Multi-Theorem Preprocessing NIZKs from Lattices[*]

Sam Kim and David J. Wu

Stanford University

Abstract. Non-interactive zero-knowledge (NIZK) proofs are fundamental to modern cryptography. Numerous NIZK constructions are known in both the random oracle and the common reference string (CRS) models. In the CRS model, there exist constructions from several classes of cryptographic assumptions such as trapdoor permutations, pairings, and indistinguishability obfuscation. Notably absent from this list, however, are constructions from standard *lattice* assumptions. While there has been partial progress in realizing NIZKs from lattices for specific languages, constructing NIZK proofs (and arguments) for all of NP from standard lattice assumptions remains open.

In this work, we make progress on this problem by giving the first construction of a *multi-theorem* NIZK argument for NP from standard lattice assumptions in the *preprocessing* model. In the preprocessing model, a (trusted) setup algorithm generates proving and verification keys. The proving key is needed to construct proofs and the verification key is needed to check proofs. In the multi-theorem setting, the proving and verification keys should be reusable for an unbounded number of theorems without compromising soundness or zero-knowledge. Existing constructions of NIZKs in the preprocessing model (or even the designated-verifier model) that rely on weaker assumptions like one-way functions or oblivious transfer are only secure in a single-theorem setting. Thus, constructing multi-theorem NIZKs in the preprocessing model does not seem to be inherently easier than constructing them in the CRS model.

We begin by constructing a multi-theorem preprocessing NIZK directly from context-hiding homomorphic signatures. Then, we show how to efficiently implement the preprocessing step using a new cryptographic primitive called *blind homomorphic signatures*. This primitive may be of independent interest. Finally, we show how to leverage our new lattice-based preprocessing NIZKs to obtain new malicious-secure MPC protocols purely from standard lattice assumptions.

1 Introduction

The concept of zero-knowledge is fundamental to theoretical computer science. Introduced in the seminal work of Goldwasser, Micali, and Rackoff [62], a zero-knowledge proof system enables a prover to convince a verifier that some statement

[*]The full version of this paper is available at https://eprint.iacr.org/2018/272.pdf.

is true without revealing *anything more* than the truth of the statement. Traditionally, zero-knowledge proof systems for NP are interactive, and in fact, interaction is essential for realizing zero-knowledge (for NP) in the standard model [61].

Non-interactive zero-knowledge. Nonetheless, Blum, Feldman, and Micali [16] showed that meaningful notions of zero-knowledge are still realizable in the non-interactive setting, where the proof consists of just a *single* message from the prover to the verifier. In the last three decades, a beautiful line of works has established the existence of NIZK proof (and argument) systems for all of NP in the random oracle model [45,81] or the common reference string (CRS) model [44,40,66,65,86], where the prover and the verifier are assumed to have access to a common string chosen by a trusted third party. Today, we have NIZK candidates in the CRS model from several classes of cryptographic assumptions:[1] (doubly-enhanced) trapdoor permutations [44,40,65], pairings [66], and indistinguishability obfuscation [86]. Notably absent from this list are constructions from lattice assumptions [6,83]. While some partial progress has been made in the case of specific languages [79,7], the general case of constructing NIZK proofs (or even arguments) for all of NP from standard lattice assumptions remains a longstanding open problem.

NIZKs in a preprocessing model. In this work, we make progress on this problem by giving the first *multi-theorem* NIZK argument for NP from standard lattice assumptions in the *preprocessing* model. In the NIZK with preprocessing model [42], there is an initial (trusted) setup phase that generates a proving key k_P and a verification key k_V. The proving key is needed to construct proofs while the verification key is needed to check proofs. In addition, the setup phase is run *before* any statements are proven (and thus, must be statement-independent). In the multi-theorem setting, we require that soundness holds against a prover who has oracle access to the verifier (but does not see k_V), and that zero-knowledge holds against a verifier who has oracle access to the prover (but does not see k_P). The NIZK with preprocessing model generalizes the more traditional settings under which NIZKs have been studied. For instance, the case where k_P is public (but k_V is secret) corresponds to designated-verifier NIZKs [36,39,34], while the case where both k_P and k_V are public corresponds to the traditional CRS setting, where the CRS is taken to be the pair (k_P, k_V).

Why study the preprocessing model? While the preprocessing model is weaker than the more traditional CRS model, constructing multi-theorem NIZK arguments (and proofs) in this model does not appear to be any easier than constructing them in the CRS model. Existing constructions of NIZKs in the preprocessing model from weaker assumptions such as one-way functions [42,75,38,69] or oblivious transfer [73] are only secure in the *single-theorem* setting. As we discuss in greater detail in Remark 4.7, the constructions from [42,75,38] only

[1]There are also NIZK candidates based on number-theoretic assumptions [16,41,15] which satisfy weaker properties. We discuss these in greater detail in Section 1.2 and Remark 4.7.

provide single-theorem zero-knowledge, while the constructions in [73,69] only provide single-theorem soundness. Even in the designated-verifier setting [36,39,34] (where only the holder of a verification key can verify the proofs), the existing constructions of NIZKs for NP based on linearly-homomorphic encryption suffer from the so-called "verifier-rejection" problem where soundness holds only against a *logarithmically-bounded* number of statements. Thus, the only candidates of multi-theorem NIZKs where soundness and zero-knowledge hold for an *unbounded* number of theorems are the constructions in the CRS model, which all rely on trapdoor permutations, pairings, or obfuscation. Thus, it remains an interesting problem to realize multi-theorem NIZKs from lattice assumptions even in the preprocessing model.

Moreover, as we show in Section 6.1, multi-theorem NIZKs in the preprocessing model suffice to instantiate many of the classic applications of NIZKs for boosting the security of multiparty computation (MPC) protocols. Thus, our new constructions of reusable NIZK arguments from standard lattice assumptions imply new constructions of round-optimal, near-optimal-communication MPC protocols purely from lattice assumptions. Our work also implies a *succinct* version of the classic Goldreich-Micali-Wigderson compiler [59,60] for boosting semi-honest security to malicious security, again purely from standard lattice assumptions. Furthermore, studying NIZKs in the preprocessing model may also serve as a stepping stone towards realizing NIZKs in the CRS model from standard lattice assumptions. For example, the starting point of the first multi-theorem NIZK construction by Feige, Lapidot, and Shamir [44] was a NIZK proof for graph Hamiltonicity in the preprocessing model.

1.1 Multi-Theorem Preprocessing NIZKs from Lattices

The focus of this work is on constructing NIZKs in the preprocessing model (which we will often refer to as a "preprocessing NIZK") from standard lattice assumptions. As we discuss in Section 1.2 and in Remark 4.7, this is the first candidate of reusable (i.e., multi-theorem) NIZK arguments from a standard lattice assumption. Below, we provide a high-level overview of our main construction.

Homomorphic signatures. A *homomorphic signature* scheme [18,19,63,5] enables computations on *signed* data. Specifically, a user can sign a message x using her private signing key to obtain a signature σ. Later on, she can delegate the pair (x, σ) to an untrusted data processor. The data processor can then compute an arbitrary function g on the signed data to obtain a value $y = g(x)$ along with a signature $\sigma_{g,y}$. The computed signature $\sigma_{g,y}$ should certify that the value y corresponds to a *correct* evaluation of the function g on the original input x. In a *context-hiding* homomorphic signature scheme [22,18], the computed signature $\sigma_{g,y}$ also *hides* the input message x. Namely, the pair $(y, \sigma_{g,y})$ reveals no information about x other than what could be inferred from the output $y = g(x)$. Gorbunov et al. [63] gave the first construction of a context-hiding homomorphic signature scheme for general Boolean circuits (with bounded depth) from standard lattice assumptions.

From homomorphic signatures to zero-knowledge. The notion of context-hiding in a homomorphic signature scheme already bears a strong resemblance to zero-knowledge. Namely, a context-hiding homomorphic signature scheme allows a user (e.g., a prover) to certify the result of a computation (e.g., the output of an NP relation) without revealing any additional information about the input (e.g., the NP witness) to the computation. Consider the following scenario. Suppose the prover has a statement-witness pair (x, w) for some NP relation $\mathcal{R}$ and wants to convince the verifier that $\mathcal{R}(x, w) = 1$ without revealing w. For sake of argument, suppose the prover has obtained a signature σ_w on the witness w (but does not have the signing key for the signature scheme), and the verifier holds the verification key for the signature scheme. In this case, the prover can construct a zero-knowledge proof for x by evaluating the relation $\mathcal{R}_x(w) := \mathcal{R}(x, w)$ on (w, σ_w). If $\mathcal{R}(x, w) = 1$, then this yields a new signature $\sigma_{\mathcal{R},x}$ on the bit 1. The proof for x is just the signature $\sigma_{\mathcal{R},x}$. Context-hiding of the homomorphic signature scheme says that the signature $\sigma_{\mathcal{R},x}$ reveals no information about the input to the computation (the witness w) other than what is revealed by the output of the computation (namely, that $\mathcal{R}(x, w) = 1$). This is precisely the zero-knowledge property. Soundness of the proof system follows by unforgeability of the homomorphic signature scheme (if there is no w such that $\mathcal{R}_x(w) = 1$, the prover would not be able to produce a signature on the value 1 that verifies according to the function $\mathcal{R}_x$).

While this basic observation suggests a connection between homomorphic signatures and zero-knowledge, it does not directly give a NIZK argument. A key problem is that to construct the proof, the prover must already possess a signature on its witness w. But since the prover does not have the signing key (if it did, then the proof system is no longer sound), it is unclear how the prover obtains this signature on w without interacting with the verifier (who could hold the signing key). This is the case even in the preprocessing model, because we require that the preprocessing be statement-independent (and in fact, reusable for arbitrarily many adaptively-chosen statements).

Preprocessing NIZKs from homomorphic signatures. Nonetheless, the basic observation shows that if we knew ahead of time which witness w the prover would use to construct its proofs, then the setup algorithm can simply give the prover a homomorphic signature σ_w on w. To support this, we add a layer of indirection. Instead of proving that it knows a witness w where $\mathcal{R}(x, w) = 1$, the prover instead demonstrates that it has an encryption ct_w of w (under some key sk), and that it knows some secret key sk such that ct decrypts to a valid witness w where $\mathcal{R}(x, w) = 1$.[2] A proof of the statement x then consists of the encrypted witness ct_w and a proof $\pi_{\mathcal{R},x,\mathsf{ct}_w}$ that ct_w is an encryption of a satisfying witness (under *some* key). First, if the encryption scheme is semantically-secure and the proof is zero-knowledge, then the resulting construction satisfies (computational) zero-knowledge. Moreover, the witness the prover uses to construct $\pi_{\mathcal{R},x,\mathsf{ct}_w}$ is always the same: the secret key sk. Notably, the witness is statement-independent

[2]This is a classic technique in the construction of non-interactive proof systems and has featured in many contexts (e.g., [87,56]).

and can be reused to prove arbitrarily many statements (provided the encryption scheme is CPA-secure).

This means we can combine context-hiding homomorphic signatures (for general circuits) with any CPA-secure symmetric encryption scheme to obtain NIZKs in the preprocessing model as follows:

- **Setup:** The setup algorithm generates a secret key sk for the encryption scheme as well as parameters for a homomorphic signature scheme. Both the proving and verification keys include the public parameters for the signature scheme. The proving key k_P additionally contains the secret key sk and a signature σ_{sk} on sk.
- **Prove:** To generate a proof that an NP statement x is true, the prover takes a witness w where $\mathcal{R}(x, w) = 1$ and encrypts w under sk to obtain a ciphertext ct_w. Next, we define the witness-checking function $\mathsf{CheckWitness}[\mathcal{R}, x, \mathsf{ct}_w]$ (parameterized by $\mathcal{R}$, x, and ct_w) that takes as input a secret key sk and outputs 1 if $\mathcal{R}(x, \mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}_w)) = 1$, and 0 otherwise. The prover homomorphically evaluates $\mathsf{CheckWitness}[\mathcal{R}, x, \mathsf{ct}_w]$ on $(\mathsf{sk}, \sigma_{\mathsf{sk}})$ to obtain a new signature σ^* on the value 1. The proof consists of the ciphertext ct_w and the signature σ^*.
- **Verify:** Given a statement x for an NP relation $\mathcal{R}$ and a proof $\pi = (\mathsf{ct}, \sigma^*)$, the verifier checks that σ^* is a valid signature on the bit 1 according to the function $\mathsf{CheckWitness}[\mathcal{R}, x, \mathsf{ct}]$. Notice that the description on the function only depends on the relation $\mathcal{R}$, the statement x, and the ciphertext ct, all of which are known to the verifier.

