage-resilient variants of verifiable secret sharing, function secret sharing, threshold encryption and signatures, and distributed zero-knowledge proofs. In all our constructions, the blowup in the share size is constant (compared to the original, non-leakage-resilient scheme), and the resultant scheme is secure against adaptive probing of a constant fraction of bits from each share separately even if the adversary adaptively corrupts parties. (We note that in certain primitives – such as TES and FSS – each party may store several shares of different secrets, in which case we assume that each share leaks separately. As discussed above, this is similar to the restriction on leakage in the OCL model. However, our constructions satisfy a stronger guarantee known as OCL+ [BCG+11] which allows to alternately leak from the shares, where each leakage query might depend on the responses to previous queries. In particular, the leakage on different shares is not necessarily independent.) All our constructions are also tamper-resilient against tampering of a constant fraction of bits in each share. The constructions and formal theorem statements appear in Section 5.

We note that *defining* leakage-resilience for these primitives is highly *non-trivial*. This is especially true for computational primitives such as TES and FSS, in which we *need to combine the computational* security guarantee for *fully-revealed shares*, with an *information-theoretic guarantee for leakage* on the other shares.[7] Even for information-theoretic objects such as dZK proofs, we encounter subtleties since standard definitions are only *statically*-secure. As an additional contribution, we generalize these definitions to the adaptive setting, and prove that existing protocols (e.g., a protocol of [BBC+19a]) are adaptively-secure, a result which may be of independent interest. See Section 1.4, and the formal definitions in Section 5, for a discussion of the subtleties of the definitions.

1.2 Applications

In this section we describe several applications of equivocal SSSs.

1.2.1 Deniable Secret Sharing Schemes

Deniability in the context of encryption. The notion of deniable encryption, introduced by Canetti et al. [CDNO97], ensures private communication in the presence of attackers that are also provided with all the private information: the plaintext, the random bits used for encryption, and any secret keys the parties have. Specifically, deniable encryption introduces an additional efficient "faking" algorithm that is not part of the standard secure communication definitions. This new algorithm generates "fake" internal states for the participating parties, that are indistinguishable from the real states, and are consistent with the (public) communication transcript and any plaintext of the parties' choice. In addition to being an interesting object, deniable encryption has important applications, the most immediate one being preventing vote-buying in electronic voting schemes. Deniability is a very strong security guarantee. Specifically, deniable encryption schemes require stronger hardness assumptions than classic encryption schemes [SW14, CPP20].[8]

[7] We note that similar security models – with information-theoretic security against leakage, and computational security for the underlying primitive itself – have been considered in the leakage resilience literature, e.g., for public-key primitives in the bounded retrieval model [Dzi06, CLW06]. However, due to the distributed nature of the primitives we consider, our security definitions are more intricate than in these aforementioned works.

[8] We note that information-theoretic symmetric-key encryption schemes are easily deniable as every ciphertext can be explained with respect to every plaintext. However, symmetric-key encryption requires the parties to agree on the secret key, i.e., a-priori agree on common randomness, and it is unclear how to do so in a deniable way.

Deniable secret-sharing. The notion of deniability can be highly useful in applications using secret sharing, where a dealer distributes the shares of a sensitive database – e.g., a clinic sharing a database containing the medical records of its patients – amongst a set of parties or servers.[9] Clearly, if the attacker obtains sufficiently many shares (specifically, of an authorized set), it can fully recover the secret. On the other hand, obtaining only the shares of an unauthorized set implies deniability since the secret is information-theoretically hidden. (We stress, however, that even in this case the "faking" algorithm may not be efficient.) Deniable secret sharing explores an intermediate scenario where the adversary holds the shares of some unauthorized set, and *additionally obtains partial knowledge of the honest parties' shares.* This model is motivated by the fact that an attacker may attack servers in various ways, in some cases succeeding in extracting the entire share, while in other cases it may only be able to gather (through physical observations) some leakage on the share.

Formally defining deniable secret sharing requires some care, in particular in deciding what information is available to the faking algorithm. In the context of public-key deniable *encryption*, the faking algorithm knows the ciphertext it is trying to "explain", so a natural adaptation to *secret sharing* is to give the faking algorithm the secret shares it is trying to equivocate. This, however, is useless – it should be intuitively clear that the faking algorithm cannot successfully explain the leakage without knowing which parts of the secret shares had leaked. Therefore, we define deniability by giving the faking algorithm *the leakage*, rather than the entire secret sharing. Specifically, the faking algorithm is given the leakage queries and responses (but *not* the secret shares in their entirety), and using them "explains" the sharing as a sharing of an arbitrary (possibly different) secret, by providing "fake" randomness for the sharing algorithm. This notion of "public leakage" arises naturally in many settings, since oftentimes leaked data becomes public (e.g., via social or other forms of media). For example, in the application scenario described above where a clinic A shares its database of patient medical records, this database might contain medical records of celebrities. A competing clinic B might be able to obtain leakage on the shares, allowing it to learn the identity and medical history of some celebrity patients. Clinic B might publish this information with the hopes of ruining the reputation of clinic A. Deniability allows clinic A to give an alternative explanation for the shares – in particular, replacing the identities of all celebrity patients – and publish it to dismiss the leaked information.

Our deniable secret-sharing scheme. We design deniable secret sharing schemes based on equivocal SSSs, where the faking algorithm emulates the equivocal SSS simulator. We note that to equivocate the shares, the equivocal SSS simulator needs to know the leakage queries and responses, and these are indeed available to the faking algorithm. Using the equivocated secret shares, the dealer can convincingly renounce the original secret, where the equivocated shares serve as a "proof" that validates an arbitrary secret, *even given the adversary's view.* Denying the secret provides a strong guarantee against coercing the dealer, especially since our focus is on *efficient* deniability. We further elaborate on this application in Section 4.2.

1.2.2 MPC Resilient to Leakage of "Data at Rest"

Equivocal SSSs are useful towards designing Multiparty Computation (MPC) protocols which guarantee leakage-resilience of private inputs (alternatively, secret local states) when they are stored in memory. That is, MPC with leakage-resilience in the same "data at rest" leakage model described above. More specifically, we consider protocols in which following a leak-free *Sharing* phase, in which each party locally secret-shares its input (either sending them to other parties, or storing them locally until they will be communicated at a later point), the shares are stored in memory until a later leak-free *Computation* phase, in which parties run an MPC protocol on the secret shares to compute

[9] We assume no a-priori common randomness between the dealer and the parties.

the outcome. We stress that the sharing phase is *function oblivious*, namely it can be executed before parties even know which function they wish to jointly compute on their inputs.

Similar to the other primitives considered in this work, this model is motivated by realistic application scenarios in which the (short) computation phase might be executed well after the (short) sharing phase. In particular, the leak-free assumption for the sharing phase is justified by situations in which parties obtain their secret inputs (e.g., bank account balance when the input is for the Millionaire's problem) while in their own homes, i.e., in locations which are less vulnerable to side-channel attacks; or because once they are given their private inputs, parties are likely to quickly "protect" them by performing a fast local computation (i.e., sharing, similar to how one would protect a sensitive word document by adding a password). The leak-free assumption for the computation phase is likewise justified because interactive protocols that require all parties to be online at the same time, and for the duration of the execution, are expected to be fast.

However, between the sharing and computation phases, the secret shares are stored in the internal state of the parties, e.g., on their laptops. Thus, the shares might be subjected to various side channel attacks (as laptop would most likely be carried around by their owners), especially since the computation phase might be executed long after the sharing phase had ended. Moreover, during this time leakage on one party's state might accumulate, up to the point that its internal state might leak entirely. Alternatively, an adversary might be able to infiltrate the party's laptop, thus obtaining its entire internal state.

Equivocal SSSs naturally give a leakage-resilient solution for MPC protocols in this leakage model, against adaptive leakage, by having parties share their inputs using an equivocal SSS. Specifically, leakage on the shares can be adaptively simulated, where if the actual amount of leakage on a party passes some a-priori bound then we consider the party as being (semi-honestly) "corrupted", which is supported by the equivocal SSS (that allows for adaptively leaking full shares). If, at the onset of the computation phase, a party is corrupted, we can equivocate the shares consistently with its actual input. (We note that if the MPC protocol run during the computation phase is adaptively-secure, then we can also handle adaptive corruptions *during* the computation phase.) Thus, equivocal SSSs can be used to protect MPC protocols from adaptive leakage and adaptive semi-honest party corruptions. We note that LR-SSSs which are *not* equivocal provides no guarantees against *adaptive* party corruptions (and sometimes not even against adaptive *leakage queries*).

1.3 Future Directions and Open Questions

We initiate the study of designing leakage-resilient cryptographic primitives by applying a light-weight procedure which respects additive reconstruction. We demonstrate the usefulness of our paradigm by applying it to various cryptographic primitives.

Our work still leaves several interesting research directions to explore. First, it would be interesting to explore which other primitives could be protected from probing leakage using our paradigm. Second, our constructions "respect" additive reconstruction, in the sense that applications relying on additive reconstruction can use our leakage-resilient primitives. As discussed above, this is helpful in the context of FSS and TES. Extending our paradigm to respect *general linear* reconstruction procedures will allow using our leakage-resilient constructions in an even wider array of applications. Moreover, devising an equivocal SSS with a multiplicity property (which enables multiplying shared secrets by operating locally on the shares, and then executing some simple "correction" protocol), would yield leakage-resilient MPC which resists leakage even during computation, thus extending our results beyond the "data at rest" model. Finally, it is natural to ask whether our results extend to more general leakage classes.

1.4 Techniques

In this section we describe our paradigm and the resultant constructions in more detail. We start by describing our compiler for SSSs.

Recall from Section 1.1 that our goal is to obtain simple SSSs with linear reconstruction and probing resilience against an adaptive adversary that can probe a constant fraction of bits from each share. More generally, we wish to design a general compiler from SSSs to LR-SSSs which preserves efficiency and respects additive reconstruction (namely, if the original scheme has additive reconstruction then the resultant LR scheme will have linear reconstruction). A natural starting point would be to use some sort of encoding, similar to how leakage-resilient compilers for general computations first encode the input for the computation. This encoding can either apply to the vector of secret shares, or to each share separately. To obtain leakage-resilience, it should posses some leakage-resilience guarantees. Moreover, to achieve linear reconstruction, this encoding should have linear reconstruction (i.e., be a linear code). However, to obtain *equivocation*, as needed by our main application of equivocal SSSs, the linear code should posses an additional equivocation property which, roughly, guarantees that given the leakage on the encoding of any message msg, and given any arbitrary message msg', the leakage can be "explained" in retrospect as the leakage on an encoding of msg'. We note that such leakage-resilience and equivocation properties necessitate *randomized* encoding. Fortunately, such encodings exist.

Main building block: Reconstructible Probabilistic Encodings (RPEs). RPEs [CDMW08, BDKM16, CDMW18, BDG+18] are, roughly, linear codes with an equivocation property. More specifically, an RPE consists of a randomized encoding procedure Encode, a deterministic linear decoder Decode, and a randomized resampler algorithm Rec, where $\mathsf{Encode}(\mathsf{msg})$ outputs a random codeword from a set of possible codewords for msg. (For example, this set can consist of all codewords obtained by applying the generator matrix of a linear code to $\mathsf{msg} \circ s$, namely to the string obtained by concatenating an arbitrary suffix s to msg.) An RPE satisfies the following properties. First, it has error correction from a constant fraction of errors. Second, it has probing resilience for a constant fraction of probed bits, in the sense that for every $\mathsf{msg}, \mathsf{msg}'$, the leakage on random encodings of $\mathsf{msg}, \mathsf{msg}'$ are statistically close. Finally, it is equivocal in the sense that for any message msg, any subset $\mathcal{I}$ of probed bits leak from a random encoding $c \leftarrow \mathsf{Encode}(\mathsf{msg})$ of msg, and any message msg', $\mathsf{Rec}(\mathcal{I}, \mathsf{leak}, \mathsf{msg}')$ outputs an encoding c' of msg' which is statistically close to an encoding of msg' that is random subject to being consistent with the leakage.[10] We note that RPEs resist non-adaptive leakage (also in the equivocation property), but since all properties are statistical, this naturally extends to resisting adaptive leakage, as we show in Section 3.1.

1.4.1 Equivocation from RPEs and Standard Secret Sharing

There is a natural connection between RPEs and equivocal secret sharing, since both offer a method of encoding a message in a way that can later be equivocated. This connection was recently used in [HVW21], who used the equivocation of RPEs to construct zero-knowledge probabilistic proof systems. More specifically, Hazay et al. [HVW21] interpreted RPEs as a degenerate form of equivocal secret sharing, in which there is a single share, which the adversary can probe. This, of course, does not provide any meaningful way of *distributing* (i.e., sharing) the data, which is the main purpose

[10] We note that RPEs are usually defined with perfect security, in the sense that leakage-resilience guarantees that the bits leaked from a random encoding of any message msg are uniformly distributed, and the output of Rec is distributed identically to a random encoding of msg' subject to being consistent with the leakage. We chose to present a relaxed version of RPEs because it suffices for our needs, and allows us to quantify exactly how errors in the RPE carry over to the resultant constructions.

of a SSS (but it does give a meaningful notion of equivocal *encoding* of the data). Moreover, equivocal SSSs can provide a much stronger LR guarantee – the secret remains protected even if an (unauthorized) subset of shares are *leaked in full*. In particular, equivocal SSSs can potentially obtain a much larger leakage rate than RPEs.

Our main observation in this work is that instead of interpreting RPEs as equivocal SSSs, one can *use* them to *transform* essentially *any* standard SSS into an equivocal one. The transformation is conceptually simple: to share a secret, first share it using the underlying SSS, then encode each share separately using the RPE. To reconstruct the secret, first decode each RPE encoding, then recover the secret using the reconstruction procedure of the underlying SSS. The resultant scheme resists adaptive probing leakage, since leakage can be simulated by answering queries using encodings of 0 (or any arbitrary message). Moreover, a combination of the equivocation of the RPE and the secrecy of the underlying SSS guarantees that the resultant SSS is equivocal. Indeed, a simulator can initially answer leakage queries according to an honestly-generated sharing of 0. At some point, the simulator is given an arbitrary secret msg'. At this point, secrecy of the underlying SSS guarantees the simulator can generate a secret sharing of msg' which is consistent with the subset of shares that were *revealed in full* to the adversary. Then, the resampler algorithm of the RPE can be used to equivocate encodings for the remaining shares, consistently with the leakage already provided on them. (We note that the actual analysis – which appears in Section 4 – is more complex.) Finally, tampering-resilience follows from the error-correction of the RPE.

The equivocal SSS described in Corollary 1.2 is obtained by applying our compiler to Shamir's secret sharing and the RPE of [DGR97, DGR99]. More specifically, Decatur et al. [DGR97, DGR99] construct a linear code with constant rate, and leakage resilience against a constant fraction of leaked bits. Linearity of the code implies it is also equivocal due to its large dual distance (see [BDG+18, Lemma 2]).

1.4.2 Share-then-Encode: A General Paradigm for Leakage- and Tampering-Resilience

We show that our technique is applicable to a wide range of distributed cryptographic primitives, constructing leakage-resilient primitives against probing leakage of data at rest, that are also tampering-resilient. For the primitives discussed below, we identify distinct sharing and reconstruction procedures, and protect the shares by RPE-encoding each share separately. These primitives include protocols and procedures during which the shares are computed on. To execute these protocols/procedures, parties first RPE-decode their share, perform the computation on it, RPE-encode the result, and finally erase the intermediate values of the computation. We note that several of the applications described below crucially rely on the linear reconstruction property of the underlying primitive.

In all our constructions, the blowup in share size (compared to the underlying non-leakage-resilient scheme) is constant, and the resultant scheme resists adaptive probing and tampering of a constant fraction of bits from each share separately. We stress that security is maintained even if the adversary simultaneously corrupts parties *and* obtains leakage on the shares of the honest parties. By default, we allow for adaptive corruptions as well as adaptive leakage queries. With the exception of verifiable secret sharing, the primitives we study below are usually defined with *static* security. Thus, to obtain security in the presence of leakage with adaptive party corruptions, we need to first define adaptive security for these primitives (*without* leakage), which we then extend to adaptive security in the presence of leakage. As we explain below, defining adaptive leakage-resilience is quite subtle, especially for primitives (such as threshold encryption and function secret sharing) with computational security. We note that if one is willing to settle for security against adaptive leakage queries but with *static party corruptions*, then we can simplify our definitions to only handle static

corruptions (and rely on the standard static-security of the underlying primitives), however defining security in the presence of leakage remains quite intricate also in this case.

Verifiable Secret Sharing (VSS). A VSS scheme strengthens standard SSSs to protect against a scenario in which the entity generating the shares (called the "dealer") is corrupted. More specifically, a VSS scheme consists of sharing and reconstruction procedures as in standard SSSs, but is also associated with a *Verification* protocol in which parties verify their shares are consistent.[11] VSS has secrecy and correctness as in standard secret sharing, and guarantees an additional *commitment* property, stating that even if a small subset of corrupted parties collude with a corrupted dealer, then following the verification phase the dealer is committed to *some* secret which will be reconstructed, regardless of the behavior of the corrupted parties during reconstruction.

We define leakage-resilient VSS as VSS schemes with a stronger secrecy guarantee that holds even in the presence of leakage on the shares. This is formalized through a distinguishing game between the adversary and a challenger, in which the adversary can adaptively corrupt parties throughout the sharing and verification phases (as in standard VSS), but once verification ends, it can additionally adaptively probe a constant fraction of bits from the share of every honest party. (Our construction also generalizes to protect against leakage between the sharing and verification phases.) We require that the adversary obtains only a negligible advantage in distinguishing between any two secrets shared by the dealer.

We obtain LR-VSS similarly to our equivocal SSS described above. Roughly, we transform a standard VSS scheme VSS into a Leakage-Resilient VSS (LR-VSS) VSS' by having the dealer generate the shares as in VSS, and then the parties verify the shares by running the verification protocol of VSS. Once verification ends, each party RPE-encodes its updated share and erases all information except for this (encoded) updated share.

Given that VSS is oftentimes used as a building block in constructing general multiparty computation protocols, as well as specific distributed tasks (e.g., coin tossing), we believe that our LR-VSS could be useful towards constructing future leakage-resilient protocols.

Distributed Zero-Knowledge (dZK) Proofs. Distributed ZK proofs [BBC+19b] generalize the notion of standard ZK proofs to a setting in which the prover interacts with a set of verifiers, where the input statement to be verified is distributed between the verifiers, and the prover holds the input statement and the corresponding witness (in cases when such a witness exists). throughout the protocol execution, the prover distributes *proof shares* between the verifiers, which the verifiers then use to determine their output (either accept or reject). Our paradigm can be used to protect the proof shares, except when they are directly computed on. We note that in existing dZK proofs (e.g., [BBC+19b]) there are extended time periods during which parties perform computations that are independent of the proof shares (e.g., running a coin-tossing sub-protocol). During such times, our paradigm protects the proof shares from leakage.

More specifically, we define leakage-resilient dZK by comparing a real-world execution, in which the adversary $\mathcal{A}$ interacts with a challenger $\mathcal{C}$, to an ideal execution in which $\mathcal{A}$ interacts with a simulator Sim. The adversary is allowed to *adaptively* corrupt verifiers, as well as adaptively probe proof shares of honest verifiers. In both executions, the challenger and simulator emulate the honest parties, whereas $\mathcal{A}$ assumes full control of the corrupted parties. We note that the standard security notion for dZK proofs is for *static* corruptions. Our constructions can be instantiated with

[11] We note that standard VSS schemes consider only a sharing protocol (which includes the verification protocol) and a reconstruction procedure. We chose to separate the sharing algorithm and the verification protocol, since this enables us to more clearly capture the leakage-resilience guarantees of our VSS protocols.

such statically-secure dZK proofs, in which case LR holds for *static* party corruptions and *adaptive* leakage queries. To obtain fully-adaptive LR-dZK proofs – in which the adversary can also adaptively corrupted verifiers – we extend the ZK definition of dZK to adaptive corruptions, and prove that a protocol of [BBC+19a] is adaptively-secure (See Section 5.2 and Appendix A for further details). This could be of independent interest.

The high-level idea of our dZK proofs is similar to the VSS scheme described above: we employ a standard dZK proof Π as follows. The prover in our LR-dZK proof emulates the prover of Π, and RPE-encodes the proof shares before sending them to the verifiers. The verifiers store these encoded proof shares, except when Π requires that they compute on them. In this case each verifier locally decodes the proof shares needed for the computation, performs the computation, RPE-encodes the proof shares and the outcome (if needed) and erases all intermediate values generated during the computation.

Function Secret Sharing (FSS). Function Secret Sharing (FFS) [BGI16, BCG+21] is a cryptographic building block that enables evaluation of a function f in a distributed manner. An FSS consists of two algorithms Gen and Eval. The former algorithm outputs secret function keys $(k_0, \ldots, k_m)$, one for each party . The latter algorithm is run locally by each party. Given a public input x, and the party's secret key k_i, it computes an output share $f_i(x)$ such that $f(x) = \sum_{i\in[m]} f_i(x)$. The security requirement ensures that the individual keys do not disclose (to a computationally-bounded adversary) any information about the function description.

Current FSS constructions are motivated by applications that involve private and distributive access to a large database, and crucially rely on the fact that FSS has linear reconstruction. (In fact, existing FSS schemes have *additive* reconstruction.) Two notable applications of FSS are Private Information Retrieval (PIR) or keyword search, and statistics collection. (See [BGI16] for further discussion of these applications.) In both cases (a set of) clients generate the keys, and upload them to corresponding servers, where the keys are then used to repeatedly execute the desired operation (e.g., keyword search). Since multiple executions are performed, this also requires aggregating the output shares $f_i(x)$ which strongly relies on the linearity of the FSS reconstruction algorithm.

Defining leakage-resilience for FSS is subtle. This is because on the one hand, our goal is to obtain leakage-resilience against the strongest type of adversarial attacks – namely against computationally unbounded adversaries, while on the other hand, FSS is a computational primitive which only guarantees *computational* indistinguishability in the real/ideal paradigm. Thus, a main challenge which one faces when modeling security of leakage-resilient FSS is to decouple FSS-secrecy from leakage-resilience. We capture this separation by splitting the adversary into two algorithmic entities, one that attacks the scheme via queries to a leakage oracle, and another that receives the state of the former adversarial entity and attacks the secrecy of the underlying FSS. We stress that the former adversarial entity may be computationally unbounded. Another obstacle is that we need to maintain consistent answers (e.g., to repeated leakage queries on output shares generated for the same input x). That is, we need to ensure all responses are consistent with prior responses. This is achieved by maintaining a list which, for every input queried to the function, specifies the randomness used to generate the corresponding output shares in Eval. This randomness is used to answer all leakage queries related to this input. (We note that when an input is queried for the first time, a new entry is added to the list.)

At a high level, our construction modifies the key generation algorithm, where function keys are generated using the FSS generation algorithm, and then RPE-encoded. Similarly, to compute the output shares from the function key shares, each party locally RPE-decodes the key share, generates the output share using the evaluation algorithm Eval of the underlying FSS, and then RPE-encodes it. Since the original FSS has additive reconstruction, and the RPE has linear decoding, the resultant

scheme has linear reconstruction. Thus, clients can remotely operate on the encoded output shares as in the original FSS scheme, and can decode the outcome $f(x)$ *after* it was reconstructed from the output shares. Thus, function and output shares can still be aggregated as required by FSS applications.

Threshold Encryption Schemes (TES) and Threshold Signatures. Lastly, we study the usefulness of our paradigm in the context of threshold cryptography [DH76, LN18, FLOP18, HMR+19] where the secret key underlying some cryptographic task (e.g., encryption or signing), is used in a distributed manner. Namely, a set of parties mutually choose the secret key, where each party picks a random share and neither party (or a strict subset) can reconstruct the secret. Furthermore, each secret key share has a corresponding public key share. The latter shares are combined into the public key of the underlying object. Consequently, each party can encrypt or verify a signature as these operations are public, but decrypting (or, alternatively, signing) a message can be performed in a distributed manner by running a secure computation protocol. A notable example is an extension of the Diffie-Hellman key exchange protocol to the multiparty setting [DH76] that serves as a threshold public key encryption scheme for El Gamal [Gam85]. (We elaborate on this scheme in Section 5.4.)

Threshold encryption is a more involved computational object than FSS since it needs to preserve the secrecy of the secret key as well as the semantic security of the underlying encryption scheme. The scheme consists of a key generation protocol which generates the secret key shares in a distributed manner, an encryption algorithm which is similar to the encryption algorithm in standard encryption schemes, and a decryption protocol in which the parties use their secret key shares (from the key generation phase) to decrypt a ciphertext. Existing security definitions come in different flavours depending of whether security is game-based or simulation-based, and are typically specially-tailored to the specific underlying encryption scheme. We thus first provide a unified security definition that captures the security properties of any threshold encryption scheme. This definition – which might be of independent interest – will later be used as the starting point for our definition of leakage-resilience for threshold encryption. Our security definition for threshold encryption is simulation-based against adaptive corruptions, but can be easily adapt to the static setting. It additionally captures the fact that the adversary can observe the decryption results of the honest parties – which is needed, for instance, to guarantee that security is preserved even when all parties learn the decryption for some plaintext – by giving the adversary access to a reveal oracle that discloses the decryption shares of a specific (valid) cipherext of the adversary's choice.[12]

Next, we extend this definition to obtain leakage-resilience by equipping the adversary with an additional leakage oracle. This oracle takes as input a leakage function, and returns its output on the key and decryption shares of the honest parties. Our leakage model protects the secret key shares of the honest parties (and of corrupted parties prior to their corruption), as well as the plaintext shares of honest parties prior to their decryption. This raises a similar challenge to the one described above in the context of FSS – we wish to protect the shares from leakage against *computationally unbounded adversaries*, while TES is computationally-secure. The assumption of separate leakage on ciphertext and key shares is motivated by the fact that these are physically separated in natural application scenarios. Indeed, different secret key shares are stored on different parties, and in many cases are also physically separated from ciphertexts. For example, a remote server (that *does not* know any secret key share) may generate a ciphertext c (this is possible because TES is a public key object). An adversary that is able to leak on the internal states of the parties will then obtain *separate* leakage on the ciphertext c, as well as on the key shares, and might also obtain several key

[12] We note that the actual formulation is more involved as it needs to eliminate chosen ciphertext attacks, see Section 5.4 for details.

shares in full. In this case, our definition guarantees that the plaintext, as well as the secret key shares that were not fully revealed, remain *information-theoretically* hidden.

To achieve information-theoretic security against leakage queries, we separate the secrecy of TES from the leakage-resilience property by splitting the adversary into three entities. The first entity is computationally-bounded and has access to an encryption oracle. The second entity is computationally-unbounded, and has access to a leakage oracle. We stress that these two entities must be kept separated, since the unbounded adversary can break the computational security of TES. Therefore, these adversarial entities do not share a state. Finally, the third adversarial entity takes the states of the former adversaries as input, and has access to corruption and reveal oracles. (The reveal oracle provides decryption shares of the honest parties on valid ciphertexts.) We note that the need to handle reveal queries makes the definition of leakage-resilient TES even more involved than that of leakage-resilient FSS.

Our definition captures both semi-honest and malicious adversaries, and can be adapted to signature schemes as well, where the property of indistinguishability of ciphertexts is replaced with unforgeability.

At a high-level, our TES construction works as follows. Key shares are generated by first generating TES key shares (by running the key generation protocol of the underlying TES), and then RPE-encoding each key share separately. Encryption is identical to the underlying TES. Finally, the decryption algorithm, given a key share and ciphertext, first RPE-decodes the key, then computes the decryption share using the decryption algorithm of the underlying TES, and then RPE-encodes the decryption share.

1.5 Related Work

There is a vast body of works on leakage-resilient cryptography. In this section we review the works which are most relevant to our model and results, in most cases only citing the first papers introducing the leakage model or LR construction. (See [KR19], and references therein, for a more detailed discussion of these models and various works in the field.) In the following, m denotes the number of parties, and t denotes the reconstruction threshold (in secret sharing schemes).

1.5.1 Leakage-Resilience for Data at Rest

Our leakage model is related to several leakage models considered in the past. In the context of secret sharing, the model most relevant to ours is that of [BDIR18], who also consider a computationally-unbounded adversary that can obtain the shares of an unauthorized set, as well as leakage computed separately on every other share. However, there are several notable differences. Specifically, in their model leakage is *non-adaptive* (whereas we protect against adaptive leakage), but can compute *any* function that is (1) sufficiently shrinking (i.e., the output is sufficiently shorter than the input) and (2) operates on each share separately, whereas our focus is on probing-resilience.

More generally, our model can be seen as a combination of the leakage model of [ISW03] (that restricts the function class from which leakage functions can be chosen), and the "Only Computation Leaks" (OCL) model of [MR04] (which assumes that different memory components leak separately), though, as noted above, our constructions achieve the stronger OCL+ property [BCG+11]. We note that similar to standard OCL, our model is likewise motivated by scenarios in which physical separation of the shares, which are stored in different locations – e.g., on different remote servers – guarantees that leakage will indeed apply to each share separately. We note that the notion of protecting data at rest was studied in the past in the context of the bounded retrieval model [Dzi06, CLW06], bounded memory leakage [AGV09], and several other models (see [KR19] and references

therein). We elaborate more on these works, and their connection to our model, in Section 1.5.3 below.

1.5.2 Leakage-resilient Secret Sharing

Most relevant to our work is the notion of Leakage-Resilient Secret-Sharing Schemes (LR-SSSs), which were first implicitly studied by Dziembowski and Pietrzak [DP07] (under the name "intrusion-resilient secret-sharing"), and explicitly (and independently) defined by [GK18a, BDIR18]. Following [GK18a, BDIR18], there has been a long line of works exploring various aspects of leakage-resilient secret-sharing [DP07, DDV10, GK18a, GK18b, BDIR18, SV19, BS19, KMS19, ADN+19, SV19, CGG+20, LCG+20, MNP+21, MPSW21, CKOS21]. These works consider different leakage models, as we now explain.

Local Leakage Model. The intrusion-resilient secret-sharing of [DP07] designed an n-out-of-n secret sharing scheme that resists adaptive leakage queries applied separately on each share, where each query is defined by an arbitrary polynomial-time computable function, and there are a-priori bounds on the total number of queries and leakage bits. Their scheme required interactive reconstruction. A subsequent work by Davi et al. [DDV10] designed a 2-out-of-2 secret-sharing scheme with *non-interactive* reconstruction. (We note that [Dzi06] allowed leakage of full shares, but [DDV10] do not.)

More recently, the works of [GK18a, BDIR18] independently (formally) introduced the notion of LR-SSSs, and studied leakage-resilience in the presence of non-adaptive local leakage, namely when the adversary can make a *single* leakage query that leaks from each share *independently* of the other shares. These works were motivated by different goals, and focus on different aspects of LR-SSSs. Specifically, Goyal and Kumar [GK18a] used LR-SSSs as a stepping stone toward constructing non-malleable secret sharing, and designed m-party 2-out-of-m LR-SSSs. LR-SSSs for *arbitrary thresholds* were later constructed by Srinivasan and Vasudevan [SV19], who achieve constant rate and leakage rate, through a general compiler that employs an extractor and Shamir's secret sharing scheme as building blocks. In particular, their scheme does not have linear reconstruction. On the other hand, Benhamouda et al. [BDIR18] focused on analyzing leakage resilience of *standard* schemes – such as additive secret sharing, and Shamir secret-sharing over large prime fields. Their motivation was the attacks of Guruswami and Wooters [GW16] on secret sharing over fields of characteristic 2.[13] Specifically, [BDIR18] proved that Shamir's scheme has constant leakage rate for thresholds $t = m - o(m)$. This was recently improved by Maji et al. [MPSW21] who extended the analysis of [BDIR18] to arbitrary thresholds (by analyzing random linear codes), with the caveat that they instantiate Shamir's scheme over *random* (as opposed to fixed) evaluation points.

