Probabilistically Checkable Proofs of Proximity with Zero-Knowledge*

Yuval Ishai and Mor Weiss

Department of Computer Science, Technion, Haifa.
{yuvali,morw}@cs.technion.ac.il

Abstract. A probabilistically Checkable Proof (PCP) allows a randomized verifier, with oracle access to a purported proof, to probabilistically verify an input statement of the form "$x \in L$" by querying only few bits of the proof. A *PCP of proximity* (PCPP) has the additional feature of allowing the verifier to query only few bits of the *input* x, where if the input is accepted then the verifier is guaranteed that (with high probability) the input is *close* to some $x' \in L$.
Motivated by their usefulness for sublinear-communication cryptography, we initiate the study of a natural zero-knowledge variant of PCPP (ZKPCPP), where the view of any verifier making a bounded number of queries can be efficiently simulated by making the same number of queries to *the input oracle alone*. This new notion provides a useful extension of the standard notion of zero-knowledge PCPs. We obtain two types of results.

- **Constructions.** We obtain the first constructions of query-efficient ZKPCPPs via a general transformation which combines standard query-efficient PCPPs with protocols for secure multiparty computation. As a byproduct, our construction provides a conceptually simpler alternative to a previous construction of honest-verifier zero-knowledge PCPs due to Dwork et al. (Crypto '92).
- **Applications.** We motivate the notion of ZKPCPPs by applying it towards sublinear-communication implementations of commit-and-prove functionalities. Concretely, we present the first sublinear-communication commit-and-prove protocols which make a *black-box* use of a collision-resistant hash function, and the first such multiparty protocols which offer *information-theoretic* security in the presence of an honest majority.

1 Introduction

In this work we initiate the study of probabilistically checkable proofs of proximity with a zero-knowledge property, and use such proofs to design efficient cryptographic protocols. Before describing our main results, we first give a short overview of probabilistic proof systems.

* Research supported by the European Union's Tenth Framework Programme (FP10/2010-2016) under grant agreement no. 259426 ERC-CaC. The first author was additionally supported by ISF grant 1361/10 and BSF grants 2008411 and 2012366.

Probabilistically Checkable Proof (PCP) systems [**?**,**?**] are proof systems that allow an efficient randomized verifier, with oracle access to a purported proof, to probabilistically verify claims such as "$x \in L$" (for some input x and an NP-language L) by probing only few bits of the proof. The verifier accepts the proof of a true claim with probability 1 (the *completeness* property), and rejects false claims with high probability (the probability that the verifier accepts a false claim is called *the soundness error*). The celebrated PCP theorem [**?**,**?**,**?**] asserts that any NP language admits a PCP system with soundness error 1/2 in which the verifier reads only a *constant* number of bits from the proof. The soundness error can be reduced to $2^{-\sigma}$ by running the verifier σ times.

Probabilistically Checkable Proofs of Proximity (PCPPs), also known as assignment testers, are proof systems that allow probabilistic verification of claims by probing few bits *of the input* and a purported proof. Needless to say, the verifier of such a system cannot generally be expected to distinguish inputs in the language from inputs that are not in the language, but rather it should accept every $x \in L$ with probability 1 and reject (with high probability) every input that is "far" from all $x' \in L$. First introduced in [**?**,**?**,**?**] as building blocks for the construction of more efficient PCPs, there are currently known PCPP systems for NP with parameters comparable to those of the best known PCP systems [**?**,**?**].

A seemingly unrelated concept is that of zero-knowledge (ZK) proofs [**?**], namely proofs that carry no extra knowledge other than being convincing. Combining the advantages of ZK proofs and PCPs, a *zero-knowledge PCP* (ZKPCP) is defined similarly to a traditional PCP, with the additional guarantee that the view of any (possibly malicious) verifier can be efficiently simulated up to a small statistical distance. ZKPCPs were first constructed by Kilian et al. [**?**], building on a previous weaker "honest-verifier" notion of ZKPCPs implicitly constructed by Dwork et al. [**?**]. More concretely, the work of [**?**] combines the weaker variant of ZKPCPs from [**?**] with an unconditionally secure oracle-based commitment primitive called a "locking scheme" to obtain ZKPCPs for NP that guarantee statistical zero-knowledge against *query-bounded* malicious verifiers, namely ones who are limited to asking at most $p(|x|)$ queries for some *fixed* polynomial p. A simpler construction of locking schemes was recently given in [**?**].

ZKPCPPs. In this work we put forward and study the new notion of *zero-knowledge PCPP* (or ZKPCPP for short), which extends the previous notion of ZKPCP and makes it more useful. A ZKPCPP is a PCPP with a probabilistic choice of proof and the additional guarantee that the view of any (possibly-malicious) verifier, making at most q^* queries to his input and his proof oracle, can be efficiently simulated by making the same number of queries *to the input alone.* ZKPCPPs are a natural extension of ZKPCPs (indeed, the existence of a ZKPCPP system implies the existence of a ZKPCP system with related parameters) and interesting objects on their own. As we explain next, they are also motivated by cryptographic applications that involve sublinear-communication zero-knowledge arguments on distributed or committed data.

To give the flavor of these applications, suppose that a database owner (prover) commits to a large sensitive database D by robustly secret-sharing it among a large number of potentially unreliable servers. At a later point in time, a user (verifier) may want to learn the answer to a query $q(D)$ on the committed database. ZKPCPPs provide the natural tool for efficiently verifying that the answer a provided by the prover is indeed consistent with the committed database, namely that $a = q(D)$, without revealing any additional information about the database to the verifier and a colluding set of servers. Concretely, the prover distributes between the servers a ZKPCPP asserting that the shares of D (the input) are indeed valid shares of some database D' such that $q(D') = a$. The verifier, by probing only few entries in the input and the proof string, is convinced that the shares held by the servers are indeed *close* to being consistent with valid shares of some database D' such that $q(D') = a$. If not "too many" servers are corrupted, the robustness of the underlying secret-sharing scheme guarantees that $D' = D$. (Unlike the ZKPCPP model, the answers provided by malicious servers may depend on the identity of the verifier's queries; this difficulty can be overcome by ensuring that with sufficiently high probability the verifier's queries are answered by honest servers.) The above approach can also be used for verifiable updates of a secret distributed database, where a ZKPCPP is used to convince a verifier that the shares of the updated version of the database are consistent with the original shares with respect to the update relation.

A similar idea can be used to get a sublinear-communication implementation of a "commit-and-prove" functionality in the two-party setting. Here the prover first succinctly commits, using a Merkle tree, to the shares of D. To later prove that $q(D) = a$, the prover again uses a Merkle tree to succinctly commit to a ZKPCPP asserting that the values it committed to in the previous phase are valid shares of some database D' such that $q(D') = a$. This gives the first sublinear-communication implementations of commit-and-prove which only make a *black-box* use of a collision-resistant hash function. (See Section 5.2 for a non-black-box alternative using standard sublinear arguments.)

1.1 Summary of Results

We introduce the notion of ZKPCPPs and construct *query-efficient* ZKPCPPs for any NP language L. More precisely, given an input $x \in L$, a corresponding witness w, and a zero-knowledge parameter q^*, the prover can efficiently generate a proof string π of length $\text{poly}(|x|, q^*)$ which is statistical zero-knowledge against (possibly malicious) verifiers that make at most q^* queries to (x, π); by making only a polylogarithmic number of queries (in $|x|$ and q^*), an honest verifier can get convinced that x is at most δ-far from L, except with negligible soundness error, where δ can be any positive constant (or even inverse polylogarithmic). We then present applications of this construction to sublinear commit-and-proof protocols in both the two-party and the multiparty setting, as discussed above.

1.2 Techniques

Our main ZKPCPP construction is obtained by combining an (efficient) PCPP system without zero-knowledge with a protocol for secure multiparty computation (MPC), inheriting the efficiency from the PCPP component and the zero-knowledge from the MPC component. The transformation has two parts. The first consists of a *general* transformation from a PCPP and a secure MPC protocol to a PCPP system with zero-knowledge against semi-honest verifiers (HVZKPCPP, for *honest-verifier* ZKPCPP). The transformation can also be applied to PCPs, yielding an HVZKPCP comparable to that of [**?**] that is conceptually simpler. (Thus, our construction also simplifies the ZKPCP of Kilian et al. [**?**] which uses the HVZKPCP of [**?**] as a building block.)

The second part strengthens the zero-knowledge property to hold against *arbitrary* (query-bounded) malicious verifiers, by forcing the queries of any such verifier to be distributed (almost) as the queries of the honest verifier of the HVZKPCPP system. This part follows the approach of [**?**] of using a locking scheme. Concretely, we use the combinatorial construction of locking schemes from [**?**], except that to achieve negligible soundness error and negligible simulation error simultaneously we need to apply a natural amplification technique for reducing the error of the previous construction.

Organization. We first give the necessary preliminaries in Section 2. In Section 3 we describe the construction of a ZKPCPP from MPC protocols, and in Section 4 we state and prove our result regarding the existence of efficient ZKPCPP systems for NP. We describe our cryptographic applications in Section 5. Due to space limitations, we defer several definitions, constructions, and proofs, as well as the discussion regarding amplification of locking schemes, to the full version.

