# Protecting Distributed Primitives Against Leakage: Equivocal Secret Sharing and More

**Carmit Hazay**
Bar-Ilan University, Ramat-Gan, Israel

**Muthuramakrishnan Venkitasubramaniam**
Georgetown University, Washington, DC, USA

**Mor Weiss**
Bar-Ilan University, Ramat-Gan, Israel

## 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.

**2012 ACM Subject Classification** Theory of computation → Cryptographic primitives; Theory of computation → Cryptographic protocols

**Keywords and phrases** Leakage Resilience, Secret Sharing, Equivocal Secret Sharing, Verifiable Secret Sharing, Function Secret Sharing, Threshold Encryption, Distributed Zero-Knowledge Proofs

**Digital Object Identifier** 10.4230/LIPIcs.ITC.2022.3

**Related Version** *Full Version*: `https://eprint.iacr.org/2022/497` [58]

**Funding** *Carmit Hazay*: 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, and by ISF grant No. 1316/18.
*Mor Weiss*: 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.

**Acknowledgements** We thank the anonymous ITC'22 reviewers for helpful comments.

## 1 Introduction

Starting with the works of Kocher et al. [65, 66], and more recently in works on Meltdown and Spectre attacks [71, 64]), 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 [60], and Micali and Reyzin [74]), or *specific non-interactive primitives* such as storage (e.g., in [42, 32, 2]), signatures [20], and encryption schemes [41, 2, 77, 39, 22, 24]. (We refer the interested reader to the excellent survey by Kalai and Reyzin [62] for a discussion of works in the field.)

### 1.0.1 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 [60, 46, 79, 76, 75, 3], or assuming leakage functions operate on disjoint sets of wires of the circuit [50, 61, 51, 43, 33, 34, 52] (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 [50, 61, 52] or even indistinguishability obfuscation [52]), or incur high (polynomial) overheads. There are also works on leakage-resilient Multi-Party Computation (MPC) protocols [18], as well as works on interactive protocols (such as MPC [10, 16, 11] and zero-knowledge proofs [49, 10]) 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 [42, 32, 2, 41, 23, 40]), signatures [20], and encryption schemes [41, 2, 77, 39, 22, 24]. 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. [19] use randomness extractors to construct coin-tossing and verifiable secret sharing secure against a single leakage query (i.e., non-adaptive leakage), and [49, 10] design leakage-tolerant zero-knowledge proofs (as noted above, leakage tolerance is a weaker notion than leakage-resilience).

### 1.0.2 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 [44, 35, 53, 54, 8, 81, 4,

[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.

67, 1, 29, 69, 72, 73, 82, 28]. Focusing on secret sharing, which only protects *information* (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 [8] (though some results appeared already in earlier works, e.g., [12]), 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 2 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 A). For example, Shamir's secret sharing scheme over fields of characteristic 2 does not resist leakage of even *a single bit* from each share [55].

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 [1, 81, 67, 29, 69, 28]. 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 [67, 29, 69, 28], or global leakage (for restricted classes of permissible leakage functions) [12, 69]. Unfortunately, the stronger leakage-resilience guarantee does not come without a price: these constructions are typically complex [67, 29, 69, 28]; some works are considerably less efficient than standard (non-LR) schemes [53, 54, 29]; 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 [44, 35, 53, 54, 1, 81, 4, 67, 29, 69, 28]. Some of these works describe a general compiler that transforms any secret sharing schem into a leakage-resilient one [81, 1, 29], while others describe specially-tailored constructions [53, 67, 69, 28].

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?*

[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 [81, 1, 29] construct general compilers. However, [81, 1] resists only non-adaptive leakage and the construction of [29] has low rate.

More generally, we study the leakage-resilience of distributed cryptographic primitives such 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 [19] 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 A for a detailed discussion of the VSS scheme of [19], 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.*

### 1.0.3 Our leakage model: adaptive probing-resilience for data at rest

We focus on protecting cryptographic primitives against *local probing leakage.* 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 [8]), and can be seen as a combination of the leakage model of [60] that restricts the function class from which leakage functions can be chosen, and the "Only Computation Leaks" (OCL) model of [74] 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+ [9].)

[4] For example, in the TES based on Diffie-Helman, parties hold public-key shares of the form $g^{\text{SK}_1}, \ldots, g^{\text{SK}_m}$, where $g$ is a public group generator, and $\text{SK}_1, \ldots, \text{SK}_m$ are the secret-key shares. In this case, the public key is $g^{\sum_i \text{SK}_i}$ and the secret key $\text{SK} = \sum_i \text{SK}_i$ is unknown to any party. See the full version [58] 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 the full version [58] for further details on the security definition.