Since the homomorphic signature scheme is context-hiding, the signature σ^* hides the input to $\mathsf{CheckWitness}[\mathcal{R}, x, \mathsf{ct}_w]$, which in this case, is the secret key sk. By CPA-security of the encryption scheme, the ciphertext hides the witness w, so the scheme provides zero-knowledge. Soundness again follows from unforgeability of the signature scheme. Thus, by combining a lattice-based homomorphic signature scheme for general circuits [63] with any lattice-based CPA-secure symmetric encryption scheme, we obtain a (multi-theorem) preprocessing NIZK from lattices. In fact, the verification key in our construction only consists of the public parameters for the homomorphic signature scheme, and thus, can be made public. This means that in our construction, only the proving key needs to be kept secret, so we can equivalently view our construction as a multi-theorem "designated-prover" NIZK. We discuss this in greater detail in Remark 4.6.

An appealing property of our preprocessing NIZKs is that the proofs are short: the length of a NIZK argument for an NP relation $\mathcal{R}$ is $|w| + \mathsf{poly}(\lambda, d)$ bits, where $|w|$ is the length of a witness for $\mathcal{R}$ and d is the depth of the circuit computing $\mathcal{R}$. The proof size in NIZK constructions from trapdoor permutations or pairings [44,40,66,65] typically scale with the *size* of the circuit computing $\mathcal{R}$ and *multiplicatively* with the security parameter. Previously, Gentry et al. [56] gave a generic approach using fully homomorphic encryption (FHE) to reduce the proof size in any NIZK construction. The advantage of our approach is that we naturally satisfy this succinctness property, and the entire construction can be based only on lattice assumptions (without needing to mix assumptions). We discuss this in greater detail in the full version of this paper [74]. We also give the complete description of our preprocessing NIZK and security analysis in Section 4.

Blind homomorphic signatures for efficient preprocessing. A limitation of preprocessing NIZKs is we require a trusted setup to generate the proving and verification keys. One solution is to have the prover and verifier run a (malicious-secure) two-party computation protocol (e.g., [76]) to generate the proving and verification keys. However, generic MPC protocols are often costly and require making *non-black-box* use of the underlying homomorphic signature scheme.

In this work, we describe a conceptually simpler and more efficient way of implementing the preprocessing without relying on general MPC. We do so by introducing a new cryptographic notion called *blind homomorphic signatures*. First, we observe that we can view the two-party computation of the setup phase as essentially implementing a "blind signing" protocol where the verifier holds the signing key for the homomorphic signature scheme and the prover holds the secret key sk. At the end of the blind signing protocol, the prover should learn σ_{sk} while the verifier should not learn anything about sk. This is precisely the properties guaranteed by a blind signature protocol [35,47]. In this work, we introduce the notion of a blind homomorphic signature scheme which combines the blind signing protocol of traditional blind signature schemes while retaining the ability to homomorphically operate on ciphertexts. Since the notion of a blind homomorphic signatures is inherently a two-party functionality, we formalize it in the model of universal composability [24]. We provide the formal definition of the ideal blind homomorphic signature functionality in Section 5.

In Section 5.1, we show how to securely realize our ideal blind homomorphic signature functionality in the presence of *malicious* adversaries by combining homomorphic signatures with any UC-secure oblivious transfer (OT) protocol [27]. Note that security against malicious adversaries is critical for our primary application of leveraging blind homomorphic signatures to implement the setup algorithm of our preprocessing NIZK candidate. At a high-level, we show how to construct a blind homomorphic signature scheme from any "bitwise" homomorphic signature scheme—namely, a homomorphic signature scheme where the signature on an ℓ-bit message consists of ℓ signatures, one for each bit of the message. Moreover, we assume that the signature on each bit position only depends on the value of that particular bit (and not the value of any of the other bits of the message); of course, the ℓ signatures can still be generated using common or correlated randomness. Given a bitwise homomorphic signature scheme, we can implement the blind signing protocol (on ℓ-bit messages) using ℓ independent 1-out-of-2 OTs. Specifically, the signer plays the role of the sender in the OT protocol and for each index $i \in [\ell]$, the signer signs both the bit 0 as well as the bit 1. Then, to obtain a signature on an ℓ-bit message, the receiver requests the signatures corresponding to the bits of its message.

While the high-level schema is simple, there are a few additional details that we have to handle to achieve robustness against a malicious signer. For instance, a malicious signer can craft the parameters of the homomorphic signature scheme so that when an evaluator computes on a signature, the resulting signatures no longer provide context-hiding. Alternatively, a malicious signer might mount a "selective-failure" attack during the blind-signing protocol to learn information about the receiver's message. We discuss how to address these problems by giving strong definitions of malicious context-hiding for homomorphic signatures in Section 3, and give the full construction of blind homomorphic signatures from oblivious transfer in Section 5.1. In particular, we show that the Gorbunov et al. [63] homomorphic signature construction satisfies our stronger security notions, and so coupled with the UC-secure lattice-based OT protocol of Peikert et al. [80], we obtain a UC-secure blind homomorphic signature scheme from standard lattice assumptions. Moreover, the blind signing protocol is a two-round protocol, and only makes black-box use of the underlying homomorphic signature scheme.

UC-secure preprocessing NIZKs. Finally, we show that using our UC-secure blind homomorphic signature candidate, we can in fact realize the stronger notion of UC-secure NIZK arguments in a preprocessing model from standard lattice assumptions. This means that our NIZKs can be arbitrarily composed with other cryptographic protocols. Our new candidates are thus suitable to instantiate many of the classic applications of NIZKs for boosting the security of general MPC protocols. As we show in Section 6, combining our preprocessing UC-NIZKs with existing lattice-based semi-malicious MPC protocols such as [78] yields malicious-secure protocols purely from standard lattice assumptions (in a reusable preprocessing model). We also show that our constructions imply a *succinct* version of the classic GMW [59,60] protocol compiler (where the total communication overhead of the compiled protocol depends only on the *depth*, rather than the *size* of the computation).

Towards NIZKs in the CRS model. In this paper, we construct the first multi-theorem preprocessing NIZK arguments from standard lattice assumptions. However, our techniques do not directly generalize to the CRS setting. While it is possible to obtain a *publicly-verifiable* preprocessing NIZK (i.e., make the verification key k_V public), our construction critically relies on the prover state being hidden. This is because the prover state contains the *secret key* the prover uses to encrypt its witness in the proofs, so publishing this compromises zero-knowledge. Nonetheless, we believe that having a better understanding of NIZKs in the preprocessing model provides a useful stepping stone towards the goal of building NIZKs from lattices in the CRS model, and we leave this as an exciting open problem.

Preprocessing NIZKs from other assumptions? Our work gives the first construction of a multi-theorem preprocessing NIZK from standard lattice assumptions. It is an interesting challenge to obtain multi-theorem preprocessing NIZKs from other assumptions that are currently not known to imply NIZKs in the CRS model. For instance, a natural target would be to construct multi-theorem NIZKs in the preprocessing model from the decisional Diffie-Hellman (DDH) assumption.

1.2 Additional Related Work

In this section, we survey some additional related work on NIZK constructions, blind signatures, and homomorphic signatures.

Other NIZK proof systems. In the CRS model, there are several NIZK constructions based on specific number-theoretic assumptions such as quadratic residuosity [16,41,15]. These candidates are also secure in the *bounded-theorem* setting where the CRS can only be used for an *a priori* bounded number of proofs. Exceeding this bound compromises soundness or zero-knowledge. In the preprocessing model, Kalai and Raz [70] gave a single-theorem *succinct* NIZK proof system for the class LOGSNP from polylogarithmic private information retrieval (PIR) and *exponentially-hard* OT. In this work, we focus on constructing multi-theorem NIZKs, where an *arbitrary* number of proofs can be constructed after an initial setup phase.

NIZKs have also been constructed for specific algebraic languages in both the publicly-verifiable setting [64,67] as well as the designated-verifier setting [33]. In the specific case of lattice-based constructions, there are several works on building hash-proof systems, (also known as smooth projective hash functions [37]) [71,91,14], which are designated-verifier NIZK proofs for a *specific* language (typically, this is the language of ciphertexts associated with a particular message). In the random oracle model, there are also constructions of lattice-based NIZK arguments from Σ-protocols [77,90]. Recently, there has also been work on instantiating the random oracle in Σ-protocols with lattice-based correlation-intractable hash functions [26]. However, realizing the necessary correlation-intractable hash functions from lattices requires making the non-standard assumption that Regev's encryption scheme [83] is *exponentially KDM-secure* against all polynomial-time adversaries. In our work, we focus on NIZK constructions for general NP languages in the plain model (without random oracles) from the *standard* LWE assumption (i.e., polynomial hardness of LWE with a subexponential approximation factor).

Very recently, Rothblum et al. [84] showed that a NIZK proof system for a decisional variant of the bounded distance decoding (BDD) problem suffices for building NIZK proof system for NP.

Blind signatures. The notion of blind signatures was first introduced by Chaum [35]. There are many constructions of blind signatures from a wide range of assumptions in the random oracle model [88,21,82,1,17,13,85,12], the CRS model [23,72,47,4,49,2,3,57], as well as the standard model [52,51,50,68].

Homomorphic signatures. There are many constructions of linearly homomorphic signatures [8,89,43,9,20,53,18,10,19,31,48,5]. Beyond linear homomorphisms, a number of works [19,11,32] have constructed homomorphic signatures for polynomial functions from lattices or multilinear maps. For general circuits, Gorbunov et al. [63] gave the first homomorphic signature scheme from lattices, and Fiore et al. [46] gave the first "multi-key" homomorphic signature scheme from lattices (where homomorphic operations can be performed on signatures signed under *different* keys).

2 Preliminaries

We begin by introducing some basic notation. For an integer $n \geq 1$, we write $[n]$ to denote the set of integers $\{1, \ldots, n\}$. For a positive integer $q > 1$, we write $\mathbb{Z}_q$ to denote the ring of integers modulo q. For a finite set S, we write $x \xleftarrow{\mathrm{R}} S$ to denote that x is sampled uniformly at random from S. For a distribution $\mathcal{D}$, we write $x \leftarrow \mathcal{D}$ to denote that x is sampled from $\mathcal{D}$. Throughout this work, we use λ to denote a security parameter. We typically use bold uppercase letters (e.g., $\mathbf{A}$, $\mathbf{B}$) to denote matrices and bold lowercase letters (e.g., $\mathbf{u}$, $\mathbf{v}$) to denote vectors.

We say that a function f is negligible in λ, denoted $\mathsf{negl}(\lambda)$, if $f(\lambda) = o(1/\lambda^c)$ for all constants $c \in \mathbb{N}$. We say that an event happens with negligible probability if the probability of the event occurring is bounded by a negligible function, and we say that an event happens with overwhelming probability if its complement occurs with negligible probability. We say an algorithm is efficient if it runs in probabilistic polynomial time in the length of its input. We write $\mathsf{poly}(\lambda)$ to denote a quantity whose value is upper-bounded by a fixed polynomial in λ. We say that two families of distributions $\mathcal{D}_1 = \{\mathcal{D}_{1,\lambda}\}_{\lambda \in \mathbb{N}}$ and $\mathcal{D}_2 = \{\mathcal{D}_{2,\lambda}\}_{\lambda \in \mathbb{N}}$ are computationally indistinguishable if no efficient algorithm can distinguish samples from either $\mathcal{D}_1$ or $\mathcal{D}_2$, except with negligible probability. We denote this by writing $\mathcal{D}_1 \overset{c}{\approx} \mathcal{D}_2$. We write $\mathcal{D}_1 \overset{s}{\approx} \mathcal{D}_2$ to denote that $\mathcal{D}_1$ and $\mathcal{D}_2$ are statistically indistinguishable (i.e., the statistical distance between $\mathcal{D}_1$ and $\mathcal{D}_2$ is bounded by a negligible function). In the full version of this paper [74], we provide additional preliminaries in on CPA-secure encryption as well as lattice-based cryptography.

3 Homomorphic Signatures

A homomorphic signature scheme enables computations on signed data. Given a function C (modeled as a Boolean circuit) and a signature σ_x that certifies a message x, one can homomorphic derive a signature $\sigma_{C(x)}$ that certifies the value $C(x)$ with respect to the function C. The two main security notions that we are interested in are unforgeability and context-hiding. We first provide a high-level description of the properties:

- **Unforgeability:** We say a signature scheme is unforgeable if an adversary who has a signature σ_x on a message x cannot produce a valid signature on any message $y \neq C(x)$ that verifies with respect to the function C.
- **Context-hiding:** Context-hiding says that when one evaluates a function C on a message-signature pair (x, σ_x), the resulting signature $\sigma_{C(x)}$ on $C(x)$ should not reveal any information about the original message x other than the circuit C and the value $C(x)$. In our definition, the homomorphic signature scheme contains an explicit "hide" function that implements this transformation.