Limitations of standard secret-sharing schemes. The aforementioned works show some limitations to the leakage-resilience properties of existing schemes. As noted above, Shamir's secret sharing scheme over fields of characteristic 2 does not resist leakage of even *a single bit* from each share [GW16], and similar insecurities hold more generally for fields of small characteristic (not necessarily 2) [BDIR18]. In terms of the reconstruction threshold t in m-party schemes, the analysis of Benhamouda et al. [BDIR18] holds only for certain parameter regimes (either $t \geq 0.85m$ when leaking a constant number of bits from each share, or $t = m - \sqrt[4]{m}$ when leaking a quarter of the bits). There are also known lower bounds on the values of t for which Shamir's scheme resists leakage (even of a single bit) [NS20], which can be circumvented by instantiating Shamir's scheme using random evaluation points [MPSW21].

[13] We note that [BDIR18] extended this result by showing attacks on secret sharing over fields of small characteristic (not necessarily 2).

Joint Leakage Model. Kumar et al. [KMS19] extend the works of [GK18a, GK18b, BDIR18] in a different direction, focusing on a stronger leakage resilience guarantee, namely against *adaptive* leakage queries that depend *jointly* on shares of multiple parties. Specifically, they introduce the Bounded Collusion Protocol (BCP) leakage model, in which the adversary can adaptively leak an a-priori bounded number of bits, where each leakage bit is computed jointly from the shares of a subset of at most p parties, for an a-priori bound p. Roughly, BCP leakage models the leakage as a communication protocol run by the leaking adversary, where in each round the adversary chooses a set $\mathcal{P}$ of at most p parties that compute a "message" based on their shares and the previous protocol messages. The leakage consists of the transcript of this protocol. The schemes of [KMS19] support any threshold, but can only withstand joint leakage of $p = O(\log m)$ shares.[14] This result was further refined recently by Chattopadhyay et al. [CGG+20], who design LR-SSSs withstanding larger collusion bounds of up to $p = t/\log t$ parties, where t is the secrecy threshold of the scheme. They also consider a weaker leakage model in which the (adaptive) leakage queries are restricted to using *disjoint* sets of parties, namely, no share can be accessed by two leakage queries. In this restricted leakage model, they obtain leakage-resilience for any collusion bound $p \leq t$, with $O(1/m)$ rate and leakage rate. This was recently improved by Chandran et al. [CKOS21], who obtain constant rate and leakage rate in this restricted leakage model.

Global Leakage Model. Another line of works [BIVW16, LCG+20] consider *global* leakage which is computed jointly on *all* shares, but restrict the class of permissible leakage functions. Bogdanov et al. [BIVW16] proved that Shamir's scheme over finite fields of characteristic 2 is leakage-resilient against constant-depth polynomial-sized circuits (i.e., AC^0 circuits), and sign polynomials of degree 2,[15] so long as the threshold $t = \omega(\mathsf{polylog}(m))$. The work of [LCG+20] designed LR-SSSs that are secure against affine leakage functions over $\mathbb{F}_2$. These works are somewhat orthogonal to ours since they allow for global leakage (i.e., on all shares) but obtain a very low leakage rate, which, in particular, does not even allow leaking a single bit (on average) from each share.

All the works mentioned above, except for [KMS19, CGG+20, CKOS21, LCG+20], only support non-adaptive queries, and the constructions secure against adaptive leakage queries do not admit efficient equivocation or linear reconstruction. In Table 1 we provide a curated list of works that are closest to our work, and identify the properties satisfied by the leakage-resilient schemes they construct.

1.5.3 Leakage-Resilient Memory and Storage

There is a vast body of works on protecting memory from leakage in different models, such as the bounded-retrieval model [Dzi06, CLW06], bounded memory leakage [AGV09], auxiliary-input memory leakage [DKL09], and continual memory leakage [BKKV10, DHLW10]. These models can be seen as protecting against leakage of data at rest (e.g., protecting the secret key while stored in memory), but differ significantly from our model in the security guarantees (they permit information leakage on the secret state as long as the security of the scheme using the key is not compromised, whereas our goal is to hide the secret), the security quality (usually computational, whereas we protect against computationally-unbounded leaking adversaries), and the permitted leakage functions (usually arbitrary shrinking functions computable in polynomial time, whereas we focus on probing leakage). The model of leakage-resilient storage [DDV10] – which aims to protect (properly encoded) storage – is similarly a model of leakage-resilience for data at rest. Similar to our model, the goal

[14] We remark that the adversary can choose a different subset of $O(\log m)$ shares for each adaptive query, as long as the total number of leaked bits is bounded.

[15] A sign polynomial is a function $f(x) = \mathsf{sgn}(p(x))$ for some polynomial $p(\cdot)$, where sgn is the sign function.

	Scheme	Equiv.	Adaptive Queries	Linear Recon.	Rate	Leakage Rate	Leakage Model
Specialized	[GK18a, GK18b]	No	No	No	$O(\frac{1}{m})$	$O(\frac{1}{m})$	Local
	[BDIR18]	No	No	Yes	$O(1)$	$O(1)$	Local
	[KMS19]	No	Yes	No	$O(\frac{1}{\mathsf{poly}(m)})$	$O(\frac{1}{\mathsf{poly}(m)})$	Joint
	[CKOS21]	No	Yes	No	$O(1)$	$O(1)$	Restricted Joint
General	[SV19]	No	No	No	$O(1)$	$O(1)$	Local
	[CGG+20]	No	Yes	No	$O(\frac{1}{\mathsf{poly}(m)})$	$O(\frac{1}{\mathsf{poly}(m)})$	Joint
	This work (Thm. 4.1/Cor. 4.2)	**Yes**	**Yes**	**Yes**	$O(1)$	$O(1)$	**Probing**

Table 1: Comparison of Existing Leakage-Resilient Secret Sharing Schemes.
Here, "Equiv." denotes whether the scheme is equivocal, "adaptive queries" states whether the scheme is secure against adaptive leakage queries, "linear recon." describes whether the reconstruction procedure of the SSS is linear, m is the number of parties, and the "restricted joint" leakage model refers to the joint leakage model in which each share can be queried by a single leakage query. The first four rows construct specific LR-SS schemes while the last three rows give a general compiler that transforms any SS to a LR one.

is to prevent *any* information leakage on the stored secret. Constructions in this model usually rely on complex primitives such as extractors, which do not result in linear reconstruction procedures.

1.5.4 Leakage-Resilient Distributed Primitives

Boyle et al. [BGK11] introduce and study the notions of leakage-resilient coin-tossing and leakage-resilient VSS. Similar to our work, their constructions are secure against a computationally unbounded, malicious adversary that corrupts t parties, as well as obtains adaptive "local" leakage, namely the leakage function applies to each party separately. However, there are a few notable differences between our leakage model and results, and those of [BGK11], which make them incomparable. First, their corruption model is *non-adaptive*, and this is inherent to their results since they delegate certain computations to committees (a-la [Bra84]), whereas we allow for adaptive corruptions. Unlike our schemes, their constructions rely on (2-source and multi-source) randomness extractors with an additional robustness guarantee,[16] and therefore do not preserve algebraic properties of the original primitives (e.g., linear reconstruction). However, they allow for general leakage of up to ℓ information bits throughout the entire lifetime of the system, where ℓ is a constant fraction of the view size, whereas we focus on probing resilience for data at rest and can handle a constant fraction of leaked bits from each share. There are also works obtaining zero-knowledge protocols [GJS11, BCH12] as well as MACs, commitment schemes, and OT [BCH12] with a weaker leakage-resilience guarantee known as *leakage-tolerance*, in which the ideal-world adversary obtains leakage on the witness/secret. Thus, these works guarantee graceful degradation of security – instead of full security – against leakage.

[16] We note that they do have a VSS protocol that does not use extractors, but it only achieves a weaker LR guarantee in which the secret retains some min-entropy given the leakage.

1.5.5 Leakage-Resilient MPC and General Computations

Following the works of [ISW03, MR04], there has been a long line of works on protecting general computations against leakage (see [KR19] and references therein). These constructions usually consider a continuous leakage model in which the adversary observes repeated executions of the computation on inputs of its choice. Due to their generality, these constructions are less efficient (incurring polynomial blowups), complex, and in particular do not preserve the structure of the original computation (e.g., linear reconstruction). There are also works on protecting MPC protocols from leakage, either with full security against leakage [BGJK12, BDIR18], or allowing some leakage in the ideal world [BCH12, BGJ+13, BDL14]. The latter protocols achieve only a weaker leakage-resilience guarantee (since the ideal-world simulator also obtains leakage), whereas the former either achieve only computational security [BGJK12], or a poor leakage rate [BDIR18].

1.5.6 Tampering-Resilience

The notion of tampering-resilience for secret-sharing, also known as *non-malleable* secret-sharing [GK18a], has received a lot of attention lately [GK18a, GK18b, SV19, BS19, FV19, ADN+19, BFV19, KMS19, LCG+19, CGG+20, CKOS21, KOST21]. Roughly speaking, non-malleable secret-sharing guarantees that even if an adversary tampers with all the shares, reconstruction either outputs the original secret, or a *completely independent secret* which is unrelated to the original secret. This should be contrasted with our notion of tampering-resilience, which guarantees that *the original secret will be reconstructed*. On the other hand, non-malleable secret-sharing can withstand higher tampering rates, in particular ones exceeding the error-correction bound for which unique recovery of the underlying secret is possible.

We note that another – very different – notion of tampering-resilience appeared in the literature in the context of protecting *secrecy* (e.g., of the input or internal state of a cryptographic scheme) in the presence of a tampering adversary. These works, originating from [IPSW06], are unrelated to our notion of tampering since they aim at protecting *secrecy*, whereas our goal is to maintain *correctness*.

2 Preliminaries

Basic notations. We denote the security parameter by κ. We say that a function $\mu : \mathbb{N} \to \mathbb{N}$ is *negligible* if for every positive polynomial $p(\cdot)$ and all sufficiently large κ's it holds that $\mu(\kappa) < \frac{1}{p(\kappa)}$. We denote the set of all negligible functions by $\mathsf{negl}(\kappa)$. We use the abbreviation PPT to denote probabilistic polynomial-time, and denote by $[n]$ the set of elements $\{1, \ldots, n\}$ for some $n \in \mathbb{N}$. For a string s of length n, and a subset $I \subseteq [n]$, we denote by $s|_I$ the restriction of s to the coordinates in I. For an NP relation $\mathcal{R}$, we denote by $\mathcal{R}_x$ the set of witnesses of x, and by $\mathcal{L}_\mathcal{R}$ its associated language. That is, $\mathcal{R}_x = \{w \mid (x, w) \in \mathcal{R}\}$ and $\mathcal{L}_\mathcal{R} = \{x \mid \exists\ w\ s.t.\ (x, w) \in \mathcal{R}\}$.

Let Σ be an alphabet, and let x, y be strings over Σ^n. We denote by $\mathrm{Ham}(x, y) = |\{i\ :\ x_i \neq y_i\}|$ the Hamming distance of x, y.

Definition 2.1. *Let X_κ and Y_κ be random variables accepting values taken from a finite domain Ω. The* statistical distance *between X_κ and Y_κ is*

$$\mathsf{SD}(X_\kappa, Y_\kappa) = \frac{1}{2} \sum_{w \in \Omega} \left| \Pr[X_\kappa = w] - \Pr[Y_\kappa = w] \right|.$$

*We say that X_κ and Y_κ are ε-*close *if their statistical distance is at most $\varepsilon(\kappa)$. We say that X_κ and Y_κ are* statistically close*, denoted $X_\kappa \approx_s Y_\kappa$, if $\varepsilon(\kappa)$ is negligible in κ.*

We use the asymptotic notation $O(\cdot)$ and $\Omega(\cdot)$. We will sometimes disregard polylogarithmic factors, using $\widetilde{O}(n)$ and $\widetilde{\Omega}(n)$ to denote $n \cdot \mathsf{poly}\log n$ and $n/\mathsf{poly}\log n$, respectively.

2.1 Leakage Classes

In the following sections we will discuss leakage-resilience of different cryptographic primitives. Leakage-resilience is defined with relation to a class of leakage functions. We now define two such classes, which we will focus on. Both classes consider *local probing* leakage with relation to a distributed primitive which involves m parties, where each party holds a secret share. The primitive is associated with an access structure defining the authorized sets (which are the sets of parties that can learn the underlying secret by pooling together their shares) and unauthorized sets (which learn nothing about the secret even given all their shares). Roughly, the local probing leakage classes consist of all functions that, given the m shares, output the shares of an unauthorized set in their entirety, as well as ℓ bits from each of the other shares. The two classes differ only in the access structure they consider.

The first class of leakage classes considers the $t+1$-threshold access structure (defined more formally in Section 2.2 below), in which all (and only) subsets of size $\geq t+1$ are authorized. Formally:

Definition 2.2 ((t,ℓ)-local probing leakage)**.** *Let $\mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_m$ be the domain of shares for some secret sharing scheme. For a subset $\mathcal{G} \subseteq [m]$ and a sequence $(\mathcal{I}_1, \ldots, \mathcal{I}_m)$ of subsets, the function $f_{\mathcal{G},\mathcal{I}_1,\ldots,\mathcal{I}_m}$ on input $(s_1, \ldots, s_m)$ outputs s_i for every $i \in \mathcal{G}$, and outputs $s_i|_{\mathcal{I}_i}$ for every $i \notin \mathcal{G}$. The* (t,ℓ)-local probing function family *corresponding to $\mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_m$ is defined as follows:*

$$\mathcal{L}_{t,\ell} = \{f_{\mathcal{G},\mathcal{I}_1,\ldots,\mathcal{I}_m} \ : \ \mathcal{G} \subseteq [m], |\mathcal{G}| \leq t, \forall i \notin \mathcal{G}, |\mathcal{I}_i| \leq \ell\}\,.$$

The second class of leakage functions generalizes the previous definition to general access structures.

Remark 2.1 (Local probing leakage: general access structures)**.** *We will sometimes consider a generalization of Definition 2.2 to arbitrary access structures. Specifically, for an access structure Acc, and a domain $\mathcal{S}_1 \times \mathcal{S}_2 \times \ldots \times \mathcal{S}_m$ of secrets, the* (Acc,ℓ)-local probing function family *corresponding to $\mathcal{S}_1 \times \mathcal{S}_2 \times \ldots \times \mathcal{S}_m$ is defined as follows:*

$$\mathcal{L}_{\mathsf{Acc},\ell} = \{f_{\mathcal{G},\mathcal{I}_1,\ldots,\mathcal{I}_m} \ : \ \mathcal{G} \subseteq [m], |\mathcal{G}| \notin \mathsf{Acc}, \forall i \notin \mathcal{G}, |\mathcal{I}_i| \leq \ell\}\,.$$

2.2 Secret Sharing Schemes (SSSs)

A Secret-Sharing Scheme (SSS) allows a dealer to distribute a secret among m parties. Specifically, during a *sharing* phase each party receives a share from the dealer, and the secret can then be recovered from the shares during a *reconstruction* phase. The scheme is associated with an *access structure* which defines subsets of authorized and unauthorized parties, where every authorized set can recover the secret from its shares, whereas unauthorized sets learn nothing about the secret even given all their shares. A *Leakage-Resilient* SSS (LR-SSS) guarantees this latter property holds even if the unauthorized set obtains some leakage on the other shares.

We will mainly be interested in *t-threshold* secret sharing schemes, in which all (and only) subsets of size at least $t+1$ are authorized to reconstruct the secret. We first define secret sharing schemes.

Definition 2.3 (Secret Sharing Scheme)**.** *An m-party Secret Sharing Scheme (SSS) for secrets in $\mathcal{S}$ consists of the following pair of algorithms.*

Sharing algorithm Share: *Takes as input a secret $s \in \mathcal{S}$ and outputs shares $(s_1, \cdots, s_m) \in \mathcal{S}_1 \times \cdots \times \mathcal{S}_m$, where s_i is called the* share *of party i, and $\mathcal{S}_i$ is the* domain of shares *of party i.*

Reconstruction algorithm Reconst: *Takes as input a description $\mathcal{G}$ of an authorized set, and shares $\{s_i \ : \ i \in \mathcal{G}\}$ and outputs $s' \in \mathcal{S}$.*

The scheme is required to satisfy the following properties:

Correctness: *For every $s \in \mathcal{S}$, and every authorized set $\mathcal{G}$,*

$$\Pr\left[\mathsf{Reconst}\left(\mathcal{G}, \mathsf{Share}\left(s\right)|_{\mathcal{G}}\right) = s\right] = 1$$

where $\mathsf{Share}\left(s\right)|_{\mathcal{G}}$ denotes the shares of the parties in the authorized set $\mathcal{G}$.

ε-Secrecy: *For any pair of secrets $s, s' \in \mathcal{S}$, and any unauthorized set $\mathcal{G}$,*

$$SD\left(\mathsf{Share}\left(s\right)|_{\mathcal{G}}, \mathsf{Share}\left(s'\right)|_{\mathcal{G}}\right) \leq \varepsilon.$$

In our constructions, we will use Shamir's secret sharing scheme [Sha79], which we review next.

Definition 2.4 (Shamir's SSS)**.** *Let $\mathbb{F}$ be a field.*

Sharing algorithm: *For any input $s \in \mathbb{F}$, pick a random polynomial $p(\cdot)$ of degree t in the polynomial-field $\mathbb{F}[x]$ with the condition that $p(0) = s$, and output $p(1), \ldots, p(m)$.*

Reconstruction algorithm: *For any input $(s'_i)_{i \in S}$ where none of the s'_i are $\perp$ and $|S| > t$, compute a polynomial $g(x)$ such that $g(i) = s'_i$ for every $i \in S$. This is possible using Lagrange interpolation where g is given by*

$$g(x) = \sum_{i \in S} s'_i \prod_{j \in S/\{i\}} \frac{x - j}{i - j} \ .$$

Finally the reconstruction algorithm outputs $g(0)$.

2.2.1 Resampleable Secret Sharing Schemes

Our equivocal SSS (described in the next section) will use the underlying SSS as a black box, but will require that it posses the additional property of being *resampleable*. Roughly, a SSS is resampleable if given the shares of an unauthorized set T, and the secret x, one can *efficiently* sample shares for the other parties which, together with the shares of the parties in T, are distributed statistically close to a (random) sharing of x. We note that if we allow for *inefficient* resampling then any SSS is resampleable because the resampler can simply recover the original sharing of x. Furthermore, Shamir's SSS (Definition 2.4) is perfectly resampleable.

Definition 2.5 (Resampleable SSS)**.** *We say that a secret sharing scheme* (Share, Reconst) *is ε-resampleable, if it is associated with an additional PPT algorithm* Resample *such that the following holds.*

Syntax. Resample *takes as input a secret x', a subset of parties $T \subseteq [m]$, and shares $(\mathsf{Sh}_i)_{i \in T}$ for the parties in T, and outputs shares $\left(\mathsf{Sh}'_1, \ldots, \mathsf{Sh}'_m\right)$.*

Semantics. *For every secret x, any adversary $\mathcal{A}$ obtains at most an ε distinguishing advantage in the following game.*

- *The challenger $\mathcal{C}$ samples $b \leftarrow \{0,1\}$ and $(\mathsf{Sh}_1, \ldots, \mathsf{Sh}_m) \leftarrow \mathsf{Share}(x)$.*
- *Repeat: $\mathcal{A}$ adaptively picks $i \in [m]$, sends i to $\mathcal{C}$, and obtains Sh_i from $\mathcal{C}$.*
- *Let $T \subseteq [m]$ denote the set of share indices which $\mathcal{A}$ obtained in the previous step of the game. If T is an authorized set then the game aborts, and $\mathcal{A}$'s advantage is 0.*
- *If $b = 0$, then $\mathcal{C}$ sends $(\mathsf{Sh}_1, \ldots, \mathsf{Sh}_m)$ to $\mathcal{A}$. Otherwise, $\mathcal{C}$ compute $\left(\mathsf{Sh}'_1 \ldots, \mathsf{Sh}'_m\right) \leftarrow \mathsf{Resample}\left(x, T, (\mathsf{Sh}_i)_{i \in T}\right)$, and sends $\left(\mathsf{Sh}'_1, \ldots, \mathsf{Sh}'_m\right)$ to $\mathcal{A}$.*
- *$\mathcal{A}$ outputs a bit b'.*
- *The* advantage *of $\mathcal{A}$ in the game is $|\Pr[b' = 1 | b = 1] - \Pr[b' = 1 | b = 0]|$.*

If a secret sharing scheme is 0-resampleable, then we simply say it is resampleable.

Remark 2.2 (Shamir's scheme is resampleable)**.** *For any $t \in \mathbb{N}$, Shamir's secret sharing scheme with threshold t is resampleable, with the following* Resample *algorithm: given an unauthorized set T of size $|T| \leq t$, the shares $(\mathsf{Sh}_i)_{i \notin T}$ of parties in T, and a secret x, the* Resample *algorithm operates as follows. Let $|T| = t'$, and let $i_1, \ldots, i_{t-t'}$ denote the $t - t'$ smallest indices which are not in T. Then* Resample *picks $\mathsf{Sh}_{i_1}, \ldots, \mathsf{Sh}_{i_{t-t'}}$ uniformly at random, and uses these, together with $(\mathsf{Sh}_i)_{i \in T}$ and x to determine the unique polynomial $p(\cdot)$ of degree $\leq t + 1$ which is consistent with all these shares. Then, it uses $p(\cdot)$ to determine the remaining shares.*

We will also need an adaptive version of secrecy of secret sharing schemes, which allows the adversary to adaptively choose the participants of the unauthorized set of parties whose shares it will obtain. We note that if the secret sharing scheme has perfect secrecy, then our adaptive notion of secrecy is equivalent to the standard notion of secrecy (Definition 2.3). If the secret sharing scheme has statistical secrecy, then it implies adaptive secrecy, but with an exponential loss in the statistical distance. This is formalized in Lemma 2.3 below, and follows using a standard adaptive-to-non-adaptive reduction, in which the non-adaptive adversary attempts to guess the set chosen by the adaptive adversary ahead of time.

Definition 2.6 (Adaptive secrecy of SSS)**.** *We say that a SSS* $(\mathsf{Share}, \mathsf{Reconst})$ *has ε-adaptive secrecy if every adversary $\mathcal{A}$ wins the following game with probability at most ε.*

- *$\mathcal{A}$ picks a pair of secrets x^0, x^1 and sends them to $\mathcal{C}$.*
- *The challenger $\mathcal{C}$ samples $b \leftarrow \{0,1\}$ and $(\mathsf{Sh}_1, \ldots, \mathsf{Sh}_m) \leftarrow \mathsf{Share}\left(x^b\right)$.*
- *Repeat: $\mathcal{A}$ adaptively picks $i \in [m]$, sends i to $\mathcal{C}$, and obtains Sh_i from $\mathcal{C}$.*
- *Let $T \subseteq [m]$ denote the set of share indices which $\mathcal{A}$ obtained in the previous step of the game.*
- *$\mathcal{A}$ outputs a bit b'.*
- *$\mathcal{A}$* wins *the game if T is unauthorized, and $b = b'$.*

Lemma 2.3 (Standard secrecy implies adaptive secrecy)**.** *Let* $(\mathsf{Share}, \mathsf{Reconst})$ *be an m-party ϵ-secret SSS. Then* $(\mathsf{Share}, \mathsf{Reconst})$ *has $2^m \cdot \varepsilon$-adaptive secrecy.*

In particular, if $(\mathsf{Share}, \mathsf{Reconst})$ *has* perfect *secrecy, then it has perfect adaptive secrecy.*

2.2.2 Probing-Resilient SSSs

In this section we describe a notion of probing-resilience for SSSs, and more generally for distributed primitives and protocols in which some secret is shared among a set of parties. This is formalized in the following definition:

Definition 2.7 (Tampering-Resilience (TR))**.** *Let $\mathcal{P}$ denote an n-party primitive that includes a sharing phase that distributes a secret among the parties (the sharing phase could potentially be executed several times throughout the execution of $\mathcal{P}$) and a reconstruction procedure* $\mathsf{Reconst}$ *which recovers a secret from the shares. Assume that $\mathcal{P}$ is secure against a set Acc of adversaries.*[17] *Let $\mathsf{Sh}_1, \ldots, \mathsf{Sh}_n$ denote the outcome of the sharing phase. We say that $\mathcal{P}$ is* τ-tampering-resilient (TR) *if for every adversary $\mathcal{A} \in \mathsf{Acc}$ which corrupts a subset T of parties, every set $\left\{\mathsf{Sh}'_i \ : \ i \in T\right\}$ of shares which $\mathcal{A}$ chooses for the corrupted parties, and any $\left\{\mathsf{Sh}'_i \ : \ i \notin T\right\}$ such that $Ham\left(\mathsf{Sh}_i, \mathsf{Sh}'_i\right) \leq \tau$ for every $i \notin T$, $\mathsf{Reconst}\left(\mathsf{Sh}'_1, \ldots, \mathsf{Sh}'_n\right) = \mathsf{Reconst}\left(\mathsf{Sh}_1, \ldots, \mathsf{Sh}_n\right)$.*

Remark 2.4 (Tamper-resilience for SSSs.)**.** *Definition 2.7 discusses a general distributed primitive $\mathcal{P}$. We note that when $\mathcal{P}$ is a secret-sharing scheme, then by default we assume the adversary is* semi-honest *(though we also discuss the more general case of a malicious adversary in Remark 4.2).*

2.2.3 Equivocal Secret Sharing

In this section, we define the notion of equivocal SSSs, which was introduced in [HVW21]. We provide a short high-level description of the definition, and refer the interested reader to [HVW21] for further details.

Intuitively, equivocation enhances the leakage-resilience guarantee such that even after some bits were leaked, one can still "explain" the sharing (by providing the entire secret sharing) as the sharing of *any* secret, consistently with the leakage. This is formalized by comparing the real world experiment with an ideal experiment. The real and ideal experiments have two phases: a leakage phase and a guessing phase. This is captures by having the adversary and simulator consist of two separate algorithms $(\mathcal{A}_1, \mathcal{A}_2)$ and $(\mathsf{Sim}_1, \mathsf{Sim}_2)$, respectively. Equivocation is against a family F of leakage functions and a leakage bound ℓ.

In the real world, the secret s is secret shared into n shares $\mathsf{Sh}_1, \ldots, \mathsf{Sh}_n$. The adversary $\mathcal{A}_1$ is then given oracle access to a $\mathcal{SHARE}$ oracle – which returns full shares, and updates the set T of "corrupted" parties whose shares the adversary has obtained — and a $\mathcal{LEAK}$ oracle which applies a leakage function $g \in F$ (specifying, for each honest share Sh_i, a leakage function g_i) to the shares of the honest parties. $\mathcal{LEAK}$ also maintains leakage counters ℓ_i of the number of leakage bits obtained on Sh_i. T and $\ell_1, \ldots, \ell_n$ are treated as global variables that can be accessed and updated by all oracles. At the end of the first phase of the experiment, $\mathcal{A}_1$ outputs a bit b_R specifying whether it wishes to learn the entire secret sharing of s (by calling the $\mathcal{REV}$ oracle). Otherwise, the adversary obtains no further information beyond what it obtained during the leakage phase. Then, in the second phase of the game, the adversary $\mathcal{A}_2$ outputs a guess b'_R as to whether it is in the real or ideal experiments. This adversarial guess is only taken into account if the adversary did not violate the leakage restrictions, i.e., T is unauthorized, and $\ell_i \leq \ell$ for every honest i. These checks are performed by calling the $\mathcal{VALID}$ oracle, where if the tests fail then the adversary automatically looses the game (by setting its "guess" to 0). The ideal experiment is similar to the real experiment, except that the $\mathcal{SHARE}, \mathcal{LEAK}$, and $\mathcal{REV}$ oracles are emulated by the simulator. Any equivocal SSS is also a LR-SSS as the adversary in the equivocal SSS game can choose *not* to receive the entire secret sharing of s at the end of the first phase (see Remark 2.5 below).

[17] The exact meaning of "security", as well as the set of permissible adversaries, depend on the primitive.

Definition 2.8 (Equivocal SSS). *Let $\varepsilon \in [0,1)$. We say that an m-party secret sharing scheme $(\mathsf{Share}, \mathsf{Reconst})$ for secrets in $\mathcal{S}$ is* ε-equivocal *for leakage class F, leakage bound ℓ and access structure Acc if for every adversary $(\mathcal{A}_1, \mathcal{A}_2)$ there exists an efficient simulator $(\mathsf{Sim}_1, \mathsf{Sim}_2)$ such that for every $s \in \mathcal{S}$,*

$$\left|\Pr\left[\mathcal{REAL}_{F,\ell,\mathsf{Acc}}(s) = 1\right] - \Pr\left[\mathcal{IDEAL}_{F,\ell,\mathsf{Acc}}(s) = 1\right]\right| \leq \varepsilon$$

where $\mathcal{REAL}_{F,\ell,\mathsf{Acc}}(s), \mathcal{IDEAL}_{F,\ell,\mathsf{Acc}}(s)$ are defined in Figure 1, and the probability is over the random coin tosses of $\mathcal{SETUP}^{\mathcal{R}}$, $(\mathcal{A}_1, \mathcal{A}_2)$ and $(\mathsf{Sim}_1, \mathsf{Sim}_2)$.

We note that Definition 2.8 naturally generalizes to the asymptotic setting, where ε is a function of, e.g., the length of secrets in $\mathcal{S}$.

Remark 2.5 (On the connection to adaptive LR secret sharing). *We note that equivocal secret sharing generalizes the notion of an adaptive LR Secret Sharing Scheme (LR-SSS) as follows. The definition of an adaptive LR-SSS can be obtained by overwriting the bits b_R, b_I in Figure 1 with 0. In this case, the $\mathcal{REV}$ oracle is not called in Figure 1, in which case the adversary does not receive the full secret shares, and the simulator does not receive the secret s. Consequently, statistical closeness of the real and ideal experiments implies that adaptive leakage queries of the adversary can be simulated without knowing the secret, as long as these queries lie in the supported class of leakage functions. This is exactly the definition of an adaptive LR-SSS.*

3 Reconstructible Probabilistic Encodings (RPEs)

The main building block underlying our constructions are Reconstructible Probabilistic Encodings (RPEs). Roughly, these are a generalized notion of error-correcting codes that have a *randomized* encoding procedure, with the following useful properties. First, a small subset of symbols from a random encoding of a secret x reveals no information on s. Second, this subset of symbols can be "explained" as the encoding of any *arbitrary* secret x', in the sense that there exists an efficient resampler algorithm Recon that, given the symbols and x', samples an encoding of s' that is random subject to being consistent with these symbols. This is foramlized in the following definition.

