ZK-PCPs from Leakage-Resilient Secret Sharing

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

Muthuramakrishnan Venkitasubramaniam ✉
University of Rochester, NY, USA

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

Abstract

Zero-Knowledge PCPs (ZK-PCPs; Kilian, Petrank, and Tardos, STOC '97) are PCPs with the additional zero-knowledge guarantee that the view of any (possibly malicious) verifier making a bounded number of queries to the proof can be efficiently simulated up to a small statistical distance. Similarly, ZK-PCPs of Proximity (ZK-PCPPs; Ishai and Weiss, TCC '14) are PCPPs in which the view of an adversarial verifier can be efficiently simulated with few queries to the input.

Previous ZK-PCP constructions obtained an exponential gap between the query complexity q of the honest verifier, and the bound q^* on the queries of a malicious verifier (i.e., $q = \mathsf{poly}\log(q^*)$), but required either exponential-time simulation, or adaptive *honest* verification. This should be contrasted with standard PCPs, that can be verified non-adaptively (i.e., with a single round of queries to the proof). The problem of constructing such ZK-PCPs, *even when* $q^* = q$, has remained open since they were first introduced more than 2 decades ago. This question is also open for ZK-PCPPs, for which no construction with non-adaptive honest verification is known (not even with exponential-time simulation).

We resolve this question by constructing the *first* ZK-PCPs and ZK-PCPPs which *simultaneously* achieve *efficient* zero-knowledge simulation and *non-adaptive* honest verification. Our schemes have a square-root query gap, namely $q^*/q = O\left(\sqrt{n}\right)$ where n is the input length.

Our constructions combine the "MPC-in-the-head" technique (Ishai et al., STOC '07) with leakage-resilient secret sharing. Specifically, we use the MPC-in-the-head technique to construct a ZK-PCP variant over a large alphabet, then employ leakage-resilient secret sharing to design a new alphabet reduction for ZK-PCPs which preserves zero-knowledge.

2012 ACM Subject Classification Security and privacy → Information-theoretic techniques; Theory of computation → Cryptographic protocols; Security and privacy → Cryptography; Theory of computation → Proof complexity

Keywords and phrases Zero Knowledge, Probabilisitically Checkable Proofs, PCPs of Proximity, Leakage Resilience, Secret Sharing

Digital Object Identifier 10.4230/LIPIcs.ITC.2021.6

Related Version *Full Version*: https://eprint.iacr.org/2021/606 [20]

Funding The first and third authors are 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 is supported by ISF grant No. 1316/18. The first and second authors are supported by DARPA under Contract No. HR001120C0087. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA.

Acknowledgements We thank the anonymous ITC'21 reviewers for their helpful comments, in particular for pointing out the connection to RPEs and noting that the ZK code of [14, Theorem 2.2] is equivocal.

1 Introduction

Probabilistically Checkable Proofs (PCPs) [2, 3] allow a probabilistic verifier to check the validity of a given statement by only querying few proof bits. Zero-Knowledge (ZK) proofs [18] allow a prover to convince a verifier of the validity of a statement, without revealing any additional information to the verifier. This work focuss on *Zero-Knowledge Probabilistically Checkable Proofs (ZK-PCPs)* (and ZK-PCPs *of proximity*), which combine the advantages of these two types of proof systems. Before describing our main results, we first give a short overview of these proof systems.

Probabilistically Checkable Proofs (PCPs)

PCPs [2, 3] allow a randomized efficient verifier $\mathcal{V}$ with oracle access to a purported proof π to verify an NP-statement of the form "$x \in \mathcal{L}$" by reading only few bits of π. The proof can be efficiently generated given the NP witness, and the verifier accepts true claims with probability 1, whereas false claims are accepted with low probability (which is called the soundness error). The celebrated PCP theorem [2, 3, 15] states that any NP language has a PCP system with soundness error 1/2, in which the verifier reads only $O(1)$ bits from a polynomial-length proof (soundness can be amplified through repetition). An attractive feature of these PCP systems is that the verifier is *non-adaptive*, namely it makes a single round of queries to the proof. *PCPs of Proximity (PCPPs)* [16, 6, 15] are a generalization of PCPs in which the verifier does not read its entire input. Instead, $\mathcal{V}$ has oracle access to x, π, and wishes to check whether x is close to $\mathcal{L}$ (in relative Hamming distance). The best PCPP constructions for NP [7, 27] obtain comparable parameters to the PCP systems described above, where any x which is δ-far from $\mathcal{L}$ in relative Hamming distance is rejected with high probability, and δ is a constant or even inverse polylogarithmic (δ is called the *proximity parameter*).

Zero-Knowledge (ZK) proofs

ZK proofs [18] allow a randomized efficient prover to prove an NP-statement of the form "$x \in \mathcal{L}$" to a randomized efficient verifier, while guaranteeing that true claims are accepted with probability 1, false claims are rejected with high probability, and the verifier learns no information about the corresponding NP-witness. This last property, known as *zero-knowledge*, is formalized by requiring that for any (possibly malicious) efficient verifier $\mathcal{V}^*$, there exists an efficient *simulator* machine that has access only to the statement x, and can simulate the interaction of $\mathcal{V}^*$ with the honest prover.

Zero-Knowledge PCPs (ZK-PCPs)

ZK-PCPs [26] combine the attractive features of PCPs and ZK proofs. Specifically, ZK-PCPs are PCPs in which the prover $\mathcal{P}$ is randomized, and the proof π has the following zero-knowledge guarantee: the view of every (possibly malicious, possibly unbounded) verifier $\mathcal{V}^*$ that makes an a-priori bounded number of queries to the proof, can be efficiently simulated up to a small statistical distance. We remark that restricting $\mathcal{V}^*$ to making an a-priori bounded number of queries is inherent to obtaining ZK with polynomial-length proofs.

The first ZK-PCP constructions for NP [26, 22] obtain ZK against any verifier $\mathcal{V}^*$ that is restricted to querying at most $q^* = q^*(|x|)$ proof bits, with proofs of length $\mathsf{poly}(q^*)$ that can be verified with $\mathsf{polylog}(q^*)$ queries and have a negligible soundness error. In particular, the *query gap* q^*/q – the ratio between the query complexities of the malicious and honest verifiers – obtained by these constructions is exponential.[1] Unfortunately, obtaining ZK in [26, 22] did not come without a cost: it required the *honest* verifier to be adaptive, namely to make *several* rounds of queries to the proof (where the queries of each round depend on the answers to previous queries). In cryptographic applications of ZK-PCPs (e.g., in [25]) this blows-up the round complexity of resultant protocols. In particular, every round of queries which the verifier makes to the ZK-PCP induces two communication rounds in the interactive protocols which rely on ZK-PCPs.

Ishai and Weiss [24] introduce the notion of *Zero-Knowledge PCPPs (ZK-PCPPs).* Similar to ZK-PCPs, the ZK-PCPP prover is randomized, and zero-knowledge means that the view of any verifier $\mathcal{V}^*$ making q^* queries to *the input* and the proof can be efficiently simulated, up to a small statistical distance, *by making only q^* queries to the input.* They use similar techniques to [26, 22] to obtain ZK-PCPPs for NP with comparable parameters to the ZK-PCPs of [26, 22], where the proximity parameter δ is constant or inverse polylogarithmic. These ZK-PCPPs also require adaptive verification, which increases the round complexity in their cryptographic applications [24, 30].

As discussed in Sections 2.1.4 and 2.2.2 below, adaptive verification is in fact inherent to the constructions of [26, 22, 24]. Indeed, these schemes are obtained by combining a PCP or PCPP with a weak zero-knowledge guarantee that only holds against the *honest* verifier, with an information-theoretic commitment primitive called locking schemes [26]. This latter primitive requires adaptive opening, which causes the resultant ZK-PCP verifier to be adaptive.

Can ZK-PCPs be verified non-adaptively?