Syntax and notation. Our construction of blind homomorphic signatures from standard homomorphic signatures (Section 5.1) will impose some additional structural requirements on the underlying scheme. Suppose the message space for the homomorphic signature scheme consists of ℓ-tuples of elements over a set $\mathcal{X}$ (e.g., the case where $\mathcal{X} = \{0,1\}$ corresponds to the setting where the message space consists of ℓ-bit strings). Then, we require that the public parameters $\overrightarrow{\mathsf{pk}}$ of the scheme can be split into a vector of public keys $\overrightarrow{\mathsf{pk}} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell)$. In addition, a (fresh) signature on a vector $\boldsymbol{x} \in \mathcal{X}^\ell$ can also be written as a tuple of ℓ signatures $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_\ell)$ where σ_i can be verified with respect to the verification key vk and the i^{th} public key pk_i for all $i \in [\ell]$. In our description below, we often use vector notation to simplify the presentation.

Definition 3.1 (Homomorphic Signatures [19,63]). *A* homomorphic signature scheme *with message space $\mathcal{X}$, message length $\ell \in \mathbb{N}$, and function class $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$, where each $\mathcal{C}_\lambda$ is a collection of functions from $\mathcal{X}^\ell$ to $\mathcal{X}$, is defined by a tuple of algorithms $\Pi_{\mathsf{HS}} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrmsEval}, \mathsf{SigEval}, \mathsf{Hide}, \mathsf{Verify}, \mathsf{VerifyFresh}, \mathsf{VerifyHide})$ with the following properties:*

- $\mathsf{PrmsGen}(1^\lambda, 1^\ell) \to \overrightarrow{\mathsf{pk}}$*: On input the security parameter λ and message length ℓ, the parameter-generation algorithm returns a set of ℓ public keys $\overrightarrow{\mathsf{pk}} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell)$.*
- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{vk}, \mathsf{sk})$*: On input the security parameter λ, the key-generation algorithm returns a verification key vk, and a signing key sk.*
- $\mathsf{Sign}(\mathsf{pk}_i, \mathsf{sk}, x_i) \to \sigma_i$*: On input a public key pk_i, a signing key sk, and a message $x_i \in \mathcal{X}$, the signing algorithm returns a signature σ_i.*
  Vector variant: For $\overrightarrow{\mathsf{pk}} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell)$, and $\boldsymbol{x} = (x_1, \ldots, x_\ell) \in \mathcal{X}^\ell$, we write $\mathsf{Sign}(\overrightarrow{\mathsf{pk}}, \mathsf{sk}, \boldsymbol{x})$ to denote component-wise signing of each message. Namely, $\mathsf{Sign}(\overrightarrow{\mathsf{pk}}, \mathsf{sk}, \boldsymbol{x})$ outputs signatures $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_\ell)$ where $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{pk}_i, \mathsf{sk}, x_i)$ for all $i \in [\ell]$.
- $\mathsf{PrmsEval}(C, \overrightarrow{\mathsf{pk}}') \to \mathsf{pk}_C$*: On input a function $C \colon \mathcal{X}^\ell \to \mathcal{X}$ and a collection of public keys $\overrightarrow{\mathsf{pk}}' = (\mathsf{pk}'_1, \ldots, \mathsf{pk}'_\ell)$, the parameter-evaluation algorithm returns an evaluated public key pk_C.*
  Vector variant: For a circuit $C \colon \mathcal{X}^\ell \to \mathcal{X}^k$, we write $\mathsf{PrmsEval}(C, \overrightarrow{\mathsf{pk}}')$ to denote component-wise parameter evaluation. Namely, let $C_1, \ldots, C_k$ be functions such that $C(x_1, \ldots, x_\ell) = \big(C_1(x_1, \ldots, x_\ell), \ldots, C_k(x_1, \ldots, x_\ell)\big)$. Then, $\mathsf{PrmsEval}(C, \overrightarrow{\mathsf{pk}}')$ evaluates $\mathsf{pk}_{C_i} \leftarrow \mathsf{PrmsEval}(C_i, \overrightarrow{\mathsf{pk}}')$ for $i \in [k]$, and outputs $\mathsf{pk}_C = (\mathsf{pk}_{C_1}, \ldots, \mathsf{pk}_{C_k})$.
- $\mathsf{SigEval}(C, \overrightarrow{\mathsf{pk}}', \boldsymbol{x}, \boldsymbol{\sigma}) \to \sigma$*: On input a function $C \colon \mathcal{X}^\ell \to \mathcal{X}$, public keys $\overrightarrow{\mathsf{pk}}' = (\mathsf{pk}'_1, \ldots, \mathsf{pk}'_\ell)$, messages $\boldsymbol{x} \in \mathcal{X}^\ell$, and signatures $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_\ell)$, the signature-evaluation algorithm returns an evaluated signature σ.*
  Vector variant: We can define a vector variant of $\mathsf{SigEval}$ analogously to that of $\mathsf{PrmsEval}$.
- $\mathsf{Hide}(\mathsf{vk}, x, \sigma) \to \sigma^*$*: On input a verification key vk, a message $x \in \mathcal{X}$, and a signature σ, the hide algorithm returns a signature σ^*.*
  Vector variant: For $\boldsymbol{x} = (x_1, \ldots, x_k)$ and $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_k)$, we write $\mathsf{Hide}(\mathsf{vk}, \boldsymbol{x}, \boldsymbol{\sigma})$ to denote component-wise evaluation of the hide algorithm. Namely, $\mathsf{Hide}(\mathsf{vk}, \boldsymbol{x}, \boldsymbol{\sigma})$ returns $(\sigma_1^*, \ldots, \sigma_k^*)$ where $\sigma_i^* \leftarrow \mathsf{Hide}(\mathsf{vk}, x_i, \sigma_i)$ for all $i \in [k]$.

- $\mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, x, \sigma) \to \{0, 1\}$*: On input a public key* pk*, a verification key* vk*, a message* $x \in \mathcal{X}$*, and a signature* σ*, the verification algorithm either accepts (returns* 1*) or rejects (returns* 0*).*

  Vector variant: For a collection of public keys $\overrightarrow{\mathsf{pk}'} = (\mathsf{pk}'_1, \ldots, \mathsf{pk}'_k)$*, messages* $\boldsymbol{x} = (x_1, \ldots, x_k)$*, and signatures* $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_k)$*, we write* $\mathsf{Verify}(\overrightarrow{\mathsf{pk}'}, \mathsf{vk}, \boldsymbol{x}, \boldsymbol{\sigma})$ *to denote applying the verification algorithm to each signature component-wise. In other words,* $\mathsf{Verify}(\overrightarrow{\mathsf{pk}'}, \mathsf{vk}, \boldsymbol{x}, \boldsymbol{\sigma})$ *accepts if and only if* $\mathsf{Verify}(\mathsf{pk}'_i, \mathsf{vk}, x_i, \sigma_i)$ *accepts for all* $i \in [k]$*.*
- $\mathsf{VerifyFresh}(\mathsf{pk}, \mathsf{vk}, x, \sigma) \to \{0, 1\}$*: On input a public key* pk*, a verification key* vk*, a message* $x \in \mathcal{X}$*, and a signature* σ*, the fresh verification algorithm either accepts (returns* 1*) or rejects (returns* 0*).*

  Vector variant: We can define a vector variant of VerifyFresh *analogously to that of* Verify*.*
- $\mathsf{VerifyHide}(\mathsf{pk}, \mathsf{vk}, x, \sigma^*) \to \{0, 1\}$*: On input a public key* pk*, a verification key* vk*, a message* $x \in \mathcal{X}$*, and a signature* σ^**, the hide verification algorithm either accepts (returns* 1*) or rejects (returns* 0*).*

  Vector variant: We can define a vector variant of VerifyHide *analogously to that of* Verify*.*

Correctness. We now state the correctness requirements for a homomorphic signature scheme. Our definitions are adapted from the corresponding ones in [63]. Our homomorphic signature syntax has three different verification algorithms. The standard verification algorithm Verify can be used to verify fresh signatures (output by Sign) as well as homomorphically-evaluated signatures (output by SigEval). The hide verification algorithm VerifyHide is used for verifying signatures output by the context-hiding transformation Hide, which may be structurally different from the signatures output by Sign or SigEval. Finally, we have a special verification algorithm VerifyFresh that can be used to verify signatures output by Sign (before any homomorphic evaluation has taken place). While Verify subsumes VerifyFresh, having a separate VerifyFresh algorithm is useful for formulating a strong version of evaluation correctness. Due to space limitations, we defer the formal correctness definitions to the full version of this paper [74].

Unforgeability. Intuitively, a homomorphic signature scheme is unforgeable if no efficient adversary who only possesses signatures $\sigma_1, \ldots, \sigma_\ell$ on messages $x_1, \ldots, x_\ell$ can produce a signature σ_y that is valid with respect to a function C where $y \neq C(x_1, \ldots, x_\ell)$. We give the formal definition in the full version.

Context-hiding. The second security requirement on a homomorphic signature scheme is *context-hiding*, which roughly says that if a user evaluates a function C on a message-signature pair $(\boldsymbol{x}, \boldsymbol{\sigma})$ to obtain a signature $\sigma_{C(\boldsymbol{x})}$, and then runs the hide algorithm on $\sigma_{C(\boldsymbol{x})}$, the resulting signature $\sigma^*_{C(\boldsymbol{x})}$ does not contain any information about $\boldsymbol{x}$ other than what is revealed by C and $C(\boldsymbol{x})$. We define this formally in the full version.

Compactness. The final property that we require from a homomorphic signature scheme is compactness. Roughly speaking, compactness requires that given a message-signature pair $(\boldsymbol{x}, \boldsymbol{\sigma})$, the size of the signature obtained from homomorphically evaluating a function C on $\boldsymbol{\sigma}$ depends only on the size of the output message $|C(\boldsymbol{x})|$ (and the security parameter) and is *independent* of the size of the original message $|\boldsymbol{x}|$.

Structural properties of homomorphic signatures. Definition 3.1 specifies a *bitwise* homomorphic signature scheme where the signature on an ℓ-bit message $x = x_1 \cdots x_\ell$ consists of ℓ separate signatures $\sigma = (\sigma_1, \ldots, \sigma_\ell)$ with respect to ℓ public keys $\overrightarrow{\mathsf{pk}} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell)$, one for each bit of the message. As discussed in Section 1.1, this property is essentially to our construction of blind homomorphic signatures from homomorphic signatures and oblivious transfer. In addition to a bitwise homomorphic signature scheme, we also require a *decomposable homomorphic signature scheme* for our full construction. In a decomposable homomorphic signature scheme, a signature σ of a message x can be decomposed into a message-independent σ^{pk} that contains no information about x, and a message-dependent component σ^{m}. In the full version of this paper [74], we use this decomposability property to show that the homomorphic signature construction of Gorbunov et al. [63] simultaneously satisfies full unforgeability and context-hiding (against malicious signers).

4 Preprocessing NIZKs from Homomorphic Signatures

In this section, we begin by formally defining the notion of a non-interactive zero-knowledge argument in the preprocessing model (i.e., "preprocessing NIZKs"). This notion was first introduced by De Santis et al. [42], who also gave the first candidate construction of a preprocessing NIZK from one-way functions. Multiple works have since proposed additional candidates of preprocessing NIZKs from one-way functions [75,38,69] or oblivious transfer [73]. However, all of these constructions are *single-theorem*: the proving or verification key cannot be reused for multiple theorems without compromising either soundness or zero-knowledge. We provide a more detailed discussion of existing preprocessing NIZK constructions in Remark 4.7.