Definition 3.1 (Reconstructible Probabilistic Encoding (RPE)). *Let $k, n, \ell, \tau \in \mathbb{N}$ and $\varepsilon \in (0,1)$. An $(\varepsilon, \ell, \tau)$-*Reconstructible Probabilistic Encoding (RPE) *is a triple $(\mathsf{Encode}, \mathsf{Decode}, \mathsf{Rec})$ where $\mathsf{Encode}, \mathsf{Rec}$ are PPT algorithms, and Decode is a polynomial-time algorithm, that satisfy the following.*

- **Syntax.** Encode *on input a secret $x \in \{0,1\}^k$ outputs a codeword $c \in \{0,1\}^n$.* Decode *on input $c \in \{0,1\}^n$ outputs $x \in \{0,1\}^k$ or a special error symbol $\perp$.* Rec *on input a secret x, a set $\mathcal{I} \subset [n]$ of size $|\mathcal{I}| \leq \ell$, and ℓ bits $(c_i)_{i \in \mathcal{I}}$, outputs $c' \in \{0,1\}^n$.*
- **τ-Error correction.** *For every $x \in \{0,1\}^k$, and every $c \in \{0,1\}^n$, if there exists $c_x \in \mathsf{Supp}(\mathsf{Encode}(x))$ such that $Ham(c, c_x) \leq \tau$ then $\mathsf{Decode}(c) = x$, otherwise Decode outputs $\perp$.*
- **(ε, ℓ)-Secrecy of partial views.** *For every pair of secrets x, x', and any subset $\mathcal{I} \subseteq [n]$ such that $|\mathcal{I}| \leq \ell$, $SD(\mathsf{Encode}(x)|_{\mathcal{I}}, \mathsf{Encode}(x')|_{\mathcal{I}}) \leq \varepsilon$.*
- **(ε, ℓ)-Reconstruction from partial views.** *For any pair of secrets x, x', any subset $\mathcal{I} \subseteq [n]$ of size $|\mathcal{I}| \leq \ell$, and any set $(c'_i)_{i \in \mathcal{I}} \in \mathsf{Supp}(\mathsf{Encode}(x'))$ of bits, $\mathsf{Rec}\left(x, \mathcal{I}, (c'_i)_{i \in \mathcal{I}}\right)$ is ε-statistically close to an encoding $c \in \mathsf{Supp}(\mathsf{Encode}(x))$ that is random subject to being consistent with $(c'_i)_{i \in \mathcal{I}}$.*

Equivocal SSS Security

$\underline{\mathcal{SETUP}^{\mathcal{R}}(s)}$:
pick a random string r for Share [a]
$(\mathsf{Sh}_1, \ldots, \mathsf{Sh}_m) \leftarrow \mathsf{Share}(s; r)$
output $(\mathsf{Sh}_1, \ldots, \mathsf{Sh}_m)$

$\underline{\mathcal{SETUP}^{\mathcal{I}}()}$:
initialize St to the empty string
$\mathsf{St} \leftarrow \mathsf{Sim}_1(\mathsf{St})$
output St

$\underline{\mathcal{SHARE}^{\mathcal{R}}(s, r, i)}$:
$T_1 \leftarrow T_1 \cup \{i\}$
output Sh_i

$\underline{\mathcal{SHARE}^{\mathcal{I}}(i)}$:
$\mathsf{Sh}_i \leftarrow \mathsf{Sim}_1(\mathsf{St}, i)$
$T_1 \leftarrow T_1 \cup \{i\}$
output Sh_i

$\underline{\mathcal{LEAK}^{\mathcal{R}}(s, r, g, T)}$:
if $g \notin F$ then return
$T_1 \leftarrow T_1 \cup T$
$(\mathsf{output}_i)_{i \notin T_1} \leftarrow g\left((\mathsf{Sh}_i)_{i \notin T_1}\right)$
for every $i \notin T_1$
  $\ell_i \leftarrow \ell_i + |\mathsf{output}_i|$
output $\left((\mathsf{output}_i)_{i \notin T_1}, (\mathsf{Sh}_i)_{i \in T}\right)$

$\underline{\mathcal{LEAK}^{\mathcal{I}}(g, T)}$:
if $g \notin F$ then return
$T_1 \leftarrow T_1 \cup T$
$\left((\mathsf{output}_i)_{i \notin T_1}, (\mathsf{Sh}_i)_{i \in T}, \mathsf{St}\right) \leftarrow \mathsf{Sim}(\mathsf{St}, g, T)$
for every $i \notin T_1$
  $\ell_i \leftarrow \ell_i + |\mathsf{output}_i|$
output $\left((\mathsf{output}_i)_{i \notin T_1}, (\mathsf{Sh}_i)_{i \in T}\right)$

$\underline{\mathcal{REV}^{\mathcal{R}}(s, r)}$:
output $(\mathsf{Sh}_1, \ldots, \mathsf{Sh}_m)$

$\underline{\mathcal{REV}^{\mathcal{I}}(s)}$:
$\mathsf{rev} \leftarrow \mathsf{Sim}_2(\mathsf{St}, s)$
output rev

$\underline{\mathcal{VALID}(\ell, \mathsf{Acc})}$:
if $T_1 \notin \mathsf{Acc}$ and $\ell_i \leq \ell$ for every $i \in [m]$
  then output true
else
  output false

$\underline{\mathcal{REAL}_{F,\ell,\mathsf{Acc}}(s)}$:
$\ell_1, \ldots, \ell_m \leftarrow 0$
$T_1 \leftarrow \emptyset$
$(\mathsf{Sh}_1, \ldots, \mathsf{Sh}_m) \leftarrow \mathcal{SETUP}^{\mathcal{R}}(s)$
$(\mathsf{St}_{\mathcal{A}}, b_R) \leftarrow \mathcal{A}_1^{\mathcal{SHARE}^{\mathcal{R}}(s,r,\cdot), \mathcal{LEAK}^{\mathcal{R}}(s,r,\cdot,\cdot)}$
if $b_R = 1$ then
  $\left(\mathsf{Sh}'_1, \ldots, \mathsf{Sh}'_m\right) \leftarrow \mathcal{REV}^{\mathcal{R}}(s, r)$
  $\mathsf{St}_{\mathcal{A}} \leftarrow \mathsf{St}_{\mathcal{A}} \circ \left(\mathsf{Sh}'_1, \ldots, \mathsf{Sh}'_m\right)$
$b'_R \leftarrow \mathcal{A}_2(\mathsf{St}_{\mathcal{A}})$
if $\mathcal{VALID}(\ell, \mathsf{Acc})$ then output b'_R
else output 0

$\underline{\mathcal{IDEAL}_{F,\ell,\mathsf{Acc}}(s)}$:
$\ell_1, \ldots, \ell_m \leftarrow 0$
$T_1 \leftarrow \emptyset$
$\mathsf{St} \leftarrow \mathcal{SETUP}^{\mathcal{I}}()$
$(\mathsf{St}_{\mathcal{A}}, b_I) \leftarrow \mathcal{A}_1^{\mathcal{SHARE}^{\mathcal{I}}(\cdot), \mathcal{LEAK}^{\mathcal{I}}(\cdot,\cdot)}$
if $b_I = 1$ then
  $\mathsf{output} \leftarrow \mathcal{REV}^{\mathcal{I}}(s)$
  $\mathsf{St}_{\mathcal{A}} \leftarrow \mathsf{St}_{\mathcal{A}} \circ \mathsf{output}$
$b'_I \leftarrow \mathcal{A}_2(\mathsf{St}_{\mathcal{A}})$
if $\mathcal{VALID}(\ell, \mathsf{Acc})$ then output b'_I
else output 0

[a] We think of the randomness r for Share, the secret s, and the simulator state St as global variables, which certain oracles can access freely (e.g., all the ideal-world oracles can access St, and $\mathcal{REV}^{\mathcal{I}}$ can also access s).

Figure 1: The Security Experiments of Equivocal SSS

Remark 3.1 (Perfect RPEs.). *We note that standard RPEs [CDMW08] have perfect secrecy of partial views and reconstruction from partial views (i.e., $\varepsilon = 0$). We chose to present a relaxed notion in Definition 3.1 because it suffices for our constructions (the fact that our constructions can use non-perfect RPEs makes them stronger), and allows us to explicitly quantify how privacy/reconstruction error in the RPE propagates to simulation error in the resultant primitives.*

Moreover, in standard RPEs the reconstruction from partial views property holds for any *set $(c'_i)_{i\in\mathcal{I}}$ of bits (not necessarily ones corresponding to an encoding of some secret x'). The weaker property defined in Definition 3.1 suffices for our constructions, and will allow us to easily obtain RPEs from random linear codes (see Lemma 3.9).*

Remark 3.2 (On the role of error-correction.). *The error-correction property of the RPE is used in our constructions only for tampering-resilience. In particular, instantiating our constructions with an RPE that* does not *have error-correction would give a leakage-resilient primitive without tampering-resilience.*

Remark 3.3 (Families of codes). *Definition 3.1 discusses a single message length k, but it generalizes naturally to* families *of codes, where for every $k \in \mathbb{N}$ there exists a code with block length $n = n(k)$, and there exists a uniform algorithm that generates the encoding procedure for message length k. (For example, for linear codes – in which encoding simply multiplies by a public generator matrix – this algorithm given input 1^k outputs the generator matrix for message length k.) We will only consider (efficiently encodable and decodable)* families *of codes in this work. For simplicity and clarity of the definitions, we do not explicitly refer to a family of codes, but k should be understood as a general input length parameter. (This is standard in the literature.)*

Remark 3.4 (RPEs over general fields and the connection to linear codes). *Recall that we defined RPEs as binary codes. This was done to simplify the presentation (and since such codes suffice for our main application of equivocal SSSs), but the notion naturally extends to general fields. Moreover, RPEs are closely related to linear codes. Indeed, any linear code over a field $\mathbb{F}$ with dual distance ℓ has $(0,\ell)$-secrecy of partial views (when this notion is generalized in the natural way to hold for $\mathbb{F}$).*[18] *Consequently, the code has $(0,\ell)$-reconstruction from partial views, since the reconstructor can efficiently solve the linear set system (because this set system necessarily has a solution).*

3.1 Adaptive Take on RPEs

In this section we present the alternative definitions of RPE properties which we will use. These definitions phrase RPE properties in terms of adaptive adversaries, describing security games between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. As we show below, these "adaptive" definitions are satisfied by any RPE.

Definition 3.2 (RPE - adaptive secrecy of partial views). *Let* $\mathsf{RPE} = (\mathsf{Encode}, \mathsf{Decode}, \mathsf{Rec})$ *be an RPE scheme, with* $\mathsf{Encode} : \{0,1\}^k \to \{0,1\}^n$. *We say that* RPE *has (ε,ℓ)-adaptive secrecy of partial views if for any adversary $\mathcal{A}$, the distinguishing advantage of $\mathcal{A}$ in the following game with a challenger $\mathcal{C}$ is at most ε.*

- *$\mathcal{A}$ picks a pair of secrets x^0, x^1 and sends them to $\mathcal{C}$.*
- *$\mathcal{C}$ draws $b \leftarrow \{0,1\}$, and encodes $c \leftarrow \mathsf{Encode}\left(x^b\right)$.*

[18] Indeed, we can set the encoding procedure, on input x, to apply the generator matrix to (x,r) for $r \leftarrow \mathbb{F}^\ell$, yielding a randomized encoding procedure. Then for any x, x', any ℓ codeword symbols in a random encoding of x are identically distributed to the corresponding ℓ codeword symbols in a random encoding of x'.

- *For* $i = 1, \ldots, \ell$*:*
  - $\mathcal{A}$ *picks* $i \in [n]$ *and sends* i *to* $\mathcal{C}$*.*
  - $\mathcal{C}$ *sends* c_i *to* $\mathcal{A}$*.*
- $\mathcal{A}$ *outputs a bit* b'*.*
- *The* advantage *of* $\mathcal{A}$ *in the game is* $|\Pr[b' = 1 | b = 1] - \Pr[b' = 1 | b = 0]|$*.*

We say that an RPE has ℓ*-adaptive secrecy of partial views if it has* $(0, \ell)$*-adaptive secrecy of partial views.*

We will sometimes omit the parameters ℓ, ε, simply saying "an RPE scheme with adaptive secrecy of partial views".

The following lemma follows by a standard adaptive-to-non-adaptive reduction, in which the reduction guesses the queries of the adaptive adversary ahead of time, and asks its (static) challenger for the answers, where if the guess is correct then the reduction can successfully emulate the adaptive secrecy game for the adaptive adversary.

Lemma 3.5 (RPEs have adaptive secrecy of partial views)**.** *Let* $\mathsf{RPE} = (\mathsf{Encode}, \mathsf{Decode}, \mathsf{Rec})$ *be an* (ε, ℓ)*-RPE scheme, with* $\mathsf{Encode} : \{0,1\}^k \to \{0,1\}^n$*. Then* RPE *has* $\left(2\binom{n}{\ell} \cdot \varepsilon, \ell\right)$*-adaptive secrecy of partial views.*

Definition 3.3 (RPE - adaptive reconstruction from partial views)**.** *Let* $\mathsf{RPE} = (\mathsf{Encode}, \mathsf{Decode}, \mathsf{Rec})$ *be an RPE scheme, with* $\mathsf{Encode} : \{0,1\}^k \to \{0,1\}^n$*. We say that* RPE *has* (ε, ℓ)*-adaptive reconstruction from partial views if for any adversary* $\mathcal{A}$*, the distinguishing advantage of* $\mathcal{A}$ *in the following game with a challenger* $\mathcal{C}$ *is at most* ε*.*

1. $\mathcal{A}$ *picks* $x \in \{0,1\}^k$ *and sends* x *to* $\mathcal{C}$*.*
2. $\mathcal{C}$ *draws* $b \leftarrow \{0,1\}$*, and encodes* $c \leftarrow \mathsf{Encode}(x)$*.*
3. *For* $i = 1, \ldots, \ell$*:*
   (a) $\mathcal{A}$ *picks* $i \in [n]$ *and sends* i *to* $\mathcal{C}$*.*
   (b) $\mathcal{C}$ *sends* c_i *to* $\mathcal{A}$*.*
4. *If* $b = 1$ *then* $\mathcal{C}$ *sends* c *to* $\mathcal{A}$*. Otherwise, let* $\mathcal{I}$ *denote the set of indices which* $\mathcal{A}$ *queried in Step 3. Then* $\mathcal{C}$ *computes* $c' \leftarrow \mathsf{Rec}\left(x, \mathcal{I}, (c_i)_{i \in \mathcal{I}}\right)$ *and sends* c' *to* $\mathcal{A}$*.*
5. $\mathcal{A}$ *outputs a bit* b'*.*
6. *The* advantage *of* $\mathcal{A}$ *in the game is* $|\Pr[b' = 1 | b = 1] - \Pr[b' = 1 | b = 0]|$*.*

We say that an RPE has ℓ*-adaptive reconstruction from partial views if it has* $(0, \ell)$*-adaptive reconstruction from partial views.*

We will sometimes omit the parameters ℓ, ε, simply saying "an RPE scheme with adaptive reconstruction from partial views".

The following lemma follows by a standard adaptive-to-non-adaptive reduction.

Lemma 3.6 (RPEs have adaptive reconstruction from partial views)**.** *Let* $\mathsf{RPE} = (\mathsf{Encode}, \mathsf{Decode}, \mathsf{Rec})$ *be an* (ε, ℓ)*-RPE scheme, with* $\mathsf{Encode} : \{0,1\}^k \to \{0,1\}^n$*. Then* RPE *has* $\left(\binom{n}{\ell} \cdot \varepsilon, \ell\right)$*-adaptive reconstruction from partial views.*

Definition 3.3 guarantees that for every secret x, no adversary can distinguish between an encoding of x and an encoding that was resampled consistently with (adaptive) leakage on the encoding. However, for our equivocal SSS construction we will need the stronger guarantee that an adversary cannot distinguish between an encoding of x, and an encoding that was resampled consistently with (adaptive) leakage on an encoding of *some arbitrary* x' (possibly different from x). This property, which we formalize next, follows from a combination of secrecy and reconstruction of RPEs. (In fact, our constructions will not directly use the adaptive reconstruction property of RPEs, it will only be used to prove the adaptive reconstruction* property in Lemma 3.7 below.)

Definition 3.4 (RPE - adaptive reconstruction* from partial views)**.** *Let* $\mathsf{RPE} = (\mathsf{Encode}, \mathsf{Decode}, \mathsf{Rec})$ *be an RPE scheme, with* $\mathsf{Encode} : \{0,1\}^k \to \{0,1\}^n$*. We say that* RPE *has* (ε, ℓ)*-adaptive reconstruction* from partial views if for any adversary* $\mathcal{A}$*, the distinguishing advantage of* $\mathcal{A}$ *in the following game with a challenger* $\mathcal{C}$ *is at most* ε*.*

1. $\mathcal{A}$ *picks* $x^0, x^1 \in \{0,1\}^k$ *and sends* x^0, x^1 *to* $\mathcal{C}$*.*
2. $\mathcal{C}$ *draws* $b \leftarrow \{0,1\}$*, and encodes* $c^0 \leftarrow \mathsf{Encode}\left(x^0\right)$ *and* $c^1 \leftarrow \mathsf{Encode}\left(x^1\right)$*.*
3. *For* $i = 1, \ldots, \ell$*:*
   (a) $\mathcal{A}$ *picks* $i \in [n]$ *and sends* i *to* $\mathcal{C}$*.*
   (b) $\mathcal{C}$ *sends* c_i^b *to* $\mathcal{A}$*.*
4. *If* $b = 1$ *then* $\mathcal{C}$ *sends* c^1 *to* $\mathcal{A}$*. Otherwise, let* $\mathcal{I}$ *denote the set of indices which* $\mathcal{A}$ *queried in Step 3. Then* $\mathcal{C}$ *computes* $c^{0\prime} \leftarrow \mathsf{Rec}\left(x^1, \mathcal{I}, \left(c_i^0\right)_{i\in\mathcal{I}}\right)$ *and sends* $c^{0\prime}$ *to* $\mathcal{A}$*.*
5. $\mathcal{A}$ *outputs a bit* b'*.*
6. *The* advantage *of* $\mathcal{A}$ *in the game is* $|\Pr[b' = 1|b = 1] - \Pr[b' = 1|b = 0]|$*.*

We say that an RPE has ℓ*-adaptive reconstruction* from partial views if it has* $(0, \ell)$*-adaptive reconstruction* from partial views.*

We will sometimes omit the parameters ℓ, ε, simply saying "an RPE scheme with adaptive reconstruction* from partial views".

Lemma 3.7 (RPEs have adaptive reconstruction* from partial views)**.** *Let* $\mathsf{RPE} = (\mathsf{Encode}, \mathsf{Decode}, \mathsf{Rec})$ *be an RPE scheme with* (ε, ℓ)*-secrecy of partial views and* (ε', ℓ)*-reconstruction from partial views, where* $\mathsf{Encode} : \{0,1\}^k \to \{0,1\}^n$*. Then* RPE *has* (ε'', ℓ)*-adaptive reconstruction* from partial views, where* $\varepsilon'' = \binom{n}{\ell} \cdot (2\varepsilon + \varepsilon')$*.*

Proof: We prove the claim using a sequence of hybrids.

$\mathcal{H}_0$**:** this is the output of the adversary in the distinguishing game of Definition 3.4, with $b = 1$.

$\mathcal{H}_1$**:** in $\mathcal{H}_1$, in Step 4 of the game of Definition 3.4, instead of returning c^1 to $\mathcal{A}$, we compute $c' \leftarrow \mathsf{Rec}\left(x^1, \mathcal{I}, \left(c_i^1\right)_{i\in\mathcal{I}}\right)$ and provide c' to $\mathcal{A}$ as the answer of the challenger $\mathcal{C}$.

Then $\mathsf{SD}(\mathcal{H}_0, \mathcal{H}_1) \leq \binom{n}{\ell}\varepsilon'$ by the reconstruction from partial views of the RPE and Lemma 3.6. Indeed, $\mathcal{H}_0$ is the output of $\mathcal{A}$ in the adaptive reconstruction from partial views game of Definition 3.3 with $x = x^1$ when the challenger picks bit 0, and $\mathcal{H}_1$ is its view when the challenger picks bit 1. Since the RPE has (ε', ℓ)-reconstruction from partial views, by Lemma 3.6 it has $\left(\binom{n}{\ell}\varepsilon', \ell\right)$*-adaptive* reconstruction from partial views, namely $\mathsf{SD}(\mathcal{H}_0, \mathcal{H}_1) \leq \binom{n}{\ell}\varepsilon'$.

$\mathcal{H}_2$: this is the output of the adversary in the distinguishing game of Definition 3.4, with $b = 0$. Notice that the only difference between $\mathcal{H}_1, \mathcal{H}_2$ is that in $\mathcal{H}_1$ we answer queries in Step 3 according to an encoding of x^1, whereas in $\mathcal{H}_2$ we answer according to an encoding of x^0.

$\mathsf{SD}(\mathcal{H}_1, \mathcal{H}_2) \leq 2\binom{n}{\ell}\varepsilon$ by the secrecy of partial views of the RPE and Lemma 3.5. Indeed, Lemma 3.5 guarantees that the RPE has $\left(2\binom{n}{\ell}\varepsilon, \ell\right)$-*adaptive* secrecy of partial views. Moreover, given an adversary $\mathcal{A}$ that obtains advantage ε^* in distinguishing $\mathcal{H}_1$ from $\mathcal{H}_2$, we can construct an adversary $\mathcal{A}'$ that obtains ε^* distinguishing advantage in the security game of Definition 3.2, which implies that $\varepsilon^* \leq 2\binom{n}{\ell}\varepsilon$ as claimed. $\mathcal{A}'$ will play the role of the challenger for $\mathcal{A}$: it will send x^0, x^1 to its own challenger $\mathcal{C}'$, and forward queries i from $\mathcal{A}$ to $\mathcal{C}'$. Let $\mathcal{I}$ denote the set of queries which $\mathcal{A}$ made throughout the game, and let $(c_i)_{i\in\mathcal{I}}$ denote the answers of $\mathcal{C}'$ to the queries. At the end of the game, $\mathcal{A}'$ computes $c' \leftarrow \mathsf{Rec}\left(x^1, \mathcal{I}, (c_i)_{i\in\mathcal{I}}\right)$ and gives c' to $\mathcal{A}$. Then, $\mathcal{A}'$ outputs the bit b' which $\mathcal{A}$ outputs. Notice that if $b = 1$ in the security game of Definition 3.2 for $\mathcal{A}'$, then $\mathcal{A}$ is emulated with $\mathcal{H}_1$, whereas if $b = 0$ in the security game of $\mathcal{A}'$ then $\mathcal{A}$ is run with $\mathcal{H}_2$.

■

In the next sections we will use the following notation:

Notation 1 (RPEs.)**.** *An* adaptive $(\varepsilon, \ell, \tau)$-RPE scheme *is an RPE scheme (As in Definition 3.1) with τ-error-correction, (ε, ℓ)-adaptive secrecy from partial views, (ε, ℓ)-adaptive reconstruction from partial views, and (ε, ℓ)-adaptive reconstruction* from partial views.*

Gammel and Mangard [GM10] and Ball et al. [BDG+18, Lemma 2] show that the existence of linear codes implies the existence of RPEs. Specifically, they prove the following, where a code $\mathcal{C} \subseteq \{0,1\}^n$ is linear if its encoding procedure simply multiplies the input with a public generator matrix, and the distance of the code is $\min_{c\in\mathcal{C}} \mathrm{Ham}(c, 0^n)$ (i.e., the minimal weight of a non-zero codeword):

Lemma 3.8 (RPEs from linear error-correcting codes [GM10, BDG+18])**.** *If there exists a linear error-correcting code $\mathcal{C} \subseteq \{0,1\}^n$ with messages in $\{0,1\}^k$ and distance d, then there exists an $(0, d-1, d-1)$-RPE for messages in $\{0,1\}^k$ and codewords in $\{0,1\}^n$.*

To obtain an RPE with good parameters, we apply Lemma 3.8 to any explicit family of linear codes with constant rate and constant relative distance (e.g., the codes of [DGR99, Thoerem 2.1], which already posses secrecy from partial views)[19], and using Lemmas 3.5-3.7, we deduce that adaptive RPEs with good parameters exist:

Corollary 3.1 (Adaptive RPEs)**.** *For every message length $k \in \mathbb{N}$, there exists an adaptive $(0, \Omega(k), \Omega(k))$-RPE for messages in $\{0,1\}^k$ with codewords of length $O(k)$.*

Finally, we note that one can obtain an RPE over an arbitrary field $\mathbb{F}$ by generating a random linear code. Indeed, a random linear code $\mathcal{C} \subseteq \mathbb{F}^n$ will have $\Omega(n)$ dual distance except with negligible probability, and by Remark 3.4, such a code is an RPE. This is summarized in the following lemma:

Lemma 3.9 (RPEs from random linear codes)**.** *For any field $\mathbb{F}$ and every message length $k \in \mathbb{N}$, there exists an adaptive $(0, \Omega(k), \Omega(k))$-RPE for messages in $\mathbb{F}^k$ with codewords of length $O(k)$. Moreover, a generator matrix for the code can be constructed in probabilistic polynomial time, except with negligible failure probability.*

We note that while Corollary 3.1 gives a fully-explicit construction, Lemma 3.9 gives a semi-explicit construction.

[19] We note that the construction of [DGR99] relies on Toeplitz matrices generating codes with good parameters, such a matrix can be efficiently found by traversing all these matrices by some pre-defined order, and using the first matrix satisfying the desired properties.

An Equivocal SSS from Resampleable Secret Sharing and RPEs

Building blocks: a resampleable m-party SSS $\left(\mathsf{Share}^{\mathsf{in}}, \mathsf{Reconst}^{\mathsf{in}}\right)$, and an RPE scheme $(\mathsf{Encode}, \mathsf{Decode}, \mathsf{Rec})$.

Share on input a secret x:

- Computes $\left(\mathsf{Sh}_1^{\mathsf{in}}, \ldots, \mathsf{Sh}_m^{\mathsf{in}}\right) \leftarrow \mathsf{Share}^{\mathsf{in}}(x)$.
- For every $i \in [m]$, encodes $\mathsf{Sh}_i \leftarrow \mathsf{Encode}\left(\mathsf{Sh}_i^{\mathsf{in}}\right)$.
- Outputs the secret shares $(\mathsf{Sh}_1, \ldots, \mathsf{Sh}_m)$.

$\mathsf{Reconst}$ on input shares $(\mathsf{Sh}_i)_{i \in T}$ of an authorized set $T \subseteq [m]$:

- For every $i \in T$, decodes $\mathsf{Sh}_i^{\mathsf{in}} = \mathsf{Decode}(\mathsf{Sh}_i)$.
- Reconstructs $x = \mathsf{Reconst}\left(\left(\mathsf{Sh}_i^{\mathsf{in}}\right)_{i \in T}\right)$, and output x.

Figure 2: An Equivocal SSS

4 Equivocal SSS from SSSs and RPEs

In this section we describe a general transformation from standard SSSs and RPEs to equivocal SSS. Our schemes will also be tampering-resilient.

4.1 The Equivocal SSS Construction

In this section we describe and analyze our equivocal SSS from (non-leakage-resilient) SSS and RPEs. Our construction is presented in Figure 2, and uses a resampleable SSS and an RPE as building blocks.

The next theorem summarizes the properties of the construction.

Theorem 4.1 (Equivocal SSS from Resampleable SSS and RPE)**.** *Assume that Construction 2 is instantiated with:*

- *An ε-resampleable ε'-adaptively secret SSS for access structure Acc.*
- *An adaptive $(\varepsilon'', \ell, \tau)$-RPE scheme.*

Then Construction 2 is a τ-tampering-resilient and $\varepsilon + \varepsilon' + m \cdot \varepsilon''$-equivocal SSS for the class of (Acc, ℓ)-local probing leakage.

Moreover, if the SSS has shares of length k, and the RPE maps messages of length k to codewords of length n_k, then the shares in Construction 2 have length n_k.

Remark 4.1 (The role of resampleability)**.** *We could also use a standard,* not resampleable *SSS in Construction 2, to obtain an equivocal SSS with an* inefficient *simulator. Indeed, as noted in Section 2.2.1, any secret sharing scheme is resampleable with an inefficient resampling algorithm, and this resampling algorithm could be used by an inefficient simulator to equivocate the secret shares.*

Remark 4.2 (Tampering-resilience against malicious corruptions)**.** *We note that if the construction of Figure 2 is instantiated with a* robust *SSS for some access structure Acc, then it is also τ-tampering-resilient (as in Definition 2.7) against any* malicious *adversary in Acc.*

Proof (of Theorem 4.1): The claim regarding the share length follows immediately from the construction. Correctness of the scheme follows directly from the correctness properties of the underlying SSS and RPE. Secrecy and leakage resilience of the scheme will follow from the equivocation property, which we prove below. (However, we note that secrecy also follows directly from the secrecy of the underlying SSS because the RPE operates on each share separately.)

As for tampering-resilience, it follows directly from the error-correction of the RPE, and from the fact that each share is encoded separately. Indeed, modifying an RPE encoding in at most τ bits does not affect the recovered message, and therefore the adversary has no affect on the shares, and so the correctness of the underlying secret sharing scheme guarantees that the (correct) secret will be reconstructed (we can use the correctness of the secret sharing scheme since the adversary is semi-honest).[20]

We now show that the scheme is equivocal. Let $\mathcal{A}$ be an adversary in the equivocation experiment of Figure 1, and we describe a simulator Sim which emulates the oracles of $\mathcal{A}$. We assume that each $\mathcal{LEAK}$ query of $\mathcal{A}$ leeks a single bit from a single share (and, in particular, full shares are leaked only through $\mathcal{SHARE}$ queries). This assumption is without loss of generality, and will simplify the description of the simulator. Sim operates as follows.

1. Generates a secret sharing of $\vec{0}$ by computing $\left(\mathsf{Sh}_1^{\mathsf{in}},\ldots,\mathsf{Sh}_m^{\mathsf{in}}\right) \leftarrow \mathsf{Share}^{\mathsf{in}}\left(\vec{0}\right)$ and then encoding each $\mathsf{Sh}_i^{\mathsf{in}}$ as $\widehat{\mathsf{Sh}}_i \leftarrow \mathsf{Encode}\left(\mathsf{Sh}_i^{\mathsf{in}}\right)$.

2. In the first phase of the experiment, Sim answers leakage queries as follows:

   (a) $\mathcal{LEAK}^{\mathcal{I}}$ queries are answered according to $\left(\widehat{\mathsf{Sh}}_1,\ldots,\widehat{\mathsf{Sh}}_m\right)$.

   (b) A $\mathcal{SHARE}^{\mathcal{I}}(i)$ query is answered as follows. Let L_i denote the set of coordinates of $\widehat{\mathsf{Sh}}_i$ that have already been leaked through $\mathcal{LEAK}^{\mathcal{I}}$ queries. Then Sim samples $\widehat{\mathsf{Sh}}_i' \leftarrow \mathsf{Rec}\left(\mathsf{Sh}_i^{\mathsf{in}}, L_i, \left(\widehat{\mathsf{Sh}}_{i,j}\right)_{j\in L_i}\right)$, where $\widehat{\mathsf{Sh}}_{i,j}$ denotes the j'th bit of $\widehat{\mathsf{Sh}}_i$, and returns $\widehat{\mathsf{Sh}}_i'$ as the oracle answer.