2 Preliminaries

We consider efficient probabilistic proof system for NP relations. (We refer to a relation rather than a language because we require the prover to be computationally efficient given an NP-witness.) Recall that an NP relation $\mathcal{R}$ is a polynomial-time recognizable binary relation which is *polynomially bounded* in the sense that there is a polynomial p such that if $(x, w) \in \mathcal{R}$ then $|w| \leq p(|x|)$. We refer to x as an *input* and to w as a *witness*.

We denote by $L_{\mathcal{R}}$ the NP language corresponding to $\mathcal{R}$, namely $L_{\mathcal{R}} = \{x : \exists w, (x, w) \in \mathcal{R}\}$. We say that x is δ-*far* from $L_{\mathcal{R}}$ (for some $\delta \in [0, 1]$) if the relative hamming distance between x and every $x' \in L_{\mathcal{R}}$ of the same length is more than δ.

We say that two distribution ensembles X_n, Y_n are computationally (resp. statistically) indistinguishable if every computationally bounded (respectively, computationally unbounded) distinguisher achieves only a negligible advantage in distinguishing a sample drawn according to X_n from a sample drawn according to Y_n.

A *probabilistic proof system* (P, V) for $\mathcal{R}$ consists of a PPT prover P, that on input (x, w) outputs a proof π, and a PPT verifier V that given input $1^{|x|}$ and oracle access to x (the *input oracle*) and π (the *proof oracle*) outputs either accept or reject. Intuitively, P tries to convince V of the claim "$x \in L_{\mathcal{R}}$" using w such that $(x, w) \in \mathcal{R}$. All the probabilistic proof systems studied in this work will have perfect completeness (i.e. V accepts true claims with probability 1). The system has soundness error ϵ if every input $x \notin L_{\mathcal{R}}$ is accepted by V with probability at most ϵ, *regardless* of the proof oracle. Our systems are sometimes parameterized by a statistical security parameter σ and a zero-knowledge parameter q^*. These parameters are given as additional inputs to both P and V.

In the following sections, we define several classes consisting of NP relations that have probabilistic proof systems with additional properties, and discuss the containment relations between these classes. (We associate each class of relations with the corresponding class of proof systems.) Every containment stated in this paper follows from *constructive transformations*, namely if we claim that $\text{Class}_1 \subseteq \text{Class}_2$, then the proof also shows an efficient transformation from a pair $(P, V) \in \text{Class}_1$ to a pair $(P', V') \in \text{Class}_2$.

PCPs. A standard probabilistically checkable proof (PCP) is a probabilistic proof system (P, V) in which V can query his input oracle x freely (that is, his queries to x are not counted towards the query complexity). We write $\mathcal{R} \in \text{PCP}_{\Sigma}[r, q, \epsilon, \ell]$ if $\mathcal{R}$ admits a PCP with verifier randomness complexity r, query complexity q, soundness error ϵ, and a proof π of length ℓ over the alphabet Σ.

PCPPs. Intuitively, the verifier of a PCPP system tries to validate the claim "$x \in L_{\mathcal{R}}$", while reading only *few* bits of x. Of course, V cannot generally be expected to distinguish the case that $x \in L_{\mathcal{R}}$ from the case that $x \notin L_{\mathcal{R}}$ but is very "close" to it. Instead, V is only expected to reject when x is "far" from $L_{\mathcal{R}}$. Concretely, A probabilistically checkable proof of proximity (PCPP) system with soundness error $0 \leq \epsilon < 1$ and proximity parameter $0 \leq \delta < 1$ is a probabilistic proof system (P, V) for which the following holds. For every x that is δ-far from $L_{\mathcal{R}}$, V accepts x with probability at most ϵ, *regardless* of his proof oracle. In this case we write $\mathcal{R} \in \text{PCPP}_{\Sigma}[r, q, \delta, \epsilon, \ell]$ where r, q, ℓ are as above. We refer to a PCPP system with proximity parameter $\delta = 0$ (i.e., in which soundness holds for every $x \notin L_{\mathcal{R}}$) as an *exact* PCPP.

We also consider systems with the following notion of *strong soundness*, where *every* $x \notin L_{\mathcal{R}}$ is rejected with probability that is proportional to its distance from $L_{\mathcal{R}}$. That is, there exists a function $\epsilon_S = \epsilon_S(\delta, |x|) : [0, 1] \times \mathbb{N} \to [0, 1]$ such that for every $\delta \in [0, 1]$, *every* x that is δ-far from $L_{\mathcal{R}}$ is accepted by V with probability at most $\epsilon_S(\delta, |x|)$. Such PCPPs are called *strong PCPPs*, see [**?**,**?**]. A strong PCPP system has *rejection ratio* β if every x that is δ-far from $L_{\mathcal{R}}$ is rejected with probability at least $\beta\delta$.

ZKPCPPs and ZKPCPs. We are interested in PCPs and PCPPs that reveal (almost) no information to verifiers who do not make "too many" queries. Intuitively, a probabilistic proof system is q^*-zero-knowledge if whatever a (possibly

malicious) verifier learns by making $q' \leq q^*$ queries to x, π can be simulated by making q' queries *to x alone*. In particular, zero-knowledge of a PCP system implies the witness is entirely hidden (the queries of the verifier to the input oracle are not counted towards the query complexity, so the simulator can query all of x, and consequently only the witness is hidden), while in a zero-knowledge PCPP system not only is the witness hidden, but so is most of x. Notice that the prover in a zero-knowledge proof system must be *probabilistic* (while the prover in standard proof systems for NP can be deterministic).

More formally, let (P, V) be a probabilistic proof system, and let V^* be a (possibly malicious) *q-bounded* verifier (namely a verifier that never makes more than q queries). We compare the real-life interaction between P and V^* with an *ideal-world* interaction, in which a simulator Sim with oracle access to V^* interacts with a trusted third party (TTP) that knows only x. Denote the distribution ensemble describing the view of V^* with oracles x, π (where π was honestly generated by P on input x, w) by $\mathsf{View}_{V^{*x,\pi}}$, and let q_{V^*} denote the total number of queries V^* sent to the input and proof oracles. Let $\mathrm{Real}_{V^*,P}(x, w) = (\mathsf{View}_{V^{*x,\pi}}, q_{V^*})$. Similarly, for an ideal-world simulator Sim let $\mathsf{Sim}(x)$ denote the distribution ensemble describing the output of Sim (after making his queries to x), and let q_S denote the number of queries Sim made. We define $\mathrm{Ideal}_{\mathsf{Sim}}(x) = (\mathsf{Sim}(x), q_S)$.

We say that (P, V) is (ϵ, q^*)-zero knowledge with respect to $\mathcal{R}$ (for some $\epsilon \in [0, 1]$ and $q^* \in \mathbb{N}$) if for every real-life q^*-bounded verifier V^* there exists an ideal-world simulator Sim such that for every $(x, w) \in \mathcal{R}$, we have $\mathrm{Real}_{V^*,P}(x, w) \approx^{\epsilon} \mathrm{Ideal}_{\mathsf{Sim}}(x)$, where $\approx^{\epsilon}$ denotes statistical distance of ϵ. If (P, V) is $(0, q^*)$-zero-knowledge we say that it has *perfect* q^*-zero-knowledge, and write $\mathrm{Real}_{V^*,P}(x, w) \equiv \mathrm{Ideal}_{\mathsf{Sim}}(x)$. We may choose ϵ, q^* to be functions of a security parameter σ and the input size $|x|$. By default, we will make the stronger requirement that there exist a single, PPT *black-box* simulator S such that for every q^*-bounded V^*, the simulator $\mathsf{Sim} = \mathsf{S}^{V^*}$ satisfies the above requirement. Moreover, S can only interact with V^* in a straight-line fashion (i.e., it cannot rewind V^*). The latter straight-line simulation requirement is useful for one of our motivating applications.

Remark 1. The above notion of zero-knowledge requires that the number of input bits read by the simulator be the same as the *total* number of bits read by the verifier. One may consider stronger notions which require that the number of input bits read by the simulator coincide with the number of input bits read by the verifier, or even that the *same* input bits are read by the verifier and the simulator. The latter is captured by letting Real and Ideal, instead of including the number of queries V^* and Sim made (respectively), include the *specific* indices V^*, Sim queried in x. Our constructions do not satisfy these stronger notions.

Notation 1 *If a system $(P, V) \in \mathrm{PCPP}_{\Sigma}[r, q, \delta, \epsilon_S, \ell]$ for relation $\mathcal{R}$ guarantees (ϵ_{ZK}, q^*)-zero-knowledge, we say that (P, V) is a q^*-zero-knowledge PCPP and write $(P, V) \in \mathrm{ZKPCPP}_{\Sigma}[r, q, \epsilon_{ZK}, \delta, \epsilon_S, \ell]$. Similarly, if $(P, V) \in \mathrm{PCP}_{\Sigma}[r, q, \epsilon_S, \ell]$ for relation $\mathcal{R}$ guarantees (ϵ_{ZK}, q^*)-zero-knowledge, we write $(P, V) \in \mathrm{ZKPCP}_{\Sigma}[r, q, \epsilon_{ZK}, \epsilon_S, \ell]$.*

We also consider the following *honest-verifier* variant of zero-knowledge which is used as a simpler building block and is also of independent interest. We say that (P, V) has *honest-verifier zero-knowledge* (HVZK) with statistical distance $\epsilon \in [0,1]$ if for the honest verifier V there exists a PPT simulator Sim such that the previous zero-knowledge requirement holds, namely for every pair $(x,w) \in \mathcal{R}$, $\mathrm{Real}_{V,P}(x,w) \approx^{\epsilon} \mathrm{Ideal}_{\mathsf{Sim}}(x)$.

