Multi-Theorem Preprocessing NIZKs from Lattices

Sam Kim
Stanford University
skim13@cs.stanford.edu

David J. Wu
Stanford University
dwu4@cs.stanford.edu

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 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 [GMR85], a zero-knowledge proof system enables a prover to convince a verifier that some statement 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 [GO94].

Non-interactive zero-knowledge. Nonetheless, Blum, Feldman, and Micali [BFM88] 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 [FS86, PS96] or the common reference string (CRS) model [FLS90, DDO+01,

GOS06, Gro10, SW14], 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 [FLS90, DDO+01, Gro10], pairings [GOS06], and indistinguishability obfuscation [SW14]. Notably absent from this list are constructions from lattice assumptions [Ajt96, Reg05]. While some partial progress has been made in the case of specific languages [PV08, APSD18], 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 (and proof[2]) system for NP from standard lattice assumptions in the *preprocessing* model. In the NIZK with preprocessing model [DMP88], 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 [CD04, DFN06, CG15], 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 [DMP88, LS90, Dam92, IKOS09] or oblivious transfer [KMO89] are only secure in the *single-theorem* setting. As we discuss in greater detail in Remark 4.11, the constructions from [DMP88, LS90, Dam92] only provide single-theorem zero-knowledge, while the constructions in [KMO89, IKOS09] only provide single-theorem soundness. Even in the designated-verifier setting [CD04, DFN06, CG15] (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

[1] There are also NIZK candidates based on number-theoretic assumptions [BFM88, DMP87, BDMP91] which satisfy weaker properties. We discuss these in greater detail in Section 1.2 and Remark 4.11.

[2] A simple variant of our construction gives a lattice-based NIZK *proof* system in the preprocessing model where soundness also holds against computationally-unbounded provers (Remark 4.10). For reasons outlined in Remark 4.10, however, we will focus primarily on our construction of NIZK arguments in the preprocessing model.

MPC protocols purely from lattice assumptions. Our work also implies a *succinct* version of the classic Goldreich-Micali-Wigderson compiler [GMW86, GMW87] 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 [FLS90] 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 4.11, 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 [BF11a, BF11b, GVW15, ABC+15] 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 [BFF+09, BF11a], 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. [GVW15] 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 $\mathcal{R}(x, w) = 1$.[3] 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 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 [GVW15] 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

[3] This is a classic technique in the construction of non-interactive proof systems and has featured in many contexts (e.g., [SP92, GGI+15]).

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 [FLS90, DDO+01, GOS06, Gro10] typically scale with the *size* of the circuit computing $\mathcal{R}$ and *multiplicatively* with the security parameter. Previously, Gentry et al. [GGI+15] 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 Remark 4.7. We also give the complete description of our preprocessing NIZK and security analysis in Section 4.

In the above construction, if the homomorphic signature scheme is unforgeable even against computationally-unbounded adversaries, then the construction gives a NIZK *proof* in the preprocessing model. In Remark 4.10, we describe how to realize this using lattice-based context-hiding statistically-binding *homomorphic commitments* [GVW15]. One of the advantages of using homomorphic signatures instead of homomorphic commitments is that they enable an efficient two-party protocol for implementing the preprocessing, which we discuss below. For this reason, we focus primarily on constructing preprocessing NIZK arguments from homomorphic signatures.

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., [LP07]) 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 [Cha82, Fis06]. 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 [Can01]. 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 [CLOS02]. 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. [GVW15] homomorphic signature construction satisfies our stronger security notions, and so coupled with the UC-secure lattice-based OT protocol of Peikert et al. [PVW08], 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 [MW16] 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 [GMW86, GMW87] 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 and proofs 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 [BFM88, DMP87, BDMP91]. 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 [KR06] 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 [Gro06, GS08] as well as the designated-verifier setting [CC17]. In the specific case of lattice-based constructions, there are several works on building hash-proof systems, (also known as smooth projective hash functions [CS02]) [KV09, ZY17, BBDQ18], 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 [LNSW13, XXW13]. Recently, there has also been work on instantiating the random oracle in Σ-protocols with lattice-based correlation-intractable hash functions [CCRR18]. However, realizing the necessary correlation-intractable hash functions from lattices requires making the non-standard assumption that Regev's encryption scheme [Reg05] 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. [RSS18] 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 [Cha82]. There are many constructions of blind signatures from a wide range of assumptions in the random oracle model [Sch89, Bra00, PS00, Abe01, Bol03, BNPS03, Rüc10, BL13], the CRS model [CKW04, KZ06, Fis06, AO09, Fuc09, AFG+10, AHO10, GS12], as well as the standard model [GRS+11, FHS15, FHKS16, HK16].