3. At the onset of the second phase, let T denote the set of indices i such that $\mathcal{A}$ queried $\mathcal{SHARE}^{\mathcal{I}}(i)$ in the first phase of the experiment. For every $i \notin T$, let L_i denote the set of coordinates of the i'th share that were leaked through $\mathcal{LEAK}^{\mathcal{I}}$ queries.

   When the experiment enters its second phase, Sim obtains a secret x. If T is not an unauthorized set, or $|L_i| > \ell$ for some $i \notin T$, then Sim aborts.

4. Sim resamples a secret sharing of x under the inner SSS by computing $\left(\widetilde{\mathsf{Sh}}_1',\ldots,\widetilde{\mathsf{Sh}}_m'\right) \leftarrow \mathsf{Resample}\left(x, T, \left(\mathsf{Sh}_i^{\mathsf{in}}\right)_{i\in T}\right)$. Then, for every $i \notin T$, Sim samples the i'th share as $\widetilde{\mathsf{Sh}}_i'' \leftarrow \mathsf{Rec}\left(\widetilde{\mathsf{Sh}}_i', L_i, \left(\widehat{\mathsf{Sh}}_{i,j}\right)_{j\in L_i}\right)$.

5. For every $i \in [m]$, if $i \in T$ then Sim sets $\widetilde{\mathsf{Sh}}_i = \widehat{\mathsf{Sh}}_i$, otherwise Sim sets $\widetilde{\mathsf{Sh}}_i = \widetilde{\mathsf{Sh}}_i''$.

6. Sim outputs the secret shares $\left(\widetilde{\mathsf{Sh}}_1,\ldots,\widetilde{\mathsf{Sh}}_m\right)$.

[20] As noted above, if the underlying SSS is robust against some access structure Acc then reconstruction is guaranteed to recover the shared secret even in the presence of a *malicious* adversary in Acc that can additionally arbitrarily modify τ bits in the share of every honest party.

Next, we bound the distance between the outputs of the ideal and simulated experiments. We will condition on the event that Sim did not abort. This is without loss of generality because whenever Sim aborts, $\mathcal{VALID}$ would return false and the output would be 0 in both experiments. To prove the claim, we will bound the statistical distance between the adversarial views in the real and ideal experiments, by a sequence of hybrids. In all hybrids, the distributions are generated through interaction with the adversary (and depend on its queries). The hybrids differ in how they generate the answers to the adversary.

$\mathcal{H}_0$**:** this is the adversarial view in the real world, where a secret x is secret shared into shares $(\mathsf{Sh}_1, \ldots \mathsf{Sh}_m) \leftarrow \mathsf{Share}(x)$ with underlying shares $\left(\mathsf{Sh}_1^{\mathsf{in}}, \ldots, \mathsf{Sh}_m^{\mathsf{in}}\right)$ of the internal SSS. (That is, Sh_i is obtained by encoding $\mathsf{Sh}_i^{\mathsf{in}}$ using the RPE scheme.)

$\mathcal{H}_1$**:** Intuitively, in $\mathcal{H}_1$ we answer $\mathcal{LEAK}$ queries using a *sharing of* $\vec{0}$ (instead of the sharing of x as in $\mathcal{H}_0$). Moreover, to answer $\mathcal{SHARE}$ queries, and to reveal the remaining shares in the second phase of the experiment, we generate (using Rec) *fresh RPE-encodings* of the *secret shares of* x, consistently with the leakage. We note that regenerating the RPE-encodings is necessary since the bits leaked through $\mathcal{LEAK}$ queries were of encodings of shares of $\vec{0}$, and might not be consistent with the encodings we wish to reveal (which are encodings of shares of x).

Specifically, in $\mathcal{H}_1$, we generate a random encoding of $\vec{0}$ by computing $\left(\overline{\mathsf{Sh}}_1^{\mathsf{in}}, \ldots, \overline{\mathsf{Sh}}_m^{\mathsf{in}}\right) \leftarrow \mathsf{Share}^{\mathsf{in}}\left(\vec{0}\right)$, and then encoding each share as $\overline{\mathsf{Sh}}_i \leftarrow \mathsf{Encode}\left(\overline{\mathsf{Sh}}_i^{\mathsf{in}}\right)$. Then, we answer $\mathcal{LEAK}$ queries according to $\left(\overline{\mathsf{Sh}}_1, \ldots, \overline{\mathsf{Sh}}_m\right)$ (i.e., according to RPE-encodings of secret shares of $\vec{0}$). Additionally, we answer $\mathcal{SHARE}(i)$ queries as follows. Let L_i denote the coordinates of the i'th share that were already queried through $\mathcal{LEAK}$ queries. Then we resample $\overline{\mathsf{Sh}}_i' \leftarrow \mathsf{Rec}\left(\mathsf{Sh}_i^{\mathsf{in}}, L_i, \left(\overline{\mathsf{Sh}}_{i,j}\right)_{j \in L_i}\right)$, where $\overline{\mathsf{Sh}}_{i,j}$ denotes the j'th bit of $\overline{\mathsf{Sh}}_i$.[21] Then, we output $\overline{\mathsf{Sh}}_i'$ as the answer of the oracle. (Subsequent $\mathcal{SHARE}$ or $\mathcal{LEAK}$ queries to the i'th share will be answered according to $\overline{\mathsf{Sh}}_i'$.) Finally, in the second phase of the game, for every i such that no $\mathcal{SHARE}(i)$ query was made in the first phase, we resample $\mathsf{Sh}_i' \leftarrow \mathsf{Rec}\left(\mathsf{Sh}_i^{\mathsf{in}}, L_i, \left(\overline{\mathsf{Sh}}_{i,j}\right)_{j \in L_i}\right)$ and provide Sh_i' as the opening of the i'th share.

We claim that $\mathsf{SD}(\mathcal{H}_0, \mathcal{H}_1) \leq m\varepsilon''$. Indeed, this follows from the adaptive reconstruction* from partial views of the RPE, using a union bound. Specifically, we define a sequence of hybrids $\mathcal{H}_0^0 = \mathcal{H}_0, \mathcal{H}_0^1, \ldots, \mathcal{H}_0^m = \mathcal{H}_1$ where in the j'th hybrid we: (1) answer $\mathcal{LEAK}$ queries on the first j shares using $\left(\overline{\mathsf{Sh}}_1, \ldots, \overline{\mathsf{Sh}}_m\right)$; (2) answer $\mathcal{LEAK}$ queries to the other shares using $(\mathsf{Sh}_1, \ldots \mathsf{Sh}_m)$; and (3) for each $k \leq j$, resample an encodings of the k'th share (by applying Rec to $\mathsf{Sh}_k^{\mathsf{in}}$) before revealing it. Then every pair of adjacent hybrids are ε''-statistically close by adaptive reconstruction* from partial views (and the claim follows from a union bound over all such pairs). Indeed, given an (adaptive)[22] distinguisher $\mathcal{D}$ between some pair $\mathcal{H}_0^{j-1}, \mathcal{H}_0^j$ we can build an adversary $\mathcal{A}'$ that wins the game of Definition 3.4: $\mathcal{A}'$ has x hard-wired into it, and can emulate the oracles of $\mathcal{D}$ on all shares except the j'th one. For the j'th share it uses

[21] Notice that here, and in the second phase of the experiment, we use Rec to generate encodings of secret shares *of* x. These encodings are sampled to be consistent with the leakage obtained through $\mathcal{LEAK}$ queries (which leaked from encodings of secret shares *of* $\vec{0}$).

[22] The distinguisher is adaptive in the sense that it can "play" the part of the adversary in the experiment, i.e., make adaptive leakage queries.

its challenger $\mathcal{C}$ (with secrets $\overline{\mathsf{Sh}}_j^{\mathsf{in}}, \mathsf{Sh}_j^{\mathsf{in}}$), and when $\mathcal{D}$ outputs a bit b', $\mathcal{A}'$ outputs b'. Then if $b = 1$ in the security game of $\mathcal{A}'$, then $\mathcal{D}$ is emulated with $\mathcal{H}_0^{j-1}$, otherwise it is emulated with $\mathcal{H}_0^j$, so $\mathcal{A}'$ obtains the same distinguishing advantage as $\mathcal{D}$.

$\mathcal{H}_2$: Intuitively, in $\mathcal{H}_2$ we change the secret shares which are revealed in the second phase of the experiment. Instead of re-encoding the (original) shares $\mathsf{Sh}_i^{\mathsf{in}}$ of x, we resample a fresh secret sharing of x, conditioned on the shares that were already revealed (through $\mathcal{SHARE}$ queries) in the first phase of the experiment. This effectively decouples the remaining secret shares, which were not revealed in the first phase of the experiment, from the secret shares already revealed through $\mathcal{SHARE}$ queries, and will allow us to use the secrecy of the secret sharing scheme in the subsequent hybrid.

Specifically, let T denote the set of shares i for which the adversary made $\mathcal{SHARE}(i)$ queries. In $\mathcal{H}_2$, in the second phase of the experiment, we compute the shares of $i \notin T$ as follows. We resample $\left(\widehat{\mathsf{Sh}}_1', \ldots, \widehat{\mathsf{Sh}}_m'\right) \leftarrow \mathsf{Resample}\left(x, T, \left(\mathsf{Sh}_i^{\mathsf{in}}\right)_{i \in T}\right)$, then reconstruct encodings of $\left(\widehat{\mathsf{Sh}}_i'\right)_{i \notin T}$ (instead of $\left(\mathsf{Sh}_i^{\mathsf{in}}\right)_{i \notin T}$).

We claim that $\mathsf{SD}(\mathcal{H}_1, \mathcal{H}_2) \leq \varepsilon$ because the underyling SSS is resampleable. Specifically, given an (adaptive) distinguisher $\mathcal{D}$ with $\widehat{\varepsilon}$ distinguishing advantage between $\mathcal{H}_1, \mathcal{H}_2$, we describe an adversary $\mathcal{A}'$ winning the resampleability game of Definition 2.5 for secret x with the same probability $\widehat{\varepsilon}$. $\mathcal{A}'$ has x hard-wired into it, and emulates the oracles for $\mathcal{D}$ as follows. During the first phase of the experiment, $\mathcal{A}'$ answers $\mathcal{LEAK}$ queries according to $\left(\overline{\mathsf{Sh}}_1, \ldots, \overline{\mathsf{Sh}}_m\right)$. It answers $\mathcal{SHARE}(i)$ queries by sending i to its challenger, to obtain a share $\mathsf{Sh}_i^{\mathsf{in}'}$. Let L_i denote the coordinates of the i'th share that were already queried through $\mathcal{LEAK}$ queries, then $\mathcal{A}'$ samples $\mathsf{Sh}_i' \leftarrow \mathsf{Rec}\left(\mathsf{Sh}_i^{\mathsf{in}'}, L_i, \left(\overline{\mathsf{Sh}}_{i,j}\right)_{j \in L_i}\right)$, and sends Sh_i' to $\mathcal{D}$ as the oracle answer. In the second phase of the game, $\mathcal{A}'$ obtains shares $\left(\mathsf{Sh}_1^{\mathsf{in}''}, \ldots, \mathsf{Sh}_m^{\mathsf{in}''}\right)$ from its challenger. For every i which was not queried as a $\mathcal{SHARE}(i)$ query in the first phase of the game, it resamples $\mathsf{Sh}_i'' \leftarrow \mathsf{Rec}\left(\mathsf{Sh}_i^{\mathsf{in}''}, L_i, \left(\overline{\mathsf{Sh}}_{i,j}\right)_{j \in L_i}\right)$, where L_i denotes the set of coordinates that were queried from the i'th share through $\mathcal{LEAK}$ queries in the first phase of the game. It provides Sh_i'' to $\mathcal{D}$. When $\mathcal{D}$ outputs a bit b', $\mathcal{A}$ outputs b'. Notice that if $b = 0$ in the game of $\mathcal{A}'$ then $\mathcal{D}$ is emulated with $\mathcal{H}_1$, otherwise it is emulated with $\mathcal{H}_2$, so $\mathcal{A}'$ obtains the same distinguishing advantage as $\mathcal{D}$.

$\mathcal{H}_3$: In $\mathcal{H}_3$, the shares $(\mathsf{Sh}_1, \ldots, \mathsf{Sh}_m)$ are computed as a sharing of $\vec{0}$ (instead of x). The difference between the hybrids is that in $\mathcal{H}_2$, $\mathcal{SHARE}$ queries are answered with secret shares of x, whereas in $\mathcal{H}_3$ they are answered with secret shares of $\vec{0}$. (We stress that when resampling the secret shares in the second phase of the game in $\mathcal{H}_3$, they are resampled as shares of x.)

We claim that $\mathsf{SD}(\mathcal{H}_2, \mathcal{H}_3) \leq \varepsilon'$ from the adaptive secrecy of the underlying secret sharing scheme (Definition 2.6). Indeed, given a distinguisher $\mathcal{D}$ between $\mathcal{H}_2, \mathcal{H}_3$ we construct a (possibly adaptive) distinguisher $\mathcal{A}'$ between secret shares of $x, \vec{0}$ for some unauthorized set. Specifically, $\mathcal{A}'$ has x hard-wired into it. It emulates the oracles of $\mathcal{D}$ as follows. At the onset of the emulation it generates secret shares $\left(\overline{\mathsf{Sh}}_1, \ldots, \overline{\mathsf{Sh}}_m\right) \leftarrow \mathsf{Share}\left(\vec{0}\right)$, and uses these secret shares to answer $\mathcal{LEAK}$ queries of $\mathcal{D}$. For every $\mathcal{SHARE}(i)$ query of $\mathcal{D}$, it forwards i to its challenger, and obtains a secret share Sh_i'. Let L_i denote the coordinates of the i'th share that were already leaked through $\mathcal{LEAK}$ queries, then $\mathcal{A}'$ samples

$\mathsf{Sh}_i \leftarrow \mathsf{Rec}\left(\mathsf{Sh}'_i, L_i, \left(\overline{\mathsf{Sh}}_{i,j}\right)_{j\in L_i}\right)$, and provides Sh_i to $\mathcal{D}$ as the answer of the oracle. Let T denote the set of parties whose shares $\mathcal{D}$ obtained in the first phase of the experiment through $\mathcal{SHARE}$ queries. At the onset of the second phase of the experiment, $\mathcal{A}'$ generates a fresh secret sharing $\left(\widetilde{\mathsf{Sh}}'_1, \ldots \widetilde{\mathsf{Sh}}'_m\right) \leftarrow \mathsf{Resample}\left(x, T, \left(\mathsf{Sh}'_i\right)_{i\in T}\right)$. Then, for every $i \notin T$ it samples $\widetilde{\mathsf{Sh}}_i \leftarrow \mathsf{Rec}\left(\widetilde{\mathsf{Sh}}'_i, L_i, \left(\overline{\mathsf{Sh}}_{i,j}\right)_{j\in L_i}\right)$, where L_i denote the coordinates that were leaked from the i'th share in the first phase of the experiment. Finally, it provides $\left(\mathsf{Sh}'_i\right)_{i\in T}, \left(\widetilde{\mathsf{Sh}}_i\right)_{i\notin T}$ to $\mathcal{D}$ as the secret sharing. Notice that if in the security game of $\mathcal{A}'$ it obtained secret shares of $\vec{0}$, then it emulates $\mathcal{H}_3$ for $\mathcal{D}$. Otherwise, it emulates $\mathcal{H}_2$. Therefore, $\mathcal{A}'$ obtains the same distinguishing advantage as $\mathcal{D}$.

We conclude the proof by noting that $\mathcal{H}_3$ is the view of the adversary in the ideal experiment. ■

The following corollary is obtained by instantiating Theorem 4.1 with Shamir's secret sharing (Definition 2.4) and the RPEs of Corollary 3.1:

Corollary 4.2. *For any natural length parameter N, number of parties m, and threshold $t < m$, there exist an m-party SSS for secrets in $\{0,1\}^N$ that is ℓ-tampering-resilient and perfectly equivocal for leakage class $\mathcal{L}_{t,\ell}$, where $\ell = \Omega(N)$. Moreover, the shares have length $O(N)$.*

Proof: We instantiate Construction 2 with Shamir's scheme for secrets in the field $\mathbb{F}_{2^N}$, and the RPEs of Corollary 3.1 with $k = N$. Shamir's scheme has perfect secrecy and perfect resampleability (see Remark 2.2). Moreover, by Corollary 3.1 the RPE is an adaptive $(0, \Omega(N), \Omega(N))$-RPE for messages in $\{0,1\}^N$ with codewords of length $O(N)$. Consequently Theorem 4.1 guarantees that the resultant SSS is $\Omega(N)$-tampering-resilient and perfectly equivocal for the class of $(t, \Omega(N))$-local probing leakage. Moreover, since the shares in Shamir's scheme are also in $\mathbb{F}_{2^N}$, and the RPE has constant rate, we obtain in Construction 2 shares of length $O(N)$. ■

4.2 Deniable Secret Sharing Schemes

In this section we elaborate on the technical side of obtaining a deniable SSS based on an equivocal SSS, staring with formally defining the primitive. Towards defining deniable secret sharing schemes we define the notion of a (t,ℓ)-transcript, which captures the information obtained by the adversary while attacking the secret sharing scheme.

Definition 4.1 ((t,ℓ)-transcript)**.** *A (t,ℓ)-transcript with respect to a t-threshold secret sharing scheme consists of the shares of the parties in an unauthorized set $\mathcal{G}$ of size $|\mathcal{G}| \leq t$, and a sequence of $m-t$ subsets $(\mathcal{I}_i)_{i\notin\mathcal{G}}$ such that $|\mathcal{I}_i| \leq \ell$ for every $i \notin \mathcal{G}$.*

Next, we formalize the notion of deniable secret sharing as follows.

Definition 4.2. *An m-party* Deniable Secret Sharing Scheme (DSSS) *for secrets in $\mathcal{S}$ is a secret sharing scheme associated with an additional* PPT *algorithm* Fake *that takes as input a secret in $\mathcal{S}$ and a (t,ℓ)-transcript, such that for every pair of secrets $s, s^* \in \mathcal{S}$, and any (t,ℓ)-transcript* trans *of a sharing of s, it holds that*

$$\Pr\left[\mathsf{Share}(s^*;r)|_{(\mathcal{G},\ell)} = \mathsf{trans} \;:\; r \leftarrow \mathsf{Fake}(s^*, \mathsf{trans})\right] = 1$$

where the probability is over the coin tosses of Fake*, and* $\mathsf{Share}(s)|_{(\mathcal{G},\ell)}$ *denotes the restriction of the shares output by* Share *to the shares of the parties in the unauthorized set $\mathcal{G}$, and for every share of party $i \notin \mathcal{G}$, the restriction of its share to the bits in $\mathcal{I}_i$.*

Our construction requires a strongly-equivocal SSS which, roughly, is an equivocal SSS with the following additional guarantee: the equivocation simulator consists of two simulators $(\mathsf{Sim}_1, \mathsf{Sim}_2)$ which operate as follows: Sim_1 is given a secret x, it honestly secret shares x and answers the leakage queries according to these shares. Next, Sim_2 is given a new secret x', and the leakage from the previous phase – namely the leakage queries and the responses of Sim_1, but *not* the state of Sim_1 – and equivocates a secret sharing of x', consistently with the leakage.

Given such a scheme, we construct a deniable SSS as follows: to share x, run the sharing algorithm of the strongly-equivocal SSS (notice that this is equivalent to running Sim_1). Then, to "explain" the adversary's view, given the leakage, as the sharing of a desired secret x', run Sim_2 with x'. We note that our equivocal SSS (Construction 2) satisfies this property. Indeed, in the proof of Theorem 4.1 we describe a simulator that, in the first phase of the game, honestly shares $\vec{0}$ and answers leakage queries according to these shares. Then, in the second phase of the game, it equivocates a sharing consistently with the given secret and previous leakage. The choice of $\vec{0}$ in our proof was arbitrary, and can be replaced with any other value. Moreover, the simulation in the second phase of the game depended only on the leakage queries, the simulated responses, and the given secret. In particular, we can generalize our simulator to take a secret x as input, and use it (instead of $\vec{0}$) in the first phase of the game, and then continue the simulation using these shares, as described in the proof of Theorem 4.1.

In summary, for any threshold t, using the (strongly) equivocal SSS implied by Corollary 4.2 for threshold t, the deniable SSS described above has constant rate, and resists (t, ℓ)-local probing leakage, and where ℓ is a constant fraction of the share size. More formally,

Corollary 4.3 (Leakage-Resilient and Tampering-Resilient DSSS)**.** *For any natural length parameter N, number of parties m, and threshold $t < m$, there exist an m-party DSSS for secrets in $\{0,1\}^N$ for leakage class $\mathcal{L}_{t,\ell}$, where $\ell = \Omega(N)$. Moreover, the shares have length $O(N)$.*

5 Leakage-Resilient and Tampering-Resilient Distributed Primitives

In this section we explore the applicability of our paradigm beyond secret sharing schemes and demonstrate its usefulness for a broad class of cryptographic building blocks. Applying our paradigm to each of these primitives gives a leakage- and tampering-resilient variant of the primitive. The high-level idea of the paradigm is to individually encode each share using an RPE. This ensures that the resulting share resists probing leakage (when stored in memory) due to the probing resilience of the RPE. To obtain tampering-resilience, we exploit the error-correction properties of RPE which guarantees that RPE decoding can correct a constant fraction of errors in the codeword. One of the main challenges is to define a meaningful notion of leakage-resilience for each such object, since we need to provide information-theoretic security against adaptive leakage queries, whereas the original primitives are sometimes only statically and non-adaptively secure (meaning we need to generalize their security notions). We elaborate more on these challenges in the following sections, when we define the leakage model for each primitive.

The leakage model. Our constructions provide *adaptive* leakage-resilience for *data at rest*. Namely, we consider distributed primitives in which the parties hold *shares* of some secret. These shares are protected against adaptive leakage while they are being stored in memory, but not when they are computed on. This model is motivated by the fact that the secret shares may be stored for extended time periods before, and following, the phase in which they are computed on. Indeed, the primitives we consider involve a fast (local) computation phase, or a short interactive protocol (during which

parties compute over their shares) which requires all parties to be online at the same time. The duration of these phases is expected to be much shorter than the lifetime of the entire process.

In the following sections, we define leakage-resilience explicitly for each primitive. We note that in all cases, the leakage class we consider is that of (t, ℓ)-local probing leakage (cf. Definition 2.2), but our definitions naturally extend to more general classes of leakage functions. Furthermore, if the underlying building block maintains adaptive security (where the adversary adaptively corrupts the parties throughout the protocol execution), then our transformation supports this security level, and enhances it to allow the adversary to *adaptively* probe a constant fraction of each share.

5.1 Verifiable Secret Sharing (VSS)

In this section, we apply our paradigm to Verifiable Secret Sharing (VSS) schemes, to obtain a leakage- and tampering-resilient VSS. Intuitively, a VSS scheme is an enhanced SSS with the added guarantee that after the sharing phase, even a corrupted dealer is committed to *some* secret. This should be contrasted with standard SSSs in which a dealer might distribute shares that are inconsistent with *any* secret, and this will only become known during the reconstruction phase. To achieve this added feature, VSS schemes have an additional *verification phase* as part of the sharing phase, which is an interactive sub-protocol in which all parties can communicate to verify global consistency of their shares.[23]

We protect against leakage of "data at rest" - meaning the shares are protected while being stored in the memory of each of the parties, and up until they are used to recover the secret during reconstruction. We do not protect the shares when they are being computed on, i.e., during the verification sub-protocol.

Definition 5.1 (Verifiable Secret Sharing (VSS))**.** *An* ε-secure (m, t)-Verifiable Secret Sharing (VSS) *scheme is a three-phase protocol between* n *parties* $P_1 \ldots, P_m$ *with a designated party* $D = P_1$ *called the* dealer. *The scheme consists of the following phases*

- **The sharing phase** *is a procedure run by the dealer* D *on input* s *(the secret).* D *generates secret shares* $\mathsf{Sh}_1, \ldots, \mathsf{Sh}_m$*, and sends* Sh_i *to party* P_i*.*
- **The verification phase** *is an interactive protocol during which each party can send private messages to every other party, as well as broadcast messages. The input of party* P_i *is its share* Sh_i *from the sharing phase. Its output is a share* $\widetilde{\mathsf{Sh}}_i$*.*
- **The reconstruction phase** *consists of applying a reconstruction function* $\mathsf{Reconst}$ *to the shares* $\widetilde{\mathsf{Sh}}_1, \ldots, \widetilde{\mathsf{Sh}}_m$*.*

A VSS scheme has the following semantics.

- ε**-Privacy.** *If* D *is honest, then for every adversary that adaptively corrupts a subset* $T \subset [m]$ *of parties of size* $|T| \leq t$*, and any pair of secrets* s, s'*,* $\mathsf{SD}(\mathbf{View}_T(s), \mathbf{View}_T(s')) \leq \varepsilon$*, where* $\mathbf{View}_T(s)$ *denotes the joint view of all parties* $P_i, i \in T$ *during the sharing and verification phases, when the dealer has input* s*.*
- **Correctness.** *If* D *is honest then for every secret* s*, any adversary that adaptively corrupts a subset* $T \subset [m]$ *of parties of size* $|T| \leq t$*, and any shares* $\widetilde{\mathsf{Sh}}_1, \ldots, \widetilde{\mathsf{Sh}}_m$ *output by the verification*

[23] We note that VSS definitions usually do not distinguish between the sharing and verification phases. However, to enable us to capture the leakage-resilience guarantees of our VSS protocols, we distinguish between the sharing algorithm and the verification sub-protocol.

phase, $\Pr\left[\mathsf{Reconst}\left(\widetilde{\mathsf{Sh}}_1 \ldots, \widetilde{\mathsf{Sh}}_m\right) = s\right] \geq 1-\varepsilon$*, even if the corrupted parties arbitrarily deviate from the protocol.*[24]

- **Commitment.** *For any adversary* $\mathcal{A}$ *that adaptively corrupts up to* t *parties (including the dealer), except with probability* ε *the execution of the sharing and verification phases determine a unique value* s^* *which will be reconstructed during the reconstruction phase, even if the corrupted parties arbitrarily deviate from the protocol.*[25]

VSS leakage model. As discussed in the beginning of Section 5, our leakage-resilient VSS scheme protects the secret shares against probing leakage when they are being stored. That is, we protect the secret shares throughout the lifetime of the VSS process, and up until the secret is reconstructed from the shares, *except* during the verification phase. This is formalized in Definition 5.2 below.

Definition 5.2 (Leakage-Resilient VSS (LR-VSS))**.** *We say that a VSS scheme is* ε-Leakage-Resilient (LR) for the class of (t, ℓ)-local probing leakage *if the distinguishing advantage of any adversary* $\mathcal{A}$ *in the following game with a challenger* $\mathcal{C}$ *is at most* ε*.*

- $\mathcal{A}$ *picks a pair of secrets* s^0, s^1*, and sends* s^0, s^1 *to* $\mathcal{C}$*.*
- $\mathcal{C}$ *draws* $b \leftarrow \{0,1\}$*, and emulates the sharing and verification phases together with* $\mathcal{A}$*. Throughout this emulation,* $\mathcal{C}$ *emulates the dealer* D *with secret* s^b*, as well as all honest parties. During the emulation,* $\mathcal{A}$ *can adaptively corrupt parties by sending their index to* $\mathcal{C}$*. Upon receiving a party index* j *from* $\mathcal{A}$*, the challenger* $\mathcal{C}$ *sends to* $\mathcal{A}$ *the view of* P_j *up to that point of the emulation. Following the corruption of* P_j*, the challenger* $\mathcal{C}$ *also sends to* $\mathcal{A}$ *the messages which honest parties send to* P_j *during the remainder of the emulation. Let* T *denote the set of parties which* $\mathcal{A}$ *corrupted throughout this process, and let* $\mathbf{View}_T$ *denote the joint view of the parties in* T *during the sharing and verification phases. Let* $(\mathsf{Sh}_j)_{j \notin T}$ *denote the shares the honest parties hold when the verification phase terminates.*
- *Repeat:*
  - $\mathcal{A}$ *picks* $i \in [n]$ *and* $j \in [m] \setminus T$*, and sends* i, j *to* $\mathcal{C}$*.*
  - $\mathcal{C}$ *sends the* i*'th bit of* Sh_j *to* $\mathcal{A}$*.*
- $\mathcal{A}$ *outputs a bit* b'*.*
- *If for some* $j \in [m] \setminus T$*,* $\mathcal{A}$ *made more than* ℓ *distinct leakage queries to* Sh_j*, then the advantage of* $\mathcal{A}$ *is set to 0. Otherwise, the advantage of* $\mathcal{A}$ *in the game is* $|\Pr[b' = 1 | b = 1] - \Pr[b' = 1 | b = 0]|$*.*

Remark 5.1 (Stronger LR)**.** *We note that Definition 5.2 can be further strengthened to allow for leakage between the sharing and verification phases, where the challenger first emulates the sharing phase with* $\mathcal{A}$*, then* $\mathcal{A}$ *is allowed to leak on the shares, then the verification phase is emulated, and finally* $\mathcal{A}$ *can again leak on the shares. We chose to present the weaker definition because it is simpler. However, we note that a slight variation of our LR-VSS of Figure 3 guarantees this stronger LR property. Specifically, at the end of the sharing phase, the dealer first RPE-encodes each share before sending the shares to the parties, and the first step during verification is for all parties to first RPE-decode their shares, where verification is executed with the decoded shares. The security proof*

[24] We note that the parties in T might provide invalid or modified shares for reconstruction. To simplify the notation, we assume that in this case the parties in T output these modified shares already in the verification phase.

[25] We note that the commitment property should hold even if D is corrupted.

then follows similarly to the proof of the equivocation property of the construction of Figure 2, where we add another hybrid in which each share is resampled (using the resampling algorithm of the RPE) consistently with the leakage before it is given to the adversary.

Our Construction. Our scheme combines together a (standard, non-LR) VSS scheme VSS with an RPE scheme. Sharing emulates the sharing procedure of the underlying VSS scheme. The verification phase is carried out by running the verification protocol of VSS to obtain shares $\widetilde{\mathsf{Sh}}_1^{\mathsf{in}}, \ldots, \widetilde{\mathsf{Sh}}_m^{\mathsf{in}}$, and finally having each party locally RPE encode its share to obtain shares $\widetilde{\mathsf{Sh}}_1, \ldots, \widetilde{\mathsf{Sh}}_m$. Upon termination of the verification phase, all parties erase all information generated during the verification phase, except for the encoded shares $\widetilde{\mathsf{Sh}}_1, \ldots, \widetilde{\mathsf{Sh}}_m$. Reconstruction is performed by having all parties provide their shares, which are then RPE decoded to obtain $\widetilde{\mathsf{Sh}}_1^{\mathsf{in}}, \ldots, \widetilde{\mathsf{Sh}}_m^{\mathsf{in}}$, and then the reconstruction procedure of VSS is executed on $\widetilde{\mathsf{Sh}}_1^{\mathsf{in}}, \ldots, \widetilde{\mathsf{Sh}}_m^{\mathsf{in}}$. This construction is formalized in Figure 3, and its properties are summarized in Theorem 5.1 below.

Theorem 5.1 (Tampering-resilient LR-VSS)**.** *Assume that Construction 3 is instantiated with:*

- *An ε-secure (m,t)-VSS.*
- *An adaptive $(\varepsilon', \ell, \tau)$-RPE scheme.*[26]

Then Construction 3 is an ε-secure (m,t)-VSS that is τ-TR and $(\varepsilon + 2(m-t)\varepsilon')$-LR for the class of (t,ℓ)-local probing leakage.