Secure MPC. We follow the terminology and notation of [**?**]. Let $P_1, ..., P_n$ be n parties, where every party P_i holds a private input z_i (we allow z_i to be empty, which we denote by $z_i = \lambda$). We consider protocols for securely realizing an n-party functionality g, that maps the tuple of inputs $(z_1, ..., z_n)$ to an output in $\{0,1\}$. All parties are expected to have the same output. The *view* of a party P_i, denoted V_i, includes his private input z_i and a random input r_i, together with all the messages that P_i received during the protocol execution. (The messages P_i sends during the execution, as well as his local output, are determined by this view.) A pair $\mathsf{V}_i, \mathsf{V}_j$ of views are *consistent with respect to* z_i, z_j *and* Π, if the outgoing messages (from P_i to P_j) implicit in V_i in an execution of Π on inputs z_i, z_j, are identical to the incoming messages (from P_i to P_j) reported in V_j, and vice versa. Consistency between a view and one of its incident communication channels is defined similarly.

We consider the execution of the protocol in the presence of an adversary $\mathcal{A}$ who may corrupt up to t parties. A *semi-honest* adversary can only passively corrupt parties (i.e., it does not modify their behavior but can learn their entire view), whereas a *malicious* adversary can arbitrarily modify the behavior of corrupted parties. A *static* adversary is restricted to pick the set of corrupted parties in advance, whereas an *adaptive* adversaries may pick them one by one, choosing the next party to corrupt based on its view so far.

A protocol Π realizes a deterministic n-party functionality $g(z_1, ..., z_n)$ with *perfect correctness* if for all inputs $z_1, ..., z_n$, when no party is corrupted, all parties output $g(z_1, ..., z_n)$. For a security threshold $1 \leq t \leq n$, we say that Π realizes g with *perfect t-privacy* if for every semi-honest adversary $\mathcal{A}$ corrupting a set $T \subseteq [n], |T| \leq t$ of parties there exists a simulator Sim that can perfectly simulate the view of $\mathcal{A}$ given only the inputs of corrupted parties and the output. One can naturally define a variant of privacy that applies to adaptive adversaries. (In the adaptive case, we require the existence of a PPT black-box straight-line simulator.) We say that Π realizes g with *perfect T-robustness* (for some subset $T \subseteq [n]$) if for every *malicious* adversary $\mathcal{A}$ corrupting the parties in T, and for every tuple $z_{\bar{T}}$ of inputs of uncorrupted parties, the following holds. If g evaluates to 0 on *all* choices of inputs z consistent with $z_{\bar{T}}$, then all uncorrupted parties are guaranteed to output 0.[1] This property is implied by the standard simulation-based notion of security against malicious adversaries.

[1] Notice that we only define robustness for the case that g evaluates to 0, which suffices for our purposes since we only consider functions g representing relations $\mathcal{R}$. More specifically, robustness is used to construct *sound* proofs systems, where the corrupted party is the party holding the witness (and the bits of the input are partitioned between the honest parties). As soundness concerns the case $x \notin L_{\mathcal{R}}$,

Locking schemes [?,?]. Informally, a locking scheme allows a sender S to commit some secret to a receiver R, such that given a key the receiver can "open" the lock and retrieve the secret, whereas without the key this is almost impossible (for a query-bounded receiver). More formally, a locking scheme (S, R) for message space $\mathcal{W}$ consists of a sender S and a receiver R that interact in two phases: *Commitment*, during which S sends a locking oracle L_w to R, thus committing to some $w \in \mathcal{W}$; and *Decommitment*, in which S decommits w by sending R a key K_w that "opens" the lock. The requirements from the locking scheme are as follows. First, for every honestly-generated pair (L_w, K_w), R with key K_w and oracle access to L_w outputs w at the end of the decommitment phase with probability 1 (this is called *perfect completeness*). Second, the scheme is *hiding*, namely without knowing the key, R learns nothing about w, even if he probes many bit coordinates of the lock. Thirdly, we require *binding*, i.e. every (possibly ill-formed) lock commits the sender to *some* value w'. (See [?] for formal definitions.)

3 From MPC Protocols to (Inefficient) EZKPCPPs

We show a general connection between secure MPC protocols and (exact) ZKPCPPs. More specifically, given an NP-relation $\mathcal{R}$, we define the *characteristic function* $g_{\mathcal{R}_m} : \{0,1\}^* \times \{0,1\}^m \to \{0,1\}$ of $\mathcal{R}_m = \{(x, w) \in \mathcal{R} : |x| = m\}$ (or simply g, when $\mathcal{R}, m$ are clear from the context) as follows. $g_{\mathcal{R}_m}(w, x_1, ..., x_m) = 1$ if and only if $(x_1 \circ ... \circ x_m, w) \in \mathcal{R}_m$. Following techniques of Ishai et al. [?], we transform a protocol Π securely realizing $g_{\mathcal{R}_m}$ into an EZKPCPP system for $\mathcal{R}_m$, with perfect zero-knowledge against malicious (query-bounded) verifiers. Concretely, for any $t = t(m)$, if the underlying n-party protocol is t-private (for some $n = n(m, t)$), then the system has perfect zero-knowledge against t-bounded verifiers.

Construction 2 (EZKPCPP from MPC.) *The system is parameterized by a length parameter $m \in \mathbb{N}$, a zero-knowledge parameter $t = t(m)$, and employs an n-party protocol Π realizing $g_{\mathcal{R}_m}$ with perfect adaptive t-privacy and perfect static 1-robustness.*[2] *We assume without loss of generality that the bits $x_1, ..., x_m$ are given as input to $P_1, ..., P_m$.*

Prover algorithm. On input $(x, w), 1^t$ the prover P_E emulates "in his head" a random execution of Π on inputs $(w, x_1, ..., x_m)$. Let $\mathsf{V}_0, ..., \mathsf{V}_n$ be the views of

---

i.e., $(x, w^*) \notin \mathcal{R}$ for every "witness" w^*, then g evaluates to 0 on *every* input of the party holding the witness.

[2] We could get the same results using secure protocols in the *semi-honest* model, by sharing the witness w between $t + 1$ parties (similar to the construction of zero-knowledge protocols from MPC of [?]). However, this solution requires a larger number of parties than in our solution. We prefer to rely on robust protocols, since it suffices to have $\{P_0\}$-robustness, and such protocols can be instantiated by more efficient protocols in the semi-honest model (e.g., [?]).

$P_0, ..., P_n$ in this execution, and for every $0 \leq i < j \leq n$, let $\mathsf{Ch}_{i,j}$ describe the messages sent over the communication channel between i, j during the execution. P_E outputs the proof π consisting of the concatenation of the views $\mathsf{V}_1, ..., \mathsf{V}_n$ and the communication channels $\mathsf{Ch}_{i,j}$ for $0 \leq i < j \leq n$, where every view and communication channel constitutes a symbol of the proof. (Notice that the proof does not *include the view V_0, since V_0 reveals the witness w.)*

Verifier algorithm. The verifier V_E with input $1^t, m$ and oracles x, π flips a random coin to decide which test to perform. If the outcome was 0, V_E picks a random view $\mathsf{V}_i, i \in_R [m]$, and verifies that the input of P_i in the protocol execution was x_i (this ensures the protocol execution is consistent with x). If the outcome was 1, V_E picks $i \in_R [n]$ and $j \in_R \{0, 1, ..., n\}, i \neq j$ and verifies that V_i is consistent with $\mathsf{Ch}_{i,j}$ (this ensures the emulated execution is consistent). In both cases V_E verifies that the output of the protocol (implicit in V_i) was 1.

Lemma 1 (From MPC to EZKPCPPs). *For any NP-relation $\mathcal{R} = \mathcal{R}(x, w)$, Construction 2 is a perfectly t-zero-knowledge exact-PCPP for $\mathcal{R}$, with soundness error $\left(1 - \Omega\left(\frac{1}{n^2}\right)\right)$, where the honest verifier makes only 2 oracle queries.*

Proof (sketch). Set some $m \in \mathbb{N}$ and let $\Pi = \Pi_m$. The perfect completeness follows from the perfect completeness of Π. As for soundness, if $x \notin L_{\mathcal{R}}$ then a malicious prover has three possible courses of action. First, he can emulate an execution of Π on some $x' \neq x$, which is detected by the verifier with probability at least $\frac{1}{2m} \geq \frac{1}{n^2}$. Second, he can provide a proof in which some view V_i is inconsistent with some incident communication channel (either $\mathsf{Ch}_{j,i}, 0 \leq j < i$ or $\mathsf{Ch}_{i,j}, 1 \leq i < j \leq n$), which V_E detects with probability at least $\frac{1}{2n(n+1)}$. Thirdly, P_E can generate a proof in which every view V_i is consistent with all incident communication channels (with respect to Π, x). In this case, it can be shown that there exists an execution of Π on x, in which all parties (except, possibly, P_0) are honest, such that the view of P_i in the execution is V_i, and the messages exchanged between P_i, P_j are according to $\mathsf{Ch}_{i,j}$. Therefore, the P_0-robustness of Π guarantees that the output implicit in $\mathsf{V}_1, ..., \mathsf{V}_n$ is 0, so V_E rejects (with probability 1). We note that the soundness error can be reduces by repetition ($\lceil \frac{t}{2} \rceil$ iterations can be performed while preserving zero-knowledge). The t-zero-knowledge follows from the privacy of Π. Indeed, for every $i, j \in [n]$, the communication channel $\mathsf{Ch}_{i,j}$ can be reconstructed from V_i (and from V_j). Therefore, the answers to the queries of every (possibly malicious) verifier V^* can be simulated given the views of (a specific subset of) t parties $P_{i_1}, ..., P_{i_t}$. By the privacy of Π, these views can be perfectly simulated given $x_{i_1}, ..., x_{i_t}$. Therefore, the view of V^* can be simulated with only t TTP-queries. □

Notice that if we only require *honest-verifier* zero-knowledge, then it suffices for Π to be 1-private. (P_E, V_E) is *weakly-sound* in the sense that its soundness error is large. (As noted above, the error can be reduced by repetition, but this increases the query complexity and again requires Π to be private against larger coalitions of parties.)