Motivated by the goal of obtaining ZK for PCPs at no additional cost, Ishai et al. [25] gave a partial answer to this question. Specifically, they construct ZK-PCPs with similar parameters to the schemes of [26, 22] in which the honest verifier is *non-adaptive*, but with a weaker zero-knowledge guarantee compared to standard ZK-PCPs: the zero-knowledge simulator is inefficient (this is also known as *witness-indistinguishability*). Alternatively, they obtain ZK with efficient simulation against *computationally-bounded* verifiers, assuming the existence of one-way functions and a common random string. The techniques of [25] diverge from the standard method of designing ZK-PCPs [26, 22] discussed above. Specifically, the ZK-PCP of [25] is based on a novel connection to *leakage-resilient circuits*, which are circuits that operate over encoded inputs, and resist certain "side channel" attacks in the sense that such attacks reveal nothing about the input other than the output. Unfortunately, the weaker ZK guarantee of the ZK-PCPs of [25] carries over to any application in which these systems are used. Moreover, [25] give evidence that inefficient simulation is inherent to their technique of using leakage-resilient circuits.

Non-adaptive honest vs. malicious verification

It is instructive to note that while having non-adaptive *(honest)* verification is a feature of the system (since it guarantees that the honest verifier can achieve soundness with a single round of queries), having zero-knowledge against *non-adaptive malicious* verifiers is a restriction of the system, since there is no ZK guarantee against *adaptive* malicious verifiers, that make several rounds of queries to the proof.

[1] We stress that a larger gap is preferable to a smaller one, since it means the proof can be verified with few queries, while guaranteeing zero-knowledge even when a malicious verifier makes many more queries (compared to the honest verifier).

We note that in [25], leakage-resilient circuits fall short of yielding ZK-PCPs with full-fledged zero-knowledge not only because the simulation is inefficient, but also because *zero-knowledge holds only against non-adaptive (malicious) verifiers.* Ishai et al. [25] obtain ZK (with inefficient simulation) against adaptive verifiers by combining leakage-resilient circuits with techniques of [9]. These techniques incur an exponential blowup in the complexity of the ZK simulator, but did not pose a problem for [25] since their simulator (even against *non-adaptive malicious* verifiers) was already inefficient.

The current landscape of ZK-PCPs is unsatisfying. Current ZK-PCP constructions either require adaptive verification [26, 22], or guarantee only a weak form of ZK with an inefficient simulator [25]. This holds regardless of the query gap, i.e., even if we restrict malicious verifiers to making *the same* number of queries as the honest verifier. For ZK-PCPPs, the situation is even worse: *no* constructions with non-adaptive verification are known (not even with inefficient simulation). This state of affairs gives rise to the following natural question:

Do there exist ZK-PCPs (and ZK-PCPPs) with non-adaptive verification and efficient simulation?

As we discuss in Sections 2.1.4 and 2.2.2 below, the limitations of existing ZK-PCP and ZK-PCPP constructions seem to be inherent to the respective techniques they employ to obtain ZK. This seems to imply that obtaining both non-adaptive verification and efficient simulation requires new techniques. Or maybe such objects do not even exist?

1.1 Our results

In this work, we answer our research question in the affirmative: we construct ZK-PCPs and ZK-PCPPs that can be verified non-adaptively and have efficient zero-knowledge simulation. Unlike the schemes of [26, 22, 24, 25], which obtain an exponential gap between the query complexities of the malicious and honest verifiers, we are only able to obtain a *polynomial* query gap (q^* vs. $(q^*)^\epsilon$, for some constant $\epsilon \in (0,1)$).

In the following, we say that a PCP (PCPP, resp.) system is a non-adaptive q-query q^*-ZK-PCP (q^*-ZK-PCPP, resp.) if it is perfectly ZK against a (possibly malicious, possibly adaptive) verifier making q^* queries, and achieves a $\mathsf{negl}(q^*)$ soundness error where the honest verifier makes q non-adaptive queries to the proof.

Specifically, we obtain the following results:

▶ **Theorem 1** (Non-Adaptive ZK-PCPs with Efficient Simulation). *There exists a constant $\epsilon \in (0,1)$ such that for any ZK parameter $q^* \in \mathbb{N}$ there exists a non-adaptive $(q^*)^\epsilon$-query $\Omega(q^*)$-ZK-PCP for NP.*

▶ **Theorem 2** (Non-Adaptive ZK-PCPPs with Efficient Simulation). *Let $n \in \mathbb{N}$ be an input length parameter. Then there exists a constant $c > 0$ such that for any proximity parameter $\delta \geq 1/\sqrt{n}$, there exists a non-adaptive q-query q^*-ZK-PCPP for NP with proximity parameter δ, $q^* = \Omega\left(n^{c+1}\right)$, and $q = \tilde{O}\left(n^{c+1/2}\right)$.*

Our non-adaptive ZK-PCPs and ZK-PCPPs can be plugged-into the applications described in [24, 25, 30], and will reduce the round complexity of the resultant protocols.[2]

[2] In this context, we note that if one only requires ZK against the *honest* verifier, then non-adaptive ZK-PCPs and ZK-PCPPs are known. (This is implicit in [26] and [24] for ZK-PCPs and ZK-PCPPs respectively, via standard soundness amplification.) Consequently, our non-adaptive ZK-PCPs and ZK-PCPPs (with ZK against *malicious* verifiers) do not improve the round complexity in applications that only require ZK against the honest-verifier (e.g., the ZK arguments of [22], and the commit-and-prove protocols of [24]).

Our constructions show that leakage-resilience techniques *can* be used to construct ZK-PCPs (and ZK-PCPPs) with both non-adaptive (honest) verification and efficient simulation. Specifically, we circumvent the negative result of [25] on the limitations of using leakage-resilient *circuits*, by relying on leakage-resilient *secret sharing* [17, 12] secure against local leakage [19, 8, 1]. Compared to leakage-resilient circuits, leakage-resilient secret sharing has the weaker guarantee of only protecting *information* from leakage, whereas leakage-resilient circuits also protect *computation*. However, this weaker guarantee suffices for our needs, and admits leaner and more efficient constructions compared to those of leakage-resilient circuits (and applications using them). Specifically, we use leakage resilient secret sharing to design a new alphabet reduction procedure that transforms a ZK-PCP over a large alphabet to a ZK-PCP over bits, while preserving zero-knowledge.

2 Our Techniques

We now give more details about our ZK-PCP and ZK-PCPP constructions.

2.1 ZK-PCPs with Non-Adaptive Verification and Efficient Simulation

Our starting point is a ZK-PCP implicit in the work of [21]. They use secure Multi-Party Computation (MPC) protocols to construct a ZK-PCP variant *over a large (poly-sized) alphabet* with efficient ZK simulation, that can be verified non-adaptively. Their ZK-PCP suffers from two disadvantages. First, strictly speaking it is not a ZK-PCP, since in standard ZK-PCPs the proof is a bit string, whereas the ZK-PCP of [21] is over a large alphabet. Second, their construction has no query gap, namely the proof is ZK against verifiers querying q^* proof symbols, but to get soundness the honest verifier must also make q^* queries.

2.1.1 Amplifying the Query Gap in the ZK-PCP of [21]

To prove that $x \in \mathcal{L}$ for some NP-language $\mathcal{L}$ with a corresponding NP relation $\mathcal{R} = \mathcal{R}(x, w)$, Ishai et al. [21] employ an n-party protocol that computes the function $f_{\mathcal{R}}(x, w_1, \ldots, w_n) = \mathcal{R}(x, \oplus w_i)$, where x is a common input, and w_i is the input of the ith party P_i. The prover executes the MPC protocol "in its head", obtaining the *views* of all parties $P_1, \ldots, P_n$ in the execution (the view of party P_i consists of its input, random input, and all messages it received during the execution). The proof consists of all these views, where each view is a symbol in the resultant proof. To verify the proof, the verifier reads several views, and checks that: (1) the output reported in all views is 1; and (2) the views are *consistent*, namely for every pair V_i, V_j of queried views of P_i, P_j (respectively), the incoming messages from P_i reported in V_j are the messages P_i would send in the protocol given its view V_i, and vice versa.

To get q^*-ZK, the protocol should be *private* against q^* (semi-honest) parties, in the sense that they learn nothing from the execution except their inputs and the output. For soundness, [21] rely on a notion of correctness against q^* corrupted parties (known as *robustness*), guaranteeing that even if q^* parties arbitrarily deviate from the protocol, they cannot cause an honest party to output 1 in a protocol execution on $x \notin \mathcal{L}$. We revisit their analysis, and show that general MPC protocols yield a square root query gap. That is, given a q^*-private and q^*-robust MPC protocol, the resultant ZK-PCP over a large alphabet is ZK against a (possibly malicious) verifier querying q^* proof symbols, and can be non-adaptively verified with only $\sqrt{q^*}$ queries, with a negligible soundness error. This already yields a non-trivial ZK-PCP *over a large alphabet*. Our tighter analysis can be found in the full version [20].