The following corollary is obtained by instantiating the construction of Figure 3 with the RPEs of Corollary 3.1.

Corollary 5.2 (Leakage-Resilient and Tampering-Resilient VSS)**.** *Any ε-secure (m,t)-VSS VSS can be transformed into an $\Omega(N)$-TR VSS which is also ε-LR for the class of $(t,\Omega(N))$-local probing leakage, where N denotes the share length in VSS. Moreover, this transformation only causes a constant blowup in the share length.*

Proof sketch (Theorem 5.1): We argue that Construction 3 satisfies the properties of Definition 5.1, the leakage-resilience guarantee of Definition 5.2, and the tampering-resilience property of Definition 2.7.

Correctness. Follows directly from a combination of the correctness of the underlying VSS and RPE schemes. Indeed, the RPE encoding has no effect in this case, because corrupted parties can overwrite their shares with arbitrary values.

Commitment. Notice that the first step of the verification phase determines *some* secret shares $\widetilde{\mathsf{Sh}}_1^{\mathsf{in}}, \ldots, \widetilde{\mathsf{Sh}}_m^{\mathsf{in}}$, which will be used in the verification protocol of VSS. These values could have been used by the parties in VSS. Indeed, a corrupted party in VSS can anyway overwrite its share with any value that it wants, and use the new value for the verification protocol. For honest parties, every dealer strategy determines which share they will use during verification (in particular, providing P_i with an invalid RPE encoding - i.e., one for which decoding fails - corresponds to a dealer strategy in VSS which gives s_i to P_i as its share). Therefore, commitment reduces to the commitment of VSS.

[26] As will be evident from the proof, we do not need the reconstruction property of the RPE, and it suffices for the RPE to only have (ε', ℓ)-adaptive secrecy of partial views.

Privacy. Follows directly from the privacy of VSS, since the RPE encoding is applied separately to each secret share.

TR. Follows from the error-correction of the RPE (since each share is encoded separately), and from the correctness and commitment of VSS, using similar arguments to the proof of Theorem 4.1. (We note that since the underlying VSS scheme VSS is secure against malicious parties, the resultant VSS scheme is also TR against malicious parties.)

LR. Follows from a combination of the privacy of VSS and the LR of the RPE scheme, using a hybrid argument. Notice that if the adversary makes more than ℓ leakage queries to some share Sh_j then its advantage in the game is 0. Therefore, we can condition on the event that the adversary leaks at most ℓ bits from each share. We denote by $\mathsf{leak}(\mathsf{Sh}_1,\ldots,\mathsf{Sh}_m)$ the bits of the secret shares $\mathsf{Sh}_1,\ldots,\mathsf{Sh}_m$ of s which the adversary queried throughout the game. We now sketch the hybrids.

$\mathcal{H}_0$**:** This is the joint distribution of $\mathbf{View}_T(s)$ and $\mathsf{leak}\left(\mathsf{Sh}_1^0,\ldots,\mathsf{Sh}_m^0\right)$ in an execution with secret s^0 (in particular, $\left(\mathsf{Sh}_1^0,\ldots,\mathsf{Sh}_m^0\right)$ is a random secret-sharing of s^0).

$\mathcal{H}_1$**:** In $\mathcal{H}_1$, we replace $\mathsf{leak}\left(\mathsf{Sh}_1^0,\ldots,\mathsf{Sh}_m^0\right)$ with $\mathsf{leak}(\mathsf{Sh}_1,\ldots,\mathsf{Sh}_m)$, where $(\mathsf{Sh}_1,\ldots,\mathsf{Sh}_m)$ is a random sharing of an arbitrary valid secret s (e.g., $\vec{0}$ if that is a valid secret). Then $\mathsf{SD}(\mathcal{H}_0,\mathcal{H}_1)\leq(m-t)\,\varepsilon'$ due to the adaptive secrecy of partial views of the RPE scheme. (We assume here that $|T|=t$, which is without loss of generality, since we can always generate the leakage on a particular share given the share in its entirety.) Indeed, this can be proved using a standard hybrid argument, where in the jth hybrid for $j\notin T$ we replace the jth share used to compute the leakage from Sh_j^0 to Sh_j. Then every pair of consecutive hybrids are ε'-statistically close by the secrecy of the RPE scheme, because a distinguisher $\mathcal{D}$ between $\mathcal{H}_j$ and the previous hybrid can be used to distinguish between encodings of $\mathsf{Sh}_j^0,\mathsf{Sh}_j$ as follows. The distinguisher has s^0,s hardwired into it, it randomly secret shares them into $\mathsf{Sh}_1^0,\ldots,\mathsf{Sh}_m^0$ and $\mathsf{Sh}_1,\ldots,\mathsf{Sh}_m$. It RPE encodes $\left(\mathsf{Sh}_l^0\right)_{l\in T},\left(\mathsf{Sh}_l^0\right)_{l\notin T,l\leq j},(\mathsf{Sh}_l)_{l\notin T,l>j}$, and asks its challenger for an encoding of either Sh_j^0 or Sh_j. Depending on the answer of the challenger, the distinguisher obtains either $\mathcal{H}_j$ or the previous hybrid, and can emulate $\mathcal{D}$ with the hybrid (and answer as $\mathcal{D}$ does) to obtain the same distinguishing advantage.

$\mathcal{H}_2$**:** In $\mathcal{H}_2$, we replace $\mathbf{View}_T\left(s^0\right)$ with $\mathbf{View}_T\left(s^1\right)$. Then $\mathsf{SD}(\mathcal{H}_1,\mathcal{H}_2)\leq\varepsilon$ due to the privacy of VSS, because $\mathsf{leak}(\mathsf{Sh}_1,\ldots,\mathsf{Sh}_m)$ is independent of s^0,s^1 and can be fixed into the distinguisher.

$\mathcal{H}_3$**:** In $\mathcal{H}_3$, we replace $\mathsf{leak}(\mathsf{Sh}_1,\ldots,\mathsf{Sh}_m)$ with $\mathsf{leak}\left(\mathsf{Sh}_1^1,\ldots,\mathsf{Sh}_m^1\right)$, where $\left(\mathsf{Sh}_1^1,\ldots,\mathsf{Sh}_m^1\right)$ is a random secret sharing of s^1. Then $\mathsf{SD}(\mathcal{H}_0,\mathcal{H}_1)\leq(m-t)\,\varepsilon'$ using the same arguments used to show that $\mathsf{SD}(\mathcal{H}_0,\mathcal{H}_1)\leq(m-t)\,\varepsilon'$. We note that $\mathcal{H}_3$ is the joint distribution of $\mathbf{View}_T\left(s^1\right)$ and $\mathsf{leak}\left(\mathsf{Sh}_1^1,\ldots,\mathsf{Sh}_m^1\right)$ in an execution with secret s^1.

■

5.2 Distributed ZK (dZK)

In this section we apply our paradigm to distributed ZK proofs, obtaining proofs which are leakage- and tampering-resilient. We first introduce the setting of distributed ZK proofs, following mostly the terminology of [BBC+19b].

The setting of distributed ZK proofs. Distributed ZK proofs are a multi-verifier variant of zero-knowledge interactive proofs. Recall that in standard interactive proofs, a prover P tries to

TR and LR-VSS

Building blocks: an (m,t)-VSS scheme VSS and an RPE scheme $(\mathsf{Encode}, \mathsf{Decode}, \mathsf{Rec})$.

The sharing phase: the dealer D, on input a secret s emulates the sharing procedure of VSS on input s to obtain secret shares $\mathsf{Sh}_1, \ldots, \mathsf{Sh}_m$, and sends Sh_i to P_i.

The verification phase: party P_i, with share Sh_i obtained from the sharing phase:

- Emulates party P_i in the verification phase of VSS with Sh_i. Let $\widetilde{\mathsf{Sh}}_i^{\mathsf{in}}$ denote the updated share which P_i obtains at the end of this emulation.
- Encodes $\widetilde{\mathsf{Sh}}_i \leftarrow \mathsf{Enc}\left(\widetilde{\mathsf{Sh}}_i^{\mathsf{in}}\right)$.
- Erases all information obtained during the emulation, except for $\widetilde{\mathsf{Sh}}_i$.
- Outputs $\widetilde{\mathsf{Sh}}_i$ as its updated share.

The reconstruction phase: on input $\widetilde{\mathsf{Sh}}_1, \ldots, \widetilde{\mathsf{Sh}}_m$, the reconstruction procedure:

- Decodes $\widetilde{\mathsf{Sh}}_i^{\mathsf{in}} = \mathsf{Dec}\left(\widetilde{\mathsf{Sh}}_i\right)$.
- Computes $s^* = \mathsf{Reconst}\left(\widetilde{\mathsf{Sh}}_1^{\mathsf{in}}, \ldots, \widetilde{\mathsf{Sh}}_m^{\mathsf{in}}\right)$, where $\mathsf{Reconst}$ is the reconstruction procedure of VSS.
- Outputs s^*.

Figure 3: TR abd LR-VSS

convince a verifier that some input x is in some language L. In the distributed setting, the single verifier is replaced with k verifiers $V_1, \ldots, V_k$, and the input x is distributed between the verifiers, where V_i holds a *piece* x^i such that $x = x^1 \circ \ldots \circ x^k$ (where $\circ$ can denote concatenation or addition, depending on the concrete implementation). The language L is replaced with its *k-distributed* version $L_k = \left\{\left(x^1, \ldots, x^k\right) : x \in L\right\}$, and a relation $\mathcal{R}(x,w)$ is replaced with its k-distributed version $\mathcal{R}_k\left(x^1, \ldots, x^k, w\right)$. We use $L(\mathcal{R})$ to denote the language corresponding to $\mathcal{R}$ (this notation can be used also for k-distributed relations and languages).

Definition 5.3 (k-Distributed Interactive Proofs (k-dIP))**.** *Let $\mathbb{F}$ be a field. A k-distributed Interactive Proof (k-dIP) consists of a prover P and verifiers $V_1, \ldots, V_k$ satisfying the following.*

1. *The input of V_i is a piece x^i, and the input of P is $x = x^1 \circ \ldots \circ x^k$.*
2. *The parties interact in rounds, where the messages sent by a party in round i are determined given a next-message function, and depend on its input, randomness, and messages it received in previous rounds.*

   More specifically, the computation can be divided into phases,[27] *where each phase has the following structure. First, the prover shares some proof string between the verifiers by sending*

[27] The notion of distributed ZK proofs was defined more generally in [BBC+19b]. However, the structure which we describe here will allow us to obtain a more meaningful LR guarantee. We note that the main protocols of [BBC+19b] (specifically, the protocols of [BBC+19b, Theorem 6.6] and [BBC+19b, Theorem 6.10]) have the structure we describe here. We also note that some other dZK proofs can be seen as consisting of several "epochs", where each epoch consists of several phases as described above.

a message to each verifier.[28] *This is followed by several rounds in which all parties can interact, but during which the messages exchanged are completely independent of the inputs of the parties, and the previous messages they have received. (For example, these rounds can be used to execute a coin-tossing sub-protocol.) Then, each verifier performs a local computation on its secret shares and the outcome it received from the previous rounds of the phase. Following the final phase, the verifiers can again exchange messages to determine whether to accept or reject.*

3. *The protocol terminates after a fixed number of rounds, and each verifier outputs either accept or reject, based on its view (which consists of its input and the messages it received throughout the execution).*

Next, we define the notion of distributed *ZK* proofs. These proofs are executed in the presence of a malicious and rushing adversary, and come in two flavors, depending on whether the adversary is allowed to simultaneously corrupt the prover *and* a subset of verifiers.

Definition 5.4 (Distributed ZK (dZK): malicious prover *or* verifiers)**.** *Let $\mathcal{R}_k\left(x^1,\ldots,x^k,w\right)$ be a k-distributed relation over a finite field $\mathbb{F}$. A k-dIP $\Pi=(P,V_1,\ldots,V_k)$ for $\mathcal{R}_k$ is a* k-distributed ZK proof (k-dZK) for $\mathcal{R}_k$ against malicious prover *or t* verifiers *with error ε, if Π satisfies the following properties.*

- **Completeness.** *For every $\left(x^1,\ldots,x^k,w\right)\in\mathcal{R}_k$, all verifiers accept in the execution of Π on $\left(x^1,\ldots,x^k,w\right)$ with probability 1.*
- **Soundness (malicious prover *or* verifiers).** *For every (possibly malicious and unbounded) prover P^*, and any $x^1\circ\ldots\circ x^k\notin L\left(\mathcal{R}_k\right)$, except with probability ε all verifiers reject in the execution of Π with prover P^* and input $x^1\circ\ldots\circ x^k$.*
- **Distributed Zero-Knowledge (dZK).** *For every adversary $\mathcal{A}$ corrupting a subset T of at most t verifiers, there exists a simulator Sim such that for every $\left(x^1,\ldots,x^k,w\right)\in\mathcal{R}_k$, it holds that*

$$SD\left(\mathsf{Sim}\left(\left(x^j\right)_{j\in T}\right),\mathbf{View}_{\Pi,\mathcal{A}}\left(x^1,\ldots,x^k,w\right)\right)\leq\varepsilon$$

where $\mathbf{View}_{\Pi,\mathcal{A}}\left(x^1,\ldots,x^k,w\right)$ denotes the view of the adversary $\mathcal{A}$ in an execution of Π on inputs $x^1,\ldots,x^k,w$.[29]

Definition 5.5 (Distributed ZK (dZK): malicious prover *and* verifiers)**.** *Let $\mathcal{R}_k\left(x^1,\ldots,x^k,w\right)$ be a k-distributed relation over a finite field $\mathbb{F}$. A k-dIP $\Pi=(P,V_1,\ldots,V_k)$ for $\mathcal{R}_k$ is a* k-distributed ZK proof (k-dZK) for $\mathcal{R}_k$ against malicious prover *and t* verifiers *with error ε, if Π satisfies the completeness and dZK properties of Definition 5.4, as well as the following soundness property.*

- **Soundness (malicious prover *and* verifiers).** *For every adversary $\mathcal{A}$ corrupting P and a subset T of at most $t-1$ verifiers, and for any $x^1\circ\ldots\circ x^k\notin L\left(\mathcal{R}_k\right)$, the following holds. If there exists no $\left(x^{1\prime},\ldots,x^{k\prime}\right)\in L\left(\mathcal{R}_k\right)$ such that $x^i=x^{i\prime}$ for every $i\notin T$, then except with probability ε all honest verifiers reject in the execution of Π with adversary $\mathcal{A}$ and input $x^1\circ\ldots\circ x^k$.*

[28] We note that as in standard ZK proofs, the messages of the prover to the verifiers depend on the messages it received from the verifiers in previous phases. For example, in the protocols of [BBC+19b], in each phase the verifiers jointly toss random coins which are used as a random verification challenge, and affect the prover's message in the next phase.

[29] We note that [BBC+19b] define dZK proofs with *perfect* dZK, whereas we allow for a statistical simulation error. This is to explicitly analyze how the simulation error grows through our transformation. However, we note that our final construction will have perfect dZK (see Corollary 5.4).

dZK leakage model. Our leakage-resilient dZK proofs protect the proof shares - which the prover sends to the verifiers throughout the execution - against probing leakage. Similar to our previous constructions, the secret shares are protected when they are stored in memory, and throughout the lifetime of the protocol, and up until the verifiers exchange messages at the end of the protocol to decide whether to accept or reject. In particular, the secret shares are *not* protected when the verifiers locally compute on them at the end of each phase. We also assume that there is no leakage on the prover.[30]

More specifically, leakage-resilience is defined by comparing a real-world execution, in which the adversary $\mathcal{A}$ interacts with a challenger $\mathcal{C}$, to an ideal execution in which $\mathcal{A}$ interacts with a simulator Sim. In both executions, the challenger and simulator emulate the honest parties, whereas $\mathcal{A}$ can *adaptively* corrupt parties and assumes full control of corrupted parties. $\mathcal{A}$ can additionally adaptively probe proof shares of honest verifiers. This is formalized in the following definition.

Definition 5.6 (Leakage-Resilient dZK (LR-dZK))**.** *We say that a dZK proof Π is an ε-leakage-resilient dZK (LR-dZK) proof against (t,ℓ)-local probing leakage if for every semi-honest adversary $\mathcal{A}$ that can adaptively corrupt up to t verifiers, there exists a simulator Sim such that for every $(x^1,\ldots,x^k,w) \in \mathcal{R}_k$ it holds that $\mathsf{SD}(\mathcal{REAL}_{t,\ell},\mathcal{IDEAL}_{t,\ell}) \leq \varepsilon$, where $\mathcal{REAL}_{t,\ell},\mathcal{IDEAL}_{t,\ell}$ are defined as follows.*

- $\mathcal{REAL}_{t,\ell} = \left(\mathbf{View}_{\Pi,\mathcal{A}}\left(x^1,\ldots,x^k,w\right), \mathsf{leak}\left(\left(\pi_{j,l}\right)_{j\in[k],l\leq r}\right)\right)$ *where $\pi_{j,l}$ is the proof shares which the prover sent to V_j in phase l which were generated through the following process. $\mathcal{A}$ interacts with a challenger $\mathcal{C}$ who emulates the execution of the dZK proof together with $\mathcal{A}$. Throughout this emulation, $\mathcal{C}$ emulates the prover P with input $\left(x^1,\ldots,x^k,w\right)$, as well as all honest verifiers. During the emulation, $\mathcal{A}$ can repeatedly perform the following operations:*
  - *adaptively corrupt a verifier V_j by sending j to $\mathcal{C}$, in which case $\mathcal{C}$ sends to $\mathcal{A}$ the view of V_j up to that point of the emulation. Following the corruption of V_j, the adversary obtains full control of V_j, meaning it sends to $\mathcal{C}$ the messages of V_j to the honest parties, and $\mathcal{C}$ sends to $\mathcal{A}$ the messages which honest parties send to V_j. We let T denote the set of verifiers that were corrupted throughout this process.*
  - *make a leakage query to the i'th bit of the proof share of the j'th verifier in round l, by sending i,j,l to $\mathcal{C}$, to which $\mathcal{C}$ responds by sending the i'th bit of $\pi_{j,l}$ to $\mathcal{A}$, where $\pi_{j,l}$ denotes the proof share which V_j received from the prover in phase l. $\mathcal{A}$ is allowed to make up to ℓ distinct leakage queries to each $\pi_{j,l}$ ($\mathcal{C}$ does not answer any further queries).*

  Then $\mathbf{View}_{\Pi,\mathcal{A}}\left(x^1,\ldots,x^k,w\right)$ denotes the view of the parties in T in this process, r is the number of phases performed during the proof, and $\mathsf{leak}(\cdot)$ denotes the symbols which $\mathcal{A}$ adaptively *probed from the proof shares throughout the execution, where each proof share was probed in at most ℓ locations.*
- *$\mathcal{IDEAL}_{t,\ell}$ is the output of the simulator Sim in the following process. Sim interacts with $\mathcal{A}$, playing the role of the challenger, and answering its corruption and leakage queries as follows.*
  - *if $\mathcal{A}$ corrupts V_j, then Sim receives the input x_j of V_j. Then, it sends to $\mathcal{A}$ a simulated view of V_j up to that point of the emulation, and simulates any further messages that V_j receives.*

[30] We note that this assumption could be removed by having the prover encode the input and witness, and whenever it needs to perform some computations on these, it will decode, compute and reencode these values. However, assuming that the prover is leak-free results in a cleaner and clearer model, so we chose to present the model in this way.

– *if $\mathcal{A}$ makes a leakage query to the i'th bit of the share of the j'th verifier in round l, then* Sim *responds with a simulated bit.*

Remark 5.2 (On adaptive security of dZK proofs.)**.** *The security definition of dZK proofs of Boneh et al. [BBC+19b] (given in Definitions 5.4 and 5.5 above) assumes the adversary is non-adaptive. It can be naturally extended to allow for adaptive corruptions in the dZK property, a property which will be needed to achieve adaptive leakage resilience in our construction (Construction 4). As we show in Lemma A.1, the protocol of [BBC+19a, Theorem 6.6] is dZK even against adaptive corruptions. In what follows, we will assume by default that the dZK proof has dZK against adaptive adversaries. We note that we could also use the standard (i.e., non-adaptive) notion of dZK, in which case we would achieve a weaker LR guarantee which allows only the leakage queries to be adaptive (namely the adversary is restricted to non-adaptive corruption of verifiers).*

Tampering-resilient dZK. Our dZK proofs will be tampering-resilient in the following sense: if all parties are honest, then the verifiers will accept a true statement even if an external adversary can arbitrarily modify at most τ symbols in *every* proof share sent by the prover. This is formalized in the next definition:

Definition 5.7 (τ-tampering-resilience for dZK proofs)**.** *We say that a dZK proof Π is τ-tampering-resilient if for every $(x^1, \ldots, x^k, w) \in \mathcal{R}_k$, and any adversary $\mathcal{A}$ that can arbitrarily modify at most τ symbols in every proof share sent by the prover, all verifiers accept in the execution of Π on $(x^1, \ldots, x^k, w)$ with probability 1.*

Our Construction. Our scheme combines together a distributed ZK proof between a prover P and k verifiers $V_1, \ldots, V_k$, with an RPE scheme. The high-level idea is to RPE encode the input pieces $x^1, \ldots, x^k$, as well as to modify the phases of the protocol as follows. The prover will RPE encode the proof shares which it sends to the verifiers. These shares can remain encoded in the following rounds in which the parties interact, because the messages exchanged during those rounds are independent of the shares. Then, when the verifiers need to locally compute on their proof shares, each verifier will decode its share, perform the computation on it, and RPE encode the result. Additionally, each verifier will erase all the intermediate values in the computation on the decoded share (including the decoded share itself). Finally, when the final phase terminates, the verifiers can decode their shares to compute the messages they need to send.[31] This construction is formalized in Figure 4, and its properties are summarized in Theorem 5.3 below.

Theorem 5.3 (LR-dZK (malicious prover *or* verifiers))**.** *Let $\mathcal{R}$ be a k-distributed relation over a field $\mathbb{F}$. Assume that Construction 4 is instantiated with:*

- *A dZK proof for $\mathcal{R}$ between prover P and k verifiers $V_1, \ldots, V_k$ with security against the prover or t verifiers and error ε.*[32] *Let r denote the number of phases in this dZK protocol.*
- *An adaptive $(\varepsilon', \ell, \tau)$-RPE scheme.*

Then Construction 4 is a dZK proof for $\mathcal{R}$ with security against the prover or t verifiers, that is τ-TR and $\varepsilon + kr\varepsilon'$-LR for the class of (t, ℓ)-local probing leakage.

[31] We note that in some cases, e.g., in the protocol of [BBC+19b], this final rounds consist of each verifier sending to a single verifier all the values it locally computed throughout the protocol, and then that verifier broadcasting the result (accept of reject). In this case, the shares can remain encoded up until the designated verifier decodes them to compute whether to accept or reject.

[32] Recall that we assume the dZK property holds against *adaptive* adversaries.

The following corollary is obtained by instantiating the construction of Figure 4 with the dZK proofs of [BBC+19a, Theorem 6.6] (using Lemma A.1 for adaptive security) and the RPEs of Corollary 3.1.

Corollary 5.4 (Leakage-Resilient and Tampering-Resilient dZK proofs)**.** *For any k-distributed relation $\mathcal{R}_k$, any k-dZK proof Π for $\mathcal{R}_k$ can be transformed into a dZK proof Π' for $\mathcal{R}_k$ which is $\Omega(N)$-TR and LR for the class of $(t, \Omega(N))$-local probing leakage, where N denotes the length of proof shares in Π. Moreover, if Π has perfect dZK then Π' has perfect dZK, and the transformation only causes a constant blowup in the length of proof shares.*

Proof sketch (Theorem 5.3): We argue that Construction 4 satisfies the properties of Definition 5.4 and the additional leakage-resilience guarantee of Definition 5.6.

Completeness. Follows directly from a combination of the completeness of the underlying dZK and the correctness of the RPE scheme.

Soundness against a malicious prover or verifiers. Notice that the first round of each phase l determines *some* proof shares $\pi_{1,l}, \ldots, \pi_{k,l}$, which will be used throughout the protocol execution. These values could have been generated by the prover P of the underlying dZK protocol. (In particular, providing V'_j with an invalid RPE encoding - i.e., one for which decoding fails - corresponds to a prover strategy in the underlying dZK which gives $\vec{0}$ to V_j as its share in the lth phase.) Therefore, soundness reduces to the soundness of the underlying dZK system.

ZK. Follows directly from the ZK of the underlying dZK system, since the RPE encoding is applied separately to each share.

TR. Follows from the error-correction of the RPE (since each share is encoded separately), and from the completeness of the underlying dZK proof, using similar arguments to the proof of Theorem 4.1.

LR. Follows from a combination of the ZK of the underlying dZK system and the LR of the RPE scheme, using a hybrid argument. The simulator Sim' will run the simulator of the underlying dZK system to answer adaptive corruption queries, and answer leakage queries using random RPE encoding of the all-zero strings. We sketch the hybrids.

$\mathcal{H}_0$**:** This is the real distribution $\mathcal{REAL}_{t,\ell}$ which is the joint distribution of $\mathbf{View}_{\Pi,\mathcal{A}}\left(x^1, \ldots, x^k, w\right)$ and $\mathsf{leak}\left(\left(\widetilde{\pi}_{j,l}\right)_{j\in[k],l\leq r}\right)$.

$\mathcal{H}_1$**:** In $\mathcal{H}_1$, we consider a hybrid simulator Sim'' which obtains $x^1, \ldots, x^k, w$, and emulates the role of $\mathcal{C}$ in $\mathcal{REAL}_{t,\ell}$, with the only difference that it answers leakage queries using RPE encodings of the all-zero strings (exactly as Sim' does).

Notice that $\mathcal{H}_1$ is the distribution obtained from $\mathcal{H}_0$ by replacing $\mathsf{leak}\left(\left(\widetilde{\pi}_{j,l}\right)_{j\in[k],l\leq r}\right)$ with the leakage obtained by replacing each $\pi_{j,l}$ with the all-zeros string (and then randomly RPE encoding it). Then $\mathsf{SD}(\mathcal{H}_0, \mathcal{H}_1) \leq kr\varepsilon'$ from the adaptive secrecy of partial views of the RPE scheme, by a standard hybrid argument in which we replace the $\pi_{j,l}$ one at a time (where $x^1, \ldots, x^k, w$ are hardwired into the distinguisher, so it can generate the $\pi_{j,l}$'s on its own).[33]

[33] We note that if $\mathcal{A}$ is restricted to non-adaptive corruptions (but is allowed to obtain adaptive leakage on the shares of the honest verifiers) then we can bound the statistical distance here by $(k-t)\,r\varepsilon$. This is because in this case we can answer leakage queries to shares of the corrupted verifiers using the actual shares, instead of using encodings of 0. (This holds when $\mathcal{A}$ corrupts t verifiers, which is without loss of generality, since we can always generate the leakage on a particular proof share given the share in its entirety.)

$\mathcal{H}_2$: $\mathcal{H}_2$ is the ideal distribution $\mathcal{IDEAL}_{t,\ell}$. Notice that the output of $\mathsf{leak}(\cdot)$ is distributed identically in $\mathcal{H}_1, \mathcal{H}_2$, so $\mathsf{SD}(\mathcal{H}_1, \mathcal{H}_2) \leq \varepsilon'$ due to the ZK (against adaptive corruptions) of the underlying dZK system.

■

Theorem 5.5 (LR-dZK (malicious prover *and* verifiers))**.** *Let $\mathcal{R}$ be a k-distributed relation over a field $\mathbb{F}$. Assume that Construction 4 is instantiated with:*

- *A dZK protocol for $\mathcal{R}$ between prover P and k verifiers $V_1, \ldots, V_k$ with security against a coalition of the prover and $t-1$ malicious verifiers, and error ε.*[34] *Let r denote the number of phases in this dZK protocol.*
- *An adaptive $(\varepsilon', \ell, \tau)$-RPE scheme.*

Then Construction 4 is a dZK protocol for $\mathcal{R}$ with security against a coalition of the prover and $t-1$ *malicious verifiers, that is τ-TR and $\varepsilon + kr\varepsilon'$-LR for the class of (t, ℓ)-local probing leakage.*

Proof sketch: The only difference from the proof of Theorem 5.3 is in the soundness property, which we now explain.

Soundness against a malicious prover *and* verifiers. The first round of each phase l determines *some* proof shares $\pi'_{1,l}, \ldots, \pi'_{k,l}$, which will be used throughout the protocol execution. These values could have been generated by the prover P of the underlying dZK protocol, since providing an invalid RPE encoding in the RL-dZK proof corresponds to a prover strategy in the underlying dZK which gives the share $\vec{0}$. Moreover, each computation phase which depends on the proof shares, uses (the decoded shares) $\pi'_{1,l}, \ldots, \pi'_{k,l}$, where malicious verifiers can of course arbitrarily modify their shares. Therefore, soundness reduces to the soundness of the underlying dZK system. ■

5.3 Function Secret Sharing (FSS)

In this section we study is a leakage- and tampering-resilient variant of Function Secret Sharing (FSS). Loosely speaking, FFS is a cryptographic building block that enables evaluation of a function f in a distributed manner, and involves two algorithms $(\mathsf{Gen}, \mathsf{Eval})$. In the first phase secret keys (k_0, k_1) are generated within $\mathsf{Gen}(1^\kappa, f)$ and handed to each party.[35] In the second phase each party P_i is given an input x, and locally computes $\mathsf{Eval}(i, k_i, x)$ to obtain a function share $f_i(x)$ such that $f(x) = f_0(x) + f_1(x)$. The security requirement ensures that the individual keys do not disclose any information about the function description (to a computationally-bounded adversary). We now define FSS and its security more formally (taken verbatim from [BGI16]).

Definition 5.8 (FSS: Syntax.)**.** *An m-party* Function Secret Sharing (FSS) *scheme is a pair of algorithms* $(\mathsf{Gen}, \mathsf{Eval})$ *with the following syntax:*

- $\mathsf{Gen}(1^\kappa, \hat{f})$ *is a* PPT key generation *algorithm, which on input 1^κ (security parameter) and $\hat{f} \in \{0,1\}^*$ (a description of a function f) outputs an m-tuple of keys $(k_1, \ldots, k_m)$. We assume that $\hat{f}$ explicitly contains an input length 1^κ, group description $\mathbb{G}$, and size parameter T.*

[34] Recall that we assume the dZK property holds against *adaptive* adversaries.

[35] While FSS can be defined more broadly for any number of parties, current applications find practical constructions mostly in the two-party setting [BGI16, BCG+21]. We note that our results naturally extend to the multi-party setting.

LR-dZK

Building blocks: a k-dZK system with prover P and verifiers $V_1, \ldots, V_k$, and an RPE scheme $(\mathsf{Encode}, \mathsf{Decode}, \mathsf{Rec})$.

Input encoding:[a] we assume that for every j, x^j is encoded as $\widetilde{x}^j \leftarrow \mathsf{Encode}\left(x^j\right)$.