We note that Construction 2 is inefficient in the sense that the alphabet size may be exponential in m, t, since it contains symbols for all possible views and communication channels in Π. This inefficiency will not pose a problem in later constructions. Indeed, the construction of honest-verifier ZKPCPPs uses EZKPCPPs only for *constant sized* claims, and the construction of a ZKPCPP (with zero-knowledge against malicious verifiers) uses EZKPCPPs for claims of size $O(\sigma)$, where σ denotes the security parameter of the ZKPCPP. Moreover, we will only use EZKPCPPs for relations in P.

Basing Construction 2 on efficient multiparty protocols that can withstand a constant fraction of corrupted parties, we obtain the following result.

Corollary 1 (EZKPCPPs for NP). *Let $\mathcal{R} = \mathcal{R}(x, w)$ be an NP-relation. Then for every zero-knowledge parameter $t = t(|x|)$, $\mathcal{R} \in \mathrm{EZKPCPP}_{\Sigma}[r, q, \epsilon_{\mathrm{ZK}}, \delta, \epsilon_S, \ell]$, where $\Sigma = 2^{\mathrm{poly}(t,|x|)}$, $r = O(\log t + \log |x|)$, $q = 2$, $\epsilon_{\mathrm{ZK}} = 0$, $\epsilon_S = 1 - \frac{1}{\mathrm{poly}(t,|x|)}$, $\ell = \mathrm{poly}(t, |x|)$, and the EZKPCPP system is t-zero-knowledge. Furthermore, $\mathcal{R}$ has an EZKPCPP system over the binary alphabet with $q = 3$ (and $r, \epsilon_{\mathrm{ZK}}, \epsilon_S, \ell$ are as above).*

The existence of the EZKPCPP over a large alphabet follows from Contruction 2. The natural approach towards reducing the alphabet size, is to define the proofs over Σ and represent every view and communication channel using several symbols, and have the verifier read all the bits corresponding to the symbol he wishes to query. However, this solution does not preserve zero-knowledge. Indeed, it increases the query complexity of the honest verifier, and consequently a malicious (even query bounded) verifier may query many *parts* of views, thus potentially breaking the privacy of the underlying protocol, and consequently the zero-knowledge of the system.

Proof (sketch). The existence of an EZKPCPP system over a large alphabet follows from Lemma 1, when Construction 2, based on an efficient multiparty protocol (e.g., the protocols of [**?**]).

The EZKPCPP over a binary alphabet, denoted $(P_{\mathrm{bin}}, V_{\mathrm{bin}})$, is obtained using techniques of Dwork et al. [**?**]. The general idea is to represent a proof generated by P_E over the binary alphabet, but avoid increasing the query complexity of the honest verifier by having V_{bin} *probabilistically* check that the oracles satisfy the decision circuit of V_E. More specifically, P_{bin} on input (x, w) uses P_E to generate a proof π_E. Then, for every random string r of V_E, P_{bin} writes down the assignment A_r to the inner gates of the verification circuit of V_E (i.e., the circuit V_E uses when he has randomness r). P_{bin} outputs the proof π_E, concatenated with the assignments A_r for all random strings r of V_E. (The proof should actually include, for every r, a proof that A_r is consistent with the verification circuit of V_E and the bits of x, π_E that V_E queries. We refer the reader to [**?**] or the full version for additional details. We note that these "proofs" have length $O(A_r)$ so they can be ignored when analyzing the efficiency properties of the system.) To verify that $x \in L_{\mathcal{R}}$, V_{bin} picks a random string r for V_E, and checks that x, π_E, A_r satisfy a random gate in the verification circuit of V_E.

Notice that V_E reads only poly $(|x|, t)$ bits from his oracles (i.e., the symbols he reads can be represented using poly $(|x|, t)$ bits), and his verification circuit has size poly $(|x|, t)$ (since V_E is efficient in the number of bits he reads). Therefore, the randomness complexity increases by only $O(\log t + \log |x|)$ (V_{bin} needs to pick a random gate in the verification circuit of V_E), and the proof increases by a factor of poly $(|x|, t)$ (there are poly $(|x|, t)$ random strings for V_E, and every random string corresponds to a circuit of size poly $(|x|, t)$). Moreover, the soundness error degrades only by a factor of $\frac{1}{\text{poly}(|x|,t)}$, since in every verification circuit of V_E which x, π_E do not satisfy, at least one gate (out of poly $(|x|, t)$) is not satisfied. Regarding zero-knowledge, notice that every t-bounded verifier algorithm V^*_{bin} in the modified system induces a verifier algorithm V^* in the original system, whose queries correspond to the queries V_E makes in t independent invocations. Therefore, the view of V^* can be simulated given the views $\mathsf{V}_{i_1}, ..., \mathsf{V}_{i_t}$ which V^* queries, and these views can be simulated given the inputs of $P_{i_1}, ..., P_{i_t}$ in Π. As the view of V^*_{bin} can be reconstructed from the view of V^*, this implies the system is t-zero-knowledge. □

Remark 2 (Strong HVZK). Both of the EZKPCPP systems mentioned above have a *stronger* honest-verifier zero-knowledge guarantee, which is formalized next. For an integer parameter c and a soundness parameter ϵ, we say a proof system has (ϵ, c)-strong honest-verifier zero-knowledge, if there exists a straight-line simulator Sim such that the following holds for every $c' \leq c$ and every $(x,) \in \mathcal{R}$. Sim interacts with $V^{c'}$ ($V^{c'}$ denotes c' random and independent invocations of the honest verifier V) without rewinding the verifier. During the simulation, Sim makes only c' TTP queries, and generates a view which is statistically close (up to distance ϵ) to the real-world view of $V^{c'}$, when $V^{c'}$ has oracle access to x and a random honestly-generated proof for x. Both our EZKPCPP systems have *perfect* t-strong honest-verifier zero-knowledge (where t is the zero-knowledge parameter), i.e., the simulated view is indistinguishable from the real world view. (This stronger zero-knowledge feature will be used in Section 4.1 to construct an HVZKPCPP with similar properties, which in turn will be used to construct a ZKPCPP in Section 4.2.)

4 From Efficient PCPPs to Efficient ZKPCPPs

We show a general transformation from PCPPs to ZKPCPPs, and construct an efficient ZKPCPP system for any NP-relation $\mathcal{R}$. (Using the same methods one can transform a PCP into a ZKPCP.) First, we use proof-composition techniques to transform a PCPP into an HVZKPCPP, using an EZKPCPP as the inner proof system. Then, we show a transformation from an HVZKPCPP and a locking scheme, into a ZKPCPP that guarantees zero-knowledge against *malicious* query-bounded verifiers. Finally, by applying the first transformation to an efficient PCPP, and the second to an efficient locking scheme and to the HVZKPCPP obtained through the first transformation, we get an efficient ZKPCPP.

4.1 From PCPPs to HVZKPCPPs

In this section we present the general transformation from PCPPs to HVZKPCPPs. We first describe a basic transformation (with weak parameters), and then improve it. The high-level idea is to use proof composition (see, e.g., [**?**,**?**,**?**]). In the context of PCPs, proof composition is used to reduce the query complexity of a PCP verifier: instead of making q queries and applying some predicate to the oracle answers, the verifier delegates the verification task to an "inner verifier", who probabilistically checks the oracle satisfies the decision circuit of the outer verifier (the query complexity is reduced since the inner verifier makes *less* queries than the original verifier). Intuitively, as the verifier of the composed system emulates the verification procedure of the *inner* verifier, then the composed system should have a zero-knowledge guarantee if the inner system does (even when the outer system has *no* zero-knowledge guarantee). The advantage in using composition in this case is similar to the advantage achieved by composition in standard PCP constructions: the inner system may be *very* inefficient, but the composed system is efficient if the outer system is.