2.1.2 Alphabet Reduction for ZK-PCPs

Next, we address the fact that the ZK-PCP of [21] is over a large alphabet. For standard PCPs, one can easily reduce the alphabet Σ over which the proof π is defined to $\{0,1\}$ by simply replacing each alphabet symbol with a bit string, thus obtaining a new proof π' over $\{0,1\}$. This would increase the proof length and the query complexity of the honest verifier by a multiplicative $\log|\Sigma|$ factor, but would not otherwise affect the system.[3]

Unfortunately, applying this transformation to *zero-knowledge* PCPs might render the resultant scheme totally insecure. Indeed, while the system would still be ZK against verifiers making q^* queries, the query gap now reduces since the query complexity of the *honest* verifier (i.e., the number of queries it *must* make to obtain soundness) increases. Specifically, depending of $|\Sigma|$, the honest verifier might now need to make $> q^*$ queries, but π' might not be ZK even against $q^* + 1$ malicious queries. As a result, π' might not be ZK even against *malicious* verifiers that make *fewer* queries than the honest verifier! Indeed, a malicious verifier $\mathcal{V}^*$ with oracle access to π' is not restricted to querying "whole" symbols of π, i.e., reading the entire substring of π' that corresponds to a symbol of π. On the contrary, $\mathcal{V}^*$ might read *"parts"* of symbols, thus potentially gaining (partial) information on $q^* + 1$ symbols of π, and possibly violating the ZK guarantee of the original system.

The trivial alphabet reduction for PCPs described above fails because querying *even a single bit* in the bit string s_σ representing a symbol $\sigma \in \Sigma$ might reveal information about σ. Therefore, to make this alphabet reduction work for *zero-knowledge* PCPs, we must guarantee that querying few bits of s_σ reveals nothing about σ. We do so using leakage-resilient secret sharing.

At a high level, a (t-threshold) *Secret Sharing Scheme (SSS)* is a method of distributing a secret s among n parties by giving each party P_i a share Sh_i, such that any t shares reveal no information about s, but any $t+1$ shares completely determine s. A *Leakage-Resilient Secret Sharing Scheme (LR-SSS)* against local leakage [19, 8, 1] has the added feature of resisting leakage on the shares, in the following sense. The secret s remains hidden given t shares, *as well as few leakage bits computed separately on every other share.*

Given a ZK-PCP system $(\mathcal{P}, \mathcal{V})$ over a large alphabet Σ, and a LR-SSS for secrets in $\{0,1\}^{\log|\Sigma|}$, our alphabet reduction works as follows. The prover $\mathcal{P}'$ uses $\mathcal{P}$ to generate a proof $\pi = \sigma_1 \ldots \sigma_N$ over Σ, replaces every σ_i with its bit-representation s_{σ_i}, which it secret shares using the LR-SSS. The proof π' consists of the secret sharings of $s_{\sigma_1}, \ldots, s_{\sigma_N}$. To verify the proof, the verifier $\mathcal{V}'$ emulates $\mathcal{V}$, where a query Q of $\mathcal{V}$ to its proof π is answered as follows. $\mathcal{V}'$ uses the secret sharing of s_{σ_Q} (from its own proof oracle π') to reconstruct σ_Q, which it then provides to $\mathcal{V}$.[4]

The PCP system obtained through the reduction preserves the completeness and soundness of $(\mathcal{P}, \mathcal{V})$, and guarantees ZK against *non-adaptive* (possibly malicious) verifiers that are restricted to making (roughly) $q^{**} = q^*\ell t$ queries to the proof, where $(\mathcal{P}, \mathcal{V})$ is ZK against verifiers querying q^* proof symbols, and the LR-SSS is private against any t shares as well as ℓ leakage bits from every other share.

[3] We note that several PCP constructions (e.g. [15]) use more elaborate alphabet reduction techniques *for efficiency reasons* (in particular, their goal is to achieve quasi-linear length proofs with $O(1)$ query complexity and a constant soundness error). A $\log|\Sigma|$ blowup is less significant in the context of *zero-knowledge* PCPs, where the query complexity is anyway $\omega(1)$ since we wish to have a negligible soundness error.

[4] Due to some technical issues, the construction is actually somewhat more involved, see Section 4 and the full version [20] for the construction and further details.

To see why ZK holds, we split the proof π' into N "sections", where the ith section contains the secret sharing of s_{σ_i}, and s_{σ_i} is the bit-representation of the ith symbol σ_i of the original proof π. Roughly (and somewhat inaccurately), the queries of any non-adaptive (possibly malicious) verifier $\mathcal{V}^*$ querying at most q^{**} proof bits divide the sections of π' into two groups.

1. "Light" sections, from which $\mathcal{V}^*$ reads at most ℓt bits. In particular, for each such section containing the secret shares $\mathsf{Sh}_1, \ldots, \mathsf{Sh}_n$ of the bit-representation s_σ of a symbol σ, there can be at most t shares from which $\mathcal{V}^*$ reads more than ℓ bits, and each other share is queried only ℓ times. Therefore, the leakage-resilience of the LR-SSS guarantees that s_σ (and consequently also σ) remains entirely hidden.
2. "Heavy" sections, from which $\mathcal{V}^*$ queries more than ℓt bits. Notice that there can only be at most q^* such sections, and $\mathcal{V}^*$ obtains no information about the symbols of π encoded in "light" sections, so the queries to the "heavy" sections can be simulated by the ZK of $(\mathcal{P}, \mathcal{V})$.

(The full – and accurate – analysis appears in the proof of Theorem 18, which can be found in the full version [20].)

In summary, combining a ZK-PCP over a large alphabet with a SSS that resists probing leakage, we obtain a ZK-PCP over $\{0, 1\}$. Instantiating the transformation with the ZK-PCP of [21] (with our improved analysis) together with the LR-SSS of [29] yields a ZK-PCP with the parameters of Theorem 1that is ZK against (possibly malicious and unbounded) verifiers that only make *non-adaptive* queries to their proof oracle.

2.1.3 Amplifying to ZK Against Adaptive Verifiers

The analysis of our alphabet reduction for ZK-PCPs crucially relied on the fact that the malicious verifier $\mathcal{V}^*$ was *non-adaptive*. Indeed, the queries to "light" and "heavy" sections are simulated differently (using the LR-simulator of the LR-SSS, and the ZK-simulator of $(\mathcal{P}, \mathcal{V})$, respectively), meaning the simulator for $(\mathcal{P}', \mathcal{V}')$ needs to know at the onset of the simulation which sections are "heavy" and which are "light". Obtaining ZK against *adaptive* verifiers seems to require a stronger leakage-resilience guarantee with a two-phase flavor similar to the locking schemes of [26, 22]. Specifically, in the first phase of the simulation, the LR-simulator $\mathsf{Sim}_{\mathsf{LR}}$ should be able to answer adaptive leakage queries as in a standard (adaptively-secure) LR-SSS. However, unlike standard LR-SSSs, at some point the simulation may move to a second phase. In the second phase the simulator $\mathsf{Sim}_{\mathsf{LR}}$ is given a secret s, and should be able to "explain" s by providing an entire secret sharing of s which is random subject to being consistent with the previously-simulated answers to leakage queries. We formalize this notion, introducing *equivocal SSSs* as a generalization of standard LR-SSSs, and provide a construction based on codes with leakage-resilience guarantees (see Section 2.3 for further details).

Applying our alphabet reduction to $(\mathcal{P}, \mathcal{V})$ and an *equivocal* SSS now yields a ZK-PCP with ZK against any – possibly malicious *and adaptive* – verifier $\mathcal{V}^*$ making at most q^{**} queries. Indeed, the ZK-PCP simulator Sim starts the simulation by answering all queries using the LR-simulator $\mathsf{Sim}_{\mathsf{LR}}$ of the equivocal SSS. Whenever the number of queries to a certain proof section passes some threshold, Sim uses the ZK-simulator $\mathsf{Sim}_{\mathsf{ZK}}$ to simulate the underlying symbol σ of π. Then, Sim provides the bit-representation of σ to $\mathsf{Sim}_{\mathsf{LR}}$ as the secret-shared secret. The equivocation property of the SSS guarantees that $\mathsf{Sim}_{\mathsf{LR}}$ can now "explain" this secret by providing an entire secret sharing which is consistent with the previous leakage. These secret shares can be used to answer any further queries to that section of the proof. The construction appears in Section 4.1, and the analysis can be found in the full version [20].