Definition 4.1 (NIZK Arguments in the Preprocessing Model). *Let $\mathcal{R}$ be an* NP *relation, and let $\mathcal{L}$ be its corresponding language. A non-interactive zero-knowledge (NIZK) argument for $\mathcal{L}$ in the preprocessing model consists of a tuple of three algorithms $\Pi_{\mathsf{PPNIZK}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ with the following properties:*

– $\mathsf{Setup}(1^\lambda) \to (k_P, k_V)$*: On input the security parameter λ, the setup algorithm (implemented in a "preprocessing" step) outputs a proving key k_P and a verification key k_V.*
– $\mathsf{Prove}(k_P, x, w) \to \pi$*: On input the proving key k_P, a statement x, and a witness w, the prover's algorithm outputs a proof π.*
– $\mathsf{Verify}(k_V, x, \pi) \to \{0, 1\}$*: On input the verification key k_V, a statement x, and a proof π, the verifier either accepts (with output 1) or rejects (with output 0).*

Moreover, Π_{PPNIZK} should satisfy the following properties:

– ***Completeness:*** *For all x, w where $\mathcal{R}(x, w) = 1$, if we take $(k_P, k_V) \leftarrow \mathsf{Setup}(1^\lambda)$;*

$$\Pr[\pi \leftarrow \mathsf{Prove}(k_P, x, w) : \mathsf{Verify}(k_V, x, \pi) = 1] = 1.$$

– ***Soundness:*** *For all efficient adversaries $\mathcal{A}$, if we take $(k_P, k_V) \leftarrow \mathsf{Setup}(1^\lambda)$, then*

$$\Pr[(x, \pi) \leftarrow \mathcal{A}^{\mathsf{Verify}(k_V, \cdot, \cdot)}(k_P) : x \notin \mathcal{L} \wedge \mathsf{Verify}(k_V, x, \pi) = 1] = \mathsf{negl}(\lambda).$$

– ***Zero-Knowledge:*** *For all efficient adversaries $\mathcal{A}$, there exists an efficient simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ such that if we take $(k_P, k_V) \leftarrow \mathsf{Setup}(1^\lambda)$ and $\tau_V \leftarrow \mathcal{S}_1(1^\lambda, k_V)$, we have that*

$$\left| \Pr[\mathcal{A}^{\mathcal{O}_0(k_P, \cdot, \cdot)}(k_V) = 1] - \Pr[\mathcal{A}^{\mathcal{O}_1(k_V, \tau_V, \cdot, \cdot)}(k_V) = 1] \right| = \mathsf{negl}(\lambda),$$

where the oracle $\mathcal{O}_0(k_P, x, w)$ outputs $\mathsf{Prove}(k_P, x, w)$ if $\mathcal{R}(x, w) = 1$ and $\perp$ otherwise, and the oracle $\mathcal{O}_1(k_V, \tau_V, x, w)$ outputs $\mathcal{S}_2(k_V, \tau_V, x)$ if $\mathcal{R}(x, w) = 1$ and $\perp$ otherwise.

Remark 4.2 (Comparison to NIZKs in the CRS Model). Our zero-knowledge definition in Definition 4.1 does *not* allow the simulator to choose the verification state k_V. We can also consider a slightly weaker notion of zero-knowledge where the simulator also chooses the verification state:

– **Zero-Knowledge:** For all efficient adversaries $\mathcal{A}$, there exists an efficient simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ such that if we take $(k_P, k_V) \leftarrow \mathsf{Setup}(1^\lambda)$ and $(\tilde{k}_V, \tilde{\tau}_V) \leftarrow \mathcal{S}_1(1^\lambda)$, we have that

$$\left| \Pr[\mathcal{A}^{\mathsf{Prove}(k_P, \cdot, \cdot)}(k_V) = 1] - \Pr[\mathcal{A}^{\mathcal{O}(\tilde{k}_V, \tilde{\tau}_V, \cdot, \cdot)}(\tilde{k}_V) = 1] \right| = \mathsf{negl}(\lambda),$$

where the oracle $\mathcal{O}(\tilde{k}_V, \tilde{\tau}_V, x, w)$ outputs $\mathcal{S}_2(\tilde{k}_V, \tilde{\tau}_V, x)$ if $\mathcal{R}(x, w) = 1$ and $\perp$ otherwise.

We note that this definition of zero-knowledge captures the standard notion of NIZK arguments in the common reference string (CRS) model. Specifically, in the CRS model, the Setup algorithm outputs a single CRS σ. The proving and verification keys are both defined to be σ.

Preprocessing NIZKs from homomorphic signatures. As described in Section 1.1, we can combine a homomorphic signature scheme (for general circuits) with any CPA-secure symmetric encryption scheme to obtain a preprocessing NIZK for general NP languages. We give our construction and security analysis below. Combining the lattice-based construction of homomorphic signatures of [63] with any lattice-based CPA-secure encryption [58,6], we obtain the first multi-theorem preprocessing NIZK from standard lattice assumptions (Corollary 4.5). In Remark 4.6, we note that a variant of Construction 4.3 also gives a *publicly-verifiable* preprocessing NIZK.

Construction 4.3 (Preprocessing NIZKs from Homomorphic Signatures). Fix a security parameter λ, and define the following quantities:

- Let $\mathcal{R}\colon \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ be an NP relation and $\mathcal{L}$ be its corresponding language.
- Let $\Pi_{\mathsf{SE}} = (\mathsf{SE.KeyGen}, \mathsf{SE.Encrypt}, \mathsf{SE.Decrypt})$ be a symmetric encryption scheme with message space $\{0,1\}^m$ and secret-key space $\{0,1\}^\rho$.
- For a message $x \in \{0,1\}^n$ and ciphertext ct from the ciphertext space of Π_{SE}, define the function $f_{x,\mathsf{ct}}(k_{\mathsf{SE}}) := \mathcal{R}(x, \mathsf{SE.Decrypt}(k_{\mathsf{SE}}, \mathsf{ct}))$.
- Let $\Pi_{\mathsf{HS}} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrmsEval}, \mathsf{SigEval}, \mathsf{Hide}, \mathsf{Verify}, \mathsf{VerifyFresh}, \mathsf{VerifyHide})$ be a homomorphic signature scheme with message space $\{0,1\}$, message length ρ, and function class $\mathcal{C}$ that includes all functions of the form $f_{x,\mathsf{ct}}$.[3]

We construct a preprocessing NIZK argument $\Pi_{\mathsf{NIZK}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ as follows:

- $\mathsf{Setup}(1^\lambda) \to (k_P, k_V)$: First, generate a secret key $k_{\mathsf{SE}} \leftarrow \mathsf{SE.KeyGen}(1^\lambda)$. Next, generate $\overrightarrow{\mathsf{pk}}_{\mathsf{HS}} \leftarrow \mathsf{PrmsGen}(1^\lambda, 1^\rho)$ and a signing-verification key-pair $(\mathsf{vk}_{\mathsf{HS}}, \mathsf{sk}_{\mathsf{HS}}) \leftarrow \mathsf{KeyGen}(1^\lambda)$. Next, sign the symmetric key $\boldsymbol{\sigma}_k \leftarrow \mathsf{Sign}(\overrightarrow{\mathsf{pk}}_{\mathsf{HS}}, \mathsf{sk}_{\mathsf{HS}}, k_{\mathsf{SE}})$ and output
$$k_P = (k_{\mathsf{SE}}, \overrightarrow{\mathsf{pk}}_{\mathsf{HS}}, \mathsf{vk}_{\mathsf{HS}}, \boldsymbol{\sigma}_k) \quad \text{and} \quad k_V = (\overrightarrow{\mathsf{pk}}_{\mathsf{HS}}, \mathsf{vk}_{\mathsf{HS}}, \mathsf{sk}_{\mathsf{HS}}).$$
- $\mathsf{Prove}(k_P, x, w) \to \pi$: If $\mathcal{R}(x,w) = 0$, output $\perp$. Otherwise, parse $k_P = (k_{\mathsf{SE}}, \overrightarrow{\mathsf{pk}}_{\mathsf{HS}}, \mathsf{vk}_{\mathsf{HS}}, \boldsymbol{\sigma}_k)$. Let $\mathsf{ct} \leftarrow \mathsf{SE.Encrypt}(k_{\mathsf{SE}}, w)$, and $C_{x,\mathsf{ct}}$ be the circuit that computes the function $f_{x,\mathsf{ct}}$ defined above. Compute the signature $\sigma'_{x,\mathsf{ct}} \leftarrow \mathsf{SigEval}(C_{x,\mathsf{ct}}, \overrightarrow{\mathsf{pk}}_{\mathsf{HS}}, k_{\mathsf{SE}}, \boldsymbol{\sigma}_k)$ and then $\sigma^*_{x,\mathsf{ct}} \leftarrow \mathsf{Hide}(\mathsf{vk}_{\mathsf{HS}}, 1, \sigma'_{x,\mathsf{ct}})$. It outputs the proof $\pi = (\mathsf{ct}, \sigma^*_{x,\mathsf{ct}})$.
- $\mathsf{Verify}(k_V, x, \pi) \to \{0,1\}$: Parse $k_V = (\overrightarrow{\mathsf{pk}}_{\mathsf{HS}}, \mathsf{vk}_{\mathsf{HS}}, \mathsf{sk}_{\mathsf{HS}})$ and $\pi = (\mathsf{ct}, \sigma^*_{x,\mathsf{ct}})$. Let $C_{x,\mathsf{ct}}$ be the circuit that computes $f_{x,\mathsf{ct}}$ defined above. Then, compute $\mathsf{pk}_{x,\mathsf{ct}} \leftarrow \mathsf{PrmsEval}(C_{x,\mathsf{ct}}, \overrightarrow{\mathsf{pk}}_{\mathsf{HS}})$, and output $\mathsf{VerifyHide}(\mathsf{pk}_{x,\mathsf{ct}}, \mathsf{vk}_{\mathsf{HS}}, 1, \sigma^*_{x,\mathsf{ct}})$.

Theorem 4.4 (Preprocessing NIZKs from Homomorphic Signatures). *Let λ be a security parameter and $\mathcal{R}$ be an NP relation (and let $\mathcal{L}$ be its corresponding language). Let Π_{NIZK} be the NIZK argument in the preprocessing model from Construction 4.3 (instantiated with a symmetric encryption scheme Π_{SE} and a homomorphic signature scheme Π_{HS}). If Π_{SE} is CPA-secure and Π_{HS} satisfies evaluation correctness, hiding correctness, selective unforgeability, and context-hiding, then Π_{NIZK} is a NIZK argument for $\mathcal{R}$ in the preprocessing model.*

We give the proof of Theorem 4.4 in the full version [74]. Combining Construction 4.3 with a lattice-based homomorphic signature scheme [63] and any LWE-based CPA-secure encryption scheme [58,6], we have the following corollary.

[3]Since it is more natural to view $x \in \{0,1\}^n$ as a string rather than a vector, we drop the vector notation $\boldsymbol{x}$ and simply write x in this section.

Corollary 4.5 (Preprocessing NIZKs from Lattices). *Under the LWE assumption, there exists a multi-theorem preprocessing NIZK for* NP*.*

Remark 4.6 (Publicly-Verifiable Preprocessing NIZK). Observe that the verification algorithm in Construction 4.3 does not depend on the signing key $\mathsf{sk}_{\mathsf{HS}}$ of the signature scheme. Thus, we can consider a variant of Construction 4.3 where the verification key does *not* contain $\mathsf{sk}_{\mathsf{HS}}$, and thus, the verification state can be made *public*. This does not compromise soundness because the prover's state already includes the other components of the verification key. However, this publicly-verifiable version of the scheme does not satisfy zero-knowledge according to the strong notion of zero-knowledge in Definition 4.1. This is because without the signing key, the simulator is no longer able to simulate the signatures in the simulated proofs. However, if we consider the weaker notion of zero-knowledge from Remark 4.2 where the simulator chooses the verification key for the preprocessing NIZK, then the publicly-verifiable version of the scheme is provably secure. Notably, when the simulator constructs the verification key, it also chooses (and stores) the signing key for the homomorphic signature scheme. This enables the simulator to simulate signatures when generating the proofs. The resulting construction is a publicly-verifiable preprocessing NIZK (i.e., a "designated-prover" NIZK).

Remark 4.7 (Preprocessing NIZKs from Weaker Assumptions). By definition, any NIZK argument (or proof) system in the CRS model is also a preprocessing NIZK (according to the notion of zero-knowledge from Remark 4.2). In the CRS model (and without random oracles), there are several main families of assumptions known to imply NIZKs: number-theoretic conjectures such as quadratic residuosity [16,41,15],[4] trapdoor permutations [44,40,65], pairings [66], or indistinguishability obfuscation [86]. In the designated-verifier setting, constructions are also known from additively homomorphic encryption [36,39,34]. A number of works have also studied NIZKs in the preprocessing model, and several constructions have been proposed from one-way functions [42,75,38,69] and oblivious transfer [73]. Since lattice-based assumptions imply one-way functions [6,83], oblivious transfer [80], and homomorphic encryption [83,55], one might think that we can already construct NIZKs in the preprocessing model from standard lattice assumptions. To our knowledge, this is not the case:

– The NIZK constructions of [42,75,38] are *single-theorem* NIZKs, and in particular, zero-knowledge does not hold if the prover uses the same proving key to prove multiple statements. In these constructions, the proving key contains secret values, and each proof reveals a subset of the prover's secret values. As a result, the verifier can combine multiple proofs together to learn additional information about each statement than it could have learned had it only seen a single proof. Thus, the constructions in [42,75,38] do not directly give a multi-theorem NIZK.

[4]Some of these schemes [16,41] are "bounded" in the sense that the prover can only prove a small number of theorems whose total size is bounded by the length of the CRS.

A natural question to ask is whether we can use the transformation by Feige et al. [44] who showed how to generically boost a NIZK (in the CRS model) with single-theorem zero-knowledge to obtain a NIZK with multi-theorem zero-knowledge. The answer turns out to be negative: the [44] transformation critically relies on the fact that the prover algorithm is publicly computable, or equivalently, that the prover algorithm does not depend on any secrets.[5] This is the case in the CRS model, since the prover algorithm depends only on the CRS, but in the preprocessing model, the prover's algorithm can depend on a (secret) proving key k_P. In the case of [42,75,38], the proving key must be kept private for zero-knowledge. Consequently, the preprocessing NIZKs of [42,75,38] do not give a general multi-theorem NIZK in the preprocessing model.