Protocol execution: the prover P' with input x and witness w, and the verifiers $V'_1, \ldots, V'_k$ with input pieces $\widetilde{x}^1, \ldots, \widetilde{x}^k$ interact in phases. Phase l of the protocol is carried out as follows:

- P' emulates P on x, w to obtain the shares $\pi_{1,l}, \ldots, \pi_{k,l}$ that P sends to $V_1, \ldots, V_k$ (respectively) in the lth phase.
- For every $j \in [k]$, P' encodes $\widetilde{\pi}_{j,l} \leftarrow \mathsf{Encode}\left(\pi_{j,l}\right)$, and sends $\widetilde{\pi}_{j,l}$ to V_j.
- The parties emulate $P, V_1, \ldots, V_k$ during the rounds of interaction of the lth phase. (We note that the shares can remain encoded throughout this process, because the messages exchanged during these rounds are independent of the shares.)
- Every $V'_j, j \in [k]$:
  - Decodes $\pi_{j,l} = \mathsf{Decode}\left(\widetilde{\pi}_{j,l}\right)$.[b] If decoding fails then V'_j sets $\pi_{j,l} = \vec{0}$.
  - Emulates V_j on $\pi_{j,l}$, and the messages received in previous rounds, to obtain an outcome $o_{j,l}$.
  - Encodes $\widetilde{o}_{j,l} \leftarrow \mathsf{Encode}\left(o_{j,l}\right)$, and erases $\pi_{j,l}, o_{j,l}$ and all intermediate values generated during the computation of $o_{j,l}$.

After the final phase of the protocol, the verifiers interact to determine the outcome of the computation. At this point the verifiers may decode any value computed previously, as needed.[c]

[a] This will be executed by the party which distributes the input between the verifiers, which could be the prover or some trusted external party.

[b] We note that the computation in this round might depend on $\pi_{j,l'}$ or $o_{j,l'}$ ($o_{j,l'}$ is defined in the following steps) for some $l' \leq l$, in which case V'_j decodes them too.

[c] We note that in the protocols of [BBC+19a, Theorem 6.6] and [BBC+19a, Theorem 6.10], the final rounds consist of all verifiers sending the outcome of the last phase to V_1, and V_1 broadcasting the result. In this case, there is no need for decoding, the verifiers can send the encoded outcomes to V_1, who will locally decode them.

Figure 4: LR-dZK

- $\mathsf{Eval}(i, k_i, x)$ *is a* PPT evaluation algorithm, *which on input* $i \in [m]$ *(party index),* k_i *(key defining* $f_i : \{0,1\}^\kappa \mapsto \mathbb{G}$*) and* $x \in \{0,1\}^\kappa$ *(input for* f_i*) outputs a group element* $y_i \in \mathbb{G}$ *(the value of* $f_i(x)$*, the* i*th share of* $f(x)$*).*

When m is omitted, it is understood to be 2, in which case we sometimes index the parties by $i \in \{0,1\}$ rather than $i \in \{1,2\}$. A function family is defined by a pair $\mathcal{F} = (P_\mathcal{F}, E_\mathcal{F})$, where $P_\mathcal{F} \subseteq \{0,1\}^*$ is an infinite collection of function descriptions $\hat{f}$, and $E_\mathcal{F} : P_\mathcal{F} \times \{0,1\}^* \mapsto \{0,1\}^*$ is a polynomial-time algorithm defining the function described by $\hat{f}$. Concretely, each $\hat{f} \in P_\mathcal{F}$ describes a corresponding function $f : D_f \mapsto R_f$ defined by $f(x) = E_\mathcal{F}(\hat{f}, x)$.

Definition 5.9 (FSS: Security.). *Let* $\mathcal{F} = (P_\mathcal{F}, E_\mathcal{F})$ *be a function family and* $\mathsf{Leak} : \{0,1\}^* \mapsto \{0,1\}^*$ *be a function specifying the allowable leakage. Let* m *(number of parties) and* t *(secrecy threshold) be positive integers. An* m*-party* t*-secure* FSS *for* $\mathcal{F}$ *with leakage* Leak *is a pair* $(\mathsf{Gen}, \mathsf{Eval})$ *as in Definition 5.8, satisfying the following requirements*

- **Correctness:** *For all* $\hat{f} \in P_{\mathcal{F}}$ *describing* $\hat{f} : \{0,1\}^\kappa \mapsto \mathbb{G}$, *and every* $x \in \{0,1\}^\kappa$, *if* $(k_1, \ldots, k_m) \leftarrow \mathsf{Gen}(1^\kappa, \hat{f})$ *then* $\Pr[\sum_{i=1}^m \mathsf{Eval}(i, k_i, x) = f(x)] = 1$.
- **Secrecy:** *For every subset* $T \subset [m]$ *of size* $|T| \leq t$, *there exists a* PPT *algorithm* Sim *(simulator), such that for every sequence* $\hat{f}_1, \hat{f}_2, \ldots$ *of polynomial-size function descriptions from* $P_{\mathcal{F}}$, *the outputs of the following experiments* Real *and* Ideal *are computationally indistinguishable:*
  - $\mathsf{Real}(1^\kappa)$: $(k_1, \ldots, k_m) \leftarrow \mathsf{Gen}(1^\kappa, \hat{f}_\kappa)$;
    Output $\{k_i\}_{i \in T}$.
  - $\mathsf{Ideal}(1^\kappa)$;
    Output $\mathsf{Sim}(1^\kappa, \mathsf{Leak}(\hat{f}_\kappa))$.

When Leak is omitted, it is understood to be the function $\mathsf{Leak}(\hat{f}) = (1^\kappa, S_{\hat{f}}, \mathbb{G})$ where $1^\kappa, S_{\hat{f}}$, and $\mathbb{G}$ are the input length, size,[36] and group description contained in $\hat{f}$.

FSS leakage model. In this work we construct leakage-resilient FSS that protects function keys and output shares against probing. Recall that our security model protects these keys during periods of time in which they are stored in the memory of the servers, but not when they are used for computation and evaluation. As opposed to other security notions considered in this work (e.g., VSS and dZK described in Sections 5.1 and 5.2, respectively), our LR-FSS definition requires considering computationally-bounded adversaries, because standard FSS has computational security. We therefore define admissible adversaries by decoupling the (computationally unbounded) adversary that leaks from the honest parties' shares from the (computationally bounded) adversary that attacks the underlying FSS, maintaining a stronger security property for the former.

Definition 5.10 (Admissible adversaries (FSS))**.** *An admissible adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *is a pair of algorithms* $(\mathcal{A}_1, \mathcal{A}_2)$ *such that* $\mathcal{A}_1$ *is computationally unbounded whereas* $\mathcal{A}_2$ *is* PPT.

Next, we introduce our leakage-resilient FSS definition. Intuitively speaking, our definition compares a real-world experiment with an ideal-world experiment. In both experiments $\mathcal{A}_1$ is allowed to adaptively leak on the honest parties' key and output shares via the oracle $\mathcal{LEAK}$, where a leakage query consists of a pair of leakage functions g, g' to be applied to the key and output shares, respectively. $\mathcal{A}_2$ is allowed to adaptively corrupt parties by calling the $\mathcal{COR}$ oracle, in which case it receives the key shares of the corrupted parties. The adversary must make all leakage queries before making any corruption queries. As noted above, this is necessary to obtain information-theoretic security against leakage queries while using a computationally-secure primitive such as FSS. To ensure consistency between repeated leakage queries, we maintain a table $\mathcal{O}$ containing, for each queried input x, all randomness required for sampling the output shares in the real execution (such a list is not needed in the ideal experiment, since it can be maintained in the simulator's state). The adversary is restricted to corrupting at most t parties, and leaking at most ℓ bits from the key share, and *all* output shares, of each party. (See Remark 5.3 below for a discussion of a stronger LR guarantee for output shares, which our construction achieves.) This is enforced by the check performed before the experiments terminate, where if the adversary violated these restrictions, then its guess b is disregarded.

Definition 5.11 (Leakage-Resilient FSS (LR-FSS))**.** *Let* $\mathsf{FSS} = (\mathsf{Gen}, \mathsf{Eval})$ *be an FSS scheme. We say that* FSS *is* Leakage-Resilient (LR) for the class of (t, ℓ)-local probing leakage *if for every* PPT

[36] The exact meaning of the size parameter depends on the computational model in use. For instance, the size of Boolean circuits is typically the number of the gates in the circuit.

adversary $\mathcal{A}$ corrupting at most t parties there exists an efficient simulator $(\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3, \mathsf{Sim}_4)$ and a negligible function $\varepsilon(\kappa)$ such that

$$\left|\Pr\left[\mathcal{REAL}_{\ell,t}\left(1^\kappa\right)=1\right]-\Pr\left[\mathcal{IDEAL}_{\ell,t}\left(1^\kappa\right)=1\right]\right|\leq\varepsilon(\kappa)$$

where $\mathcal{REAL}_{\ell,t}\left(1^\kappa\right), \mathcal{IDEAL}_{\ell,t}\left(1^\kappa\right)$ are defined in Figure 5, and the probability is over the random coin tosses of $\mathcal{SETUP}^{\mathcal{R}}$, Eval, $\mathcal{A}$, and $(\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3, \mathsf{Sim}_4)$.

Remark 5.3 (Better leakage bound for output shares)**.** *Notice that we allow the adversary in Figure 5 to obtain leakage on the output shares, for an unbounded number of inputs x.*[37] *However, for every party $i \in [m]$, we allow only ℓ leakage bits* in total *on* all *output shares $f_i(x)$ generated throughout the experiment. This was done to simplify the definition, but we stress that our construction will obtain a stronger leakage guarantee, which allows the adversary to leak ℓ bits* from every output share*. For example, if during the execution the adversary calls $\mathcal{LEAK}$ with two different inputs x, x^*, resulting in two sets of output shares $f_1(x), \ldots, f_m(x)$ and $f_1(x^*), \ldots, f_m(x^*)$ (respectively), then leakage resilience is guaranteed even if the adversary leaks ℓ bits from $f_i(x)$* and *ℓ (possibly different) bits from $f_i(x^*)$, for every i. We chose to present the weaker version in Figure 5 to keep the definition simpler and clearer.*

Remark 5.4 (Adaptive corruptions)**.** *Our definition supports adaptive corruptions, so to satisfy it, our construction will require that the underlying FSS is also adaptively secure. Specifically, we need the underlying FSS to have an* adaptive *secrecy guarantee which is stronger than the secrecy of Definition 5.9. Roughly, in adaptive secrecy the experiments* $\mathsf{Real}, \mathsf{Ideal}$ *of Definition 5.9 are modified to be a game between the adversary and a challenger or simulator (respectively), where the adversary can adaptively ask for key shares (by sending a party index i to the challenger or simulator) and obtains the corresponding function key. Secrecy is required to hold so long as the set of shares the adversary obtains throughout the process is of an unauthorized set. We note that current schemes from the literature are analyzed in the presence of static corruptions, but could possibly be compiled into the adaptive setting using known secure computation techniques. We leave such adaptations to future work.*

We stress that we could also make due with statically-secure FSS schemes (i.e., ones satisfying the standard static-secrecy property of Definition 5.9), by considering a weaker security definition for LR-FSS in which corruptions are fixed at the onset of the protocol, but the adversary can still adaptively leak on the shares of the honest parties.

Remark 5.5 (Enhancing FSS security)**.** *We can enhance the security definition of (both standard and LR) FSS by adding an additional $\mathcal{REV}$ oracle, allowing the adversary to obtain the function shares of honest parties (for instance, this is useful in applications where all parties are required to learn the output of the function). In this case, the FSS simulator needs to simulate the honest parties' function shares as well. In standard FSS, this can easily be done because the output shares form a (fresh) additive sharing of the output. We can also extend LR-FSS to support $\mathcal{REV}$ queries by having the leakage-resilience simulator sample a random additive sharing of the output subject to the shares of the corrupted parties, and resample RPE-encodings of the freshly-sampled shares consistently with the previous leakage.*

[37] When using RPEs with a statistical error ε, the statistical error will accumulate over calls to $\mathcal{LEAK}$ with different inputs, however, if the RPE is perfect (as is the case in the RPE used in this work, see Corollary 3.1) the number of inputs does not affect the quality of simulation (i.e., the error does not accumulate, since there is no simulation error). See also Remark 5.6 below.

LR FSS

$\underline{\mathcal{SETUP}^{\mathcal{R}}(1^\kappa)}$:

pick a random string r for Gen[a]

$\underline{\mathcal{SETUP}^{\mathcal{I}}\left(1^\kappa, \mathsf{Leak}(\hat{f}_\kappa)\right)}$:

$\mathsf{St} \leftarrow \mathsf{Sim}_1\left(1^\kappa, \mathsf{Leak}(\hat{f}_\kappa)\right)$

$\underline{\mathcal{LEAK}^{\mathcal{R}}(r, g, g', x)}$:

$(k_1, \ldots, k_m) \leftarrow \mathsf{Gen}(1^\kappa, \hat{f}_\kappa; r)$:
if $(x, \cdot, \ldots, \cdot) \notin \mathcal{O}$
  then pick random strings $r_1^x, \ldots, r_m^x$
  else let $(x, r_1^x, \ldots, r_m^x) \in \mathcal{O}$
for every $i \in [m]$, $f_i(x) \leftarrow \mathsf{Eval}(i, k_i, x; r_i^x)$
$(\mathsf{output}_i)_{i\in[m]} \leftarrow g\left((k_i)_{i\in[m]}\right)$
$(\mathsf{output}'_i)_{i\in[m]} \leftarrow g'\left((f_i(x))_{i\in[m]}\right)$
$\mathcal{O} \leftarrow \mathcal{O} \cup \{(x, r_1^x, \ldots, r_m^x)\}$
for every $i \in [m]$
  $\ell_i \leftarrow \ell_i + |\mathsf{output}_i|$
  $\ell'_i \leftarrow \ell'_i + |\mathsf{output}'_i|$
output $\left((\mathsf{output}_i, \mathsf{output}'_i)_{i\in[m]}\right)$

$\underline{\mathcal{LEAK}^{\mathcal{I}}(\mathsf{St}, g, g', x)}$:

$\left((\mathsf{output}_i)_{i\in[m]}, \mathsf{St}\right) \leftarrow \mathsf{Sim}_2(\mathsf{St}, g)$
$\left((\mathsf{output}'_i)_{i\in[m]}, \mathsf{St}\right) \leftarrow \mathsf{Sim}_3(\mathsf{St}, g', x)$
for every $i \in [m]$
  $\ell_i \leftarrow \ell_i + |\mathsf{output}_i|$
  $\ell'_i \leftarrow \ell'_i + |\mathsf{output}'_i|$
output $\left((\mathsf{output}_i, \mathsf{output}'_i)_{i\in[m]}\right)$

$\underline{\mathcal{COR}^{\mathcal{R}}(r, T)}$:

$T_1 \leftarrow T_1 \cup T$
output $(k_i)_{i\in T}$

$\underline{\mathcal{COR}^{\mathcal{I}}(\mathsf{St}, T)}$:

$T_1 \leftarrow T_1 \cup T$
$(k'_i)_{i\in T} \leftarrow \mathsf{Sim}_4(\mathsf{St}, T)$
output $(k'_i)_{i\in T}$

$\underline{\mathcal{REAL}_{\ell,t}(1^\kappa)}$:

$\ell_1, \ell'_1, \ldots, \ell_m, \ell'_m \leftarrow 0$
$T_1, \mathcal{O} \leftarrow \emptyset$
$\mathcal{SETUP}^{\mathcal{R}}(1^\kappa)$
$\mathsf{St}^{\mathcal{A}} \leftarrow \mathcal{A}_1^{\mathcal{LEAK}^{\mathcal{R}}(r,\cdot,\cdot,\cdot)}()$
$b \leftarrow \mathcal{A}_2^{\mathcal{COR}^{\mathcal{R}}(r,\cdot)}(\mathsf{St}^{\mathcal{A}})$
if $\ell_i, \ell'_i \le \ell$ for every $i \in [m]$ and $|T_1| \le t$
  then output b
  else output 0

$\underline{\mathcal{IDEAL}_{\ell,t}\left(1^\kappa, \mathsf{Leak}(\hat{f}_\kappa)\right)}$:

$\ell_1, \ell'_1, \ldots, \ell_m, \ell'_m \leftarrow 0$
$T_1, \mathcal{O} \leftarrow \emptyset$
$\mathsf{St} \leftarrow \mathcal{SETUP}^{\mathcal{I}}\left(1^\kappa, \mathsf{Leak}(\hat{f}_\kappa)\right)$
$\mathsf{St}^{\mathcal{A}} \leftarrow \mathcal{A}_1^{\mathcal{LEAK}^{\mathcal{I}}(\mathsf{St},\cdot,\cdot,\cdot)}()$
$b \leftarrow \mathcal{A}_2^{\mathcal{COR}^{\mathcal{I}}(\mathsf{St},\cdot)}(\mathsf{St}^{\mathcal{A}})$
if $\ell_i, \ell'_i \le \ell$ for every $i \in [m]$ and $|T_1| \le t$
  then output b
  else output 0

[a] We consider the randomness r, the table $\mathcal{O}$, the simulator state St, the set T_1 of corrupted parties, and the counters ℓ_i, ℓ'_i as global variables that the relevant oracles can freely access and update.

Figure 5: The Security Experiments of LR FSS

Our Construction. Our modified key generation algorithm is similar to the sharing phase described in Figure 2, where we generate the function keys using Gen and then apply an RPE encoding separately to each share. Next, whenever the servers need to run the Eval algorithm, each server locally decodes its key, computes $y_i = \mathsf{Eval}(i, k_i, x)$, RPE encodes y_i, and then deletes the decoded key and y_i upon completing the evaluation. Recall that according to our leakage model, the decoding and evaluation phases are leak-free. y_i remains encoded for the remaining lifetime of this process,

and up until the clients reconstruct $f(x)$. Importantly, due to the linearity of the underlying reconstruction algorithm, the clients can remotely operate on the encoded output shares y_i, and only decode the outcome $f(x)$. Thus, function and output shares can still be aggregated as required for FSS applications.

We provide the formal construction in Figure 6 for a one-time invocation of the FSS algorithms. Our construction naturally extends to multiple executions of both algorithms. This requires another aggregation step where each server computes the sum $\sum_j \tilde{y}_i^j$ of the encoded function shares.

Theorem 5.6 (Leakage-Resilient and Tampering-Resilient FSS)**.** *Assume that Construction 6 is instantiated with:*

- *An m-party t-secure FSS scheme* $\mathsf{FSS} = (\mathsf{Gen}, \mathsf{Eval})$.
- *An adaptive $(\varepsilon, \ell, \tau)$-RPE scheme.*

Then the construction defined in Figure 6 is an m-party t-secure FSS which is τ-TR and is also leakage-resilient for the class of (t, ℓ)-local probing leakage, with

$$\left|\Pr\left[\mathcal{REAL}_{\ell,t}\left(1^\kappa\right)=1\right]-\Pr\left[\mathcal{IDEAL}_{\ell,t}\left(1^\kappa\right)=1\right]\right| \leq (h+1)m\varepsilon+\delta(\kappa)$$

where $\mathcal{REAL}_{\ell,t}, \mathcal{IDEAL}_{\ell,t}$ were defined in Figure 5, h is the number of distinct inputs to the FSS on which the adversary called the $\mathcal{LEAK}$ oracle, and $\delta(\kappa)$ is the secrecy error of FSS.

Remark 5.6 (LR-FSS for an unbounded number of input queries)**.** *The error in Theorem 5.6 grows with the number h of inputs x the adversary queried from the $\mathcal{LEAK}$ oracle, which requires some a-priori bound on h. However, we note that this bound can be removed by employing* perfect *RPEs, such as the RPE of Corollary 3.1.*

The following corollary follows immediately from Theorem 5.6 by instantiating the Construction of Figure 6 with the RPE of Lemma 3.9.

Corollary 5.7 (Leakage-resilient and tampering-resilient FSS)**.** *Let $\mathbb{F}$ be a field. Any m-party t-secure FSS* FSS *over $\mathbb{F}$ can be transformed into an m-party t-secure FSS* FSS' *over $\mathbb{F}$ that is $\Omega(N)$-TR and LR for the class of $(t, \Omega(N))$-local probing leakage, where the LR simulation error is exactly the secrecy error of* FSS*, and N is the length of key and output shares in* FSS*. Moreover, this transformation only causes a constant blowup in the key and output share sizes. Furthermore, if* FSS *has additive reconstruction over $\mathbb{F}$, then* FSS' *has linear reconstruction over $\mathbb{F}$.*

Remark 5.7. *We note that if one does not care to preserve linear reconstruction, then one can instantiate Theorem 5.6 with the RPEs of Corollary 3.1, resulting in a LR-FSS scheme with the properties specified in the corollary (except for linear reconstruction). Thus, one can obtain a LR-FSS scheme from* any *FSS scheme. (This should be contrasted with the LR-FSS scheme of Corollary 5.7, in which the (original) FSS scheme and the RPE are required to be over the same field.)*

Proof sketch (of Theorem 5.6): We argue that the construction defined in Figure 6 satisfies the properties of Definition 5.9, the leakage-resilience guarantee of Definition 5.11, and the tamper-resilience property of Definition 2.7.

Correctness. Follows directly from a combination of the correctness of the underlying FSS and RPE schemes.

Secrecy. Follows directly from the privacy of FSS, since the RPE encoding is applied separately to each key share. Specifically, the simulator can run the FSS simulator to simulate the shares, then

honestly RPE encode each share to obtain simulated encoded shares. More specifically, the flavor of secrecy – namely static or adaptive – is inherited from the underlying FSS. This is proved via a reduction to the security of the underlying FSS as follows. In the reduction, the shares are generated according to the real or simulated experiments, where in the static setting these are generated at the onset of the experiment, whereas in the adaptive setting they are generated on the fly. The shares are then RPE-encoded and handed to the distinguisher.

TR. Follows from the error-correction of the RPE (since each share is encoded separately), and the correctness of FSS, using similar arguments to the proof of Theorem 4.1. We note that in the FSS setting, TR holds against adversaries that can only change τ bits in each of the shares (but do not corrupt parties).

LR. Follows from a combination of the secrecy of FSS, the adaptive secrecy of partial views of the RPE, and the adaptive reconstruction* from partial views of the RPE, using a sequence of hybrid distributions. Specifically, the adversary is leaking from a single set of key shares, and h sets of output shares. The simulator will answer leakage queries using RPE-encodings of $\vec{0}$. When the adversary corrupts parties, the simulator will use the FSS simulator to simulate the key shares, and will use the resampling algorithm Rec of the RPE scheme to sample RPE-encodings of the simulated key shares which are consistent with the leakage. (Here, whether or not the adversary can adaptively corrupt parties depends on the flavor of security of the underlying FSS scheme.) Indistinguishability between the real and the simulated executions can be shown via a sequence of hybrids, as we now explain. In the following, we assume that the adversary corrupts at most t parties, and obtains at most ℓ leakage bits from every RPE encoding. This is without loss of generality, since otherwise both experiments output 0.

Let $\mathcal{H}_0$ denote the real execution of the LR-FSS. The next hybrid $\mathcal{H}_1$ is obtained from $\mathcal{H}_0$ by (1) answering leakage queries on key shares using RPE-encodings of $\vec{0}$, and (2) answering corruption queries by using the resampling algorithm Rec of the RPE to resample a fresh RPE-encoding of the key share, consistently with the leakage. We stress that $\mathcal{H}_1$ encodes *the same values* as in $\mathcal{H}_0$, so (apart from the leakage) the only difference between the hybrids is in the *RPE encoding itself*. Then $\mathsf{SD}(\mathcal{H}_0, \mathcal{H}_1) \leq m\varepsilon$ by the adaptive secrecy of partial views, and the adaptive reconstruction* from partial views of the RPE. Specifically, we define a sequence of hybrids where in the i'th hybrid we replace only the RPE-encodings of the first i key shares. When comparing hybrid $i-1$ with hybrid i, if the ith key share was not revealed by a corruption query, then the two hybrids are ε-statistically close by the adaptive secrecy of partial views, otherwise they are ε-statistically close by the adaptive reconstruction* from partial views. In particular, in the reduction to the secrecy/reconstruction of RPE, the entire hybrid distribution can be generated given the leakage (and, if the ith party is corrupted, the encoding) of the ith key. We note that in this reduction, all output shares of the ith party – and leakage on them – are generated using the *actual* values of these output shares (as in the real execution).

Next, we define $\mathcal{H}_2$ which is obtained from $\mathcal{H}_1$ by answering leakage queries on output shares using RPE-encodings of $\vec{0}$. (In particular, in $\mathcal{H}_2$ all leakage queries – on key or output shares – are answered according to RPE-encodings of $\vec{0}$.) Then $\mathsf{SD}(\mathcal{H}_0, \mathcal{H}_1) \leq hm\varepsilon$ by the adaptive secrecy of partial views of the RPE. Specifically, we order the (at most) hm output shares generated throughout the experiment in some arbitrary order, and define a sequence of hybrids where in the ith hybrid we switch the leakage (from real to leakage on encodings of $\vec{0}$) only in the first i shares. When comparing hybrid $i-1$ with hybrid i, the only difference is in leakage on the RPE-encoding of the ith output share, and so indistinguishability follows from the adaptive secrecy of partial views of the RPE. (Indeed, output shares are never revealed in full throughout the execution. In particular, we stress that even if the ith output share belongs to some corrupted party j, the adversary can compute –

Leakage-Resilient and Tampering-Resilient FSS

Building blocks: an m-party FSS scheme $(\mathsf{Gen}, \mathsf{Eval})$ and an RPE scheme $(\mathsf{Encode}, \mathsf{Decode}, \mathsf{Rec})$.

Keys generation: on input a security parameter 1^κ:

- Computes $(k_1, \ldots, k_m) \leftarrow \mathsf{Gen}\,(1^\kappa)$.
- For every $i \in [m]$, encodes $\tilde{k}_i \leftarrow \mathsf{Encode}\,(k_i)$.
- Outputs the key shares $\left(\tilde{k}_1, \ldots, \tilde{k}_m\right)$.

Evaluation: on a key share $\tilde{k}_i, i \in [m]$ and an input x:

- Decodes $k_i = \mathsf{Decode}(\tilde{k}_i)$.
- Evaluates $f_i(x) \leftarrow \mathsf{Eval}\,(i, k_i, x)$.
- Encodes $\tilde{f}_i(x) \leftarrow \mathsf{Encode}\,(f_i(x))$.
- Erases k_i and $f_i(x)$, together with all intermediate computations except for $\tilde{f}_i(x)$.
- Outputs $\tilde{f}_i(x)$.

Figure 6: LR and TR FSS

from party j's key share – the corresponding output share *in the original FSS* FSS, however since output shares of FSS are randomly RPE-encoded, the adversary does not learn the *actual* output share in full.)

Next, we define $\mathcal{H}_3$ that is identical to the simulated execution as defined above. The only difference between $\mathcal{H}_2, \mathcal{H}_3$ is in how corruption queries are answered – in $\mathcal{H}_2$ they are answered by resampling an RPE-encoding of the *actual* key shares, whereas in $\mathcal{H}_3$ they are resampled using *simulated* key shares. Then $\mathcal{H}_2$ and $\mathcal{H}_3$ are computationally indistinguishable by the secrecy of the underlying FSS, through the following reduction. The reduction emulates the leakage on RPE encodings of $\vec{0}$ (which, in particular, are independent of the honest parties' real shares). When it receives a corrupt query, it forwards this query to its oracle receiving back the key shares for the corrupted parties, for which it honestly resamples the RPE-encodings (using Rec) consistently with the leakage, and hands these shares to the adversary. Finally, it returns the bit b output by the adversary. (We note that the leakage is – always! – adaptively simulated, whereas the corruption pattern is either static or adaptive, depending on the permissible corruption strategies in the underlying FSS.) ■

5.4 Threshold Cryptography

In this section we study the usefulness of our paradigm in the context of threshold cryptography [DH76, LN18, FLOP18, HMR+19] where the secret key underlying some cryptographic task (e.g., encryption or signing), is used in a distributed manner. For clarity of exposition, we will focus in the following description on threshold encryption schemes. Our construction easily extends to other cryptographic primitives (e.g., signature schemes), as we explain below. Let $(\mathsf{TES}_{\mathrm{GEN}}, \mathsf{Enc}, \mathsf{TES}_{\mathrm{DEC}})$ denote a threshold encryption scheme with distributed key generation protocol $\mathsf{TES}_{\mathrm{GEN}}$ whereas the encryption algorithm Enc is unchanged compared to the standard non-distributed setting. Moreover, decryption is done locally, namely each party P_i can locally decrypt its plaintext share using a local deterministic algorithm Dec_i. Our starting point will be m-party semi-honest secure schemes.

We will later briefly explain how to handle arbitrary attacks. We first formally define threshold encryption schemes.

Definition 5.12 (Threshold Encryption Scheme (TES))**.** *An m-party Threshold Encryption Scheme (TES) for message space $\mathcal{M}$ is a tuple $(\mathsf{TES}_{\text{GEN}}, \mathsf{Enc}, \mathsf{TES}_{\text{DEC}})$ with the following syntax and correctness guarantee:*

- $\mathsf{TES}_{\text{GEN}}(1^\kappa)$ *is an m-party protocol, which on input 1^κ (where κ is a security parameter) outputs an m-tuple of secret key shares $(\text{SK}_1, \ldots, \text{SK}_m)$ and a public key* PK.
- $\mathsf{Enc}(\text{PK}, \mathsf{msg})$ *is a* PPT *algorithm, which on input a public key* PK *and a plaintext $\mathsf{msg} \in \mathcal{M}$ outputs a ciphertext c.*
- $\mathsf{TES}_{\text{DEC}}$ *is a tuple of m local* PPT *algorithms $(\mathsf{Dec}_1, \ldots, \mathsf{Dec}_m)$, where Dec_i on input a secret key share SK_i and a ciphertext c, output a plaintext share msg_i.*[38]

– *Correctness.*

$$\Pr\left[\sum_i \mathsf{msg}_i = \mathsf{msg} \;\middle|\; \begin{array}{l} ((\text{SK}_1, \ldots, \text{SK}_m), \text{PK}) \leftarrow \mathsf{TES}_{\text{GEN}}(1^\kappa) \\ c \leftarrow \mathsf{Enc}(\text{PK}, \mathsf{msg}) \\ \forall i \in [m] : \mathsf{msg}_i = \mathsf{Dec}_i(\text{SK}_i, c) \end{array}\right] = 1$$

The security of TES is defined as follows, extending the standard indistinguishability security of public key encryption schemes for multiple ciphertexts.

Definition 5.13 (TES: Security.)**.** *Let $\mathsf{TES} = (\mathsf{TES}_{\text{GEN}}, \mathsf{Enc}, \mathsf{TES}_{\text{DEC}})$ be a TES. We say that* TES *is an m-*party *t-*secure TES *if for every* PPT *adversary $\mathcal{A}$ corrupting at most t parties there exists an efficient simulator $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3)$ and a negligible function $\delta(\kappa)$ such that:*

$$\left|\Pr\left[\mathcal{REAL}_t(1^\kappa) = 1\right] - \Pr\left[\mathcal{IDEAL}_t(1^\kappa) = 1\right]\right| \leq \delta(\kappa)$$

where $\mathcal{REAL}_t(1^\kappa), \mathcal{IDEAL}_t(1^\kappa)$ are defined in Figure 7, and the probability is over the random coin tosses of $\mathcal{SETUP}^{\mathcal{R}}$, Enc, $\mathcal{A}$ and Sim.