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 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 [83]). 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 [17]) – 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 [7] 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 the full version [58], and note that in some cases (e.g., for certain TES constructions) we are able to protect *the entire process* – including computation – from leakage.

### 1.0.4 Beyond leakage-resilience

In the context of secret sharing, we also explore a stronger leakage-resilience guarantee known as *equivocation*, recently introduced by [57] 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 [57] is a degenerate one-party scheme,[6] so the adversary cannot obtain the share in full. Therefore, we ask:

*Can we construct* multiparty *equivocal SSSs secure against probing leakage?*

## 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.*

### 1.1.1 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 $\mathsf{Acc}$, 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 A 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* [30, 6, 31, 5], see Section 2).

Our construction is summarized in the following theorem, where $(\mathsf{Acc}, \ell)$-equivocation means the adversary can *adaptively* obtain the shares of any unauthorized set $T \notin \mathsf{Acc}$, as well as *adaptively* probe $\ell$ bits from every other share, and $\tau$-tampering-resilience guarantees that the correct secret will be reconstructed even if an external adversary can arbitrarily modify $\tau$ bits in *every* share. (See the full version [58] for the formal statement.)

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

▶ **Theorem 1** (Equivocal SSS from standard SSS – informal). *Let* $\mathsf{SSS}$ *be an* $m$*-party secret sharing scheme for secrets in* $\mathcal{S}$ *and an access structure* $\mathsf{Acc}$*, with shares of length* $N$*. Then there exists an* $\ell = \Omega(N)$ *for which there exists an* $m$*-party secret sharing scheme* $\mathsf{SSS}'$ *for secrets in* $\mathcal{S}$ *and access structure* $\mathsf{Acc}$*, with* $\ell$*-tampering-resilience, and* $(\mathsf{Acc}, \ell)$*-equivocation. Moreover, the secret shares have length* $O(N)$*. Furthermore, if* $\mathsf{SSS}$ *has additive reconstruction, then* $\mathsf{SSS}'$ *has linear reconstruction.*

In Table 1 (Page 12) we compare our equivocal SSS with existing LR-SSSs. Applying Theorem 1 to Shamir's secret sharing, we obtain the following (see the full version [58] for the formal statement).

▶ **Corollary 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* $(\mathsf{Acc}, \ell)$*-equivocal and* $\ell$*-tampering-resilient secret sharing scheme, where* $\mathsf{Acc}$ *consists of all subsets of* $[m]$ *of size more than* $t$*. Moreover,* $\ell = \Omega(N)$*, where* $N$ *is the share length.*

Corollary 2 demonstrates that by applying our transformation to Shamir's secret sharing scheme, we can circumvent the impossibility result of [55] on the leakage-insecurity of Shamir's scheme over fields of characteristic 2, as well as the lower bounds of [8] and [78] (which are discussed in Section A), while still preserving the main features of Shamir's scheme.

### 1.1.2 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+ [9] 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. (See the full version [58] for the constructions and formal theorem statements.)

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,

[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 [42, 32]. However, due to the distributed nature of the primitives we consider, our security definitions are more intricate than in these aforementioned works.

and prove that existing protocols (e.g., a protocol of [13]) are adaptively-secure, a result which may be of independent interest. See Section 2, and the formal definitions in the full version [58], 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. [26], 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 [80, 27].[8]

*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

[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.

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

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 the full version [58].

### 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 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.

## 2 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 $\mathsf{msg}$, and given any arbitrary message $\mathsf{msg}'$, the leakage can be "explained" in retrospect as the leakage on an encoding of $\mathsf{msg}'$. We note that such leakage-resilience and equivocation properties necessitate *randomized* encoding. Fortunately, such encodings exist.