More specifically, let $\mathcal{R}$ be a relation, and let $(P_{\text{out}}, V_{\text{out}})$ be a PCPP for $\mathcal{R}$ with soundness error ϵ and proximity parameter δ, where V_{out} makes q oracle queries and uses r random bits. Then every random string rand of V_{out} corresponds to a set of q queries, and a predicate $\varphi_{\mathsf{rand}} : \{0,1\}^q \to \{0,1\}$ describing the decision of V_{out}. Denote the vector of the 2^r predicates corresponding to *all* random strings of V_{out} by $(\varphi_1, ..., \varphi_{2^r})$, then the following holds. If $x \in L_{\mathcal{R}}$ and π was honestly-generated by P_{out} for x, then (x, π) satisfies φ_i for every $1 \leq i \leq 2^r$, and if x is δ-far from $L_{\mathcal{R}}$ then for *any* "proof" π^*, (x, π^*) satisfies at most an ϵ-fraction of $\varphi_1, ..., \varphi_{2^r}$. In standard proof-composition constructions, the prover concatenates π with proofs $\pi^1_{\text{in}}, ..., \pi^{2^r}_{\text{in}}$, where π^i_{in} should convince the inner verifier that (x, π) satisfies φ_i. The verifier then runs the outer verifier to generate *rand* and φ_{rand}, and the inner verifier to check that (x, π) satisfies φ_{rand}. However, the verification procedure of the inner verifier may query π, and is consequently *not* zero-knowledge (since π may reveal additional information about x). Therefore, we need to use proof composition, together with a form of *secret-sharing* which guarantees that π also remains hidden. Concretely, we replace every proof bit π_i (i.e., every predicate variable corresponding to a proof bit) with a set of bits $\{\pi_{i,j}\}$ (i.e., with a set of *new* predicate variables) such that π_i is reconstructable given all the new bits $\pi_{i,j}$, but (any) subset of the new bits $\{\pi_{i,j}\}$ reveals *no* information about π_i. We refer to a predicate obtained thorough this "secret-sharing" transformation as a *private* predicate, since a partial assignment to few predicate variables reveals no information about the proof π.

More specifically, given a predicate $\varphi : \{0,1\}^q \to \{0,1\}$ over variables $v_1, ..., v_q$, we partition its variables to a set V_{inp} of *input variables* (i.e., variables corresponding to bits of the input oracle) and a set V_{pf} of *proof variables*. The *k-private form of* φ (for any $k \in \mathbb{N}$), denoted $\varphi(k)$, is obtained from φ by replacing every proof variable $v_i \in V_{\text{pf}}$ with the exclusive-or of $k+1$ new variables $y_{i,1}, ..., y_{i,k+1}$ (i.e., every appearance of v_i is replaced with $\bigoplus_{j=1}^{k+1} y_{i,j}$). The

private-predicates relation $\mathcal{R}_{\text{priv}}$ consists of all pairs of private predicates and satisfying assignments for them, i.e.,

$$\mathcal{R}_{\text{priv}} = \{((w, \varphi(k)), \lambda) : \ w \text{ satisfies } \varphi(k)\}.$$

We now describe the basic transformation from PCPPs to (weakly-sound) HVZKPCPPs.

Construction 3 (HVZKPCPPs from PCPPs.) *The basic HVZKPCPP system, denoted (P_B, V_B), will be the composition of a PCPP system $(P_{\text{out}}, V_{\text{out}})$ for $\mathcal{R}$ as the outer PCPP system, and the inner EZKPCPP system $(P_{\text{in}}, V_{\text{in}})$ for the relation $\mathcal{R}_{\text{priv}}$. The system is parameterized by $d \in \mathbb{N}$ which determines the zero-knowledge requirement from the inner system. (Without loss of generality, $d \geq 3$.)*

Prover algorithm. On input $1^d, x, w$ such that $(x, w) \in \mathcal{R}$, P_B:

- *Generates the verification predicates $\varphi_1, ..., \varphi_m$ of V_{out} (for $m := 2^r$, where r denotes the length of the randomness of V_{out}), and a proof $\pi \in P_{\text{out}}(x, w)$.*
- *Generates the d-private form $\varphi_i(d)$ of every predicate φ_i, and replaces every proof variable $v_j \in V_{\text{pf}}$ with the exclusive-or of $d+1$ new variables $y_{j,1}, ..., y_{j,d+1}$, such that $\bigoplus_{k=1}^{d+1} \pi(d)_{y_{j,k}} = \pi_{v_j}$. (As (x, π) is interpreted as an assignment to the predicates $\varphi_1, ..., \varphi_m$, this transforms (x, π) into an assignment to the private predicates. We denote this partial assignment to $\varphi_1(d) \wedge ... \wedge \varphi_m(d)$ by $\pi(d)$.)*[3]
- *"Proves" that $(x, \pi(d))$ satisfies the private predicates. Concretely, let $(x, \pi(d))_i$ denote the restriction of $(x, \pi(d))$ to the variables of the private predicate $\varphi_i(d)$. Then P_B generates a proof $\pi^i_{\text{in}} \in P_{\text{in}}\left(1^d, ((x, \pi(d))_i, \varphi_i(d)), \lambda\right)$ for the claim $(((x, \pi(d))_i, \varphi_i(d)), \lambda) \in \mathcal{R}_{\text{priv}}$.*
- *Outputs the proof $\pi_B = \pi^1_{\text{in}} \circ ... \circ \pi^m_{\text{in}} \circ \pi(d)$.*

Verifier algorithm. V_B on input $1^d, |x|$ and given oracle access to x and a proof $\pi_B = \pi^1_{\text{in}} \circ ... \circ \pi^m_{\text{in}} \circ \pi(d)$, picks an $i \in_R [m]$, uses V_{out} to generate the predicate φ_i, and transforms it into the d-private predicate $\varphi_i(d)$. Then, V runs V_{in} to check that $(x, \pi(d))_i$ satisfies $\varphi_i(d)$ ($(x, \pi(d))_i$ is used as the input oracle, and π^i_{in} as the proof oracle, of V_{in}).

Lemma 2. *Let $\mathcal{R} \in \text{PCPP}[r, q, \delta, \epsilon_{\text{out}}, \ell]$ with the PCPP system $(P_{\text{out}}, V_{\text{out}})$. Let $(P_{\text{in}}, V_{\text{in}})$ be a perfectly d-zero-knowledge EZKPCPP system for $\mathcal{R}_{\text{priv}}$ with soundness error $\epsilon_{\text{in}}(\ell, d)$ (where ϵ_{in} is non-decreasing and ℓ is the length of the input to the EZKPCPP system), in which the honest verifier makes $q_{\text{in}} \leq d$ queries. Then Construction 3, based on $(P_{\text{out}}, V_{\text{out}})$ and $(P_{\text{in}}, V_{\text{in}})$, is a PCPP system for $\mathcal{R}$ with perfect completeness, perfect honest-verifier zero-knowledge, and soundness error $\epsilon_{\text{out}}(1 - \epsilon_{\text{in}}(\ell(d+1), d)) + \epsilon_{\text{in}}(\ell(d+1), d)$. Moreover, V_B makes only q_{in} queries, and the prover generates proofs of length $O_{q,d}(\ell + 2^r)$.*

[3] We say that π is a *partial* assignment to the predicates, since some of the variables are assigned values by the input x.

Proof (sketch). The completeness follows directly from the completeness of the underlying proof systems. As for zero-knowledge, the zero-knowledge of the inner EZKPCPP system guarantees there exists a simulator Sim_{in} that can perfectly simulate the view of the honest verifier V_{in} (since V_{in} is d-query bounded), given oracle access to the input oracle of V_{in} (through the TTP). Notice that the "input oracle" of V_{in} is of the form $(x, \pi(d))_i$ for some $i \in [m]$, i.e., Sim_{in} may query bits of $\pi(d)$. However, in a *random* proof $\pi_B \in_R P_B\left(1^d, x, w\right)$, $\pi(d)$ is a *random* sharing of π (that is, every set of bits $\pi_{j,1}, ..., \pi_{j,d+1}$ that correspond to a bit π_j of π, is random such that $\pi_{j,1} \oplus ... \oplus \pi_{j,d+1} = \pi_j$). Therefore, Sim can simulate the view of V_B by running Sim_{in}, and answering his TTP queries with random bits. These bits are distributed as the answers Sim_{in} would have been given by its TTP, so it suffices to prove indistinguishability conditioned on the "input" oracle $\pi(d)$. In this case, indistinguishability follows from the zero-knowledge of $(P_{\text{in}}, V_{\text{in}})$.

Regarding soundness, if x is δ-far from $L_{\mathcal{R}}$ then the soundness of $(P_{\text{out}}, V_{\text{out}})$ guarantees that for every "proof" π^* at most an ϵ_{out}-fraction of the predicates $\varphi_1, ..., \varphi_m$ are satisfied by (x, π^*). Consequently, for every "proof" $\pi^*(d)$, $(x, \pi^*(d))$ satisfies at most an ϵ_{out}-fraction of the private predicates $\varphi_1(d), ..., \varphi_m(d)$. If V_B chooses to verify a predicate $\varphi(d)_i$ that is *not* satisfied by $x \circ \pi^*(d)$, then the soundness of $(P_{\text{in}}, V_{\text{in}})$ guarantees that he accepts with probability at most $\epsilon_{\text{in}}(q(d+1), d)$. (Indeed, every predicate contains at most q proof variables, so V_{in} has input of length at most $q(d+1)$, and ϵ_{in} is non-decreasing.) □

It is clear from Lemma 2 that the soundness error degrades through this transformation (since the soundness error of the composed system depends not only on the soundness error of the outer PCPP system, but also on that of the inner EZKPCPP system). Therefore, our next goal is to reduce the soundness error.

Reducing the soundness error. The main idea is to have the verifier repeat the verification procedure of V_B. However, we must change the ZKPCPP itself since repetition does not necessarily preserve zero-knowledge. (That is, if the verifier simply repeats the verification procedure, then his queries may exceed the upper bound for which zero-knowledge is guaranteed.) Intuitively, the prover can generate several independent copies of *basic proofs* (i.e., a proof generated by P_B), and the verifier can repeat the basic verification scheme, using a "fresh" proof in every iteration. This "assumption" that the verifier uses every proof at most once, is the reason the system guarantees only honest-verifier zero-knowledge. Indeed, we increase the query complexity of the verifier without increasing the zero-knowledge guarantee of the basic system (since increasing the zero-knowledge parameter will also increase the soundness error). Therefore, a malicious verifier can potentially break the zero-knowledge by using the same proof in several iterations.