Homomorphic signatures. There are many constructions of linearly homomorphic signatures [ABC+07, SW08, DVW09, AKK09, BFKW09, GKKR10, BF11a, AL11, BF11b, CFW12, Fre12, ABC+15]. Beyond linear homomorphisms, a number of works [BF11b, BFR13, CFW14] have constructed homomorphic signatures for polynomial functions from lattices or multilinear maps. For general circuits, Gorbunov et al. [GVW15] gave the first homomorphic signature scheme from lattices, and Fiore et al. [FMNP16] 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 $x \xleftarrow{R} S$ 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). We now review the definition of a CPA-secure encryption scheme.

Definition 2.1 (CPA-Secure Encryption)**.** An encryption scheme with message space $\mathcal{M}$ is a tuple of efficient algorithms $\Pi_{\mathsf{SE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with the following properties:

- $\mathsf{KeyGen}(1^\lambda) \to \mathsf{sk}$: On input the security parameter λ, the key-generation algorithm outputs a secret key sk.
- $\mathsf{Encrypt}(\mathsf{sk}, m) \to \mathsf{ct}$: On input a secret key sk and a message $m \in \mathcal{M}$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}) \to m$: On input a secret key sk and a ciphertext ct, the decryption algorithm either outputs a message $m \in \mathcal{M}$ or a special symbol $\perp$ (to denote that decryption failed).

A CPA-secure encryption scheme should satisfy the following properties:

- **Correctness:** For all messages $m \in \mathcal{M}$, if we take $\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda)$, then
$$\Pr[\mathsf{Decrypt}(\mathsf{sk}, \mathsf{Encrypt}(\mathsf{sk}, m)) = m] = 1.$$
- **CPA-Security:** For all efficient adversaries $\mathcal{A}$, if we take $\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda)$, then
$$\left| \Pr\left[\mathcal{A}^{\mathcal{O}_0(\mathsf{sk},\cdot,\cdot)}(1^\lambda) = 1\right] - \Pr\left[\mathcal{A}^{\mathcal{O}_1(\mathsf{sk},\cdot,\cdot)}(1^\lambda) = 1\right] \right| = \mathsf{negl}(\lambda),$$
where $\mathcal{O}_b(\mathsf{sk}, m_0, m_1)$ outputs $\mathsf{Encrypt}(\mathsf{sk}, m_b)$ and $b \in \{0, 1\}$.

CPA-secure encryption can be built from any one-way function. Here, we state one candidate that follows from any lattice-based PRF that can be computed by a circuit of depth independent of its output length (c.f., [GGM84, Ajt96]).

Fact 2.2 (CPA-Secure Encryption from LWE)**.** Let λ be a security parameter. Under the LWE assumption (see Section 2.1), there exists a CPA-secure encryption scheme $\Pi_{\mathsf{SE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ over a message space $\mathcal{M}$ with the following properties:

- For all $m \in \mathcal{M}$, $\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda)$, and $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{sk}, m)$, we have that $|\mathsf{ct}| = |m| + \mathsf{poly}(\lambda)$.
- The decryption algorithm $\mathsf{Decrypt}$ can be computed by a circuit of depth $\mathsf{poly}(\lambda)$.

2.1 Lattice Preliminaries

In this section, we describe known results for lattice-based cryptography that we use throughout the paper.

Norms for vectors and matrices. Throughout this work, we will always use the infinity norm for vectors and matrices. This means that for a vector $\mathbf{x}$, the norm $\|\mathbf{x}\|$ is the maximal absolute value of an element in $\mathbf{x}$. Similarly, for a matrix $\mathbf{A}$, $\|\mathbf{A}\|$ is the maximal absolute value of any of its entries. If $\mathbf{x} \in \mathbb{Z}_q^n$ and $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, then $\|\mathbf{x}^T\mathbf{A}\| \leq n \cdot \|\mathbf{x}\| \cdot \|\mathbf{A}\|$.