### 2.0.1 Main building block: Reconstructible Probabilistic Encodings (RPEs)

RPEs [30, 6, 31, 5] are, roughly, linear codes with an equivocation property. More specifically, an RPE consists of a randomized encoding procedure $\mathsf{Encode}$, a deterministic linear decoder $\mathsf{Decode}$, and a randomized resampler algorithm $\mathsf{Rec}$, where $\mathsf{Encode}\,(\mathsf{msg})$ outputs a random codeword from a set of possible codewords for $\mathsf{msg}$. (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 $\mathsf{msg}$.) 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 $\mathsf{msg}$, any subset $\mathcal{I}$ of probed bits $\mathsf{leak}$ from a random encoding $c \leftarrow \mathsf{Encode}\,(\mathsf{msg})$ of $\mathsf{msg}$, and any message $\mathsf{msg}'$, $\mathsf{Rec}\,(\mathcal{I}, \mathsf{leak}, \mathsf{msg}')$ outputs an encoding $c'$ of $\mathsf{msg}'$ which is statistically close to an encoding of $\mathsf{msg}'$ 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 the full version [58].

## 2.1 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 [57], who used the equivocation of RPEs to construct zero-knowledge probabilistic proof systems. More specifically, Hazay et al. [57] 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 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

[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 $\mathsf{msg}$ are uniformly distributed, and the output of $\mathsf{Rec}$ is distributed identically to a random encoding of $\mathsf{msg}'$ 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.

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 $\mathsf{msg}'$. At this point, secrecy of the underlying SSS guarantees the simulator can generate a secret sharing of $\mathsf{msg}'$ 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 is more complex; see the full version [58] for details.) Finally, tampering-resilience follows from the error-correction of the RPE.

The equivocal SSS described in Corollary 2 is obtained by applying our compiler to Shamir's secret sharing and the RPE of [36, 37]. More specifically, Decatur et al. [36, 37] 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 [5, Lemma 2]).

Table 1 compares our equivocal SSS construction to known LR-SSSs. We elaborate on these schemes further in Section A.

**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.

| | Scheme | Equiv. | Adaptive Queries | Linear Recon. | Rate | Leakage Rate | Leakage Model |
|---|---|---|---|---|---|---|---|
| Specialized | [53, 54] | No | No | No | $O(\frac{1}{m})$ | $O(\frac{1}{m})$ | Local |
| | [8] | No | No | Yes | $O(1)$ | $O(1)$ | Local |
| | [67] | No | Yes | No | $O(\frac{1}{\mathsf{poly}(m)})$ | $O(\frac{1}{\mathsf{poly}(m)})$ | Joint |
| | [28] | No | Yes | No | $O(1)$ | $O(1)$ | Restricted Joint |
| General | [81] | No | No | No | $O(1)$ | $O(1)$ | Local |
| | [29] | No | Yes | No | $O(\frac{1}{\mathsf{poly}(m)})$ | $O(\frac{1}{\mathsf{poly}(m)})$ | Joint |
| | **This work** | **Yes** | **Yes** | **Yes** | $O(1)$ | $O(1)$ | **Probing** |

## 2.2 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.

#### 2.2.0.1 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 $\mathsf{VSS}$ into a Leakage-Resilient VSS (LR-VSS) $\mathsf{VSS}'$ by having the dealer generate the shares as in $\mathsf{VSS}$, and then the parties verify the shares by running the verification protocol of $\mathsf{VSS}$. 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.

[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.

#### 2.2.0.2 Distributed Zero-Knowledge (dZK) Proofs

Distributed ZK proofs [14] 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., [14]) 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 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 – in the full version [58] we extend the ZK definition of dZK to adaptive corruptions, and prove that a protocol of [13] is adaptively-secure. 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 $\Pi$ as follows. The prover in our LR-dZK proof emulates the prover of $\Pi$, and RPE-encodes the proof shares before sending them to the verifiers. The verifiers store these encoded proof shares, except when $\Pi$ 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.

#### 2.2.0.3 Function Secret Sharing (FSS)

Function Secret Sharing (FFS) [17, 15] 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 [17] 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 $\mathsf{Eval}$. 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 $\mathsf{Eval}$ 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.

#### 2.2.0.4 Threshold Encryption Schemes (TES) and Threshold Signatures

Lastly, we study the usefulness of our paradigm in the context of threshold cryptography [38, 70, 47, 56] 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 [38] that serves as a threshold public key encryption scheme for El Gamal [48].

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 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.

[12] We note that the actual formulation is more involved as it needs to eliminate chosen ciphertext attacks.

## References

1 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.

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

3 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.

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

5 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.

6 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.

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

8 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.

9 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.

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

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

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

13 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. URL: `https://eprint.iacr.org/2019/188`.

14 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.

15 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.

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

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

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

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

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

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

22 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.

23 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.

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

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

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

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

28 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.

29 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.

30 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.

31 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.

32 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.

33 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.

34 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.

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

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

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

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

39 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, 2010.

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

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

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

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

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

45 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.