To sum up, combining a ZK-PCP over a large alphabet with an *equivocal* SSS against probing leakage yields a ZK-PCP over $\{0,1\}$, where zero-knowledge holds against *adaptive* verifiers. Instantiating the transformation with the ZK-PCP of [21] and the equivocal scheme described in section 2.3.2 yields a ZK-PCP with the parameters of Theorem 1.

2.1.4 Why Do Previous Approaches of Constructing Non-Adaptive ZK-PCPs Fail?

It is instructive to discuss why our approach of combining alphabet reduction with LR secret sharing succeeds in simultaneously obtaining non-adaptive verification and efficient simulation, whereas previous approaches [26, 22, 25] could only achieve one of these properties.

As noted above, the ZK-PCPs of [26, 22] are obtained by combining PCPs that are ZK against the honest verifier with locking schemes. In effect, the locking schemes are used to "force" the queries of a malicious verifier to be distributed (almost) as the queries of *the honest verifier.* This transformation causes adaptive verification due to two reasons: first, the original proof is altered in such a way that necessitates adaptive queries to verify it. Second, the locking schemes themselves require adaptive opening. We are faced with a similar challenge, where the queries of the malicious verifier might drastically deviate from those of an honest verifier (namely, $\mathcal{V}^*$ might query "parts" of symbols, whereas the honest verifier always queries whole symbols). However, instead of "forcing" the queries of $\mathcal{V}^*$ to "look" honest, we allow $\mathcal{V}^*$ to make any set of queries, but guarantee that queries to "parts" of symbols reveal no information on the underlying symbol.

The ZK-PCP of [25] uses a different approach. Their starting point is a *non-ZK* PCP, and they use leakage-resilient circuits to protect *the entire PCP generation.* That is, the queries of the verifier are interpreted as leakage on the process of generating the PCP from the NP-witness, and by protecting this entire computation from leakage, they obtain ZK. More specifically, they change the NP statement which is being verified: instead of verifying that (x,w) is a satisfying input for the verification circuit C of the NP-relation $\mathcal{R}$, the honest verifier checks whether the leakage-resilient version $\widehat{C}$ of C is satisfiable. Therefore, soundness of the resultant ZK-PCP system crucially relies on the fact that if there exists no w such that $C(x,w)=1$, then there exists no w' such that $\widehat{C}(x,w')=1$,[5] a notion which they call *SAT-respecting.* Ishai et al. [25] give evidence that SAT-respecting leakage-resilient circuits with efficient LR-simulation (for the leakage classes needed to construct ZK-PCPs) exist only for languages in BPP. The inefficient LR-simulation is the cause of ZK with inefficient simulation in their ZK-PCPs. We circumvent their negative results by using LR-*SSSs* to protect *information* – instead of using LR *circuits* to protect *computation* – and apply the LR-SSS to PCPs with ZK guarantees (whereas [25] use standard PCPs).

2.2 ZK-PCPPs with Non-Adaptive Verification and Efficient Simulation

We extend our techniques to apply to PCPs *of Proximity.* Specifically, our alphabet reduction could also be applied to ZK-PCPPs, which reduces the task of designing ZK-PCPPs with non-adaptive verification to designing such ZK-PCPPs over a large alphabet.

[5] In fact, $\widehat{C}$ operates on encoded inputs, however to simplify the discussion we disregard this at this point, and provide a more accurate discussion in Section 2.2.2 below.

2.2.1 A ZK-PCPP with Non-Adaptive Verification Over Large Alphabets

The first step is to design a ZK-PCPP over a large alphabet. We do so by presenting a variant of the system of [21] in which the verifier does not read its entire input. Recall that the proof in the ZK-PCP of [21] consists of the views of all parties in an execution of an MPC protocol for the function $f_{\mathcal{R}}(x, w_1, \ldots, w_n) = \mathcal{R}(x, \oplus w_i)$, where x is a common input, and w_i is the input of the ith party P_i. In particular, a single proof symbol (i.e., a single view) reveals the entire input x. This is problematic in the context of ZK-PCPPs, in which the proof is required to hide not only the NP-witness w, but also most of the input x, in the sense that a verifier making few queries learns only few physical bits of x.

Thus, no single party can hold the entire input x. Instead, following [24] we introduce m additional "input parties" (where $m = |x|$), such that the MPC protocol is over $m+n$ parties $P_1, \ldots, P_{m+n}$. The inputs of parties $P_1, \ldots, P_m$ are $x_1, \ldots, x_m$ (respectively), and the inputs of parties $P_{m+1}, \ldots, P_{m+n}$ are $w_1, \ldots, w_n$ (respectively). Proof generation from this revised protocol is similar to the original construction of [21], and verification is also similar, except that whenever the verifier queries a view of $P_i, i \in [m]$, it also queries x_i from its input oracle, and checks that x_i is the input of P_i reported in the view.

If the MPC protocol is q^*-private, then we are guaranteed that q^* queries to the proof reveal at most q^* bits of x. Indeed, the q^*-privacy of the protocol guarantees that the views of q^* parties reveal no information other than their inputs (which is at most q^* bits of x) and their output (which is 1).

As for soundness, we show that our improved analysis (discussed in Section 2.1.1 above) extends to PCPPs. Specifically, we show that if the MPC protocol is q^*-robust then the resultant ZK-PCPP is sound with proximity parameter $\delta \geq 1/\sqrt{|x|}$, namely the verifier rejects (with high probability) inputs that are δ-far from the language. Indeed, let x be δ-far from $\mathcal{L}$, and notice that any (possibly ill-formed) proof π^* for x determines an effective input x^* for the underlying MPC protocol (x^* is obtained by concatenating the inputs reported in the views of $P_1, \ldots, P_m$). We show that if x^* is δ-far from x then with overwhelming probability the verifier will query a view on which x, x^* differ, in which case it rejects with probability 1. Otherwise, x^* is δ-close to x, implying $x^* \notin \mathcal{L}$, in which case our improved analysis of Section 2.1.1 essentially shows that the PCPP verifier rejects with overwhelming probability. This yields the first ZK-PCPP with non-adaptive verification and efficient simulation, but the proof is over a large alphabet.

Combining this ZK-PCPP over a large alphabet with our alphabet reduction, we obtain *the first* ZK-PCPPs over $\{0, 1\}$ with non-adaptive verification. Moreover, the system has efficient ZK simulation. Specifically, combining the ZK-PCPP over a large alphabet with a LR-SSS gives a ZK-PCPP with ZK against *non-adaptive malicious* verifiers, whereas combining it with an equivocal SSS gives a full-fledged ZK-PCPP. Instantiating the equivocal SSS with the scheme of Section 2.3.2 gives a ZK-PCPP with the properties of Theorem 2. Due to space limitations, the construction and analysis of our ZK-PCPP system is deferred to the full version [20].

2.2.2 Why Do Previous Approaches of Constructing Non-Adaptive ZK-PCPPs Fail?

We now give some intuition as to why LR-SSS is useful to obtaining ZK-PCPPs with non-adaptive verification, whereas ZK-PCPP constructions based on leakage-resilient circuits seem to fail. Recall that a leakage-resilient circuit operates over encoded inputs, where leakage from some function class $\mathcal{F}$ on the internal wire values of the circuit reveals no information about the input other than the output. In particular, this implies that the input encoding should resist leakage from $\mathcal{F}$, since leakage can be applied to the input wires (which carry the encoded inputs).