Learning with errors. We first review the learning with errors (LWE) assumption [Reg05]. Let $n, m, q \in \mathbb{N}$ be positive integers and χ be a noise distribution over $\mathbb{Z}_q$. In the $\mathsf{LWE}(n, m, q, \chi)$ problem, the adversary's goal is to distinguish between the two distributions

$$(\mathbf{A}, \mathbf{A}^T\mathbf{s} + \mathbf{e}) \quad \text{and} \quad (\mathbf{A}, \mathbf{u})$$

where $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times m}$, $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$, $\mathbf{e} \leftarrow \chi^m$, and $\mathbf{u} \xleftarrow{R} \mathbb{Z}_q^m$.

When the error distribution χ is β-bounded[4], and under mild assumptions on the modulus q, the $\mathsf{LWE}(n, m, q, \chi)$ problem is as hard as approximating certain worst-case lattice problems such as GapSVP and SIVP on n-dimensional lattices to within a $\tilde{O}(n \cdot q/\beta)$ factor [Reg05, Pei09, ACPS09, MM11, MP12, BLP+13].

Short integer solutions. We also review the short integers solution (SIS) assumption [Ajt96]. Let $n, m, q, \beta \in \mathbb{N}$ be positive integers. In the $\mathsf{SIS}(n, m, q, \beta)$ problem, the adversary is given a uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and its goal is to find a vector $\mathbf{u} \in \mathbb{Z}_q^m$ with $\mathbf{u} \neq \mathbf{0}$ and $\|\mathbf{u}\| \leq \beta$ such that $\mathbf{A}\mathbf{u} = \mathbf{0}$.

For any $m = \mathsf{poly}(n)$, $\beta > 0$, and any sufficiently large $q \geq \beta \cdot \mathsf{poly}(n)$, solving the $\mathsf{SIS}(n, m, q, \beta)$ problem is as hard as approximating certain worst-case lattice problems such as GapSVP and SIVP on n-dimensional lattices to within a $\beta \cdot \mathsf{poly}(n)$ factor [Ajt96, Mic04, MR07, MP13]. It is also implied by the hardness of the LWE problem.

The gadget matrix. We define the "gadget matrix" $\mathbf{G} = \mathbf{g} \otimes \mathbf{I}_n \in \mathbb{Z}_q^{n \times n \cdot \lceil \log q \rceil}$ where $\mathbf{g} = (1, 2, 4, \ldots, 2^{\lceil \log q \rceil - 1})$. We define the inverse function $\mathbf{G}^{-1} \colon \mathbb{Z}_q^{n \times m} \to \mathbb{Z}_q^{n\lceil \log q \rceil \times m}$ which expands each entry $x \in \mathbb{Z}_q$ in the input matrix into a column of size $\lceil \log q \rceil$ consisting of the bits of the binary representation of x. To simplify the notation, we always assume that $\mathbf{G}$ has width m (in our construction, $m = \Theta(n \log q)$). Note that this is without loss of generality since we can always extend $\mathbf{G}$ by appending zero columns. For any matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, we have that $\mathbf{G} \cdot \mathbf{G}^{-1}(\mathbf{A}) = \mathbf{A}$.

Lattice trapdoors. Although solving the SIS problem for a uniformly random matrix $\mathbf{A}$ is believed to be hard, with some auxiliary trapdoor information (e.g., a set of short generating vectors for

[4]We say that a distribution $\mathcal{D}$ is β-bounded if the support of $\mathcal{D}$ is $\{-\beta, \ldots, \beta - 1, \beta\}$ with probability 1.

the lattice induced by $\mathbf{A}$), the problem becomes easy. Lattice trapdoors have featured in many applications and have been extensively studied [Ajt99, GPV08, AP09, MP12, LW15]. Since the specific details of the constructions are not essential for understanding this work, we just recall the main properties that we require in the following theorem.