– The (preprocessing) NIZK constructions based on oblivious transfer [73], the "MPC-in-the-head" paradigm [69], and the ones based on homomorphic encryption [36,39,34] are designated-verifier, and in particular, are vulnerable to the "verifier rejection" problem. Specifically, soundness is compromised if the prover can learn the verifier's response to multiple adaptively-chosen statements and proofs. For instance, in the case of [73], an oblivious transfer protocol is used to hide the verifier's challenge bits; namely, the verifier's challenge message is fixed during the preprocessing, which means the verifier uses the *same* challenge to verify every proof. A prover that has access to a proof-verification oracle is able to reconstruct the verifier's challenge bit-by-bit and compromise soundness of the resulting NIZK construction. A similar approach is taken in the preprocessing NIZK construction of [69].

From the above discussion, the only candidates of general multi-theorem NIZKs in the preprocessing model are the same as those in the CRS model. Thus, this work provides the first candidate construction of a multi-theorem NIZK in the preprocessing model from standard lattice assumptions. It remains an open problem to construct multi-theorem NIZKs from standard lattice assumptions in the standard CRS model.

In the full version of this paper [74], we highlight several additional properties of our multi-theorem preprocessing NIZK. We also describe another approach for instantiating our construction using context-hiding homomorphic MACs [54,29,30,28]. While existing homomorphic MAC constructions from one-way functions do not suffice for our constructions (they are not context-hiding), they do provide another potential avenue towards realizing multi-theorem preprocessing NIZKs from weaker assumptions.

[5] At a high-level, the proof in [44] proceeds in two steps: first show that single-theorem zero knowledge implies single-theorem witness indistinguishability, and then that single-theorem witness indistinguishability implies multi-theorem witness indistinguishability. The second step relies on a hybrid argument, which requires that it be possible to *publicly* run the prover algorithm. This step does not go through if the prover algorithm takes in a secret state unknown to the verifier.

5 Blind Homomorphic Signatures

One limitation of preprocessing NIZKs is that we require a trusted setup to generate the proving and verification keys. One solution is to have the prover and the verifier run a (malicious-secure) two-party computation protocol (e.g., [76]) to generate the proving and verification keys. However, generic MPC protocols are often costly and require making *non-black-box* use of the underlying homomorphic signature scheme. In this section, we describe how this step can be efficiently implemented using a new primitive called *blind homomorphic signatures*. We formalize our notion in the model of universal composability [24]. This has the additional advantage of allowing us to realize the stronger notion of a preprocessing universally-composable NIZK (UC-NIZK) from standard lattice assumptions. We give our UC-NIZK construction and then describe several applications to boosting the security of MPC in Section 6. We refer to the full version for a review of the UC model.

We now define the ideal blind homomorphic signature functionality $\mathcal{F}_{\text{BHS}}$. Our definition builds upon existing definitions of the ideal signature functionality $\mathcal{F}_{\text{SIG}}$ by Canetti [25] and the ideal blind signature functionality $\mathcal{F}_{\text{BLSIG}}$ by Fischlin [47]. To simplify the presentation, we define the functionality in the two-party setting, where there is a special signing party (denoted $\mathbf{S}$) and a single receiver who obtains the signature (denoted $\mathbf{R}$). While this is a simpler model than the multi-party setting considered in [25,47], it suffices for the applications we describe in this work.

Ideal signature functionalities. The $\mathcal{F}_{\text{SIG}}$ functionality from [25] essentially provides a "registry service" where a distinguished party (the signer) is able to register message-signature pairs. Moreover, any party that possesses the verification key can check whether a particular message-signature pair is registered (and thus, constitutes a valid signature). The ideal functionality does not impose any restriction on the structure of the verification key or the legitimate signatures, and allows the adversary to choose those values. In a blind signature scheme, the signing process is replaced by an interactive protocol between the signer and the receiver, and the security requirement is that the signer does not learn the message being signed. To model this, the $\mathcal{F}_{\text{BLSIG}}$ functionality from [47] asks the adversary to provide the description of a *stateless* algorithm $\mathsf{IdealSign}$ in addition to the verification key to the ideal functionality $\mathcal{F}_{\text{BLSIG}}$. For blind signing requests involving an *honest* receiver, the ideal functionality uses $\mathsf{IdealSign}$ to generate the signatures. The message that is signed (i.e., the input to $\mathsf{IdealSign}$) is not disclosed to either the signer or the adversary. This captures the intuitive requirement that the signer does not learn the message that is signed in a blind signature scheme. Conversely, if a corrupt user makes a blind signing request, then the ideal functionality asks the adversary to supply the signature that could result from such a request.

Capturing homomorphic operations. In a homomorphic signature scheme, a user possessing a signature σ on a message x should be able to compute a function g on σ to obtain a new signature σ^* on the message $g(x)$. In turn, the verification algorithm checks that σ^* is a valid signature on the value $g(x)$ *and* importantly, that it is a valid signature with respect to the function g. Namely, the signature is bound not only to the computed value $g(x)$ but also to the function g.[6] To extend the ideal signature functionality to support homomorphic operations on signatures, we begin by modifying the ideal functionality to maintain a mapping between *function-message pairs* and signatures (rather than a mapping between messages and signatures). In this case, a fresh signature σ (say, output by the blind signing protocol) on a message x would be viewed as a signature on the function-message pair (f_{id}, x), where f_{id} here denotes the identity function. Then, if a user subsequently computes a function g on σ, the resulting signature σ^* should be viewed as a signature on the new pair $(g \circ f_{\mathsf{id}}, g(x)) = (g, g(x))$. In other words, in a homomorphic signature scheme, signatures are bound to a function-message pair, rather than a single message.

Next, we introduce an additional *signature-evaluation* operation to the ideal functionality. There are several properties we desire from our ideal functionality:

- The ideal signature functionality allows the adversary to decide the structure of the signatures, so it is only natural that the adversary also decides the structure of the signatures output by the signature evaluation procedure.
- Signature evaluation should be compatible with the blind signing process. Specifically, the receiver should be able to compute on a signature it obtained from the blind signing functionality, and moreover, the computation (if requested by an honest receiver) should not reveal to the adversary on which signature or message the computation was performed.
- The computed signature should also hide the input message. In particular, if the receiver obtains a blind signature on a message x and later computes a signature σ^* on $g(x)$, the signature σ^* should not reveal the original (blind) message x.

To satisfy these properties, the ideal functionality asks the adversary to additionally provide the description of a *stateless* signature evaluation algorithm $\mathsf{IdealEval}$ (in addition to $\mathsf{IdealSign}$). The ideal functionality uses $\mathsf{IdealEval}$ to generate the signatures when responding to evaluation queries. We capture the third property (that the computed signatures hide the input message to the computation) by setting the inputs to $\mathsf{IdealEval}$ to only include the function g that is computed and the output value of the computation $g(x)$. The input message x is not provided to $\mathsf{IdealEval}$.

Under our definition, the signature evaluation functionality takes as input a function-message pair (f_{id}, x), a signature σ on (f_{id}, x) (under the verification key vk of the signature scheme), and a description of a function g (to compute on x). The output is a new signature σ^* on the pair $(g, g(x))$. That is, σ^* is a signature on the value $g(x)$ with respect to the function g. When the evaluator is honest, the signature on $(g, g(x))$ is determined by $\mathsf{IdealEval}(g, g(x))$ (without going through the adversary). As discussed above, $\mathsf{IdealEval}$ only takes as input the function g

[6] If there is no binding between σ^* and the function g, then we cannot define a meaningful notion of unforgeability.

and the value $g(x)$, and not the input; this means that the computed signature σ^* hides all information about x other than what is revealed by $g(x)$. When the evaluator is corrupt, the adversary chooses the signature on $(g, g(x))$, subject to basic consistency requirements.[7] Once an evaluated signature is generated, the functionality registers the new signature σ^* on the pair $(g, g(x))$. Our definition implicitly requires that homomorphic evaluation be non-interactive. Neither the adversary nor the signer is notified or participates in the protocol.

Preventing selective failures. In our definition, the functionalities IdealSign and IdealEval must either output $\perp$ on *all* inputs, or output $\perp$ on *none* of the inputs. This captures the property that a malicious signer cannot mount a *selective failure* attack against an honest receiver, where the function of whether the receiver obtains a signature or not in the blind signing protocol varies depending on its input message. In the case of the blind signing protocol, we do allow a malicious signer to cause the protocol to fail, but this failure event must be *independent* of the receiver's message. We capture this in the ideal functionality by allowing a corrupt signer to dictate whether a blind signing execution completes successfully or not. However, the corrupt signer must decide whether a given protocol invocation succeeds or fails *independently* of the receiver's message.

Simplifications and generalizations. In defining our ideal blind homomorphic signature functionality, we impose several restrictions to simplify the description and analysis. We describe these briefly here, and note how we could extend the functionality to provide additional generality. Note that all of the applications we consider (Section 6) only require the basic version of the functionality (Figure 1), and not its generalized variants.

- **One-time signatures.** The ideal blind homomorphic signature functionality supports blind signing of a *single* message. Namely, the ideal blind signing functionality only responds to the first signing request from the receiver and ignores all subsequent requests. Moreover, the ideal functionality only supports signature evaluation requests after a signature has been successfully issued by the ideal signing functionality. We capture this via a ready flag that is only set at the conclusion of a successful signing operation. We can relax this single-signature restriction, but at the cost of complicating the analysis.
- **Single-hop evaluation.** Our second restriction on the ideal blind homomorphic signature functionality is we only consider "single-hop" homomorphic operations: that is, we only allow homomorphic operations on fresh signatures. In the ideal functionality, we capture this by having the signature evaluation functionality ignore all requests to compute on function-message pairs (f, x) where $f \neq f_{\mathsf{id}}$ is not the identity function. A more general definition would also consider "multi-hop" evaluation where a party can perform arbitrarily many sequential operations on a signature. The reason we present our definition in the simpler single-hop setting is because existing constructions of homomorphic signatures [63] (which we leverage in our construction) do

[7]The adversary is not allowed to re-register a signature that was previously declared invalid (according to the verification functionality) as a valid signature.

not support the multi-hop analog of our definition. This is because under our definition, the ideal evaluation functionality essentially combines the homomorphic evaluation with the context-hiding transformation in standard homomorphic signature schemes. The current homomorphic signature candidate [63] does not support homomorphic computation after performing context-hiding, and so, cannot be used to realize the more general "multi-hop" version of our functionality. For this reason, we give our definition in the single-hop setting.

We give the formal specification of the ideal blind homomorphic signature functionality $\mathcal{F}_{\text{BHS}}$ in Figure 1.

5.1 Constructing Blind Homomorphic Signatures

In Figure 2, we give the formal description of our blind homomorphic signature protocol Π_{BHS} in the $\mathcal{F}_{\text{OT}}^{\ell,s}$-hybrid model.[8] Here, we provide a brief overview of the construction. As discussed in Section 1.1, our construction combines homomorphic signatures with any UC-secure oblivious transfer protocol [27]. The key-generation, signature-verification, and signature-evaluation operations in Π_{BHS} just correspond to running the underlying Π_{HS} algorithms.

The blind signing protocol is interactive and relies on OT. Since we use a bitwise homomorphic signature scheme, a signature on an ℓ-bit message consists of ℓ signatures, one for each bit of the message. In the first step of the blind signing protocol, the signer constructs two signatures (one for the bit 0 and one for the bit 1) for each bit position of the message. The receiver then requests the signatures corresponding to the bits of its message using the OT protocol. Intuitively, the OT protocol ensures that the signer does not learn which set of signatures the receiver requested and the receiver only learns a single signature for each bit position. However, this basic scheme is vulnerable to a "selective-failure" attack where the signer strategically generates *invalid* signatures for certain bit positions of the message $\boldsymbol{x}$. As a result, whether the receiver obtains a valid signature on its entire message becomes *correlated* with its message itself. To prevent this selective-failure attack, we use the standard technique of having the receiver first split its message $\boldsymbol{x}$ into a number of random shares $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_t$ where $\boldsymbol{x} = \bigoplus_{i \in [t]} \boldsymbol{w}_i$. Instead of asking for a signature on $\boldsymbol{x}$ directly, it instead asks for a signature on the shares $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_t$. Since the signatures on the shares $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_t$ are homomorphic, the receiver can still compute a signature on the original message $\boldsymbol{x}$ and hence, correctness of signing is preserved. Moreover, as we show in the proof of Theorem 5.1, unless the malicious signer correctly guesses

[8] For the protocol description and its security proof, we use the vector notation $\boldsymbol{x}$ to represent the messages (in order to be consistent with the homomorphic signature notation).