Recall that the verifier wishes to verify that $(x, w) \in \mathcal{R}$, namely that (x, w) satisfies the verification circuit C of $\mathcal{R}$. When using leakage-resilient circuits to verify this claim, the circuit C is replaced with its leakage-resilient variant $\widehat{C}$, which operates over encoded inputs, where the queries of the verifier are interpreted as leakage on the wire values of $\widehat{C}$. This raises the question of how to incorporate x into the computation. Syntactically, $\widehat{C}$ cannot operate directly on the unencoded input x, but if $\widehat{C}$ operates on an encoding of x, the prover can cheat by providing an encoding of some $x^* \neq x$. (The verifier will not be able to tell the difference because the input encoding is resilient against leakage, namely against the verifier queries.) The solution of [25] is to first *hard-wire* x into C, i.e. replace C with $C_x = C(x, \cdot)$, and then generate the leakage-resilient version $\widehat{C_x}$ of C_x. The verifier will now verify that $\widehat{C_x}$ is satisfiable.

While this solves the problem for ZK-PCPs, it cannot be applied in the context of ZK-*PCPPs*. Indeed, verifying that $\widehat{C_x}$ is satisfiable requires knowing the structure of $\widehat{C_x}$. This is indeed the case for ZK-PCPs, since x is known to the verifier in its entirety, so the verifier can locally construct $\widehat{C_x}$. However, the ZK-PCPP verifier does not know all of x, nor do we want it to – the advantage of ZK-PCPPs over ZK-PCPs (which is crucially exploited by cryptographic applications of ZK-PCPPs) is that the verifier can be sublinear in the input length, and verify the claim without learning "too much" about the input. Therefore, we cannot hard-wire x into the verification circuit, and so it is unclear how the verifier would verify consistency of its own input with the one used to evaluate C.

Finally, we note that even if one were to solve this issue of how to handle the input, using leakage-resilient circuits would incur inefficient ZK simulation in the resultant ZK-PCPP, due to the negative results of [25].

2.3 Equivocal Secret Sharing

We generalize the notion of LR-SSS [17, 12] secure against local leakage [19, 8, 1] by introducing *equivocal SSSs*. We then construct a 1-party equivocal SSS based on codes with probing-resilience. We note that while a 1-party SSS is useless as a means of sharing a secret, its equivocation property gives a meaningful way of encoding a secret in a leakage-resilient (in fact, equivocal) manner. In particular, such schemes suffice for constructing ZK-PCPs and ZK-PCPPs as described in Sections 2.1 and 2.2 above. Morevoer, since 1-party schemes can be more efficient than multi-party schemes, using 1-party schemes results in more efficient ZK-PCPs and ZK-PCPPs.

2.3.1 Equivocal SSS: Definition

Recall that a standard t-threshold n-party SSS guarantees that the secret remains entirely hidden given any t of the n shares. LR-SSS enhances this privacy property to hold even given leakage on the other shares. We will focus on resisting adaptive (t, ℓ)-local probing leakage, in which the leakage reveals t shares, as well as ℓ bits from every other share. This can be formalized by comparing the real execution with an ideal experiment in which an efficient simulator Sim, that has no knowledge of the secret, answers the leakage queries.

An equivocal SSS generalizes the notion of (adaptive) LR-SSS by considering a two-phase ideal experiment, where the first phase is similar to the ideal experiment of LR-SSS. At the end of the first phase, the adversary can choose whether to continue to the second phase.

In the second phase, the simulator is given a secret s, and must generate an entire secret sharing of s, which is given to the adversary. The adversary should have only a negligible advantage in distinguishing the real execution from the ideal experiment, as long as it didn't violate the leakage restrictions. That is, as long as at the onset of the second phase, the adversary obtained at most t shares, and probed at most ℓ bits from every other share. Since the adversary can choose *not* to continue to the second phase of the simulation, this notion strictly generalizes the notion of a LR-SSS. Notice that we make no restriction on the computational power of the adversary. The definition appears in Section 3.3.

2.3.2 Equivocal SSS: Construction

We use a 1-party equivocal SSS that resists $(0, \ell)$-local probing leakage [19, 8, 1], where ℓ is a constant fraction of the share size. Considering 1-party schemes suffices for the ZK-PCP and ZK-PCPP application, and admit lean constructions that result in more efficient ZK-PCPs and ZK-PCPPs (in terms of query complexity and proof length).

2.3.2.1 Existing leakage-resilient encodings

A 1-party equivocal SSS gives a method of encoding data such that the resultant encoding is equivocal, and consequently also leakage-resilient. We note that leakage-resilient encodings have been considered before under different names. ZK codes [13, 14, 23] are encodings that resists *non adaptive probing* leakage, i.e., these are 0-threshold 1-party SSSs that resist non-adaptive probing leakage. Leakage-resilient storage [17, 12] encodes the data into two parts that resist *adaptive* leakage from *each part separately.* Thus, leakage-resilient storage can be though of as a ramp 2-party SSS which is private against 0 parties, reconstructible given both shares, and resists adaptive leakage. We note that the schemes of [17, 12] resists *general* local leakage (i.e., leakage which operates on each share separately, and has short output), and not just probing. An Equivocal SSS generalizes these notions – while ZK codes and leakage-resilient storage are only secure as long as the number of leakage bits does not pass an a-priori bound, equivocal schemes guarantee security even beyond the leakage threshold. Another related notion is that of a Reconstructable Probabilistic Encoding (RPE) [10, 5, 11, 4]. Informally, these are leakage resilient encodings that are also equivocal (with perfect leakage resilience and equivocation), with an additional error correction property. RPEs and equivocal SSSs are incomparable: while RPEs are a strengthening of *1-party* equivocal SSS (due to their error correction), (multi-party) equivocal SSSs guarantee leakage resilience even when full shares are leaked. In particular, they can potentially achieve a better leakage rate.

2.3.2.2 A specific 1-party equivocal SSS construction

Our constructions will employ the ZK code of [13, 14]. Thy construct a linear code in $\{0, 1\}^n$ with constant rate, and leakage resilience against a constant fraction of leaked bits. It is also equivocal, which follows from linearity using its dual distance (see [4, Lemma 2]). Therefore, it is a 1-party equivocal SSS with constant rate and security against probing of a constant fraction of codeword bits.

2.3.2.3 Why do equivocal SSSs yield non-adaptive ZK-PCPs?

We note that the locking schemes used in the constructions of [26, 22, 24] posses an equivocation property which is similar to our equivocal SSS, but applying them towards constructing ZK-PCPs and ZK-PCPPs causes *the honest* verifier to be adaptive. The reason is that reconstructing (i.e., opening) the secret in a locking scheme requires making several rounds of queries to the locking scheme. This is because one should be able to recover the locked secret *without reading the entire lock*, which is needed since locking schemes are generally much longer than the locked secret. (The blowup is inherent to obtaining equivocation in locking schemes.) On the other hand, in an equivocal SSS the secret is reconstructed by reading all (or most) of the shares, which can be done non-adaptively. The fact that reconstruction requires reading many shares is not problematic in terms of efficiency, since the total length of all shares is usually relatively short compared to the secret.

2.4 Future Directions

Our work still leaves several open questions for future research. First, it is natural to ask whether the query gap (between the query complexity of the malicious and honest verifiers) in our constructions could be improved – to a better polynomial gap, or even exponential? This could potentially be achieved by instantiating the ZK-PCP of [21] with an MPC protocol with stringent communication requirements, in which the communication complexity (more specifically, the size of the views) is sublinear in the number of parties. Another natural research direction is to construct *multi-party* equivocal SSSs, and in particular ones that withstand *general* local leakage (and not just probing leakage). Finally, it would be interesting to find further applications of equivocal SSSs in other contexts, e.g., for adaptively-secure MPC.

3 Preliminaries

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 $\delta \in (0, 1)$, let Σ be an alphabet, and let x, y be strings over Σ^n. We say that x, y are *δ-close* if $\frac{|\{i \,:\, x_i \neq y_i\}|}{n} < \delta$, otherwise we say that x, y are *δ-far*. We say that x is δ-close to a language $\mathcal{L}$ if there exists $x' \in \mathcal{L}$ such that x, x' are δ-close. Otherwise, we say that x is δ-far from $\mathcal{L}$.

▶ **Definition 3.** *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*

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

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.