Theorem 2.3 (Lattice Trapdoors [Ajt99, GPV08, AP09, MP12, LW15])**.** *Fix a security parameter λ and lattice parameters n, q, m and a norm bound β where $m = O(n \log q)$ and $\beta = O(n\sqrt{\log q})$. Then, there exists a tuple of efficient algorithms* $(\mathsf{TrapGen}, \mathsf{Sample}, \mathsf{SamplePre})$ *with the following properties:*

- $\mathsf{TrapGen}(1^\lambda) \to (\mathbf{A}, \mathsf{td})$*: On input the security parameter λ, the trapdoor generation algorithm outputs a rank-n matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and a trapdoor* td*.*
- $\mathsf{Sample}(\mathbf{A}) \to \mathbf{U}$*: On input a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, the sampling algorithm returns a matrix $\mathbf{U} \in \mathbb{Z}_q^{m \times m}$.*
- $\mathsf{SamplePre}(\mathbf{A}, \mathbf{V}, \mathsf{td}) \to \mathbf{U}$*: On input a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, a target matrix $\mathbf{V} \in \mathbb{Z}_q^{n \times m}$, and a trapdoor* td*, the preimage-sampling algorithm outputs a matrix $\mathbf{U} \in \mathbb{Z}_q^{m \times m}$.*
- *The above algorithms satisfy the following properties. Take $(\mathbf{A}, \mathsf{td}) \leftarrow \mathsf{TrapGen}(1^\lambda)$. Then,*
  1. *For $\mathbf{U} \leftarrow \mathsf{Sample}(\mathbf{A})$, we have $\|\mathbf{U}\| \leq \beta$.*
  2. *For any $\mathbf{V} \in \mathbb{Z}_q^{n \times m}$ and $\mathbf{U} \leftarrow \mathsf{SamplePre}(\mathbf{A}, \mathbf{V}, \mathsf{td})$, we have $\mathbf{AU} = \mathbf{V}$ and $\|\mathbf{U}\| \leq \beta$.*
  3. *For $(\mathbf{A}, \mathsf{td}) \leftarrow \mathsf{TrapGen}(1^\lambda)$, $\mathbf{A}' \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times m}$, $\mathbf{U} \leftarrow \mathsf{Sample}(\mathbf{A})$, $\mathbf{V} = \mathbf{AU}$, $\mathbf{V}' \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times m}$, and $\mathbf{U}' \leftarrow \mathsf{SamplePre}(\mathbf{A}, \mathbf{V}', \mathsf{td})$, we have*

$$\mathbf{A} \overset{s}{\approx} \mathbf{A}' \qquad \text{and} \qquad (\mathbf{A}, \mathsf{td}, \mathbf{U}, \mathbf{V}) \overset{s}{\approx} (\mathbf{A}, \mathsf{td}, \mathbf{U}', \mathbf{V}').$$

Traditionally, lattice trapdoors consist of a set of short generating vectors of the lattice that is induced by a public SIS matrix $\mathbf{A}$. In this work, we make use of an alternative form of lattice trapdoors called a $\mathbf{G}$-trapdoor formalized in [MP12]. A $\mathbf{G}$-trapdoor of a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ consists of a full-rank, low-norm matrix $\mathbf{R} \in \mathbb{Z}_q^{m \times m}$ satisfying the relation $\mathbf{AR} = \mathbf{G}$. These types of trapdoor matrices have additional statistical properties that we use in our blind homomorphic signature constructions in Sections 3 and 5. We summarize these properties in the following theorem.

Theorem 2.4 (Lattice Sampling [CHKP10, ABB10, MP12, BGG+14, LW15])**.** *Fix a security parameter λ and lattice parameters n, q, m, and a norm bound β, where $m = O(n \log q)$ and $\beta = O(n\sqrt{\log q})$. Then, in addition to the algorithms* $(\mathsf{TrapGen}, \mathsf{Sample}, \mathsf{SamplePre})$ *from Theorem 2.3, there exists a pair of algorithms* $(\mathsf{SampleLeft}, \mathsf{SampleRight})$ *with the following properties:*