Functionality $\mathcal{F}_{\text{BHS}}$

The ideal blind homomorphic signature functionality $\mathcal{F}_{\text{BHS}}$ runs with a signer $\mathbf{S}$, a receiver $\mathbf{R}$, and an ideal adversary $\mathcal{S}$. The functionality is parameterized by a message length ℓ and a function class $\mathcal{H}$. We write f_{id} to denote the identity function.

Key Generation: Upon receiving a value $(\mathsf{sid}, \mathsf{keygen})$ from the signer $\mathbf{S}$, send $(\mathsf{sid}, \mathsf{keygen})$ to the adversary $\mathcal{S}$. After receiving $(\mathsf{sid}, \mathsf{vkey}, \mathsf{vk})$ from $\mathcal{S}$, give $(\mathsf{sid}, \mathsf{vkey}, \mathsf{vk})$ to $\mathbf{S}$ and record vk. Then, initialize an empty list $\mathcal{L}$, and a ready flag (initially unset).

Signature Generation: If a signature-generation request has already been processed, ignore the request. Otherwise, upon receiving a value $(\mathsf{sid}, \mathsf{sign}, \mathsf{vk}, x)$ from the receiver $\mathbf{R}$ (for some message $x \in \{0,1\}^{\ell}$), send $(\mathsf{sid}, \mathsf{signature})$ to $\mathcal{S}$, and let $(\mathsf{sid}, \mathsf{IdealSign}, \mathsf{IdealEval})$ be the response from $\mathcal{S}$, where $\mathsf{IdealSign}$ and $\mathsf{IdealEval}$ are functions that either output $\perp$ on *all* inputs or on *no* inputs. Record the tuple $(\mathsf{IdealSign}, \mathsf{IdealEval})$. If $\mathbf{S}$ is honest, send $(\mathsf{sid}, \mathsf{signature})$ to $\mathbf{S}$ to notify it that a signature request has taken place. If $\mathbf{S}$ is corrupt, then send $(\mathsf{sid}, \mathsf{sig\text{-}success})$ to $\mathcal{S}$ and let (sid, b) be the response from $\mathcal{S}$. If $b \neq 1$, send $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, x), \perp)$ to $\mathbf{R}$. Otherwise, proceed as follows:

– If $\mathbf{R}$ is honest, generate $\sigma \leftarrow \mathsf{IdealSign}(x)$, and send $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, x), \sigma)$ to $\mathbf{R}$.
– If $\mathbf{R}$ is corrupt, send $(\mathsf{sid}, \mathsf{sign}, x)$ to $\mathcal{S}$ to obtain $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, x), \sigma)$.

If $(\mathsf{vk}, (f_{\mathsf{id}}, x), \sigma, 0) \in \mathcal{L}$, abort. Otherwise, add $(\mathsf{vk}, (f_{\mathsf{id}}, x), \sigma, 1)$ to $\mathcal{L}$, and if $\sigma \neq \perp$, set the flag ready.

Signature Verification: Upon receiving an input $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}', (f, x), \sigma)$ from a party $\mathbf{P} \in \{\mathbf{S}, \mathbf{R}\}$, proceed as follows:

– *Correctness:* If $f \notin \mathcal{H}$, then set $t = 0$. If $\mathsf{vk} = \mathsf{vk}'$ and $(\mathsf{vk}, (f, x), \sigma, 1) \in \mathcal{L}$, then set $t = 1$.
– *Unforgeability:* Otherwise, if $\mathsf{vk} = \mathsf{vk}'$, the signer $\mathbf{S}$ has not been corrupted, and there does not exist $(\mathsf{vk}, (f_{\mathsf{id}}, x'), \sigma', 1) \in \mathcal{L}$ for some x', σ' where $x = f(x')$, then set $t = 0$, and add $(\mathsf{vk}, (f, x), \sigma, 0)$ to $\mathcal{L}$.
– *Consistency:* Otherwise, if there is already an entry $(\mathsf{vk}', (f, x), \sigma, t') \in \mathcal{L}$ for some t', set $t = t'$.
– Otherwise, send $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}', (f, x), \sigma)$ to the adversary $\mathcal{S}$. After receiving $(\mathsf{sid}, \mathsf{verified}, (f, x), \sigma, \tau)$ from $\mathcal{S}$, set $t = \tau$ and add $(\mathsf{vk}', (f, x), \sigma, \tau)$ to $\mathcal{L}$.

Send $(\mathsf{sid}, \mathsf{verified}, (f, x), \sigma, t)$ to $\mathbf{P}$. If $t = 1$, we say the signature successfully verified.

Fig. 1: The $\mathcal{F}_{\text{BHS}}$ functionality. The description continues on the next page.

Functionality $\mathcal{F}_{\text{BHS}}$ (Continued)

Signature Evaluation: If the ready flag has not been set, then ignore the request. Otherwise, upon receiving an input $(\mathsf{sid}, \mathsf{eval}, \mathsf{vk}, g, (f, x), \sigma)$ from a party $\mathbf{P} \in \{\mathbf{S}, \mathbf{R}\}$, ignore the request if $f \neq f_{\mathsf{id}}$. If $f = f_{\mathsf{id}}$, then apply the signature verification procedure to $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}, (f, x), \sigma)$, but do *not* forward the output to $\mathbf{P}$. If the signature does not verify, then ignore the request. Otherwise, proceed as follows:

- If $g \notin \mathcal{H}$, then set $\sigma^* = \perp$.
- Otherwise, if $\mathbf{P}$ is honest, compute $\sigma^* \leftarrow \mathsf{IdealEval}(g, g(x))$.
- Otherwise, if $\mathbf{P}$ is corrupt, send $(\mathsf{sid}, \mathsf{eval}, g, (f, x), \sigma)$ to $\mathcal{S}$ to obtain $(\mathsf{sid}, \mathsf{signature}, (g, g(x)), \sigma^*)$.

Finally, send $(\mathsf{sid}, \mathsf{signature}, (g, g(x)), \sigma^*)$ to $\mathbf{P}$. If $\sigma^* \neq \perp$ and $(\mathsf{vk}, (g, g(x)), \sigma^*, 0) \in \mathcal{L}$, abort. If $\sigma^* \neq \perp$ and $(\mathsf{vk}, (g, g(x)), \sigma^*, 0) \notin \mathcal{L}$, add $(\mathsf{vk}, (g, g(x)), \sigma^*, 1)$ to $\mathcal{L}$.

Fig. 1 (Continued): The $\mathcal{F}_{\text{BHS}}$ functionality.

all of the shares of $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_t$ the receiver chose, the probability that the receiver aborts (due to receiving an invalid signature) is *independent* of $\boldsymbol{x}$ no matter how the malicious signer generates the signatures. We formally summarize the security properties of Π_{BHS} in the following theorem, but defer its proof to the full version [74].

Theorem 5.1 (Blind Homomorphic Signatures). *Fix a security parameter λ. Define parameters ℓ, t, and s as in Π_{BHS} (Figure 2) where $t = \omega(\log \lambda)$. Let $\mathcal{H}$ be a function class over $\{0,1\}^\ell$ and let Π_{HS} be a homomorphic signature scheme for the message space $\{0,1\}$ and function class $\mathcal{H}'$ such that for any function $f \in \mathcal{H}$, we have $f \circ f_{\mathsf{recon}} \in \mathcal{H}'$, where f_{recon} is the share-reconstruction function from Figure 2. Suppose that Π_{HS} satisfies correctness, unforgeability, and context-hiding. Then, the protocol Π_{BHS} (when instantiated with Π_{HS}) securely realizes the ideal functionality $\mathcal{F}_{\text{BHS}}$ (Figure 1) with respect to function class $\mathcal{H}$ in the presence of (static) malicious adversaries in the $\mathcal{F}_{\text{OT}}^{\ell,s}$-hybrid model.*

Blind homomorphic signatures from LWE. Combining the fully-secure homomorphic signature scheme described in the full version [74] (based on [63]) with the lattice-based UC-secure oblivious transfer protocol from [80], we obtain a blind homomorphic signature scheme from standard lattice assumptions. We describe our instantiation below.

Fact 5.2 (Oblivious Transfer from LWE [80]). Let λ be a security parameter and define parameters $\ell, s = \mathsf{poly}(\lambda)$. Then, under the LWE assumption, there exists a protocol Π_{OT} that security realizes the ideal OT functionality $\mathcal{F}_{\text{OT}}^{\ell,s}$ in the presence of malicious adversaries in the CRS model (and assuming static corruptions). Moreover, the protocol Π_{OT} is *round-optimal*: it consists of one message from the receiver to the signer and one from the receiver to the signer.

Protocol $\mathit{\Pi}_{\text{BHS}}$ in the $\mathcal{F}_{\text{OT}}^{\ell,s}$-Hybrid Model

Let λ be a security parameter and $\mathcal{H}$ be a class of functions from $\{0,1\}^{\ell}$ to $\{0,1\}$. For a parameter $t \in \mathbb{N}$, we define $f_{\mathsf{recon}} \colon \{0,1\}^{t\ell} \to \{0,1\}^{\ell}$ to be a share-reconstruction function $(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_t) \mapsto \bigoplus_{i \in [t]} \boldsymbol{w}_i$. Let $\mathit{\Pi}_{\mathsf{HS}} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrmsEval}, \mathsf{SigEval}, \mathsf{Hide}, \mathsf{Verify}, \mathsf{VerifyFresh}, \mathsf{VerifyHide})$ be a decomposable homomorphic signature scheme with message space $\{0,1\}$, message length ℓ, and function class $\mathcal{H}'$ where $\mathcal{H}'$ contains all functions of the form $f \circ f_{\mathsf{recon}}$ where $f \in \mathcal{H}$. We assume that the signer $\mathbf{S}$ and receiver $\mathbf{R}$ has access to the ideal functionality $\mathcal{F}_{\text{OT}}^{\ell,s}$ where s is the length of the signatures in $\mathit{\Pi}_{\mathsf{HS}}$.

Key Generation: Upon receiving an input $(\mathsf{sid}, \mathsf{keygen})$, the signer $\mathbf{S}$ computes a set of public parameters $\overrightarrow{\mathsf{pk}} = \{\mathsf{pk}_{i,j}\}_{i \in [t], j \in [\ell]} \leftarrow \mathsf{PrmsGen}(1^{\lambda}, 1^{t\ell})$, and a pair of keys $(\mathsf{vk}', \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^{\lambda})$. It stores $(\mathsf{sid}, \mathsf{sk})$, sets $\mathsf{vk} = (\overrightarrow{\mathsf{pk}}, \mathsf{vk}')$, and outputs $(\mathsf{sid}, \mathsf{vkey}, \mathsf{vk})$. Finally, the signer initializes the ready flag (initially unset).

Signature Generation: If the signer or receiver has already processed a signature-generation request, then they ignore the request. Otherwise, they proceed as follows:

– **Receiver:** On input $(\mathsf{sid}, \mathsf{sign}, \mathsf{vk}, \boldsymbol{x})$, where $\mathsf{vk} = (\overrightarrow{\mathsf{pk}}, \mathsf{vk}')$ and $\boldsymbol{x} \in \{0,1\}^{\ell}$, the receiver chooses t shares $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_t \xleftarrow{\text{R}} \{0,1\}^{\ell}$ where $\bigoplus_{i \in [t]} \boldsymbol{w}_i = \boldsymbol{x}$. Then, for each $i \in [t]$, it sends $\big((\mathsf{sid}, i), \mathsf{receiver}, \boldsymbol{w}_i\big)$ to $\mathcal{F}_{\text{OT}}^{\ell,s}$. It also initializes the ready flag (initially unset). Note that if vk is not of the form $(\overrightarrow{\mathsf{pk}}, \mathsf{vk}')$ where $\mathsf{pk}' = \{\mathsf{pk}_{i,j}\}_{i \in [t], j \in [\ell]}$, the receiver outputs $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \boldsymbol{x}), \perp)$.

– **Signer:** On input $(\mathsf{sid}, \mathsf{signature})$, the signer generates signatures $\sigma_{i,j}^{\mathsf{pk}} \leftarrow \mathsf{SignPK}(\mathsf{pk}_{i,j}, \mathsf{sk})$ and $\sigma_{i,j,b}^{\mathsf{m}} \leftarrow \mathsf{SignM}(\mathsf{pk}_{i,j}, \mathsf{sk}, b, \sigma_{i,j}^{\mathsf{pk}})$, and sets $\sigma_{i,j,b} = (\sigma_{i,j}^{\mathsf{pk}}, \sigma_{i,j,b}^{\mathsf{m}})$ for all $i \in [t]$, $j \in [\ell]$ and $b \in \{0,1\}$. The signer then sends $\big((\mathsf{sid}, i), \mathsf{sender}, \{(\sigma_{i,j,0}, \sigma_{i,j,1})\}_{j \in [\ell]}\big)$ to $\mathcal{F}_{\text{OT}}^{\ell,s}$. In addition, $\mathbf{S}$ sends the message-independent components $\{\sigma_{i,j}^{\mathsf{pk}}\}_{i \in [t], j \in [\ell]}$ to $\mathbf{R}$, and sets the ready flag.