46 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.

47 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.

48 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.

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

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

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

52 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.

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

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

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

56 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.

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

58 Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, and Mor Weiss. Protecting distributed primitives against leakage: Equivocal secret sharing and more. *IACR Cryptol. ePrint Arch.*, 2022:497, 2022.

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

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

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

62 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. ACM, 2019.

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

64 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.

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

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

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

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

69 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.

70 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.

71 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.

72 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.

73 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.

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

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

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

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

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

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

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

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

82 Ivan Tjuawinata and Chaoping Xing. Leakage-resilient secret sharing with constant share size. *CoRR*, abs/2105.03074, 2021. `arXiv:2105.03074`.

83 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 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 [62], 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).

## A.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 [8], 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 [60] (that restricts the function class from which leakage functions can be chosen), and the "Only Computation Leaks" (OCL) model of [74] (which assumes that different memory components leak separately), though, as noted above, our constructions achieve the stronger OCL+ property [9]. 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 [42, 32], bounded memory leakage [2], and several other models (see [62] and references therein). We elaborate more on these works, and their connection to our model, in Section A.3 below.

## A.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 [44] (under the name "intrusion-resilient secret-sharing"), and explicitly (and independently) defined by [53, 8]. Following [53, 8], there has been a long line of works exploring various aspects of leakage-resilient secret-sharing [44, 35, 53, 54, 8, 81, 4, 67, 1, 81, 29, 69, 72, 73, 28]. These works consider different leakage models, as we now explain.

**Local Leakage Model.** The intrusion-resilient secret-sharing of [44] 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. [35] designed a 2-out-of-2 secret-sharing scheme with *non-interactive* reconstruction. (We note that [42] allowed leakage of full shares, but [35] do not.)

More recently, the works of [53, 8] 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 [53] 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 [81], 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. [8] 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 [55] on secret sharing over fields of characteristic 2.[13] Specifically, [8] proved that Shamir's scheme has constant leakage rate for thresholds $t = m - o(m)$. This was recently improved by Maji et al. [73] who extended the analysis of [8] 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 [55], and similar insecurities hold more generally for fields of small characteristic (not necessarily 2) [8]. In terms of the reconstruction threshold $t$ in $m$-party schemes, the analysis of Benhamouda et al. [8] 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) [78], which can be circumvented by instantiating Shamir's scheme using random evaluation points [73].

**Joint Leakage Model.** Kumar et al. [67] extend the works of [53, 54, 8] 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 [67] 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. [29], who design LR-SSSs withstanding larger collusion bounds of up to $p = t/\log t$ parties, 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. [28], who obtain constant rate and leakage rate in this restricted leakage model.

**Global Leakage Model.** Another line of works [12, 69] consider *global* leakage which is computed jointly on *all* shares, but restrict the class of permissible leakage functions. Bogdanov et al. [12] 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 $t = \omega\left(\mathsf{polylog}\left(m\right)\right)$. The work of [69] designed LR-SSSs that are secure against affine leakage functions over $\mathbb{F}_2$. These works are

[13] We note that [8] extended this result by showing attacks on secret sharing over fields of small characteristic (not necessarily 2).

[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 $\mathsf{sgn}$ is the sign function.

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 [67, 29, 28, 69], 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.

## A.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 [42, 32], bounded memory leakage [2], auxiliary-input memory leakage [41], and continual memory leakage [23, 40]. 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 [35] – which aims to protect (properly encoded) storage – is similarly a model of leakage-resilience for data at rest. Similar to our model, the goal 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.

## A.4 Leakage-Resilient Distributed Primitives

Boyle et al. [19] 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 [19], 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 [21]), 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 $\ell$ information bits throughout the entire lifetime of the system, where $\ell$ 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 [49, 10] as well as MACs, commitment schemes, and OT [10] 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.

## A.5 Leakage-Resilient MPC and General Computations

Following the works of [60, 74], there has been a long line of works on protecting general computations against leakage (see [62] 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 [18, 8], or allowing some leakage in the ideal world [10, 16, 11]. 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 [18], or a poor leakage rate [8].

## A.6 Tampering-Resilience

The notion of tampering-resilience for secret-sharing, also known as *non-malleable* secret-sharing [53], has received a lot of attention lately [53, 54, 81, 4, 45, 1, 25, 67, 68, 29, 28, 63]. 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 [59], are unrelated to our notion of tampering since they aim at protecting *secrecy*, whereas our goal is to maintain *correctness.*