3.1 Zero-Knowledge Probabilistically Checkable Proofs (PCPs) and PCPs of Proximity

Informally, a Probabilistically Checkable Proof (PCP) system for a language $\mathcal{L}$ consists of a probabilistic polynomial time prover that given $x \in \mathcal{L}$ and a corresponding witness generates a proof for x, and a probabilistic polynomial-time verifier having direct access to individual symbols of the proof. This proof string (called oracle) will be accessed only partially by the verifier. The oracle queries are determined by the verifier's input and coin tosses. Formally,

▶ **Definition 4** (PCP)**.** *A Probabilistically Checkable Proof (PCP) for a language $\mathcal{L}$ consists of a PPT prover $\mathcal{P}$ and a* PPT *verifier $\mathcal{V}$ such that the following holds for some negligible function* $\mathsf{negl} = \mathsf{negl}(\kappa)$*.*

1. Syntax. *The prover $\mathcal{P}$ has input $1^\kappa, x, w$, and outputs a proof π_x for x over some alphabet Σ. The verifier $\mathcal{V}$ has input $1^\kappa, x$, and oracle access to π. It makes q queries to π, and outputs either 0 or 1 (representing reject or accept, respectively).*
2. Completeness: *For every $x \in \mathcal{L}$, every $w \in \mathcal{R}_x$, and every proof $\pi_x \in \mathcal{P}(1^\kappa, x, w)$,*

   $$\Pr[\mathcal{V}^{\pi_x}(1^\kappa, x) = 1] \geq 1 - \mathsf{negl}(\kappa)$$

   where the probability is over the randomness of $\mathcal{V}$, and κ is the security parameter.
3. Soundness: *For every $x \notin \mathcal{L}$ and every oracle π^*,*

   $$\Pr[\mathcal{V}^{\pi^*}(1^\kappa, x) = 1] \leq \mathsf{negl}(\kappa)$$

   where the probability is over the coin tosses of the verifier, and κ is a security parameter. negl *is called the soundness error of the system.*

Efficiency measures of a PCP system

We associate with a PCP system the following efficiency measures: the alphabet size $|\Sigma|$, the query complexity q, and the proof length $|\pi|$. We will call such a system a q-query PCP over alphabet Σ. We are generally interested in obtaining PCPs with $\Sigma = \{0, 1\}$, in which the proof length $|\pi|$ is polynomial in $|x|$, and q is significantly smaller than $|\pi|$. We note that a PCP prover is usually deterministic, but allowing for randomized provers will be useful when discussing *zero-knowledge* PCPs, as we do next.

Next, we define *zero-knowledge* PCPs. Intuitively, these are PCPs in which the witness remains entirely hidden throughout the verification process, even when the verifier is malicious and can potentially make many more queries to the proof compared to the honest verifier. We guarantee zero-knowledge against any, possibly malicious and unbounded, verifier - the only restriction is on the number of queries the verifier makes to the proof (this restriction is inherent to obtaining zero-knowledge). Thus, we first define the notion of a *query bounded* verifier.

▶ **Definition 5** (Query-bounded verifier)**.** *We say that a (possibly malicious) verifier $\mathcal{V}^*$ with oracle access to a proof π is q^*-query-bounded if it makes only q^* queries to π.*

▶ **Definition 6** (Non adaptive verifier)**.** *We say that a (possibly malicious) verifier $\mathcal{V}^*$ is non adaptive if its queries are determined solely by its input x and its randomness (in particular, $\mathcal{V}^*$ can make a single round of queries to its proof oracle).*

We will use the following notation. For a PCP system $(\mathcal{P}, \mathcal{V})$ and a (possibly malicious) verifier $\mathcal{V}^*$, we use $\mathbf{View}_{\mathcal{V}^*,\mathcal{P}}(x, w)$ to denote the view of $\mathcal{V}^*$ when it has input x and oracle access to a proof that was randomly generated by $\mathcal{P}$ on input (x, w). We are now ready to define zero-knowledge PCPs.

▶ **Definition 7** (ZK-PCP). *We say that a PCP system $(\mathcal{P}, \mathcal{V})$ for $\mathcal{L}$ is a (q^*, ε)-Zero-Knowledge PCP (or ZK-PCP for short) if for any (possibly malicious and adaptive) q^*-query-bounded verifier $\mathcal{V}^*$ there exists a* PPT *simulator* Sim*, such that for any $(x, w) \in \mathcal{R}$, $\mathsf{Sim}(1^\kappa, x)$ is distributed ε-statistically close to $\mathbf{View}_{\mathcal{V}^*,\mathcal{P}}(x, w)$.*

If $(\mathcal{P}, \mathcal{V})$ is a (q^*, ε)-ZK-PCP for $\varepsilon = \mathsf{negl}(\kappa)$ then we simply say that $(\mathcal{P}, \mathcal{V})$ is a q^*-ZK-PCP. If $(\mathcal{P}, \mathcal{V})$ is a $(q^*, 0)$-ZK-PCP then we say it is a perfect q^*-ZK-PCP. If the ZK property of the system only holds against PPT verifiers $\mathcal{V}^*$ then we say the system is a computational (q^*, ε)-ZK-PCP (or CZK-PCP for short). If the zero-knowledge property is only guaranteed against non-adaptive verifiers then we say the system is a ZK-PCP for non-adaptive verifiers. If the honest ZK-PCP verifier is non-adaptive then we say that $(\mathcal{P}, \mathcal{V})$ is a non-adaptive ZK-PCP.

We stress that having a non-adaptive honest verifier is a desirable feature of the system, whereas having ZK against non-adaptive verifiers is a weaker form of ZK (since the system has no guarantee against malicious *adaptive* verifiers).

We remark that although this definition requires a weaker notion with a non-universal simulator, all our constructions obtain the stronger notion with a universal simulator. Furthermore, our constructions will rely on the MPC-in-the-head approach, where the quality of ZK will be inherited from the level of security of the underlying MPC protocol employed by the construction.

3.2 Leakage-Resilient Secret Sharing Schemes (LR-SSS)

A Secret-Sharing Scheme (SSS) allows a dealer to distribute a secret among n 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 8** (Secret Sharing Scheme). *An n-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 $(s_1, \cdots, s_n) \in \mathcal{S}_1 \times \cdots \times \mathcal{S}_n$, 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 $\mathsf{Share}(s)|_{\mathcal{G}}$ denotes the shares of the parties in the authorized set $\mathcal{G}$.

Secrecy: *For any pair of secrets $s, s' \in \mathcal{S}$, and any unauthorized set $\mathcal{G}$, $\mathsf{Share}(s)|_{\mathcal{G}}$ and $\mathsf{Share}(s')|_{\mathcal{G}}$ are statistically close.*

In our constructions, we will use Shamir's secret sharing scheme [28], which we review next.

▶ **Definition 9** (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 $p(1), \ldots, p(n)$.*

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 $g(0)$.

We will actually require a stronger correctness property for SSSs which, informally, guarantees unique reconstruction for any set of (possibly ill-formed) shares. This can be thought of as a weak form of unique decoding, where we only require error detection (i.e., identifying whether or not an error occurred), and not error correction. Alternatively, this is a weaker form of verifiable secret sharing, which need only be secure against a corrupted dealer (i.e., all the share holders are assumed to be honest). Formally,

▶ **Definition 10** (Strongly-correct SSS). *We say that an n-party secret sharing scheme* $(\mathsf{Share}, \mathsf{Reconst})$ *for secrets in $\mathcal{S}$ is strongly correct if* $\mathsf{Reconst}$ *is deterministic, and the only authorized set is $[n]$ (the set of all parties).*

We note that for any $t < n$, t-out-of-n Shamir's scheme (with the access structure in which the only authorized set is $[n]$) is strongly correct. For this, we assume that the shares are numbered in some arbitrary way, and reconstruction always uses the "first" $t + 1$ shares, see Remark 11 below.

▶ Remark 11 (Strong correctness implies unique reconstruction in threshold schemes). The strong correctness property of Definition 10 implies unique reconstruction in threshold schemes, when these are thought of as ramp secret sharing schemes which are private for sets of size at most t, and reconstructible for the set of all parties. Indeed, we assume without loss of generality that the shares are numbered (in some arbitrary way). Given all n shares, reconstruction is performed with the first $t + 1$ shares. These $t + 1$ shares determine *some* secret, and the fact that $\mathsf{Reconst}$ is deterministic guarantees its uniqueness. In particular, we note that in this case Shamir's secret sharing scheme has unique reconstruction.