Let $\{\tilde{\sigma}_{i,j}^{\mathsf{pk}}\}_{i \in [t], j \in [\ell]}$ be the message-independent signatures that $\mathbf{R}$ receives from $\mathbf{S}$, and $\{\tilde{\sigma}_{i,j}\}_{i \in [t], j \in [\ell]}$ be the signatures $\mathbf{R}$ receives from the different $\mathcal{F}_{\text{OT}}^{\ell,s}$ invocations. For all $i \in [t]$ and $j \in [\ell]$, the receiver checks that $\mathsf{VerifyFresh}(\mathsf{pk}_{i,j}, \mathsf{vk}', w_{i,j}, \tilde{\sigma}_{i,j}) = 1$, and moreover, that the message-independent component of $\tilde{\sigma}_{i,j}$ matches $\tilde{\sigma}_{i,j}^{\mathsf{pk}}$ it received from the signer. If any check fails, then $\mathbf{R}$ outputs $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \boldsymbol{x}), \perp)$. Otherwise, it evaluates $\boldsymbol{\sigma} \leftarrow \mathsf{SigEval}\big(f_{\mathsf{recon}}, \overrightarrow{\mathsf{pk}}, (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_t), (\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_t)\big)$, where $\boldsymbol{\sigma}_i = (\tilde{\sigma}_{i,1}, \ldots, \tilde{\sigma}_{i,\ell})$ for all $i \in [t]$. The receiver also sets the ready flag and outputs $\big(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \boldsymbol{x}), \boldsymbol{\sigma}\big)$.

Fig. 2: The $\mathit{\Pi}_{\text{BHS}}$ protocol. The protocol description continues on the next page.

Protocol $\mathit{\Pi}_{\text{BHS}}$ in the $\mathcal{F}_{\text{OT}}^{\ell,s}$-Hybrid Model (Continued)

Signature Verification: Upon receiving an input $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}, (f, \boldsymbol{x}), \boldsymbol{\sigma})$ where $\mathsf{vk} = (\overrightarrow{\mathsf{pk}}, \mathsf{vk}')$, party $\mathbf{P} \in \{\mathbf{S}, \mathbf{R}\}$ first checks if $f \notin \mathcal{H}$ and sets $t = 0$ if this is the case. Otherwise, it computes $\mathsf{pk}_f \leftarrow \mathsf{PrmsEval}(f \circ f_{\mathsf{recon}}, \overrightarrow{\mathsf{pk}})$. If $f = f_{\mathsf{id}}$, then it sets $t \leftarrow \mathsf{Verify}(\mathsf{pk}_f, \mathsf{vk}', \boldsymbol{x}, \boldsymbol{\sigma})$, and if $f \neq f_{\mathsf{id}}$, it sets $t \leftarrow \mathsf{VerifyHide}(\mathsf{pk}_f, \mathsf{vk}', \boldsymbol{x}, \boldsymbol{\sigma})$. It outputs $(\mathsf{sid}, \mathsf{verified}, \boldsymbol{x}, \boldsymbol{\sigma}, t)$.

Signature Evaluation: If the ready flag has not been set, then ignore the request. Otherwise, upon receiving an input $(\mathsf{sid}, \mathsf{eval}, \mathsf{vk}, g, (f, \boldsymbol{x}), \boldsymbol{\sigma})$, party $\mathbf{P} \in \{\mathbf{S}, \mathbf{R}\}$ ignores the request if $f \neq f_{\mathsf{id}}$. If $f = f_{\mathsf{id}}$, $\mathbf{P}$ runs the signature-verification procedure on input $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}, (f, \boldsymbol{x}), \boldsymbol{\sigma})$ (but does not produce an output). If the signature does not verify, then ignore the request. Otherwise, it parses $\mathsf{vk} = (\overrightarrow{\mathsf{pk}}, \mathsf{vk}')$, computes $\mathsf{pk}_{\mathsf{recon}} \leftarrow \mathsf{PrmsEval}(f_{\mathsf{recon}}, \overrightarrow{\mathsf{pk}})$ and computes $\sigma' \leftarrow \mathsf{SigEval}(g, \mathsf{pk}_{\mathsf{recon}}, \boldsymbol{x}, \boldsymbol{\sigma})$, and $\sigma^* \leftarrow \mathsf{Hide}(\mathsf{vk}', g(\boldsymbol{x}), \sigma')$. It outputs $(\mathsf{sid}, \mathsf{signature}, (g, g(\boldsymbol{x})), \sigma^*)$.

Fig. 2 (Continued): The $\mathit{\Pi}_{\text{BHS}}$ protocol.

Corollary 5.3 (Blind Homomorphic Signatures from LWE). *Let λ be a security parameter. Then, under the LWE assumption, for all $d = \mathsf{poly}(\lambda)$, there exists a protocol $\mathit{\Pi}'_{\text{BHS}}$ that securely realizes $\mathcal{F}_{\text{BHS}}$ for the class of depth-d Boolean circuits in the presence of malicious adversaries in the CRS model (and assuming static corruptions). Moreover, the protocol $\mathit{\Pi}'_{\text{BHS}}$ satisfies the following properties:*

- *The key-generation, signature-verification, and signature-evaluation protocols are non-interactive.*
- *The signature-generation protocol (i.e., blind signing) is a two-round interactive protocol between the signer and the receiver (one message each way).*
- *The length of a signature is $\mathsf{poly}(\lambda, d)$.*

Proof. Let $\mathit{\Pi}_{\text{BHS}}$ be the protocol from Figure 2 instantiated with a lattice-based homomorphic signature scheme (see the full version [74]). By Theorem 5.1, protocol $\mathit{\Pi}_{\text{BHS}}$ securely realizes $\mathcal{F}_{\text{BHS}}$ in the $\mathcal{F}_{\text{OT}}^{\ell,s}$-hybrid model, for some $\ell, s = \mathsf{poly}(\lambda)$. We let $\mathit{\Pi}'_{\text{BHS}}$ be the protocol obtained by instantiating the functionality $\mathcal{F}_{\text{OT}}^{\ell,s}$ in $\mathit{\Pi}_{\text{BHS}}$ with the protocol from Fact 5.2. Security of $\mathit{\Pi}'_{\text{BHS}}$ then follows from the universal composition theorem. Key generation, signature verification, and signature evaluation in $\mathit{\Pi}'_{\text{BHS}}$ simply corresponds to invoking the associated functionalities of the underlying homomorphic signature scheme, and thus, are non-interactive. The signature length is also inherited from $\mathit{\Pi}_{\mathsf{HS}}$. The blind signing protocol reduces to a single invocation of $\mathcal{F}_{\text{OT}}^{\ell,s}$, which by Fact 5.2, can be implemented by just two rounds of interaction.

Protocol $\mathit{\Pi}_{\mathsf{ZK}}$ in the $\mathcal{F}_{\text{BHS}}$-Hybrid Model

Let λ be a security parameter and $\mathit{\Pi}_{\mathsf{SE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a CPA-secure encryption scheme. We assume that the prover $\mathcal{P}$ and the verifier $\mathcal{V}$ have access to the ideal functionality $\mathcal{F}_{\text{BHS}}$, where $\mathcal{P}$ is the receiver $\mathbf{R}$ and $\mathcal{V}$ is the signer $\mathbf{S}$. For any NP relation $\mathcal{R}$, define the Boolean-valued function $\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},x}$, parameterized by $\mathcal{R}$, a statement x, and a ciphertext ct as follows: on input a secret key sk, $\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},x}(\mathsf{sk})$ outputs 1 if and only if $\mathcal{R}(x, \mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct})) = 1$, and 0 otherwise. We implicitly assume that $\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},x} \in \mathcal{H}$, where $\mathcal{H}$ is the function class associated with $\mathcal{F}_{\text{BHS}}$.

Preprocessing phase: In the preprocessing phase, the prover and verifier do the following:

1. The verifier sends $(\mathsf{sid}, \mathsf{keygen})$ to $\mathcal{F}_{\text{BHS}}$ and receives in response a verification key vk. The verifier sends vk to the prover. Subsequently, when the verifier receives $(\mathsf{sid}, \mathsf{signature})$ from $\mathcal{F}_{\text{BHS}}$, it sets the ready flag.
2. The prover begins by sampling a secret key $\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda)$. Then, it requests a signature on sk under vk by sending $(\mathsf{sid}, \mathsf{sign}, \mathsf{vk}, \mathsf{sk})$ to $\mathcal{F}_{\text{BHS}}$. The prover receives a signature σ_{sk} from $\mathcal{F}_{\text{BHS}}$. If $\sigma_{\mathsf{sk}} = \perp$, then the prover aborts.

Prover: On input a tuple $(\mathsf{sid}, \mathsf{ssid}, \mathsf{prove}, \mathcal{R}, x, w)$ where $\mathcal{R}(x, w) = 1$, the prover proceeds as follows:

1. Encrypt the witness w to obtain a ciphertext $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{sk}, w)$.
2. Submit $(\mathsf{sid}, \mathsf{eval}, \mathsf{vk}, \mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},x}, (f_{\mathsf{id}}, \mathsf{sk}), \sigma_{\mathsf{sk}})$ to $\mathcal{F}_{\text{BHS}}$ to obtain a signature σ^*.
3. Set $\pi = (\mathsf{ct}, \sigma^*)$ and send $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathcal{R}, x, \pi)$ to the verifier.

Verifier: When the verifier receives a tuple $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathcal{R}, x, \pi)$, it ignores the request if the ready flag has not been set. Otherwise, it parses $\pi = (\mathsf{ct}, \sigma)$, and ignores the message if π does not have this form. Otherwise, it submits $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}, (\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},x}, 1), \sigma)$ to $\mathcal{F}_{\text{BHS}}$. If the signature is valid (i.e., $\mathcal{F}_{\text{BHS}}$ replies with 1), then the verifier accepts and outputs $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathcal{R}, x)$. Otherwise the verifier ignores the message.

Fig. 3: Preprocessing ZK argument in the $\mathcal{F}_{\text{BHS}}$-hybrid model.

6 Universally-Composable Preprocessing NIZKs

In this section, we show how to combine blind homomorphic signatures with CPA-secure encryption to obtain UC-NIZKs in the preprocessing model from standard lattice assumptions. We give our protocol $\mathit{\Pi}_{\mathsf{ZK}}$ in the $\mathcal{F}_{\text{BHS}}$-hybrid model in Figure 3. Next, we state the formal security theorem and describe how to instantiate it from standard lattice assumptions. We give the proof of Theorem 6.1 in the full version of this paper [74].

Theorem 6.1 (Preprocessing Zero-Knowledge Arguments). *Let $\Pi_{\mathsf{SE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a CPA-secure encryption scheme. Then, the protocol Π_{ZK} in Figure 3 (instantiated with Π_{SE}) securely realizes $\mathcal{F}_{\mathsf{ZK}}$ in the presence of (static) malicious adversaries in the $\mathcal{F}_{\text{BHS}}$-hybrid model.*

Corollary 6.2 (Preprocessing UC-NIZKs from LWE). *Let λ be a security parameter. Then, under the LWE assumption, for all $d = \mathsf{poly}(\lambda)$, there exists a protocol Π'_{NIZK} that securely realizes $\mathcal{F}_{\mathsf{ZK}}$ in the presence of (static) malicious adversaries in the CRS model for all NP relations $\mathcal{R}$ that can be computed by a circuit of depth at most d. The protocol Π'_{NIZK} satisfies the following properties:*

- *The (one-time) preprocessing phase is a two-round protocol between the prover and the verifier.*
- *The prover's and verifier's algorithms are both non-interactive.*
- *If $\mathcal{R}$ is an NP relation, then the length of a proof of membership for the language associated with $\mathcal{R}$ is $m + \mathsf{poly}(\lambda, d)$, where m is the size of the witness associated with $\mathcal{R}$.*

Proof. Fix a depth bound $d = \mathsf{poly}(\lambda)$. First, we can instantiate the CPA-secure encryption scheme $\Pi_{\mathsf{SE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ in Figure 3 from lattices using any lattice-based CPA-secure symmetric encryption scheme [58,6]. Let d' be a bound on the depth of the circuit that computes the $\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},x}$ function in Figure 3. Note that $d' = \mathsf{poly}(\lambda, d)$, since the depth of the relation $\mathcal{R}$ is bounded by d and the depth of the $\mathsf{Decrypt}$ function is $\mathsf{poly}(\lambda)$. By Corollary 5.3, under the LWE assumption, there exists a protocol Π'_{BHS} that securely realizes $\mathcal{F}_{\text{BHS}}$ for the class of all depth-d' Boolean circuits in the presence of (static) malicious adversaries. The claim then follows by combining Theorem 6.1 with Corollary 5.3 and the universal composition theorem. We now check the additional properties:

- The preprocessing phase corresponds to the blind signing protocol of Π'_{BHS}, which is a two-round protocol between the signer and the verifier.
- The prover's algorithm corresponds to signature evaluation while the verifier's algorithm corresponds to signature verification. Both of these are non-interactive in Π'_{BHS}.
- The length of a proof for an NP relation $\mathcal{R}$ consists of an encryption of the witness under Π_{SE} (of size $m + \mathsf{poly}(\lambda)$) and a signature under Π'_{BHS} (of size $\mathsf{poly}(\lambda, d)$). The total size is bounded by $m + \mathsf{poly}(\lambda, d)$. $\square$

6.1 Applications to MPC

In the full version of this paper, we describe several applications of our preprocessing UC-NIZKs to boosting the security of MPC protocols. Specifically, we show that combining our construction with the round-optimal, semi-malicious MPC protocol of Mukherjee-Wichs [78] yields a round-optimal, malicious-secure MPC protocol from lattices in a *reusable preprocessing* model where the communication complexity only depends on the size of the inputs/outputs. Then, we show how to obtain a *succinct* version of the GMW [59,60] compiler from lattice assumptions.