Example: Threshold Additive El Gamal [Gam85]. We first recall the (additive variant of the) standard El Gamal scheme. Let $\mathbb{G}$ be a group of prime order p in which the Decisional Diffie-Hellman problem is hard. The public key is a tuple $\text{PK} = \langle \mathbb{G}, p, g, h \rangle$, where g is a generator of $\mathbb{G}$, and the corresponding secret key is $\text{SK} = s$, s.t. $g^s = h$. Encryption is performed by choosing $r \leftarrow \mathbb{Z}_p$ and computing $\mathsf{Enc}(\text{PK}, m; r) = \langle g^r, h^r \cdot g^m \rangle$. Decryption of a ciphertext $c = \langle \alpha, \beta \rangle$ is performed by computing $g^m = \beta \cdot \alpha^{-s}$ and then finding m by running an exhaustive search. Consequently, this variant is only applicable for small plaintext domains, which is the case in our work. (We note that to encode elements from a domain of size 2^p using RPEs we implicitly view each such element as a bit string of length p).

Next, we discuss the *threshold* variant of the El Gamal scheme where the parties first agree on a group $\mathbb{G}$ of order p and a generator g. Then, each party P_i picks $s_i \leftarrow \mathbb{Z}_p$ and sends $h_i = g^{s_i}$ to the others. Finally, the parties compute $h = \prod_{i=1}^m h_i$ and set $\text{PK} = \langle \mathbb{G}, p, g, h \rangle$. Clearly, the secret key $s = \sum_{i=1}^m s_i$ associated with this public key is correctly shared amongst the parties. In the setting of malicious corruptions, to ensure honest behavior the parties must prove knowledge of their secret key share s_i by running a zero-knowledge proof of a discrete logarithm on (g, h_i). To decrypt a ciphertext $c = \langle c_1, c_2 \rangle$, party P_i computes $c_2 \cdot (\prod_{i=1}^m c_1^{s_i})^{-1}$, and sends $c_1^{s_i}$ together with a corresponding proof of the Diffie-Hellman relation (the proof is used to ensure correctness).

[38] We assume that the randomness used by each party within Dec_i is derived locally.

Secure TES

$\underline{\mathcal{SETUP}^{\mathcal{R}}(1^\kappa)}$:
pick a random string r for $\mathsf{TES}_{\text{GEN}}$ [a]
$(\text{SK}_1, \ldots, \text{SK}_m, \text{PK}) \leftarrow \mathsf{TES}_{\text{GEN}}(1^\kappa; r)$:
output PK

$\underline{\mathcal{ENC}^{\mathcal{R}}(r, \text{PK}, \mathsf{msg}, \mathsf{msg}')}$:
$c \leftarrow \mathsf{Enc}(\text{PK}, \mathsf{msg})$
$(\mathsf{msg}_1, \ldots, \mathsf{msg}_m) \leftarrow \mathsf{TES}_{\text{DEC}}(\text{SK}_1, \ldots, \text{SK}_m, c)$
$\mathsf{id} \leftarrow \mathcal{R}$ [b]
$\mathcal{O} \leftarrow \mathcal{O} \cup \{(\mathsf{msg}_1, \ldots, \mathsf{msg}_m, \mathsf{id})\}$
output (c, id)

$\underline{\mathcal{COR}^{\mathcal{R}}(r, T)}$:
$T_1 \leftarrow T_1 \cup T$
output $(\text{SK}_i)_{i \in T}$

$\underline{\mathcal{REV}^{\mathcal{R}}(r, \mathsf{id})}$:
if $(\cdot, \ldots, \cdot, \mathsf{id}) \notin \mathcal{O}$ then return
let $(\mathsf{msg}_1, \ldots, \mathsf{msg}_m, \mathsf{id}) \in \mathcal{O}$
output $(\mathsf{msg}_i)_{i \notin T_1}$

$\underline{\mathcal{REAL}_{\ell,t}(1^\kappa)}$:
$T_1, \mathcal{O} \leftarrow \emptyset$
Let $\mathcal{O}'$ denote the column of id's within $\mathcal{O}$
$\text{PK} \leftarrow \mathcal{SETUP}^{\mathcal{R}}(1^\kappa)$
$b \leftarrow \mathcal{A}^{\mathcal{ENC}^{\mathcal{R}}(r, \text{PK}, \cdot, \cdot), \mathcal{COR}^{\mathcal{R}}(r, \cdot), \mathcal{REV}^{\mathcal{R}}(r, \cdot)}(\text{PK}, \mathcal{O}')$
if $|T_1| \leq t$
then output b, else, output 0

$\underline{\mathcal{SETUP}^{\mathcal{I}}(1^\kappa)}$:
initialize St to the empty string
$(\mathsf{St}, \text{PK}) \leftarrow \mathsf{Sim}_1(1^\kappa)$
output PK

$\underline{\mathcal{ENC}^{\mathcal{I}}(\mathsf{St}, \text{PK}, \mathsf{msg}, \mathsf{msg}')}$:
$c \leftarrow \mathsf{Enc}(\text{PK}, \mathsf{msg}')$
$\mathsf{id} \leftarrow \mathcal{R}$
$\mathcal{O} \leftarrow \mathcal{O} \cup \{(c, \mathsf{msg}, \mathsf{msg}', \mathsf{id})\}$
output (c, id)

$\underline{\mathcal{COR}^{\mathcal{I}}(\mathsf{St}, T)}$:
$T_1 \leftarrow T_1 \cup T$
$(\text{SK}'_i)_{i \in T} \leftarrow \mathsf{Sim}_2(\mathsf{St}, T)$
output $(\text{SK}'_i)_{i \in T}$

$\underline{\mathcal{REV}^{\mathcal{I}}(\mathsf{St}, \mathsf{id})}$:
if $(\cdot, \cdot, \cdot, \mathsf{id}) \notin \mathcal{O}$ then return
let $(c, \mathsf{msg}, \cdot, \mathsf{id}) \in \mathcal{O}$
$\left(\mathsf{msg}_i^{\mathcal{S}}\right)_{i \notin T_1} \leftarrow \mathsf{Sim}_3(\mathsf{St}, c, \mathsf{msg})$
output $\left(\mathsf{msg}_i^{\mathcal{S}}\right)_{i \notin T_1}$

$\underline{\mathcal{IDEAL}_{\ell,t}(1^\kappa)}$:
$T_1, \mathcal{O} \leftarrow \emptyset$
Let $\mathcal{O}'$ denote the column of id's within $\mathcal{O}$
$(\mathsf{St}, \text{PK}) \leftarrow \mathcal{SETUP}^{\mathcal{I}}(1^\kappa)$
$b \leftarrow \mathcal{A}^{\mathcal{ENC}^{\mathcal{I}}(\mathsf{St}, \text{PK}, \cdot, \cdot), \mathcal{COR}^{\mathcal{I}}(\mathsf{St}, \cdot), \mathcal{REV}^{\mathcal{I}}(r, \cdot)}(\text{PK}, \mathcal{O}')$
if $|T_1| \leq t$
then output b, else, output 0

[a] We assume that the randomness r, the table $\mathcal{O}$, the public key PK, the set T_1 of corrupted parties, and the simulator state St are global parameters that the relevant oracles can access and update.

[b] We assume that $\mathcal{R}$ is a sufficiently large domain of binary strings, so that with overwhelming probability all id's are unique.

Figure 7: The Security Experiments of TES

Our leakage model. We protect secret keys and plaintext shares against *adaptive* leakage attacks of *computationally unbounded adversaries* when the keys and plaintexts are stored in memory (i.e., "data at rest"), but not during computations on them. In particular, we assume that the computations performed on the shares, namely within the subprotocols $\mathsf{TES}_{\text{GEN}}$ and $\mathsf{TES}_{\text{DEC}}$, are carried out in a leak-free environment (see discussion below on cases in which this leak-free assumption can be removed).

Our leakage model is motivated by application scenarios in which the ciphertext and/or plaintext shares are stored in memory for extended time periods. The most natural application scenario is to protect the secret key shares from leakage while they are being stored in memory throughout the lifetime of the system, which could be quite extensive. Our paradigm additionally protects the plaintext shares. In particular, note that though when the (distributed) decryption process ends, the

adversary learns all plaintext shares (and consequently also the plaintext), it can potentially learn information about the plaintext *earlier on* by (adaptively) leaking on the plaintext shares of honest parties *before* the decryption process ends. This is of particular concern when the plaintext shares are expected to be stored for a long time before the decryption is completed. Our paradigm protects against such attacks by guaranteeing that the honest parties' plaintext shares remain entirely hidden even given the leakage. To capitalize on this feature of our paradigm, we explicitly separate the decryption phase into two phases: a phase in which parties compute plaintext shares (on which the adversary can leak), and a reveal phase in which all plaintext shares are revealed to the adversary (and, in particular, the adversary learns the message).

Another example in which our paradigm is useful is when the ciphertext c is generated on some server who knows only the public key (this is possible because TES is a public key object), whereas c is unknown to the adversary, who can only (adaptively) leak on the server's storage, but cannot obtain c in full. In this case, our paradigm guarantees that the plaintext remains *information-theoretically* hidden, because our leakage guarantee holds against unbounded adversaries.

We note that our paradigm is most useful when the shares in the underlying TES protocol form an additive secret sharing of the secret key (resp., multiplicative secret sharing of the public key). This is because our paradigm respects additive sharing, and so this allows us to aggregate the shares before decoding them, as explained in detail in the following paragraph.

Obtaining a stronger leakage guarantee for TES. In some cases, we can achieve a stronger leakage-resilience guarantee for TES. Specifically, if $\mathsf{TES}_{\mathrm{GEN}}, \mathsf{TES}_{\mathrm{DEC}}$ are non-interactive (i.e., each party broadcasts a single message) and the scheme supports additive sharing, then our paradigm can protect *the entire process of public key generation from the shares* (as well as decryption from plaintext shares) from adaptive leakage. We note that this is indeed the case in many TES instantiations (such as the TES described in the example above). These properties eliminate the need for leak-free assumptions since non-interactive sub-protocols introduce no secret intermediate values,[39] and the encoded shares can be aggregated such that decoding is performed only after (an encoding of) the final outcome (the public key or the plaintext) has been computed.

In more detail, using our paradigm we can RPE encode the secret key shares, aggregate the encoded shares to obtain an encoding of the public key, and finally RPE decode to obtain the public key. (We note that in most cases – such as the Diffie-Hellman example described above – decoding is carried out in the exponent relative to some group description. Thus, each party can apply an RPE encoding to each key share in the exponent before broadcasting it to the other parties.) For example, in the El Gamal scheme described above, each party will RPE encode s_i to obtain an encoding s'_i from which it computes the value $h'_i = g^{s'_i}$ that it broadcasts to all other parties. Then, the parties will (locally) compute $h' = \prod_{i=1}^{m} h'_i = g^{\sum_{i=1}^{m} s'_i}$ and finally RPE decode h' – by applying RPE decoding in the exponent – to obtain the public key $h = g^{\sum_{i=1}^{m} s_i}$. (The fact that RPE decoding does indeed give h follows from the linearity of the RPE.)

We run a similar process during the distributed decryption protocol, where we encode the shares of the decrypted ciphertext (i.e., the plaintext shares). As noted above, encoding the plaintext shares is useful when they are expected to be stored in memory for a long period of time before completing the decryption (i.e., executing $\mathsf{TES}_{\mathrm{DEC}}$ which, when the TES has additive reconstruction, is done by adding the plaintext shares) at a later point. Recall that encoding the plaintext shares protects them from leakage until the decryption process is completed.

[39] We note that for decryption, broadcast is required only if all parties should learn the decrypted plaintext. Otherwise, the parties send their message via a point-to-point channel to the designated party.

The leakage-resilience definition. We are now ready to formally define the notion of LR-TES. Similar to Definition 5.10, we first define an admissible adversary for TES, where here the adversary is comprised of three parts since the TES definition is more involved.

Definition 5.14 (Admissible adversaries (TES)). *An admissible adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ is a tuple of algorithms $(\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ such that $\mathcal{A}_1, \mathcal{A}_3$ are* PPT *whereas $\mathcal{A}_2$ is computationally unbounded.*

Before introducing our formal leakage-resilience definition, we provide the high-level idea. Similar to equivocal SSS and LR-FSS, we compare a real-world execution with an ideal execution. In the real-world execution the adversary can use the $\mathcal{COR}^{\mathcal{R}}$ oracle to adaptively corrupt (at most) t parties, where by corrupting a party the adversary learns its secret key share, from which it can conclude the corresponding plaintext shares for any ciphertext. The adversary can also ask for plaintexts to be encrypted (by calling the $\mathcal{ENC}^{\mathcal{R}}$ oracle), and to leak on the key and plaintext shares of honest parties (by calling the $\mathcal{LEAK}^{\mathcal{R}}$ oracle). To capture the fact that the adversary can learn decrypted plaintexts (since plaintexts might be decrypted during the lifetime of the system), we allow the adversary to make $\mathcal{REV}^{\mathcal{R}}$ queries. Throughout the experiment, T_1 maintains the set of corrupted parties, and ℓ_i, ℓ'_i denotes the total number of bits leaked from the key share and plaintext shares of party i, respectively. LR is only guaranteed against an adversary that corrupts at most t parties, and leaks at most ℓ bits from the key share, and all plaintext shares, of each party. (See Remark 5.9 for a discussion of a stronger leakage guarantee for plaintext shares, which our scheme satisfies.) This is captured by the checks performed in the one but last line of $\mathcal{REAL}$, where if the leakage restrictions are violated then the adversarial guess b is disregarded. We note that the definition poses some (necessary) restrictions on the order in which oracles are called, see Remark 5.8 for a discussion. An analogous ideal experiment is defined using a four-step simulator to simulate the setup, as well as the leakage, corruption, and reveal capabilities of the adversary. These simulators extend the simulators for standards TES schemes to support leakage from the secret key and plaintext shares.

Definition 5.15 (Leakage-Resilient TES (LR-TES)). *Let $\mathsf{TES} = (\mathsf{TES}_{\mathrm{GEN}}, \mathsf{Enc}, \mathsf{TES}_{\mathrm{DEC}})$ be a TES. We say that* TES *is* Leakage-Resilient (LR) for the class of (t, ℓ)-local probing leakage *if for every admissible adversary $\mathcal{A}$ corrupting at most t parties there exists an efficient simulator $(\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3, \mathsf{Sim}_4)$ and a negligible function $\varepsilon(\kappa)$ such that:*

$$\left|\Pr\left[\mathcal{REAL}_{\ell,t}\left(1^{\kappa}\right) = 1\right] - \Pr\left[\mathcal{IDEAL}_{\ell,t}\left(1^{\kappa}\right) = 1\right]\right| \leq \varepsilon(\kappa)$$

where $\mathcal{REAL}_{\ell,t}\left(1^{\kappa}\right), \mathcal{IDEAL}_{\ell,t}\left(1^{\kappa}\right)$ are defined in Figure 8, and the probability is over the random coin tosses of $\mathcal{SETUP}^{\mathcal{R}}$, Enc, $\mathcal{A}$ and $(\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3, \mathsf{Sim}_4)$.

A few remarks are in order regarding our LR-TES definition.

Remark 5.8. *First, currently the adversary can only leak on the secret key shares if it had already made an $\mathcal{ENC}$ query. We have chosen to define the leakage oracle this way to simplify the definition. However, we note that this restriction can be removed by replacing the $\mathcal{LEAK}$ oracle with two oracles – $\mathcal{LEAK}_{key}$ and $\mathcal{LEAK}_{cipher}$ – which allow the adversary to leak separately on the key and plaintext shares, respectively. Second, in the current definition the adversary is restricted in its query pattern because it has to first make all $\mathcal{ENC}$ queries, then all $\mathcal{LEAK}$ queries, and finally all $\mathcal{REV}$ queries. Removing this restriction is subtle since we need to guarantee that the state passed to the $\mathcal{LEAK}$ adversary does not provide any information on ciphertexts, because the "$\mathcal{LEAK}$ adversary" can break TES security. We therefore chose to provide the simpler definition with restricted query pattern. We note, however, that if one is willing to have only computational security (even against leakage attacks), then the three adversarial entities can be united into a single, computationally-bounded, adversary with access to all three oracles. In particular, this means the adversary can query the oracles in an arbitrary order.*

Remark 5.9 (Better leakage bound for message shares). *Notice that in Figure 8, for every party $i \in [m]$, we allow a* total *of ℓ leakage bits on* all *message shares msg_i generated throughout the experiment. However, similar to the FSS case (Remark 5.3), our construction will obtain a stronger leakage guarantee, which allows the adversary to leak ℓ bits* from every message share. *For example, if during the execution the adversary calls $\mathcal{ENC}$ with two different messages $\mathsf{msg}, \mathsf{msg}^*$, resulting in two sets of message shares $\mathsf{msg}_1, \ldots, \mathsf{msg}_m$ and $\mathsf{msg}_1^*, \ldots, \mathsf{msg}_m^*$ (respectively), then leakage resilience is guaranteed even if the adversary leaks ℓ bits from msg_i* and *ℓ (possibly different) bits from msg_i^*, for every i. We chose to present the weaker version in Figure 8 to keep the definition simpler and clearer.*

Remark 5.10 (Protecting ciphertexts against leakage). *We note that our LR-TES definition can naturally be extended to also guarantee leakage-resilience for* ciphertexts, *by having the $\mathcal{LEAK}$ oracle take a third function g'', and apply it to the ciphertext associated with id. Our construction can easily be adapted to this case by having the encryption algorithm RPE-encode the ciphertext, where the decryption algorithm first RPE-decodes the ciphertext before generating the plaintext share from it. This scheme can be proved secure by slightly modifying the proof of Theorem 5.8 to also handle ciphertext leakage by simulating it using RPE-encodings of $\vec{0}$, and using the secrecy of partial views of the RPE. We chose not to include this strengthening in our LR-TES definition to keep the definition as simple as possible.*

We present our construction in Figure 9, and summarize its properties in the following theorem:

Theorem 5.8 (Leakage-Resilient and Tampering-Resilient TES). *Assume that the Construction of Figure 9 is instantiated with:*

- *An m-party t-secure TES $\mathsf{TES} = (\mathsf{TES}_{\mathrm{GEN}}, \mathsf{Enc}, \mathsf{TES}_{\mathrm{DEC}})$.*
- *An adaptive $(\varepsilon, \ell, \tau)$-RPE scheme.*

Then the construction defined in Figure 9 is a τ-TR TES which is also leakage-resilient for the class of (t, ℓ)-local probing leakage, where

$$\left|\Pr\left[\mathcal{REAL}_{\ell,t}\left(1^{\kappa}\right)=1\right]-\Pr\left[\mathcal{IDEAL}_{\ell,t}\left(1^{\kappa}\right)=1\right]\right| \leq 2(h+1)m\varepsilon+\delta(\kappa)$$

for some negligible function $\delta(\kappa)$, where $\mathcal{REAL}_{\ell,t}, \mathcal{IDEAL}_{\ell,t}$ were defined in Figure 8, and h is the number of encrypted plaintexts the adversary obtained throughout the execution.

Remark 5.11 (LR-TES for an unbounded number of encrypted plaintext). *The error in Theorem 5.8 grows with the number h of encrypted plaintexts that the adversary queried from the $\mathcal{ENC}$ oracle, which requires some a-priori bound on h. However, we note that this bound can be removed by employing* perfect *RPEs, such as the RPE of 3.1.*

The following corollary follows immediately from Theorem 5.8 by instantiating the Construction of Figure 9 with the RPE of Lemma 3.9.

Corollary 5.9 (Leakage-Resilient and Tampering-Resilient TES). *Let $\mathbb{F}$ be a field. Any m-party t-secure TES TES over $\mathbb{F}$ can be transformed into an $\Omega(N)$-TR TES TES' over $\mathbb{F}$ which is also LR for the class of $(t, \Omega(N))$-local probing leakage, where N denotes the length of key and output shares, and the LR simulation error is exactly the security error of TES. Moreover, this transformation only causes a constant blowup in the key and plaintext share sizes. Furthermore, if TES has additive reconstruction over $\mathbb{F}$, then TES' has linear reconstruction over $\mathbb{F}$.*

LR TES

$\underline{\mathcal{SETUP}^{\mathcal{R}}(1^\kappa)}$:
pick a random string r for $\mathsf{TES}_{\text{GEN}}$ [a]
$(\text{SK}_1, \ldots, \text{SK}_m, \text{PK}) \leftarrow \mathsf{TES}_{\text{GEN}}(1^\kappa; r)$:
output PK

$\underline{\mathcal{ENC}^{\mathcal{R}}(r, \text{PK}, \mathsf{msg}, \mathsf{msg}')}$:
$c \leftarrow \mathsf{Enc}(\text{PK}, \mathsf{msg})$
$(\mathsf{msg}_1, \ldots, \mathsf{msg}_m) \leftarrow \mathsf{TES}_{\text{DEC}}(\text{SK}_1, \ldots, \text{SK}_m, c)$
$\mathsf{id} \leftarrow \mathcal{R}$ [b]
$\mathcal{O} \leftarrow \mathcal{O} \cup \{(\mathsf{msg}_1, \ldots, \mathsf{msg}_m, \mathsf{id})\}$
output c

$\underline{\mathcal{LEAK}^{\mathcal{R}}(r, g, g', \mathsf{id})}$:
if $g \notin \mathcal{L}_{t,\ell}$ [c] or $g' \notin \mathcal{L}_{t,\ell}$ or $(\cdot, \ldots, \cdot, \mathsf{id}) \notin \mathcal{O}$ return
let $(\mathsf{msg}_1, \ldots, \mathsf{msg}_m, \mathsf{id}) \in \mathcal{O}$
$(\mathsf{output}_i)_{i\in[m]} \leftarrow g\left((\text{SK}_i)_{i\in[m]}\right)$
$(\mathsf{output}'_i)_{i\in[m]} \leftarrow g'\left((\mathsf{msg}_i)_{i\in[m]}\right)$
for every $i \in [m]$
$\quad \ell_i \leftarrow \ell_i + |\mathsf{output}_i|, \;\; \ell'_i \leftarrow \ell'_i + |\mathsf{output}'_i|$
output $(\mathsf{output}_i, \mathsf{output}'_i)_{i\in[m]}$

$\underline{\mathcal{COR}^{\mathcal{R}}(r, T)}$:
$T_1 \leftarrow T_1 \cup T$
output $(\text{SK}_i)_{i\in T}$

$\underline{\mathcal{REV}^{\mathcal{R}}(r, \mathsf{id})}$:
if $(\cdot, \ldots, \cdot, \mathsf{id}) \notin \mathcal{O}$ then return
let $(\mathsf{msg}_1, \ldots, \mathsf{msg}_m, \mathsf{id}) \in \mathcal{O}$
output $(\mathsf{msg}_i)_{i\notin T_1}$

$\underline{\mathcal{REAL}_{\ell,t}(1^\kappa)}$:
$\ell_1, \ell'_1, \ldots, \ell_m, \ell'_m \leftarrow 0, \;\; T_1, \mathcal{O} \leftarrow \emptyset$
$\text{PK} \leftarrow \mathcal{SETUP}^{\mathcal{R}}(1^\kappa)$
$\mathsf{St}_1^{\mathcal{A}} \leftarrow \mathcal{A}_1^{\mathcal{ENC}^{\mathcal{R}}(r,\text{PK},\cdot,\cdot)}(\text{PK})$
Let $\mathcal{O}'$ denote the column of id's within $\mathcal{O}$
$\mathsf{St}_2^{\mathcal{A}} \leftarrow \mathcal{A}_2^{\mathcal{LEAK}^{\mathcal{R}}(r,\cdot,\cdot,\cdot)}(\text{PK}, \mathcal{O}')$
$b \leftarrow \mathcal{A}_3^{\mathcal{COR}^{\mathcal{R}}(r,\cdot),\mathcal{REV}^{\mathcal{R}}(r,\cdot)}(\mathsf{St}_1^{\mathcal{A}}, \mathsf{St}_2^{\mathcal{A}})$
if $\ell_i, \ell'_i \le \ell$ for every $i \in [m]$ and $|T_1| \le t$
then output b, else, output 0

$\underline{\mathcal{SETUP}^{\mathcal{I}}(1^\kappa)}$:
initialize St to the empty string
$(\mathsf{St}, \text{PK}) \leftarrow \mathsf{Sim}_1(1^\kappa)$
output PK

$\underline{\mathcal{ENC}^{\mathcal{I}}(\mathsf{St}, \text{PK}, \mathsf{msg}, \mathsf{msg}')}$:
$c \leftarrow \mathsf{Enc}(\text{PK}, \mathsf{msg}')$
$\mathsf{id} \leftarrow \mathcal{R}$
$\mathcal{O} \leftarrow \mathcal{O} \cup \{(c, \mathsf{msg}, \mathsf{msg}', \mathsf{id})\}$
output c

$\underline{\mathcal{LEAK}^{\mathcal{I}}(\mathsf{St}, g, g', \mathsf{id})}$:
if $g \notin \mathcal{L}_{t,\ell}$ or $g' \notin \mathcal{L}_{t,\ell}$ or $(\cdot, \cdot, \cdot, \mathsf{id}) \notin \mathcal{O}$ return
$\left(\mathsf{St}, (\mathsf{output}_i, \mathsf{output}'_i)_{i\in[m]}\right) \leftarrow \mathsf{Sim}_2(\mathsf{St}, g, g')$
for every $i \in [m]$
$\quad \ell_i \leftarrow \ell_i + |\mathsf{output}_i|, \;\; \ell'_i \leftarrow \ell'_i + |\mathsf{output}'_i|$
output $\left((\mathsf{output}_i, \mathsf{output}'_i)_{i\in[m]}\right)$

$\underline{\mathcal{COR}^{\mathcal{I}}(\mathsf{St}, T)}$:
$T_1 \leftarrow T_1 \cup T$
$(\text{SK}'_i)_{i\in T} \leftarrow \mathsf{Sim}_3(\mathsf{St}, T)$
output $(\text{SK}'_i)_{i\in T}$

$\underline{\mathcal{REV}^{\mathcal{I}}(\mathsf{St}, \mathsf{id})}$:
if $(\cdot, \cdot, \cdot, \mathsf{id}) \notin \mathcal{O}$ then return
let $(c, \mathsf{msg}, \cdot, \mathsf{id}) \in \mathcal{O}$
$\left(\mathsf{msg}_i^{\mathcal{S}}\right)_{i\notin T_1} \leftarrow \mathsf{Sim}_4(\mathsf{St}, c, \mathsf{msg})$
output $\left(\mathsf{msg}_i^{\mathcal{S}}\right)_{i\notin T_1}$

$\underline{\mathcal{IDEAL}_{\ell,t}(1^\kappa)}$:
$\ell_1, \ell'_1, \ldots, \ell_m, \ell'_m \leftarrow 0, \;\; T_1, \mathcal{O} \leftarrow \emptyset$
$(\mathsf{St}, \text{PK}) \leftarrow \mathcal{SETUP}^{\mathcal{I}}(1^\kappa)$
$\mathsf{St}_1^{\mathcal{A}} \leftarrow \mathcal{A}_1^{\mathcal{ENC}^{\mathcal{I}}(\mathsf{St},\text{PK},\cdot,\cdot)}(\text{PK})$
Let $\mathcal{O}'$ denote the column of id's within $\mathcal{O}$
$\mathsf{St}_2^{\mathcal{A}} \leftarrow \mathcal{A}_2^{\mathcal{LEAK}^{\mathcal{I}}(\mathsf{St},\cdot,\cdot,\cdot)}(\text{PK}, \mathcal{O}')$
$b \leftarrow \mathcal{A}_3^{\mathcal{COR}^{\mathcal{I}}(\mathsf{St},\cdot),\mathcal{REV}^{\mathcal{I}}(r,\cdot)}(\mathsf{St}_1^{\mathcal{A}}, \mathsf{St}_2^{\mathcal{A}})$
if $\ell_i, \ell'_i \le \ell$ for every $i \in [m]$ and $|T_1| \le t$
then output b, else, output 0

[a] We assume that the randomness r, the table $\mathcal{O}$, the public key PK, the set T_1 of corrupted parties, and the simulator state St are global parameters that the relevant oracles can access and update.

[b] We assume that $\mathcal{R}$ is a sufficiently large domain of binary strings, so that with overwhelming probability all id's are unique.

[c] Recall that $\mathcal{L}_{t,\ell}$ was defined in Definition 2.2 in Section 2.1.

Figure 8: The Security Experiments of LR TES

Remark 5.12. *We note that if one does not care to preserve linear reconstruction, then one can instantiate Theorem 5.8 with the RPEs of Corollary 3.1, resulting in a LR-TES with the properties specified in the corollary (except for linear reconstruction). Thus, one can obtain a LR-TES from* any *TES scheme. (This should be contrasted with the LR-TES scheme of Corollary 5.9, in which the (original) TES scheme and the RPE are required to be over the same field.)*

Proof sketch: We argue that the construction defined in Figure 9 satisfies the properties of Definition 5.13, the leakage-resilience guarantee of Definition 5.15, and the tampering-resilience property of Definition 2.7.

Correctness. Follows directly from a combination of the correctness of the underlying TES and RPE schemes.

Secrecy. Follows directly from the privacy of TES, since the RPE encoding is applied separately to each secret key and plaintext shares. Specifically, the simulator can use the TES simulator to generate secret key/plaintext shares, and then honestly RPE encode each individual simulated share. We note that the static/adaptive flavor of secrecy is inherited from the underlying TES scheme, namely if the underlying scheme is only statically-secure, then the resultant scheme is also statistically-secure, whereas if the underlying scheme is adaptively-secure, then so is the resultant scheme.

TR. Follows from the error-correction of the RPE (since each share is encoded separately), and the correctness of TES (since we assume semi-honest corruptions), using similar arguments to the proof of Theorem 4.1.

LR. Follows from a combination of the secrecy of TES, the adaptive secrecy of partial views of the RPE, and the adaptive reconstruction* from partial views of the RPE, using a sequence of hybrid arguments. Specifically, the adversary is leaking from a single set of key shares, and h sets of plaintext shares. The simulator will answer leakage queries using RPE encodings of $\vec{0}$. When the adversary corrupts a subset T of parties, the simulator will use the TES simulator to simulate the key shares of parties in T, whereas when the adversary calls $\mathcal{REV}$, the simulator will use the TES simulator to simulate the plaintext shares of honest parties. In both cases, the simulator will then run the resampling algorithm Rec of the RPE scheme to sample RPE-encodings of the simulated key or plaintext shares which are consistent with the leakage. (Here, whether or not the adversary can adaptively corrupt parties depends on the security flavor of the underlying TES scheme.) Indistinguishability between the real and the simulated executions can be shown via a sequence of hybrid arguments, as we now explain. In the following, we assume that the adversary corrupts at most t parties, and obtains at most ℓ leakage bits from every RPE-encoding. This is without loss of generality, since otherwise both experiments output 0.