▶ Remark 12. We note that for our alphabet reduction for ZK-PCPs (Construction 17, Section 4) and ZK-PCPPs (which appears in the full version [20]) we can make due with *any* SSS with deterministic reconstruction, by having the verifier use some arbitrary *fixed* minimal authorized set for reconstruction. Notice that if such a set can be efficiently found then the verifier in Construction 17 will be PPT. We further note that for threshold SSSs (such as the one used in our final constructions in Theorems 1 and 2), such a minimal authorized set can be found efficiently.

Next, we define *leakage-resilient* SSS.

▶ **Definition 13** (Leakage-resilient SSS). *We say that a secret sharing scheme* $(\mathsf{Share}, \mathsf{Reconst})$ *for $\mathcal{S}$ is ε-leakage resilient against a family F of leakage functions if for every $f \in F$, and every pair of secrets $s, s' \in \mathcal{S}$, $f(\mathsf{Share}(s))$ and $f(\mathsf{Share}(s'))$ are ε-statistically close.*

We will be particularly interested in the *local probing* leakage family, which consists of all functions that, given the n shares, output the shares of an unauthorized set in their entirety, as well as ℓ bits from each of the other shares. More specifically, we will only consider the $t+1$-threshold access structure mentioned above, in which all (and only) subsets of size $\geq t+1$ are authorized. Formally:

► **Definition 14** ((t,ℓ)-local probing leakage). *Let $\mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_n$ be the domain of shares for some secret sharing scheme. For a subset $\mathcal{G} \subseteq [n]$ and a sequence $(\mathcal{I}_1, \ldots, \mathcal{I}_n)$ of subsets of $[n]$, the function $f_{\mathcal{G},\mathcal{I}_1,\ldots,\mathcal{I}_n}$ on input $(s_1, \ldots, s_n)$ 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 $\mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_n$ is defined as follows:*

$$F_{t,\ell} = \{f_{\mathcal{G},\mathcal{I}_1,\ldots,\mathcal{I}_n} \ : \ \mathcal{G} \subseteq [n], |\mathcal{G}| \leq t, \forall i \notin \mathcal{G}, |\mathcal{I}_i| \leq \ell\} .$$

3.3 Equivocal Secret Sharing

In this section we define the notion of equivocal secret sharing, compare it to leakage-resilient secret sharing, and present a 1-party equivocal SSS based on coding. We start with the definition.

At a high level, an equivocal SSS is a leakage-resilient SSS with the additional guarantee that even after some bits are leaked from the shares, one can still "open" the secret (by providing the entire secret sharing) consistently with the previous leakage. This is formalized (in Definition 15) in the simulation-based paradigm, by comparing the real world experiment with an ideal experiment, which are described in Figure 1.

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. Leakage resilience is guaranteed 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, and a $\mathcal{LEAK}$ oracle. The Share oracle, given an index i, returns the i'th share Sh_i. Each call to $\mathcal{SHARE}$ updates the set T of secret shares which the adversary has queried so far, by adding i to T (T is initialized to $\emptyset$). The $\mathcal{LEAK}$ oracle takes as input a function $g \in F$, which specifies, for each share $\mathsf{Sh}_i, i \notin T$, a leakage function g_i. It applies these leakage functions to the shares $\mathsf{Sh}_i, i \notin T$, and returns the outputs output_i. For each such share $\mathsf{Sh}_i, i \notin T$, it also updates the counter ℓ_i of the number of leakage bits obtained on Sh_i, by increasing it by $|\mathsf{output}_i|$. T and $\ell_1, \ldots, \ell_n$ are treated as global parameters that can be accessed and updated by all oracles.

At the end of the first phase of the experiment (the adversary $\mathcal{A}_1$ decides when to end the first phase and move to the second phase), $\mathcal{A}_1$ outputs a bit b_R, which specifies whether it wishes to learn the entire secret sharing of s. If $b_R = 1$, i.e., the adversary chose to proceed to the second phase, then it learns the entire secret sharing of s (this is done by calling the $\mathcal{REVEAL}$ oracle). Otherwise, the adversary obtains no further information beyond what it obtained during the leakage phase.

At 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 guess depends on the leakage in the first phase of the game, and can either depend on the secret shares of s (if $b_R = 1$) or not (if $b_R = 0$). The adversarial guess is only taken into account if the adversary did not violate the leakage restrictions, i.e., only if the following two conditions are satisfied. First, the set T of shares which the adversary received throughout the experiments (through $\mathcal{SHARE}$ queries) is an unauthorized set. Second, for every share i, the number of bits ℓ_i leaked from Sh_i (through

$\mathcal{LEAK}$ queries) does not exceed the leakage bound ℓ. 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 $\mathcal{REVEAL}$ oracles are emulated by the simulator. In particular, the simulator needs to simulate shares and leakage on shares (through the $\mathcal{SHARE}$ and $\mathcal{LEAK}$ oracles). Additionally, if $b_I = 1$ (i.e., the adversary chose to learn the entire secret sharing in the ideal experiment) then the simulator is given the secret s, and needs to emulate the entire secret sharing of s consistently with the previous leakage (this is done in the $\mathcal{REVEAL}$ oracle).

We note that by allowing the adversary to choose *not* to receive the entire secret sharing of s at the end of the first phase, we capture leakage-resilient secret sharing as a special case of equivocal secret sharing. Indeed, if the adversary chooses not to learn the secret sharing, then the simulator is only required to adaptively simulate the leakage (with no knowledge of the secret). We elaborate more on this in Remark 16 below.

▶ **Definition 15** (Equivocal SSS). *We say that an n-party secret sharing scheme* $(\mathsf{Share}, \mathsf{Reconst})$ *for secrets in $\mathcal{S}$ is $\varepsilon(n)$-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 $(\mathsf{Sim}_1, \mathsf{Sim}_2)$ and a negligible function $\varepsilon(n)$ such that for every $s \in \mathcal{S}$,*

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

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 say that the scheme is perfectly equivocal, if it is ε-equivocal with $\varepsilon = 0$.

▶ Remark 16 (On the connection between equivocal and LR secret sharing). We note that equivocal secret sharing captures LR secret sharing as a special case, in the following sense. If a t-threshold secret sharing scheme is ε-equivocal with leakage bound ℓ, then it is also 2ε-leakage resilient against (t, ℓ)-local probing leakage. Indeed, if the $\mathcal{REVEAL}$ oracle is not called in Figure 1 (which happens when $b = 0$) then the simulator never receives the secret s, in which case the simulated answers to the leakage queries are required to be distributed ε-statistically close to the real execution. As this holds for any secret, the real leakage on the shares of two different secrets must be 2ε-statistically close.

4 Alphabet Reduction for ZK-PCPs

In this section we describe a reduction for ZK-PCPs over any alphabet Σ into a ZK-PCP over $\{0, 1\}$. In particular, this reduction preserves the zero-knowledge property. We note that for standard (non-ZK) PCPs, one can easily transform a PCP over any alphabet Σ into a PCP over $\{0, 1\}$ by simply representing every symbol of Σ as a binary string. However, this reduction *does not* preserve zero-knowledge since a malicious verifier given access to the binary proof can read "parts" of symbols of the original proof, and thus potentially violate the zero-knowledge guarantee of the underlying ZK-PCP over Σ (which only guarantees zero-knowledge when most symbols are not accessed at all).

We begin by describing a general reduction, then instantiate it to obtain ZK-PCPs over $\{0, 1\}$ with a square root gap.

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

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

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

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

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

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

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

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

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

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

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

Figure 1 The Security Experiments of Equivocal SSS.