- $\mathsf{SampleLeft}(\mathbf{A}, \mathbf{B}, \mathbf{R}, \mathbf{v}, \beta^*) \to \mathbf{u}$*: On input matrices $\mathbf{A}, \mathbf{B} \in \mathbb{Z}_q^{n \times m}$, a matrix $\mathbf{R} \in \mathbb{Z}_q^{m \times m}$ (trapdoor of $\mathbf{A}$), a target vector $\mathbf{v} \in \mathbb{Z}_q^n$, and a norm bound β^*,* $\mathsf{SampleLeft}$ *returns a vector $\mathbf{u} \in \mathbb{Z}_q^{2m}$.*
- $\mathsf{SampleRight}(\mathbf{A}, \mathbf{B}, \mathbf{U}, \mathbf{v}, \beta^*) \to \mathbf{u}$*: On input matrices $\mathbf{A}, \mathbf{B} \in \mathbb{Z}_q^{n \times m}$, a matrix $\mathbf{U} \in \mathbb{Z}_q^{m \times m}$, a target vector $\mathbf{v} \in \mathbb{Z}_q^n$, and a norm bound β^*,* $\mathsf{SampleRight}$ *returns a vector $\mathbf{u} \in \mathbb{Z}_q^{2m}$.*
- *The algorithms above satisfies the following properties. For* any *rank-n matrices $\mathbf{A}, \mathbf{B} \in \mathbb{Z}_q^{n \times m}$ and a target vector $\mathbf{v} \in \mathbb{Z}_q^n$, we have:*

1. *Let* $\mathbf{R} \in \mathbb{Z}_q^{m \times m}$ *be any matrix satisfying* $\mathbf{AR} = \mathbf{G}$ *and* $\|\mathbf{R}\| \cdot \omega(m\sqrt{\log m}) \leq \beta^* \leq q$. *Then, for* $\mathbf{u}_0 \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{B}, \mathbf{R}, \mathbf{v}, \beta^*)$, *we have that* $[\mathbf{A} \mid \mathbf{B}] \cdot \mathbf{u}_0 = \mathbf{v}$ *and* $\|\mathbf{u}_0\| \leq \beta^*$.
2. *Let* $\mathbf{U} \in \mathbb{Z}_q^{m \times m}$ *be any matrix satisfying* $\mathbf{AU} + y\mathbf{G} = \mathbf{B}$ *for some* $y \neq 0$ *where* $y \in \mathbb{Z}_q$ *and* $\|\mathbf{U}\| \cdot \omega(m\sqrt{\log m}) \leq \beta^* \leq q$. *Then, for* $\mathbf{u}_1 \leftarrow \mathsf{SampleRight}(\mathbf{A}, \mathbf{B}, \mathbf{U}, \mathbf{v}, \beta^*)$, *we have that* $[\mathbf{A} \mid \mathbf{B}] \cdot \mathbf{u}_1 = \mathbf{v}$ *and* $\|\mathbf{u}_1\| \leq \beta^*$.
3. *The distributions of* $\mathbf{u}_0, \mathbf{u}_1$ *above are statistically indistinguishable.*

GSW homomorphic operations. In this work, we use the homomorphic structure from the fully homomorphic encryption (FHE) scheme by Gentry, Sahai, and Waters [GSW13]. Since we do not require the specific details of the homomorphic operations, we summarize the properties we need in the theorem below. In the language of FHE, the algorithm EvalPK corresponds to homomorphic evaluation over ciphertexts, while EvalU corresponds to homomorphic evaluation over the encryption randomness.

Theorem 2.5 (GSW Homomorphic Operations [GSW13, BV14, AP14, GV15])**.** *Fix a security parameter* λ, *lattice parameters* n, q, m, *a norm bound* β, *a depth bound* d, *and a message length* ℓ, *where* $m = O(n \log q)$ *and* $\beta \cdot 2^{\tilde{O}(d)} < q$. *Then, there exists a pair of efficient deterministic algorithms* $(\mathsf{EvalPK}, \mathsf{EvalU})$ *with the following properties:*