Construction 4 (HVZKPCPP) *The modified HVZKPCPP system (P_H, V_H) uses the the system (P_B, V_B) as a building block, and is parameterized by l, the*

number of basic proofs in a proof generated by P_H; t, the number of runs (of V_B) that V_H emulates; and d, to be passed on to the underlying HVZKPCPP system. We assume without loss of generality that $l \geq t$.

Prover algorithm. P_H on input $1^l, 1^d$ and $(x,w) \in \mathcal{R}$, uses P_B to generate l independent *(basic) proofs $\pi_B^1, ..., \pi_B^l$ for the claim $(x,w) \in \mathcal{R}$, and outputs the proof $\pi_H = \pi_B^1 \circ ... \circ \pi_B^l$.*

Verifier algorithm. V_H on input $1^t, l, 1^d, |x|$, and given access to oracles x, π_H, picks at random t different *basic proofs $\pi_B^1, ..., \pi_B^t$, and for every $1 \leq i \leq t$, runs V_B with parameter d and oracles x, π_B^i (all emulations of V_B are performed in parallel). V_H accepts if V_B accepted in all t iterations, otherwise he rejects.*

Theorem 5 (HVZKPCPPs from PCPPs). *Let σ be a security parameter. Then for any $q \in \mathbb{N}$, $\epsilon_S = \epsilon_S(\sigma, |x|)$, $\delta = \delta(\sigma, |x|)$, $r = r(\sigma, |x|)$ and $\ell = \ell(\sigma, |x|)$,*

$$\mathrm{PCPP}\left[r, q, \delta, \frac{1}{2}, \ell\right] \subseteq \mathrm{HVZKPCPP}\left[r', q', \epsilon'_{\mathrm{ZK}} = 0, \delta' = \delta, \epsilon_S, \ell'\right]$$

where $r' = O_q\left(r \cdot \mathrm{poly}\log\frac{1}{\epsilon_S}\right)$, $q' = O_q\left(\log\frac{1}{\epsilon_S}\right)$ and $\ell' = O_q\left((\ell + 2^r)\log\frac{1}{\epsilon_S}\right)$.

Proof (sketch). We take $d = O(1)$ and $t = l = O_q\left(\log\frac{1}{\epsilon_S}\right)$ in Construction 4, which increase the query complexity and proof length (of the basic system) by a factor of $\log\frac{1}{\epsilon_S}$, and the randomness complexity by a factor of $\mathrm{poly}\log\frac{1}{\epsilon_S}$ (since in every iteration the verifier needs to pick a new basic proof to use). Completeness follows from the completeness of the basic HVZKPCPP system. Regarding soundness, the soundness error of (P_B, V_B) is $\frac{1}{2}(1 + \epsilon_{\mathrm{in}})$ (where $\epsilon_{\mathrm{in}} < 1$ is the soundness error of the EZKPCPP, and depends only on q since d is constant). As V_H emulates t *independent* runs of V_B, and accepts only if all iterations succeed, then V_H accepts an x that is δ-far from $L_{\mathcal{R}}$ with probability at most $\left(\frac{1}{2}(1 + \epsilon_{\mathrm{in}})\right)^t = \epsilon_S$ (for an appropriate choice of the constant defining t). As for zero-knowledge, every emulation of V_B can be perfectly simulated (by some simulator Sim_B) while making at most $d = O(1)$ TTP queries, and as the emulations are independent (and use independent basic proof), a simulator Sim for V_H can run Sim_B t independent times, and forward the TTP queries of Sim_B to his own TTP. □

Remark 3 (Strong HVZK). The strong honest-verifier zero-knowledge feature of Construction 2 (see Section 3, Remark 2) implies that both the HVZKPCPP systems described in this section also guarantee $(\epsilon_{\mathrm{ZK}}, q^*)$-strong honest-verifier zero-knowledge, as defined in Remark 2. More specifically, to get $(\epsilon_{\mathrm{ZK}}, q^*)$-strong HVZK it suffices to take $l = \mathrm{poly}(q^*)$ in Construction 4, and use the EZKPCPP (over a binary alphabet) of Section 3 with zero-knowledge parameter $d = O\left(\log\frac{1}{\epsilon_{\mathrm{ZK}}}\right)$. Consequently, the proof length increases by a factor of $\mathrm{poly}\left(q^*, \log\frac{1}{\epsilon_{\mathrm{ZK}}}\right)$, the randomness complexity by a factor of

poly $\left(\log q^*, \log\log \frac{1}{\epsilon_{\mathrm{ZK}}}\right)$, and the query complexity by a factor of $\mathrm{poly}\log \frac{1}{\epsilon_{\mathrm{ZK}}}$. Moreover, if the original PCPP has strong soundness then so does the HVZKPCPP. (To get soundness error ϵ_S on inputs that are δ-far from the relation for some $\delta \in (0,1)$, the proof length, query complexity and randomness complexity increase by a factor of $O\left(\frac{1}{\delta}\right)$.)

It is evident from Theorem 5 that Construction 4 inherits many of its properties from the underlying PCPP system, so efficient PCPPs yield efficient HVZKPCPPs. More specifically, we can use the following PCPP due to Dinur [**?**], to obtain an efficient HVZKPCPP.

Theorem 6 (PCPP, implicit in [?]). *Let $\mathcal{R} = \mathcal{R}(x,w) \subseteq \mathrm{DTIME}(t(n))$, then $\mathcal{R}$ has a strong PCPP system (P,V) with constant rejection ratio, such that V on inputs of length n tosses $O(\log t(n))$ coins and reads $O(1)$ bits from his oracles.*

Plugging the PCPP system of Theorem 6 into Theorem 5, we get the following result.

Corollary 2 (Efficient HVZKPCPP). *Let ϵ be a soundness parameter and let δ be a proximity parameter. Then every relation $\mathcal{R} = \mathcal{R}(x,w) \in \mathrm{DTIME}(t(n))$ has an HVZKPCPP system (P_H, V_H) with perfect completeness, perfect honest-verifier zero-knowledge, and soundness error ϵ with proximity parameter δ. On input x, P_H generates a proof of size $\mathrm{poly}\left(t(|x|), \log\frac{1}{\epsilon}, \frac{1}{\delta}\right)$ and V_H makes $O\left(\frac{1}{\delta}\log\frac{1}{\epsilon}\right)$ queries.*

4.2 From HVZKPCPPs and Locking Schemes to ZKPCPPs

In this section we construct a ZKPCPP with zero-knowledge against *arbitrary* query-bounded verifiers, from a locking scheme and an HVZKPCPP with strong honest-verifier zero-knowledge (see Remark 2 for a discussion of this zero-knowledge property). We first give a high-level description of the transformation. For $q^* \in \mathbb{N}$, let (P_H, V_H) be an HVZKPCPP with q^*-strong honest-verifier zero-knowledge (e.g., the system of Construction 4, see Remark 3). Intuitively, all we need to do to achieve zero-knowledge against *arbitrary* (q^*-bounded) verifiers is to force the queries of every (possibly malicious) verifier to be distributed as the queries in q^* random and independent invocations of V_H. Following techniques of Kilian et al. [**?**], we achieve this by employing a locking scheme. Hiding a few technical details, the high-level idea is as follows. The proof consists of three sections: the PCPP section in which the prover P locks (using the locking scheme) every bit of the HVZKPCPP; the PERM section which contains a locked permutation of the random strings of the honest verifier V_H (namely, P picks a random permutation τ over the space of random strings of V_H, and for every possible random string r of V_H, P lock the image $\tau(r)$ in the PERM section); and the MIX section, where the location indexed by $\tau(r)$ contains r and the collection of keys for the locks holding the HVZKPCPP bits V_H (with

randomness r) queries. To verify the proof, V picks a random string r', queries $\mathrm{MIX}_{r'}$, and retrieves some (other) random string r and a set of keys, which he uses to unlock the corresponding locks. Then, V verifies that the lock PERM_r holds the string r' and that V_H would accept (if he was given the HVZKPCPP bits locked in the PCPP section of the proof).

Applying this transformation to an efficient HVZKPCPP and an efficient locking scheme, we get the following result (full details are deferred to the full version).

Theorem 7 (Efficient ZKPCPP). *Let ϵ be a soundness parameter, let δ be a proximity parameter and let $q^* \in \mathbb{N}$. Then every relation $\mathcal{R}(x,w) \in \mathrm{DTIME}(t(n))$ has a ZKPCPP system (P,V) with soundness error ϵ, proximity parameter δ, and straight-line (ϵ, q^*)-zero-knowledge. P on input x generates proofs of length $\mathrm{poly}\left(t(|x|), q^*, \log\frac{1}{\epsilon}, \frac{1}{\delta}\right)$ and V on input $|x|$ makes $\mathrm{poly}\left(\log t(|x|), \log q^*, \log\frac{1}{\epsilon}, \frac{1}{\delta}\right)$ queries.*

(P,V) inherits its properties from those of the HVZKPCPP and the locking scheme combined. More specifically, perfect completeness follows from the perfect completeness of both building blocks. As for soundness, the binding of the locking scheme guarantees the proof oracle V uses to emulate V_H is consistent with *some* proof oracle for V_H, and therefore (by the soundness of (P_H, V_H)) if x is far from $L_{\mathcal{R}}$ then V_H rejects (with high probability). As for zero-knowledge, the hiding of the locking scheme guarantees that by probing the locks, V learns almost nothing about the values locked within them. Therefore, V can only "hope" to gather some information by retrieving the keys and using them to open the locks (i.e. by reading MIX entries and then the corresponding PCPP entries). However, in this case the random permutation τ guarantees that his queries to π_H are distributed as in random and independent emulations of V_H, so the oracle-answers to his queries can be simulated (by the strong honest-verifier zero-knowledge of (P_H, V_H)).