A General Transformation

Our starting point is the trivial transformation described above, in which every proof symbol is replaced with a corresponding bit-string. As discussed above, this alone does not guarantee zero-knowledge since a malicious verifier may read parts of symbols of the original proof. The high-level idea of preventing such malicious strategies from leaking additional information is to "protect" each bit-string by secret-sharing it (equivalently, encoding it) using a *leakage-resilient* secret sharing scheme (equivalently, leakage-resilient encoding). Recall that, very roughly, a probing-resilient secret sharing scheme hides the secret from an adversary that sees several secret shares, and can probe few bits in each of the other shares. The zero-knowledge property of the new PCP system now follows from a combination of leakage-resilience and the zero-knowledge property of the original ZK-PCP. To see why, given a malicious query-bounded verifier $\mathcal{V}^*$, we partition the symbols of the original proof into two groups, based on the number of bits $\mathcal{V}^*$ reads from the secret-sharing of the bit-string representing the symbol. Since $\mathcal{V}^*$ is query-bounded, there are only few symbols from whose secret shares $\mathcal{V}^*$ can read many bits (having many such symbols would have violated the query bound). The zero-knowledge property of the original ZK-PCP system guarantees that $\mathcal{V}^*$ learns nothing about the witness even if it is given all these symbols in their entirety. For the rest of the symbols, since $\mathcal{V}^*$ reads only few bits from their secret shares, the leakage-resilience of the secret sharing scheme guarantees that the secret shared symbol remains entirely hidden. The actual analysis is slightly more involved, see the proof of Theorem 18 below for details.

We now formally describe the transformation.

▶ **Construction 17** (Alphabet reduction for ZK-PCPs). *Let κ be a security parameter. The system $(\mathcal{P}', \mathcal{V}')$ is over alphabet $\{0,1\}$.*

Building blocks:

- *A PCP system $(\mathcal{P}, \mathcal{V})$ over alphabet Σ of size $|\Sigma| = 2^m$.*
- *A strongly-correct secret sharing scheme* (Share, Reconst) *for secrets in $\{0,1\}^m$.*

Prover algorithm. *$\mathcal{P}'$ has input $1^\kappa, x, w$. It runs $\mathcal{P}$ with input $1^\kappa, x, w$ to obtain a proof π over Σ. For every proof symbol σ, it uses* Share *to secret-share the bit-representation of σ. (That is, the length-m bit representation of σ is treated as the secret.) Then, $\mathcal{P}'$ outputs the concatenation of all secret shares. We denote the proof generated by $\mathcal{P}'$ by π'.*

Verifier algorithm. *$\mathcal{V}'$ is given input $1^\kappa, x$ and oracle access to π'. It runs $\mathcal{V}$ with input $1^\kappa, x$, and emulates the oracle π for $\mathcal{V}$ as follows. Whenever $\mathcal{V}$ reads a symbol σ from π, $\mathcal{V}'$ reads the entire secret sharing of σ from π'. Then, it uses* Reconst *to recover the symbol σ, and provides σ to $\mathcal{V}$ as the answer of the oracle.*

The following theorem summarizes the properties of Construction 17. Its proof, and several relevant corollaries, can be found in the full version [20].

▶ **Theorem 18** (Non-adaptive ZK-PCPs for non-adaptive verifiers). *Assume Construction 17 is instantiated with:*

- *A non-adaptive q-query (q^*, ϵ)-ZK-PCP $(\mathcal{P}, \mathcal{V})$ over alphabet Σ for a language $\mathcal{L}$, with proofs of length N.*[6]
- *A strongly-correct k-party secret sharing scheme* (Share, Reconst) *for secrets in $\{0,1\}^m$ with secret shares in $\{0,1\}^M$ which is ϵ'-leakage-resilient against (t, ℓ)-local probing leakage.*

[6] In fact, as will be evident from the proof, it suffices that $(\mathcal{P}, \mathcal{V})$ is ZK against *non-adaptive* malicious verifiers.

Then Construction 17 is a non-adaptive q'-query (q^{**}, ϵ'')-ZK-PCP for non-adaptive verifiers, where:

$$q' = q \cdot M \cdot k \qquad q^{**} = (q^* + 1)(\ell + 1)(t + 1) - 1 \qquad \epsilon'' = \epsilon + \epsilon' \cdot (N - q^*)\,.$$

Moreover, the transformation preserves the soundness and completeness of $(\mathcal{P}, \mathcal{V})$.

4.1 Upgrading to ZK Against Adaptive Verifiers

Our ZK-PCP (Theorem 18) obtained through the alphabet reduction of Construction 17 can be verified non-adaptively, but guarantee ZK only against *non-adaptive* verifiers. Ideally, we would like a ZK-PCP which can be verified non-adaptively, but guarantees ZK even against *adaptive* malicious verifiers.

In this section, we show that when Construction 17 is instantiated with an equivocal SSS (see Definition 15) instead of a leakage-resilient SSS then the resultant ZK-PCP retains its ZK even when the malicious verifier is adaptive. Concretely, we prove the following:

▶ **Theorem 19.** *Assume Construction 17 is instantiated with:*

- *A non-adaptive q-query (q^*, ϵ)-ZK-PCP $(\mathcal{P}, \mathcal{V})$ over alphabet Σ for a language $\mathcal{L}$, with proofs of length N.*[7]
- *A strongly-correct k-party ϵ'-equivocal (ramp) secret sharing scheme against (t, ℓ)-local probing leakage, for secrets in $\{0,1\}^m$ with secret shares in $\{0,1\}^M$.*

Then Construction 17 is a non-adaptive q'-query (q^{**}, ϵ'')-ZK-PCP, where

$$q' = q \cdot M \cdot k \qquad q^{**} = (q^* + 1)(\ell + 1)(t + 1) - 1 \qquad \epsilon'' = \epsilon + \epsilon' \cdot N.$$

Moreover, the transformation preserves the soundness and completeness of $(\mathcal{P}, \mathcal{V})$.

Theorem 1 now follows from Theorem 19 by an appropriate instantiation of the building blocks. See the full version [20] for details.

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, Proceedings, Part II*, pages 510–539, 2019.

2 Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and hardness of approximation problems. In *FOCS, Proceedings*, pages 14–23, 1992.

3 Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs; A new characterization of NP. In *FOCS, Proceedings*, pages 2–13, 1992.

4 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.

5 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.

6 Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Robust PCPs of proximity, shorter PCPs and applications to coding. In *STOC, Proceedings*, pages 1–10, 2004.

[7] We stress that $(\mathcal{P}, \mathcal{V})$ is non-adaptive in the sense that the *honest* verifier is non-adaptive, but ZK holds against possibly *adaptive* verifiers.

7 Eli Ben-Sasson and Madhu Sudan. Short PCPs with polylog query complexity. *SIAM J. Comput.*, 38(2):551–607, 2008.

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 Ran Canetti, Ivan Damgård, Stefan Dziembowski, Yuval Ishai, and Tal Malkin. On adaptive vs. non-adaptive security of multiparty protocols. In *EUROCRYPT, Proceedings*, pages 262–279, 2001.

10 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.

11 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.

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

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

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

15 Irit Dinur. The PCP theorem by gap amplification. In *STOC, Proceedings*, pages 241–250, 2006.

16 Irit Dinur and Omer Reingold. Assignment testers: Towards a combinatorial proof of the PCP-theorem. In *FOCS, Proceedings*, pages 155–164, 2004.

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

18 Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *STOC, Proceedings*, pages 291–304, 1985.

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

20 Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, and Mor Weiss. ZK-PCPs from leakage-resilient secret sharing. *IACR Cryptol. ePrint Arch.*, 2021:606, 2021. URL: `https://eprint.iacr.org/2021/606`.

21 Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *STOC, Proceedings*, pages 21–30, 2007.

22 Yuval Ishai, Mohammad Mahmoody, and Amit Sahai. On efficient zero-knowledge PCPs. In *TCC, Proceedings*, pages 151–168, 2012.

23 Yuval Ishai, Amit Sahai, Michael Viderman, and Mor Weiss. Zero knowledge LTCs and their applications. In *RANDOM, Proceedings*, pages 607–622, 2013.

24 Yuval Ishai and Mor Weiss. Probabilistically checkable proofs of proximity with zero-knowledge. In *TCC, Proceedings*, pages 121–145, 2014.

25 Yuval Ishai, Mor Weiss, and Guang Yang. Making the best of a leaky situation: Zero-knowledge PCPs from leakage-resilient circuits. In *TCC, Proceedings*, pages 3–32, 2016.

26 Joe Kilian, Erez Petrank, and Gábor Tardos. Probabilistically checkable proofs with zero knowledge. In *STOC, Proceedings*, pages 496–505, 1997.

27 Thilo Mie. Short PCPPs verifiable in polylogarithmic time with $O(1)$ queries. *Ann. Math. Artif. Intell.*, 56(3-4):313–338, 2009.

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

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

30 Mor Weiss. Secure computation and probabilistic checking. *PhD Thesis*, 2016.