Acknowledgments

We thank Dan Boneh and Akshayaram Srinivasan for many insightful comments and discussions on this work. We thank the anonymous reviewers for helpful comments on the presentation. This work was funded by NSF, DARPA, a grant from ONR, and the Simons Foundation. Opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

References

1. M. Abe. A secure three-move blind signature scheme for polynomially many signatures. In *EUROCRYPT*, 2001.
2. M. Abe, G. Fuchsbauer, J. Groth, K. Haralambiev, and M. Ohkubo. Structure-preserving signatures and commitments to group elements. In *CRYPTO*, 2010.
3. M. Abe, K. Haralambiev, and M. Ohkubo. Signing on elements in bilinear groups for modular protocol design. *IACR Cryptology ePrint Archive*, 2010, 2010.
4. M. Abe and M. Ohkubo. A framework for universally composable non-committing blind signatures. In *ASIACRYPT*, 2009.
5. J. H. Ahn, D. Boneh, J. Camenisch, S. Hohenberger, A. Shelat, and B. Waters. Computing on authenticated data. *J. Cryptology*, 28(2), 2015.
6. M. Ajtai. Generating hard instances of lattice problems. In *STOC*, 1996.
7. N. Alamati, C. Peikert, and N. Stephens-Davidowitz. New (and old) proof systems for lattice problems. In *PKC*, 2018.
8. G. Ateniese, R. C. Burns, R. Curtmola, J. Herring, L. Kissner, Z. N. J. Peterson, and D. X. Song. Provable data possession at untrusted stores. In *ACM CCS*, 2007.
9. G. Ateniese, S. Kamara, and J. Katz. Proofs of storage from homomorphic identification protocols. In *ASIACRYPT*, 2009.
10. N. Attrapadung and B. Libert. Homomorphic network coding signatures in the standard model. In *PKC*, 2011.
11. M. Backes, D. Fiore, and R. M. Reischuk. Verifiable delegation of computation on outsourced data. In *ACM CCS*, 2013.
12. F. Baldimtsi and A. Lysyanskaya. Anonymous credentials light. In *ACM CCS*, 2013.
13. M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko. The one-more-rsa-inversion problems and the security of chaum's blind signature scheme. *J. Cryptology*, 16(3), 2003.
14. F. Benhamouda, O. Blazy, L. Ducas, and W. Quach. Hash proof systems over lattices revisited. In *PKC*, 2018.
15. M. Blum, A. De Santis, S. Micali, and G. Persiano. Noninteractive zero-knowledge. *SIAM J. Comput.*, 20(6), 1991.
16. M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications. In *STOC*, 1988.
17. A. Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *PKC*, 2003.
18. D. Boneh and D. M. Freeman. Homomorphic signatures for polynomial functions. In *EUROCRYPT*, 2011.
19. D. Boneh and D. M. Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In *PKC*, 2011.

20. D. Boneh, D. M. Freeman, J. Katz, and B. Waters. Signing a linear subspace: Signature schemes for network coding. In *PKC*, 2009.
21. S. A. Brands. *Rethinking public key infrastructures and digital certificates: building in privacy.* MIT Press, 2000.
22. C. Brzuska, M. Fischlin, T. Freudenreich, A. Lehmann, M. Page, J. Schelbert, D. Schröder, and F. Volk. Security of sanitizable signatures revisited. In *PKC*, 2009.
23. J. Camenisch, M. Koprowski, and B. Warinschi. Efficient blind signatures without random oracles. In *SCN*, 2004.
24. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, 2001.
25. R. Canetti. Universally composable signature, certification, and authentication. In *CSFW*, 2004.
26. R. Canetti, Y. Chen, L. Reyzin, and R. D. Rothblum. Fiat-shamir and correlation intractability from strong KDM-secure encryption. In *EUROCRYPT*, 2018.
27. R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, 2002.
28. D. Catalano. Homomorphic signatures and message authentication codes. In *SCN*, 2014.
29. D. Catalano and D. Fiore. Practical homomorphic MACs for arithmetic circuits. In *EUROCRYPT*, 2013.
30. D. Catalano, D. Fiore, R. Gennaro, and L. Nizzardo. Generalizing homomorphic MACs for arithmetic circuits. In *PKC*, 2014.
31. D. Catalano, D. Fiore, and B. Warinschi. Efficient network coding signatures in the standard model. In *PKC*, 2012.
32. D. Catalano, D. Fiore, and B. Warinschi. Homomorphic signatures with efficient verification for polynomial functions. In *CRYPTO*, 2014.
33. P. Chaidos and G. Couteau. Efficient designated-verifier non-interactive zero-knowledge proofs of knowledge. *IACR Cryptology ePrint Archive*, 2017, 2017.
34. P. Chaidos and J. Groth. Making sigma-protocols non-interactive without random oracles. In *PKC*, 2015.
35. D. Chaum. Blind signatures for untraceable payments. In *CRYPTO*, 1982.
36. R. Cramer and I. Damgård. Secret-key zero-knowlegde and non-interactive verifiable exponentiation. In *TCC*, 2004.
37. R. Cramer and V. Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *EUROCRYPT*, 2002.
38. I. Damgård. Non-interactive circuit based proofs and non-interactive perfect zero-knowledge with proprocessing. In *EUROCRYPT*, 1992.
39. I. Damgård, N. Fazio, and A. Nicolosi. Non-interactive zero-knowledge from homomorphic encryption. In *TCC*, 2006.
40. A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai. Robust non-interactive zero knowledge. In *CRYPTO*, 2001.
41. A. De Santis, S. Micali, and G. Persiano. Non-interactive zero-knowledge proof systems. In *CRYPTO*, 1987.
42. A. De Santis, S. Micali, and G. Persiano. Non-interactive zero-knowledge with preprocessing. In *CRYPTO*, 1988.
43. Y. Dodis, S. P. Vadhan, and D. Wichs. Proofs of retrievability via hardness amplification. In *TCC*, 2009.
44. U. Feige, D. Lapidot, and A. Shamir. Multiple non-interactive zero knowledge proofs based on a single random string. In *FOCS*, 1990.

45. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, 1986.
46. D. Fiore, A. Mitrokotsa, L. Nizzardo, and E. Pagnin. Multi-key homomorphic authenticators. In *ASIACRYPT*, 2016.
47. M. Fischlin. Round-optimal composable blind signatures in the common reference string model. In *CRYPTO*, 2006.
48. D. M. Freeman. Improved security for linearly homomorphic signatures: A generic framework. In *PKC*, 2012.
49. G. Fuchsbauer. Automorphic signatures in bilinear groups and an application to round-optimal blind signatures. *IACR Cryptology ePrint Archive*, 2009, 2009.
50. G. Fuchsbauer, C. Hanser, C. Kamath, and D. Slamanig. Practical round-optimal blind signatures in the standard model from weaker assumptions. In *SCN*, 2016.
51. G. Fuchsbauer, C. Hanser, and D. Slamanig. Practical round-optimal blind signatures in the standard model. In *CRYPTO*, 2015.
52. S. Garg, V. Rao, A. Sahai, D. Schröder, and D. Unruh. Round optimal blind signatures. In *CRYPTO*, 2011.
53. R. Gennaro, J. Katz, H. Krawczyk, and T. Rabin. Secure network coding over the integers. In *PKC*, 2010.
54. R. Gennaro and D. Wichs. Fully homomorphic message authenticators. In *ASIACRYPT*, 2013.
55. C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, 2009.
56. C. Gentry, J. Groth, Y. Ishai, C. Peikert, A. Sahai, and A. D. Smith. Using fully homomorphic hybrid encryption to minimize non-interative zero-knowledge proofs. *J. Cryptology*, 28(4), 2015.
57. E. Ghadafi and N. P. Smart. Efficient two-move blind signatures in the common reference string model. In *ISC*, 2012.
58. O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. In *FOCS*, 1984.
59. O. Goldreich, S. Micali, and A. Wigderson. How to prove all np-statements in zero-knowledge, and a methodology of cryptographic protocol design. In *CRYPTO*, 1986.
60. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, 1987.
61. O. Goldreich and Y. Oren. Definitions and properties of zero-knowledge proof systems. *J. Cryptology*, 7(1), 1994.
62. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *STOC*, 1985.
63. S. Gorbunov, V. Vaikuntanathan, and D. Wichs. Leveled fully homomorphic signatures from standard lattices. In *STOC*, 2015.
64. J. Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In *ASIACRYPT*, 2006.
65. J. Groth. Short non-interactive zero-knowledge proofs. In *ASIACRYPT*, 2010.
66. J. Groth, R. Ostrovsky, and A. Sahai. Perfect non-interactive zero knowledge for NP. In *EUROCRYPT*, 2006.
67. J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT*, 2008.
68. L. Hanzlik and K. Kluczniak. A short paper on blind signatures from knowledge assumptions. In *Financial Cryptography*, 2016.
69. Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge proofs from secure multiparty computation. *SIAM J. Comput.*, 39(3), 2009.

70. Y. T. Kalai and R. Raz. Succinct non-interactive zero-knowledge proofs with preprocessing for LOGSNP. In *FOCS*, 2006.
71. J. Katz and V. Vaikuntanathan. Smooth projective hashing and password-based authenticated key exchange from lattices. In *ASIACRYPT*, 2009.
72. A. Kiayias and H. Zhou. Concurrent blind signatures without random oracles. In *SCN*, 2006.
73. J. Kilian, S. Micali, and R. Ostrovsky. Minimum resource zero-knowledge proofs. In *CRYPTO*, 1989.
74. S. Kim and D. J. Wu. Multi-theorem preprocessing NIZKs from lattices. *IACR Cryptology ePrint Archive*, 2018:272, 2018.
75. D. Lapidot and A. Shamir. Publicly verifiable non-interactive zero-knowledge proofs. In *CRYPTO*, 1990.
76. Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT*, 2007.
77. S. Ling, K. Nguyen, D. Stehlé, and H. Wang. Improved zero-knowledge proofs of knowledge for the ISIS problem, and applications. In *PKC*, 2013.
78. P. Mukherjee and D. Wichs. Two round multiparty computation via multi-key FHE. In *EUROCRYPT*, 2016.
79. C. Peikert and V. Vaikuntanathan. Noninteractive statistical zero-knowledge proofs for lattice problems. In *CRYPTO*, 2008.
80. C. Peikert, V. Vaikuntanathan, and B. Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO*, 2008.
81. D. Pointcheval and J. Stern. Security proofs for signature schemes. In *EUROCRYPT*, 1996.
82. D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *J. Cryptology*, 13(3), 2000.
83. O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, 2005.
84. R. D. Rothblum, A. Sealfon, and K. Sotiraki. Towards non-interactive zero-knowledge for NP from LWE. *IACR Cryptology ePrint Archive*, 2018, 2018.
85. M. Rückert. Lattice-based blind signatures. In *ASIACRYPT*, 2010.
86. A. Sahai and B. Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC*, 2014.
87. A. D. Santis and G. Persiano. Zero-knowledge proofs of knowledge without interaction. In *FOCS*, 1992.
88. C. Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO*, 1989.
89. H. Shacham and B. Waters. Compact proofs of retrievability. In *ASIACRYPT*, 2008.
90. X. Xie, R. Xue, and M. Wang. Zero knowledge proofs from ring-lwe. In *CANS*, 2013.
91. J. Zhang and Y. Yu. Two-round PAKE from approximate SPH and instantiations from lattices. In *ASIACRYPT*, 2017.