The adaptivity of the honest verifier. Unlike our HVZKPCPP systems (Section 4.1), the verifier in Theorem 7 is inherently adaptive. Indeed, to decommit the locks the verifier must first retrieve the corresponding keys from the appropriate MIX entry, and therefore cannot make his queries non-adaptively. However, all iterations of the verification procedure may be executed in parallel (i.e. all MIX-queries are asked simultaneously, all locks are then simultaneously unlocked etc.), giving a verifier with adaptivity $k_{\mathrm{lock}} + 1$, where k_{lock} is the adaptivity of the locking scheme receiver.

5 Cryptographic Applications

In this section we describe several applications of ZKPCPPs. Concretely, we construct two-party and multiparty protocols that allow a dealer to commit to a secret and prove (with sublinear communication) that it satisfies an NP predicate.

5.1 Certifiable VSS

Motivated by applications that require verification with no information leakage, we focus on reducing the communication complexity of verifying the shares in a verifiable secret sharing (VSS) protocol [**?**,**?**,**?**,**?**,**?**]. Roughly speaking, VSS allows a dealer D to distribute a secret s among n servers in a way that prevents a coalition of up to t servers from learning or modifying the secret, while on the other hand guaranteeing unique reconstruction, even if D and up to t servers can collude. We study a *certifiable* variant of VSS (which we call cVSS) which is similar to traditional VSS, except that it provides the additional guarantee that the secret satisfies some NP predicate. Similar to [**?**], to achieve sublinear verification we consider networks that include an additional receiver entity R who eventually receives the secret, and may assist in the verification. We now provide more details about the model we consider.

We assume that the participating parties can interact over a synchronous network of secure point-to-point channels. The parties also have access to a *broadcast* channel, where a message sent over this channel is received by all other parties. When measuring communication complexity, we count a message sent over a broadcast channel only once towards the total communication. Alternatively, our protocols can be implemented with similar communication complexity using a public bulletin board, where every time a message is written to or read from the board is counted towards the communication complexity.

The security of protocols is defined by considering their execution in the presence of a malicious, static adversary, who may corrupt and control a subset of the parties. The adversary is capable of *rushing*, namely sending his messages only after receiving all messages sent by honest parties in the same round.

A cVSS protocol for an NP-relation $\mathcal{R}$ consists of three phases. In the *sharing* phase, the dealer D is give input $(x, w) \in \mathcal{R}$ and sends a message to each server. In the *verification* phase, the receiver R can freely interact with the servers, possibly using a broadcast channel. Finally, in the *reconstruction* phase, each server sends a single message to R, and R reconstructs the secret. We note that R and the servers are given $1^{|x|}$ as input.

A protocol as above is said to be (t, ϵ)-secure if it satisfies the following requirements:

- *Correctness.* For every adversary $\mathcal{A}$ corrupting t out of n servers and for every $(x, w) \in \mathcal{R}$, in the end of the reconstruction phase R outputs x, except with at most ϵ probability.
- *Secrecy.* For every adversary $\mathcal{A}$ corrupting R and t servers there exists a PPT simulator Sim such that for every $(x, w) \in \mathcal{R}$, $\mathsf{View}_{\mathcal{A}}(x, w) \approx^{\epsilon} \mathsf{Sim}(|x|)$, where $\mathsf{View}_{\mathcal{A}}(x, w)$ denotes the view of $\mathcal{A}$ during the sharing and verification phases.
- *Commitment.* For every adversary $\mathcal{A}$ corrupting R and t servers and for every $(x, w) \in \mathcal{R}$, the following holds except with at most ϵ failure probability over the randomness of the sharing and verification phases. In the end of the verification phase, either R outputs $\perp$, or there is a unique secret x^* (determined by the messages exchanged up to this point), such that $x^* \in L_{\mathcal{R}}$,

$|x^*| = |x|$, and R will output x^* regardless of the messages sent by the adversary during the reconstruction phase.

We note that traditional VSS is stronger than our certifiable VSS in that the verification phase does not involve the receiver R. Thus, when there are multiple receivers, traditional VSS can guarantee that the same secret x^* is reconstructed by all receivers. However, traditional VSS protocols do not guarantee that the reconstructed secret possess any specific properties, as guaranteed by certifiable VSS, and also do not achieve sublinear verification. (We note that certifiable VSS can be implemented using general MPC protocols, but the communication complexity required to verify the shares will not be sublinear.)

CERTIFIABLE VSS FROM ZKPCPPS. The protocol uses a ZKPCPP system (P, V), and a robust secret sharing scheme. (A robust secret sharing scheme maps a secret x into a vector $(s_1, ..., s_m)$ of shares such that "few" shares reveal no information on x, but x can be reconstructed from the shares even if "few" of them are replaced with incorrect values.) We note that for the protocol to be efficient, P, V should be efficient, as well as the sharing and reconstruction algorithms of the secret sharing scheme.

In the sharing phase, the dealer D secret shares $x \in L_{\mathcal{R}}$ using the secret sharing scheme, generates a ZKPCPP for the claim 'the secret shares are "close" to a vector of "legal" secret shares and $x \in L_{\mathcal{R}}$', and partitions the shares and the proof between the servers. In the verification phase, the receiver R verifies that the shares D distributed are close to a sharing of some $x' \in L_{\mathcal{R}}$ by emulating a the verifier V, where R broadcasts the queries of V and the servers answer. (The use of broadcast prevents R from contacting too many servers, which would violate the secrecy requirement.) If the verification fails, R outputs $\perp$ and ignores further messages. For reconstruction, the servers holding the secret shares send them to R, who reconstructs the secret x.

This description is in fact an over-simplification of the actual protocol. Indeed, the verification procedure of the ZKPCPP cannot be used as-is since in the context of VSS, verification is executed in the presence of an adversary that can determine the answers of the corrupted servers *after seeing the queries of the verifier*, while ZKPCPPs guarantee completeness and soundness when the verification is performed *with oracles* (in particular, the oracle answers are *independent* of the queries). Intuitively, to restrict the influence the adversary has on the verification procedure, it suffices to guarantee that symbols held by corrupted servers are queried with low probability, which can be done as follows. The dealer distributes *several copies* of the secret shares and the proof, and the value of every specific symbol V queries is determined by the majority vote over the corresponding symbols in several *randomly selected* copies. (This can be thought of as applying a sort of "error correction" to the symbols of the secret shares and the proof.) In addition, the verification procedure is repeated several times, and the verification phase passes (i.e., R does not abort) only if V accepted in most of the iterations. Further details are differed to the full version.

The secrecy property follows from the secrecy of the secret sharing scheme and from the straight-line zero-knowledge property of (P, V). Indeed, the zero

knowledge implies R learns only few shares, and the secrecy of the secret sharing scheme guarantees that these shares reveal no information on x. (Straight-line zero-knowledge is required to guarantee that the view of the adversary can be simulated.) The "error correction" applied to the secret shares and the proof guarantees that with high probability, corrupted servers are queried only in few of the emulations of V. Therefore, in most emulations we can think of the verification as being performed *with oracles*, which is useful both for correctness and for binding. Indeed, for correctness, if D is honest then with high probability V accepts in most iterations (by the completeness of the ZKPCPP), and the robustness of the secret sharing guarantees that R will reconstruct $x^* = x \in L_{\mathcal{R}}$ in the end of the reconstruction phase, even if t servers are corrupted. As for binding, a corrupted D has 2 possible courses of actions. First, if he distributes a shares vector that is far from every "legal" shares-vector, or close to a shares vector of some $x^* \notin L_{\mathcal{R}}$, then the soundness of the ZKPCPP implies that V rejects in most of the emulations (since corrupted servers are queried only in few of these emulations), so R outputs $\perp$ with high probability. Second, if he distributes a shares vector that is close to a legal shares vector of some $x^* \in L_{\mathcal{R}}$, then either R outputs $\perp$ at the end of the verification phase, or x^* will be reconstructed (due to the robustness of the secret sharing). Thus, we obtain the following result.

Theorem 8 (verification-efficient certifiable VSS). *Let $\mathcal{R} = \mathcal{R}(x, w)$ be an NP-relation. Then for every corruption threshold $t \in \mathbb{N}$ and every soundness parameter ϵ, there exists a (t, ϵ)-secure cVSS protocol for $\mathcal{R}$, with $n =$ poly $\left(|x|, t, \log \frac{1}{\epsilon}\right)$ servers, total communication complexity* poly $\left(|x|, t, \log \frac{1}{\epsilon}\right)$*, and a verification phase that uses* poly $\left(\log |x|, \log t, \log \frac{1}{\epsilon}\right)$ *bits of communication.*

Our certified VSS protocol has non-interactive single-round sharing and reconstruction phases, and a 6-round verification phase. During the sharing phase D sends a single bit to each server, and during the reconstruction phase every server sends a single bit to R. The servers are deterministic and the communication complexity of every server (throughout the protocol) is $O(1)$. Moreover, there is no direct communication between the servers.