Let $\mathcal{H}_0$ denote the real execution of the LR-TES. The next hybrid $\mathcal{H}_1$ is obtained from $\mathcal{H}_0$ by (1) answering leakage queries on key and plaintexts shares using RPE-encodings of $\vec{0}$, and (2) answering corruption queries by using the resampling algorithm Rec of the RPE to resample a fresh RPE-encoding of the key consistently with the leakage, and (3) answering reveal queries by using the resampling algorithm Rec of the RPE to resample a fresh RPE-encoding of the plaintext share. We stress that $\mathcal{H}_1$ encodes *the same values* (of key and plaintext shares) as in $\mathcal{H}_0$, so (apart from the leakage) the only difference between the hybrids is in the *RPE encoding itself*. Then $\mathsf{SD}(\mathcal{H}_0, \mathcal{H}_1) \leq (h+1)m\varepsilon$ by the adaptive secrecy of partial views, and the adaptive reconstruction* from partial views of the RPE. Specifically, we order the key and plaintext shares generated throughout the experiment in some arbitrary order, and define a sequence of hybrids where in the i'th hybrid we switch the leakage (from real to leakage on encodings of $\vec{0}$), and resmaple the encoding (if the share was queried to $\mathcal{COR}$ or $\mathcal{REV}$), only in the first i shares. When comparing hybrid $i-1$ with hybrid

Leakage-Resilient and Tampering-Resilient TES

Building blocks: an m-party TES $(\mathsf{TES}_{\text{GEN}}, \mathsf{Enc}, \mathsf{TES}_{\text{DEC}})$ and an RPE scheme $(\mathsf{Encode}, \mathsf{Decode}, \mathsf{Rec})$.

Key shares generation: on input a security parameter 1^κ:

- Computes $(\text{SK}_1, \ldots, \text{SK}_m, \text{PK}) \leftarrow \mathsf{TES}_{\text{GEN}}(1^\kappa)$ where party P_i obtains SK_i and all parties obtain PK.
- For every $i \in [m]$, encodes $\widetilde{\text{SK}}_i \leftarrow \mathsf{Encode}(\text{SK}_i)$.
- Outputs the key shares $\left(\widetilde{\text{SK}}_1, \ldots, \widetilde{\text{SK}}_m\right)$ and the public key PK.

Encryption: identical to Enc – on input a plaintext msg and a public key PK outputs $c \leftarrow \mathsf{Enc}(\text{PK}, \mathsf{msg})$.

Decryption: on input a key share $\widetilde{\text{SK}}_i, i \in [m]$ and a ciphertext c:

- Decodes $\text{SK}_i = \mathsf{Decode}\left(\widetilde{\text{SK}}_i\right)$.
- Decrypts $\mathsf{msg}_i \leftarrow \mathsf{Dec}_i(\text{SK}_i, c)$.
- Encodes $\widetilde{\mathsf{msg}}_i \leftarrow \mathsf{Encode}(\mathsf{msg}_i)$.
- Erases SK_i, c and msg_i, together with all intermediate computations except for $\widetilde{\mathsf{msg}}_i$.
- Outputs $\widetilde{\mathsf{msg}}_i$.

Figure 9: LR and TR Threshold Encryption Scheme

i, where the ith share is a (key or plaintext) share of the jth party, there are three possible cases. First, if the encoding of the ith share was never queried (i.e., by a $\mathcal{LEAK}$, $\mathcal{COR}$ or $\mathcal{REV}$ query) then the hybrids are identical. Second, if the ith share was queried by a $\mathcal{LEAK}$ query, but not by a $\mathcal{REV}$ or $\mathcal{COR}$ query, then the hybrids are ε-statistically close by the adaptive secrecy of partial views of the RPE. Finally, if the ith share is a plaintext share that was queried by a $\mathcal{REV}$ query, or it is a key share of a *corrupted* party j (i.e., j was queried through a $\mathcal{COR}$ query), then the hybrids are ε-statistically close by the adaptive reconstruction* from partial views of the RPE.

Next, we define $\mathcal{H}_2$ that is identical to the simulated execution as defined above. The only difference between $\mathcal{H}_1, \mathcal{H}_2$ is in how corruption and reveal queries are answered – in $\mathcal{H}_1$ they are answered by resampling an RPE-encoding of the *actual* key and plaintext shares, whereas in $\mathcal{H}_2$ they are resampled using *simulated* key and plaintexts shares. Then $\mathcal{H}_1$ and $\mathcal{H}_2$ are computationally indistinguishable by the secrecy of the underlying TES, through the following reduction. The reduction emulates the leakage on RPE encodings of $\vec{0}$ (which, in particular, are independent of the honest parties' real shares). When it receives a corrupt query, it forwards this query to its oracle receiving back the key shares for the corrupted parties, for which it honestly resamples the RPE-encodings (using Rec) consistently with the leakage, and hands these shares to the adversary. When it receives a reveal query, it forwards this query to its oracle receiving back the plaintext shares for the honest parties, for which it honestly resamples the RPE encodings (using Rec) consistently with the leakage, and hands these shares to the adversary. Finally, it returns the bit b output by the adversary. (We note that the leakage is – always! – adaptively simulated, whereas the corruption pattern is either static or adaptive, depending on the permissible corruption strategies in the underlying TES.) ■

Achieving leakage-resilient TES in the malicious setting. Threshold crytosystems in the presence of malicious attackers are harder to achieve and require heavier tools for ensuring correctness

e.g., zero-knowledge proofs of knowledge or additional encoding mechanisms. For systems that are amplified using such proofs without modifying the messages of the underlying semi-honest protocol, e.g., the Diffie-Hellman protocol, we can apply our paradigm in a similar manner. Namely, each party attaches a proof for every message of the semi-honest protocol. In the Diffie-Hellman protocol this boils down to a proof of knowledge of the exponent which can be applied to the encoded value.

Threshold signature schemes. Threshold signature schemes are used in applications where multiple signers are required to generate a signature. A notable example is ECDSA, a standardized signing algorithm that is widely deployed in practice. Its threshold variant attracted much attention lately due to its usage in Bitcoin and other cryptocurrencies. Other examples for threshold singing protocols exist for RSA and Schnorr signatures. Our paradigm is also useful for enhancing the security of signature schemes (guaranteeing leakage-resilience of the key shares) in which the secret decryption key is additively shared between the parties, which is indeed the case for these schemes.

Acknowledgments

We thank the anonymous ITC'22 reviewers for their helpful comments. The first and third authors were supported by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office. The first author was also supported by ISF grant No. 1316/18.

References

[ADF16] Marcin Andrychowicz, Stefan Dziembowski, and Sebastian Faust. Circuit compilers with $o(1/\log(n))$ leakage rate. In *EUROCRYPT, Proceedings, Part II*, pages 586–615, 2016.

[ADN+19] Divesh Aggarwal, Ivan Damgård, Jesper Buus Nielsen, Maciej Obremski, Erick Purwanto, João L. Ribeiro, and Mark Simkin. Stronger leakage-resilient and non-malleable secret sharing schemes for general access structures. In *CRYPTO*, pages 510–539, 2019.

[AGV09] Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *TCC, Proceedings*, pages 474–495, 2009.

[BBC+19a] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. How to prove a secret: Zero-knowledge proofs on distributed data via fully linear PCPs. *IACR Cryptol. ePrint Arch.*, 2019:188, 2019.

[BBC+19b] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Zero-knowledge proofs on secret-shared data via fully linear PCPs. In *CRYPTO, Proceedings, Part III*, pages 67–97, 2019.

[BCG+11] Nir Bitansky, Ran Canetti, Shafi Goldwasser, Shai Halevi, Yael Tauman Kalai, and Guy N. Rothblum. Program obfuscation with leaky hardware. In *ASIACRYPT, Proceedings*, pages 722–739, 2011.

[BCG+21] Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. Function secret sharing for mixed-mode and fixed-point secure computation. In *EUROCRYPT*, pages 871–900, 2021.

[BCH12] Nir Bitansky, Ran Canetti, and Shai Halevi. Leakage-tolerant interactive protocols. In *TCC, Proceedings*, pages 266–284, 2012.

[BDG+18] Marshall Ball, Dana Dachman-Soled, Siyao Guo, Tal Malkin, and Li-Yang Tan. Non-malleable codes for small-depth circuits. In *FOCS*, pages 826–837, 2018.

[BDIR18] Fabrice Benhamouda, Akshay Degwekar, Yuval Ishai, and Tal Rabin. On the local leakage resilience of linear secret sharing schemes. In *CRYPTO, Proceedings*, pages 531–561, 2018.

[BDKM16] Marshall Ball, Dana Dachman-Soled, Mukul Kulkarni, and Tal Malkin. Non-malleable codes for bounded depth, bounded fan-in circuits. In *EUROCRYPT*, pages 881–908, 2016.

[BDL14] Nir Bitansky, Dana Dachman-Soled, and Huijia Lin. Leakage-tolerant computation with input-independent preprocessing. In *CRYPTO, Proceedings, Part II*, pages 146–163, 2014.

[Bea91] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *CRYPTO*, volume 576, pages 420–432. Springer, 1991.

[BFV19] Gianluca Brian, Antonio Faonio, and Daniele Venturi. Continuously non-malleable secret sharing for general access structures. In *TCC*, pages 211–232, 2019.

[BG10] Zvika Brakerski and Shafi Goldwasser. Circular and leakage resilient public-key encryption under subgroup indistinguishability - (or: Quadratic residuosity strikes back). In *CRYPTO, Proceedings*, pages 1–20, 2010.

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *CCS*, pages 1292–1303, 2016.

[BGJ+13] Elette Boyle, Sanjam Garg, Abhishek Jain, Yael Tauman Kalai, and Amit Sahai. Secure computation against adaptive auxiliary information. In *CRYPTO*, pages 316–334, 2013.

[BGJK12] Elette Boyle, Shafi Goldwasser, Abhishek Jain, and Yael Tauman Kalai. Multiparty computation secure against continual memory leakage. In *STOC, Proceedings*, pages 1235–1254, 2012.

[BGK11] Elette Boyle, Shafi Goldwasser, and Yael Tauman Kalai. Leakage-resilient coin tossing. In *DISC, Proceedings*, pages 181–196, 2011.

[BIVW16] Andrej Bogdanov, Yuval Ishai, Emanuele Viola, and Christopher Williamson. Bounded indistinguishability and the complexity of recovering secrets. In *CRYPTO*, pages 593–618, 2016.

[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. In *FOCS*, pages 501–510, 2010.

[Bra84] Gabriel Bracha. An asynchronous $(n-1)/3$-resilient consensus protocol. In *PODC*, pages 154–162, 1984.

[BS11] Zvika Brakerski and Gil Segev. Better security for deterministic public-key encryption: The auxiliary-input setting. In *CRYPTO, Proceedings*, pages 543–560, 2011.

[BS19] Saikrishna Badrinarayanan and Akshayaram Srinivasan. Revisiting non-malleable secret sharing. In *EUROCRYPT*, pages 593–622, 2019.

[BSW11] Elette Boyle, Gil Segev, and Daniel Wichs. Fully leakage-resilient signatures. In *EUROCRYPT, Proceedings*, pages 89–108, 2011.

[CDMW08] Seung Geol Choi, Dana Dachman-Soled, Tal Malkin, and Hoeteck Wee. Black-box construction of a non-malleable encryption scheme from any semantically secure one. In *TCC*, pages 427–444, 2008.

[CDMW18] Seung Geol Choi, Dana Dachman-Soled, Tal Malkin, and Hoeteck Wee. A black-box construction of non-malleable encryption from semantically secure encryption. *J. Cryptol.*, 31(1):172–201, 2018.

[CDNO97] Ran Canetti, Cynthia Dwork, Moni Naor, and Rafail Ostrovsky. Deniable encryption. In *CRYPTO*, pages 90–104, 1997.

[CGG+20] Eshan Chattopadhyay, Jesse Goodman, Vipul Goyal, Ashutosh Kumar, Xin Li, Raghu Meka, and David Zuckerman. Extractors and secret sharing against bounded collusion protocols. In *FOCS*, pages 1226–1242, 2020.

[CKOS21] Nishanth Chandran, Bhavana Kanukurthi, Sai Lakshmi Bhavana Obbattu, and Sruthi Sekar. Adaptive extractors and their application to leakage resilient secret sharing. In *CRYPTO*, pages 595–624, 2021.

[CLW06] Giovanni Di Crescenzo, Richard J. Lipton, and Shabsi Walfish. Perfectly secure password protocols in the bounded retrieval model. In *TCC, Proceedings*, pages 225–244, 2006.

[CPP20] Ran Canetti, Sunoo Park, and Oxana Poburinnaya. Fully deniable interactive encryption. In *CRYPTO*, pages 807–835, 2020.

[DDN15] Ivan Damgård, Frédéric Dupuis, and Jesper Buus Nielsen. On the orthogonal vector problem and the feasibility of unconditionally secure leakage-resilient computation. In *ICITS, Proceedings*, pages 87–104, 2015.

[DDV10] Francesco Davì, Stefan Dziembowski, and Daniele Venturi. Leakage-resilient storage. In *SCN, Proceedings*, pages 121–137, 2010.

[DF12] Stefan Dziembowski and Sebastian Faust. Leakage-resilient circuits without computational assumptions. In *TCC, Proceedings*, pages 230–247, 2012.

[DGK+] Yevgeniy Dodis, Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Public-key encryption schemes with auxiliary inputs. In *TCC, Proceedings*, pages 361–381.

[DGR97] Scott E. Decatur, Oded Goldreich, and Dana Ron. A probabilistic error-correcting scheme. *IACR Cryptol. ePrint Arch.*, 1997:5, 1997.

[DGR99] Scott E. Decatur, Oded Goldreich, and Dana Ron. Computational sample complexity. *SIAM J. Comput.*, 29(3):854–879, 1999.

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

[DHLW10] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Cryptography against continuous memory attacks. In *FOCS*, pages 511–520, 2010.

[DKL09] Yevgeniy Dodis, Yael Tauman Kalai, and Shachar Lovett. On cryptography with auxiliary input. In *STOC, Proceedings*, pages 621–630, 2009.

[DLZ15] Dana Dachman-Soled, Feng-Hao Liu, and Hong-Sheng Zhou. Leakage-resilient circuits revisited - optimal number of computing components without leak-free hardware. In *EUROCRYPT, Proceedings, Part II*, pages 131–158, 2015.

[DP07] Stefan Dziembowski and Krzysztof Pietrzak. Intrusion-resilient secret sharing. In *FOCS, Proceedings*, pages 227–237, 2007.

[Dzi06] Stefan Dziembowski. Intrusion-resilience via the bounded-storage model. In *TCC, Proceedings*, pages 207–224, 2006.

[FLOP18] Tore Kasper Frederiksen, Yehuda Lindell, Valery Osheter, and Benny Pinkas. Fast distributed RSA key generation for semi-honest and malicious adversaries. In *CRYPTO*, pages 331–361, 2018.

[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, Proceedings*, pages 135–156, 2010.

[FV19] Antonio Faonio and Daniele Venturi. Non-malleable secret sharing in the computational setting: Adaptive tampering, noisy-leakage resilience, and improved rate. In *CRYPTO*, pages 448–479, 2019.

[Gam85] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[GIM+16] Vipul Goyal, Yuval Ishai, Hemanta K. Maji, Amit Sahai, and Alexander A. Sherstov. Bounded-communication leakage resilience via parity-resilient circuits. In *FOCS*, pages 1–10, 2016.

[GJS11] Sanjam Garg, Abhishek Jain, and Amit Sahai. Leakage-resilient zero knowledge. In *CRYPTO, Proceedings*, pages 297–315, 2011.

[GK18a] Vipul Goyal and Ashutosh Kumar. Non-malleable secret sharing. In *STOC, Proceedings*, pages 685–698, 2018.

[GK18b] Vipul Goyal and Ashutosh Kumar. Non-malleable secret sharing for general access structures. In *CRYPTO, Proceedings, Part I*, pages 501–530, 2018.

[GM10] Berndt M. Gammel and Stefan Mangard. On the duality of probing and fault attacks. *J. Electron. Test.*, 26(4):483–493, 2010.

[GR10] Shafi Goldwasser and Guy N. Rothblum. Securing computation against continuous leakage. In *CRYPTO, Proceedings*, pages 59–79, 2010.

[GR12] Shafi Goldwasser and Guy N. Rothblum. How to compute in the presence of leakage. In *FOCS*, pages 31–40, 2012.

[GW16] Venkatesan Guruswami and Mary Wootters. Repairing Reed-Solomon codes. In *STOC, Proceedings*, pages 216–226, 2016.

[HMR+19] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, Tomas Toft, and Angelo Agatino Nicolosi. Efficient RSA key generation and threshold paillier in the two-party setting. *J. Cryptol.*, 32(2):265–323, 2019.

[HVW21] Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, and Mor Weiss. ZK-PCPs from leakage-resilient secret sharing. In *ITC*, 2021.

[IPSW06] Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and David A. Wagner. Private circuits II: keeping secrets in tamperable circuits. In *EUROCRYPT*, pages 308–327, 2006.

[ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO*, pages 463–481, 2003.

[JV10] Ali Juma and Yevgeniy Vahlis. Protecting cryptographic keys against continual leakage. In *CRYPTO, Proceedings*, pages 41–58, 2010.

[KHF+19] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *SP*, pages 1–19, 2019.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *CRYPTO, Proceedings*, pages 388–397, 1999.

[KMS19] Ashutosh Kumar, Raghu Meka, and Amit Sahai. Leakage-resilient secret sharing against colluding parties. In *FOCS*, pages 636–660, 2019.

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *CRYPTO, Proceedings*, pages 104–113, 1996.

[KOST21] Bhavana Kanukurthi, Sai Lakshmi Bhavana Obbattu, Sruthi Sekar, and Jenit Tomy. Locally reconstructable non-malleable secret sharing. *IACR Cryptol. ePrint Arch.*, 2021:657, 2021.

[KR19] Yael Tauman Kalai and Leonid Reyzin. A survey of leakage-resilient cryptography. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 727–794. 2019.

[LCG+19] Fuchun Lin, Mahdi Cheraghchi, Venkatesan Guruswami, Reihaneh Safavi-Naini, and Huaxiong Wang. Non-malleable secret sharing against affine tampering. *CoRR*, abs/1902.06195, 2019.

[LCG+20] Fuchun Lin, Mahdi Cheraghchi, Venkatesan Guruswami, Reihaneh Safavi-Naini, and Huaxiong Wang. Leakage-resilient secret sharing in non-compartmentalized models. In *ITC*, pages 7:1–7:24, 2020.

[LN18] Yehuda Lindell and Ariel Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In *CCS*, pages 1837–1854, 2018.

[LSG+18] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *USENIX Security*, pages 973–990, 2018.

[Mil14] Eric Miles. Iterated group products and leakage resilience against NC^1. In *ITCS*, pages 261–268, 2014.

[MNP+21] Hemanta K. Maji, Hai H. Nguyen, Anat Paskin-Cherniavsky, Tom Suad, and Mingyuan Wang. Leakage-resilience of the shamir secret-sharing scheme against physical-bit leakages. In *EUROCRYPT*, pages 344–374, 2021.

[MPSW21] Hemanta K. Maji, Anat Paskin-Cherniavsky, Tom Suad, and Mingyuan Wang. Constructing locally leakage-resilient linear secret-sharing schemes. In *CRYPTO*, pages 779–808, 2021.

[MR04] Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In *TCC*, pages 278–296, 2004.

[MV13] Eric Miles and Emanuele Viola. Shielding circuits with groups. In *STOC*, pages 251–260, 2013.

[NS09] Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. In *CRYPTO, Proceedings*, pages 18–35, 2009.

[NS20] Jesper Buus Nielsen and Mark Simkin. Lower bounds for leakage-resilient secret sharing. In *EUROCRYPT, Proceedings, Part I*, pages 556–577, 2020.

[Rot12] Guy N. Rothblum. How to compute under $\mathcal{AC}^0$ leakage without secure hardware. In *CRYPTO, Proceedings*, pages 552–569, 2012.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[SV19] Akshayaram Srinivasan and Prashant Nalini Vasudevan. Leakage resilient secret sharing and applications. In *CRYPTO, Proceedings*, pages 480–509, 2019.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC*, pages 475–484, 2014.

[TX21] Ivan Tjuawinata and Chaoping Xing. Leakage-resilient secret sharing with constant share size. *CoRR*, abs/2105.03074, 2021.

[YWZ20] Kang Yang, Xiao Wang, and Jiang Zhang. More efficient MPC from improved triple generation and authenticated garbling. In *CCS*, pages 1627–1646. ACM, 2020.

A Adaptive Zero-Knowledge of the dZK proof of [BBC+19b]

To obtain *fully-adaptive* leakage-resilience in our LR-dZK proofs of Section 5.2, we relied on (non-leakage-resilient) dZK proofs with dZK against *adaptive* adversaries. For completeness, in this section we show that the dZK proof of [BBC+19a, Theorem 6.6] based on public-coin Fully-Linear Interactive Oracle Proofs (FLIOPs) with strong honest-verifier ZK, has dZK against *adaptive* adversaries. Combined with the construction of Figure 4 (and Theorem 5.3), this gives a TR and adaptively LR dZK proof for malicious prover or verifiers.

Overview of the dZK proof of [BBC+19a]. The dZK proof of [BBC+19a, Thoerem 6.6] uses as a building block a public-coin FLIOP that has strong honest-verifier ZK. Roughly, in an IOP system the verifier has oracle access to the prover messages, where in a fully-linear IOP the answer to a verifier query q to a proof oracle π is computed as $\langle x \circ \pi, q \rangle$, where $\circ$ denotes concatenation, and x is the input statement (i.e., the verifier is trying to verify that x is in some language L). The public-coin property of the FLIOP guarantees that the verifier queries are simply random coins. Strong honest-verifier zero-knowledge means there exists an efficient simulator that can simulate – *with no knowledge of x or the corresponding witness* – the answers to the oracle queries of the honest verifier.

At a high level, the dZK proof between a prover P and verifiers $V_1, \ldots, V_k$ operates in epochs, where in each epoch the prover P emulates the FLIOP prover P_{in} to obtain a proof oracle, which it additively secret shares, sending the jth proof share to V_j. Then, all parties invoke an ideal coin-tossing functionality to determine the queries which the FLIOP verifier V_{in} makes in this round. Then, each verifier V_j locally applies the queries to its proof share and input piece $x^{(j)}$ to obtain an answer share. When the FLIOP execution terminates, all verifiers send their answer shares to V_1,[40] who recovers from them the oracle answers for V_{in} (this is done by summing the answer shares, which form an additive sharing of the oracle answers). Finally, V_1 emulates V_{in} with these oracle answers, and broadcasts the output of V_{in} to all verifiers, who output it as their own output. In particular, the only direct communication between the verifiers is in the last two rounds: in the one but last round all verifiers send their answer shares to V_1, and in the last round V_1 broadcasts the output. (We refer the interested reader to [BBC+19a] for the formal description of the dZK proof, as well as the definition of strong HVZK and the underlying FLIOP system.)

The adaptive security of [BBC+19a, Theorem 6.6] is stated in the following lemma.

Lemma A.1 (Adaptive dZK proof)**.** *The dZK proof of [BBC+19b, Theorem 6.6] has dZK against adversaries that* adaptively *corrupt $t \leq k-1$ verifiers.*[41]

Proof: Let Π_{dist} denote the dZK of [BBC+19a, Theorem 6.6] (page 32 in [BBC+19a]), and recall that Π_{dist} employs an FLIOP that is public-coin and has strong honest-verifiers zero-knowledge. That is, the FLIOP verifier is public-coin, and the zero-knowledge simulator can simulate a proof for an input x in the language, *without* knowing x. Recall also that we consider a slightly modified version of the proof, in which the verifiers defer their messages to V_1 until the last round. Let $\mathcal{A}$ be an adversary that adaptively corrupts $t \leq k-1$ verifiers. We describe a simulator Sim for $\mathcal{A}$, which uses the simulator $\mathsf{Sim}_{\mathsf{in}}$ of the FLIOP, interacts with $\mathcal{A}$, and operates as follows.

1. Sim emulates the simulator $\mathsf{Sim}_{\mathsf{in}}$ of the FLIOP to obtain the randomness $\mathsf{R}_1, \ldots \mathsf{R}_r$ of the FLIOP-verifier (here, r determines the total number of phases in the proof, and each R_l determines verifier queries $\vec{q}_{l,1}, \ldots, \vec{q}_{l,s_l}$ because the FLIOP is public-coin), and the oracle answers $\vec{a}_1, \ldots, \vec{a}_r$ (where $\vec{a}_l, 1 \leq l \leq r$ is a length-s_l vector).

2. For every phase $l = 1, \ldots, r$:

   (a) For every corrupted verifier V_j, Sim picks its proof share $\pi_{j,l}$ uniformly at random, and gives it to $\mathcal{A}$ as the answer of the oracle.

[40] This is a slightly modified version of the dZK proof of [BBC+19b, Theorem 6.6], where in the original protocol the verifiers send their answer shares to V_1 as they are computed throughout the execution. Deferring the messages to V_1 to the last round does not affect any of the properties of the protocol (because the underlying FLIOP is public-coin), but will allow for a cleaner analysis of adaptive security.

[41] Recall that we consider a slightly modified version of the dZK proof of [BBC+19b, Theorem 6.6], in which all verifiers defer their messages to V_1 to the last round.

Whenever $\mathcal{A}$ corrupts a verifier V_j, Sim provides $\mathcal{A}$ with the shares $\pi_{j,l'}$ for $l' \leq l$, which are sampled uniformly at random. We note that when $\mathcal{A}$ corrupts V_j then Sim is given the input piece x^j of V_j.

(b) Sim provides R_l to $\mathcal{A}$ as the output of the coin-tossing oracle.

3. Let T denote the set of corrupted verifiers at the onset of the one but last round (in which all verifiers send their answer shares to V_1). If $1 \in T$ (i.e., V_1 is corrupted), then Sim needs to simulate the messages which the honest verifiers send to V_1 in this round. For every *corrupted* verifier $V_j, j \in T$, Sim honestly computes the answer shares $\vec{a}_{j,1}, \ldots, \vec{a}_{j,r}$ which V_j would have computed in Π_{dist} *had it been honest* (this is possible because Sim knows both x^j and $\pi_{j,1}, \ldots, \pi_{j,r}$). More specifically, for every $1 \leq l \leq r, 1 \leq h \leq s_l$, the hth coordinate of $\vec{a}_l$ is computed as $\langle \vec{q}_{l,h}, \left(\tilde{x}^1, \ldots, \tilde{x}^{j-1}, x^j, \tilde{x}^{j+1}, \ldots, x^k\right) \circ \pi_{j,l} \rangle$, where $\vec{q}_{l,h}$ is the lth query of the FLIOP verifier in round l, $\tilde{x}^z = \vec{0}$ for every $j \neq z \in [k]$, and $\pi_{j,l}$ is the proof share that V_j received in phase l. Then, Sim picks the answer shares of the honest verifiers uniformly at random subject to the constraint that $\vec{a}_l = \sum_{j \in [k]} \vec{a}_{j,l}$ for every $1 \leq l \leq r$.

4. In the last round, Sim needs to simulate the message which V_1 broadcasts only if V_1 is honest. (Notice that if V_1 is honest at this point of the simulation, then it was also honest in the previous round, so Sim did not need to simulate the incoming message to V_1 in Step 3.) For every malicious V_j, Sim obtained from $\mathcal{A}$ the answer shares $\vec{a}^*_{j,1}, \ldots, \vec{a}^*_{j,r}$ (one share for each phase) which the corrupted V_j sent to V_1. (This was sent in the one but last round, during which V_1 is honest.) Then, Sim computes the answer shares $\vec{a}_{j,1}, \ldots, \vec{a}_{j,r}$ which V_j *should* have sent to V_1 (the answer shares depend only on x^j and $\pi_{j,1}, \ldots, \pi_{j,r}$, which are both known to Sim). Next, for every $1 \leq l \leq r$, Sim computes $\vec{a}^*_l = \vec{a}^l + \sum_{i \in T} \left(\vec{a}^*_{j,l} - \vec{a}_{j,l}\right)$. Finally, Sim emulates the FLIOP-verifier with randomness $\mathsf{R}_1, \ldots \mathsf{R}_r$ and oracle answers $\vec{a}^*_1, \ldots, \vec{a}^*_r$ and sends the output of the FLIOP-verifier as the message from V_1.

We now show that the simulated and real-world adversarial views are identically distributed. First, the randomness $\mathsf{R}_1, \ldots, \mathsf{R}_l$ of the FLIOP-verifier, as well as the proof shares $\pi_{j,l}$ provided to the corrupted verifiers throughout the execution, are identically distributed in the simulation and the real-world execution, so we can condition both views on these values. Additionally, the simulated oracle answers $\vec{a}_1, \ldots, \vec{a}_l$ of the FLIOP-simulator $\mathsf{Sim}_{\mathsf{in}}$ are identically distributed to the real-world FLIOP oracle answers, by the perfect strong HVZK of the FLIOP, so we can further condition both views on the oracle answers.

Next, we show that the answer shares $(\vec{a}_{j,l})_{j \in [k], l \in [r]}$ are identically distributed in the real execution and the simulation. For this, we show that in both cases the shares are uniformly random subject to the constraint that they sum up to $(\vec{a}_l)_{l \in [r]}$. To prove this, it suffices to show that each strict subset of answer shares is uniformly distributed.[42]

For every round $l \in [r]$, let Q_l denote the matrix whose columns are $\vec{q}_{l,1}, \ldots, \vec{q}_{l,s_l}$. (Notice that the oracle answer $\vec{a}_l$ is computed as $\vec{a} = (x \circ \pi_l) \cdot Q_l$, where $\circ$ denotes concatenation.) We assume that each Q_l has full rank, which is without loss of generality, otherwise queries are redundant in the sense that the answer to some queries can be computed from the answer to the other queries. For every $j \in [k]$, let $Q_l^{(j)}$ denote the matrix obtained from Q_l by restricting the first n rows to the x^j entries.

To show that the answer shares $\vec{a}_{j,l}$ are indeed random subject to the constraint $\vec{a}_l = \sum_{j \in [k]} \vec{a}_{j,l}$, it suffices to show that the restriction $\widetilde{Q}_l$ of $Q_l^{(j)}$ to all but the first n rows – i.e., to the rows that are

[42] We note that this property was indeed proven in [BBC+19a, Theorem 6.6], we include a proof here for completeness.

multiplied by $\pi_{j,l}$ when computing the oracle answers – has linearly independent columns. (Notice that the $Q_l^{(j)}, j \in [k]$ matrices differ only in the first n rows, so $\widetilde{Q}_l$ is well defined.) Indeed, since Q_l has full rank then in this case the linear system has a solution, so every $\vec{a}_l$ has the same number of sources under the linear mapping defined by $f(y) = y \cdot \widetilde{Q}_l$. Since the $\pi_{j,l}$ are uniformly random – because only a strict subset of them is revealed in each round l – this means that the $\vec{a}_{j,l}$ are also random.

Therefore, it remains to show that $\widetilde{Q}_l$ has linearly independent columns. This follows from the strong HVZK of the underlying FLIOP. Indeed, assume towards contradiction that the columns of $\widetilde{Q}_l$ are linearly dependent with some positive probability. Then conditioned on this event, there is a linear combination $\vec{\alpha}$ such that $\widetilde{Q}_l \cdot \vec{\alpha} = \vec{0}$. Given the answer shares $\vec{a}_1, \ldots, \vec{a}_l$ of the FLIOP, we can use $\vec{\alpha}$ to cancel out the dependency of $\vec{a}_1, \ldots, \vec{a}_l$ on $\pi_1, \ldots, \pi_j$, in which case we learn a non-trivial linear combination of x, in contradiction to the strong HVZK of the FLIOP (which guarantees that the answers $\vec{a}_l$ can be computed with no knowledge of x).

Finally, it follows directly from the protocol description that the simulated oracle answers $\vec{a}_1^*, \ldots, \vec{a}_r^*$ computed in Step 4 of the simulation are distributed identically to the oracle answers on which the FLIOP-verifier is emulated in the real-world proof, so the message broadcasted by V_1 is identically distributed to the real world. ■