5.2 Two-party Commit-and-Prove

We use ZKPCPPs to construct a "2-party analog" of cVSS, or alternatively, a "certifiable" generalization of a commitment scheme, which we call *Commit-and-Prove*. A commitment scheme is a two-phase protocol between a sender S and a receiver R. In the first phase, called the *commit* phase, the server on input x freely interacts with R (who has input $1^{|x|}$). The messages exchanged between S, R during the phase are called the *commitment*. In the second phase, called the *reveal* pahse, S sends x, together with a decommitment string dec to R, and R decides whether to accept of reject x, based on dec and the commitment.

A commitment scheme should have the following properties. First, it should be *hiding*, in the sense that a (possibly malicious) receiver interacting with the

honest sender learns nothing about the secret x during the commit phase. Second, it should be *binding*, namely there exists no efficient malicious sender that, after the interaction with R during the commit phase, can find distinct x, x' of the same length, and two decommitment strings $\mathsf{dec}, \mathsf{dec}'$, such that R would have accepted x, x' with decommitment $\mathsf{dec}, \mathsf{dec}'$, respectively.

A commit-and-prove protocol is *certifiable* in the sense that S not only commits to x, but also proves it satisfies some predicate. (The relation between commitment schemes and commit-and-prove protocols is similar to the relation between VSS and cVSS.) Specifically, it is similar to a commitment scheme, but at the end of the reveal phase R either outputs x and $x \in L_{\mathcal{R}}$ (for some relation $\mathcal{R}$), or aborts. As R, S are both efficient algorithms, the sender cannot generally be expected to find on its own a "witness" to the fact that x satisfies the predicate (think, for example, of an NP predicate). Therefore, S is given a witness w (in addition to the input x).

We say a commit-and-prove protocol for a relation $\mathcal{R}$ is *secure* if it satisfies the following requirements:

- *Correctness.* For every $(x, w) \in \mathcal{R}$, if S, R are honest then R outputs x at the end of the reveal phase.
- *Binding.* Every efficient (possibly malicious) sender algorithm S^* wins the following game with only negligible probability. First, S^* interacts with R in the commit phase, with common input 1^n. Then, S^* outputs two pairs $(x, \mathsf{dec}), (x', \mathsf{dec}')$ such that $|x| = |x'| = n$. S^* wins if R would have accepted x, x' given the decommitments $\mathsf{dec}, \mathsf{dec}'$ (respectively), and in addition either $x \neq x'$ or $x \notin L_{\mathcal{R}}$.
- *Hiding.* There exists a PPT oracle machine Sim such that for every (possibly malicious) PPT receiver algorithm R^* and for every sender input $(x, w) \in \mathcal{R}$, $\mathsf{Sim}^{R^*}(|x|)$ is computationally indistinguishable from the view of R^* during the commitment phase, when interacting with $S(x, w)$.
- *Zero-knowledge after reveal.* There exists a PPT oracle machine Sim such that for every (possibly malicious) PPT receiver algorithm R^* and for every sender input $(x, w) \in \mathcal{R}$, $\mathsf{Sim}^{R^*}(x)$ is computationally indistinguishable from the view of R^* during the *entire* interaction with $S(x, w)$.

Similar to standard commitments, one can also consider stronger variants in which the binding or the hiding property is statistical. Our construction in fact satisfies the statistical variant of hiding and zero-knowledge after reveal.

Using techniques similar to those employed by [**?**] to construct sublinear ZK arguments, we construct a commit-and-prove protocol with *polylogarithmic* communication during the commit phase, and the protocol makes a *black-box* use of an exponentially-hard collision-resistant hash function. (By relaxing the communication requirements such that the communication during commit is sublinear, instead of polylogarithmic, the protocol can be based on a super-polynomially hard hash function.)

Commit-and-prove from ZKPCPPs. As in the cVSS protocol described in Section 5.1, the protocol is based on a robust secret sharing scheme, and an

HVZKPCPP system (P, V), e.g., the HVZKPCPP system of Construction 3. (Notice that *honest-verifier* zero-knowledge suffices in this case, since the sender can refuse to answer queries the honest ZKPCPP verifier would not make.) In addition, the protocol employs a family $\mathcal{H}$ of collision resistent hash functions. In the commit phase, R chooses a function $h \in \mathcal{H}$ and sends (the index of) h to S. S secret-shares $x \in L_{\mathcal{R}}$ into shares $s_1, ..., s_n$ and, using P, generates a proof π for the claim that the secret shares are "close" to the shares of some $x^* \in L_{\mathcal{R}}$. Next, S commits to π using a computationally-binding and statistically-hiding commitment scheme Com_h,[4], and "compresses" the commitments, using a "Merkle Hash Tree" [**?**]. (That is, the commitments are compressed by repeatedly applying the hash function h to pairs of adjacent strings, where every application of h shrinks the input. Thus, a "tree" of hash values is generated, and its root is used as the compressed commitment.) S commits to $(s_1, ..., s_n)$ in a similar manner. Then, S sends the compressed commitments C_π (of π) and C_x (of $(s_1, ..., s_n)$) to R, and R verifies the commitments as follows. R picks a randomness r for V and sends it to S. S determines the set Q of queries V, with randomness r, would have made, and answers every query $q \in Q$ by decommitting the corresponding bit of $\pi, s_1, ..., s_n$ (using the reveal phase of Com_h) and sending the pre-images of all the hash values computed along the path in the Merkle hash tree leading from that bit to the root. R verifies that the values on the paths are consistent with C_π, C_x and h, that V makes the queries Q when using randomness r, and that V would accept given these oracle answers. In the reveal phase, S sends R the entire hash tree used to compress the commitments to $s_1, ..., s_n$, together with the random strings used to generates the commitments (through Com_h) of $s_1, ..., s_n$. R verifies that the commitments and the Merkle tree are consistent with $s_1, ..., s_n$, and if so reconstructs x from the shares, and outputs it.

The properties of the protocol follow from a combination of the properties of the HVZKPCPP, the secret sharing scheme, and the collision-resistent hash function. More specifically, hiding follows from the secrecy of the secret sharing scheme and from the zero-knowledge property of (P, V), and holds even if the commitments to $s_1, ..., s_n, \pi$ are not compressed (the compression is required to "save" on communication during commit). Indeed, by the zero-knowledge of (P, V) even a malicious R^* learns only few shares, which reveal no information on x. Zero-knowledge after reveal follows in a similar manner, since a simulator for the entire view of a (possibly malicious) receiver R^* receives x, and can therefore emulate a simulation of a malicious verifier in the underlying HVZKPCPP system. As for binding, the collision-resistance of h and the binding of Com_h guarantee that except with negligible probability, S is committed to some shares vector $(s_1^*, ..., s_n^*)$. (If C_x is inconsistent with all possible Merkle hash tree commitments to all vectors $(s_1^*, ..., s_n^*)$ then R necessarily aborts during the reveal phase.) Therefore, if $(s_1^*, ..., s_n^*)$ is far from every "legal" shares-vector, or close to a shares vector of some $x^* \notin L_{\mathcal{R}}$, then R detects this during the commit phase

[4] Such a scheme can be constructed from a collision-resistant hash function, with no additional assumptions. Moreover, if the hash function has exponential hardness then the resultant commitments can be polylogarithmic (in the length of the input).

with high probability (even when interacting with a malicious sender S^*). Otherwise, $(s_1^*, ..., s_n^*)$ is "close" to a legal shares-vector of some $x^* \in L_{\mathcal{R}}$, meaning the only value a (possibly malicious) S^* can successfully decommit during the reveal phase, is x^*.

We note that the protocol as described above achieves a constant error, which can be reduced (while preserving hiding and zero-knowledge after reveal) by sequential repetition of the commit phase (further details are differed to the full version). Consequently, we obtain the following result.

Theorem 9 (Sublinear Commit-and-Prove). *Let $\mathcal{H}$ be any family of exponentially-hard collision-resistant hash functions. Then there exists a computationally-binding and statistically-hiding Commit-and-Prove protocol with negligible soundness error and polylogarithmic communication complexity during the Commit phase. Moreover, the protocol makes only black-box use of $\mathcal{H}$.*

We note that if $\mathcal{H}$ only satisfies the usual notion of super-polynomial hardness, the communication complexity (during commit) of the resulting Commit-and-Prove protocol can be $O(n^\epsilon)$, for an arbitrarily small $\epsilon > 0$.

A NON-BLACK-BOX ALTERNATIVE. We have shown how to apply ZKPCPPs towards obtaining sublinear-communication commit-and-prove protocols that make a black-box use of any collision-resistant hash function. Settling for a non-black-box use of the hash function, one could avoid the use of ZKPCPPs by combining a sublinear commitment Com with sublinear zero-knowledge arguments of knowledge [**?**,**?**]. Concretely, during the commit phase S first commits to x using Com, and then proves to R that he knows a witness w and randomness r such that (x, r) are consistent with the transcript of Com and $(x, w) \in \mathcal{R}$. For the reveal phase, S sends (x, r) to R. Both of the above primitives can be based on a collision-resistant hash function. However, the commit phase of the protocol is inherently non-black-box because the sublinear argument applies to an NP-relation which depends on the hash function (since Com depends on the hash function).