- $\mathsf{EvalPK}(\mathbf{V}_1, \ldots, \mathbf{V}_\ell, C) \to \mathbf{V}_C$*: On input matrices* $\mathbf{V}_1, \ldots, \mathbf{V}_\ell \in \mathbb{Z}_q^{n \times m}$ *and a circuit* $C \colon \{0,1\}^\ell \to \{0,1\}$ *of depth at most* d, EvalPK *returns an evaluated matrix* $\mathbf{V}_C \in \mathbb{Z}_q^{n \times m}$.
- $\mathsf{EvalU}\big((\mathbf{V}_1, x_1, \mathbf{U}_1), \ldots, (\mathbf{V}_\ell, x_\ell, \mathbf{U}_\ell), C\big) \to \mathbf{U}_C$*: On input tuples* $(\mathbf{V}_i, x_i, \mathbf{U}_i)$, *where* $\mathbf{V}_i \in \mathbb{Z}_q^{n \times m}$, $x_i \in \{0,1\}$, *and* $\mathbf{U}_i \in \mathbb{Z}_q^{m \times m}$ *for all* $i \in [\ell]$, *and a circuit* $C \colon \{0,1\}^\ell \to \{0,1\}$, EvalU *returns an evaluated matrix* $\mathbf{U}_C \in \mathbb{Z}_q^{m \times m}$.
- *For all circuits* $C \colon \{0,1\}^\ell \to \{0,1\}$ *of depth at most* d, *and all matrices* $\mathbf{A}, \mathbf{V}_1, \ldots, \mathbf{V}_\ell \in \mathbb{Z}_q^{n \times m}$, *inputs* $x_1, \ldots, x_\ell \in \{0,1\}$, *and matrices* $\mathbf{U}_1, \ldots, \mathbf{U}_\ell \in \mathbb{Z}_q^{m \times m}$ *where*

$$\mathbf{AU}_i + x_i \cdot \mathbf{G} = \mathbf{V}_i \quad \forall i \in [\ell],$$

and $\|\mathbf{U}_i\| \leq \beta$, *the* EvalPK *and* EvalU *algorithms satisfy the following property. For* $\mathbf{V}_C \leftarrow \mathsf{EvalPK}(\mathbf{V}_1, \ldots, \mathbf{V}_\ell, C)$, *and* $\mathbf{U}_C \leftarrow \mathsf{EvalU}((\mathbf{V}_1, x_1, \mathbf{U}_1), \ldots, (\mathbf{V}_\ell, x_\ell, \mathbf{U}_\ell), C)$, *we have that*

$$\mathbf{AU}_C + C(x) \cdot \mathbf{G} = \mathbf{V}_C \quad \text{and} \quad \|\mathbf{U}_C\| \leq \beta \cdot 2^{\tilde{O}(d)} < q.$$

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 $\vec{x} \in \mathcal{X}^\ell$ can also be written as a tuple of ℓ signatures $\vec{\sigma} = (\sigma_1, \ldots, \sigma_\ell)$ 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 [BF11b, GVW15])**.** 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 $\vec{x} = (x_1, \ldots, x_\ell) \in \mathcal{X}^\ell$, we write $\mathsf{Sign}(\overrightarrow{\mathsf{pk}}, \mathsf{sk}, \vec{x})$ to denote component-wise signing of each message. Namely, $\mathsf{Sign}(\overrightarrow{\mathsf{pk}}, \mathsf{sk}, \vec{x})$ outputs signatures $\vec{\sigma} = (\sigma_1, \ldots, \sigma_\ell)$ 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 $\mathsf{pk}_C = (\mathsf{pk}_{C_1}, \ldots, \mathsf{pk}_{C_k})$.

- $\mathsf{SigEval}(C, \overrightarrow{\mathsf{pk}'}, \vec{x}, \vec{\sigma}) \to \sigma$: 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 $\vec{x} \in \mathcal{X}^\ell$, and signatures $\vec{\sigma} = (\sigma_1, \ldots, \sigma_\ell)$, 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 $\vec{x} = (x_1, \ldots, x_k)$ and $\vec{\sigma} = (\sigma_1, \ldots, \sigma_k)$, we write $\mathsf{Hide}(\mathsf{vk}, \vec{x}, \vec{\sigma})$ to denote component-wise evaluation of the hide algorithm. Namely, $\mathsf{Hide}(\mathsf{vk}, \vec{x}, \vec{\sigma})$ 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 $\vec{x} = (x_1, \ldots, x_k)$, and signatures $\vec{\sigma} = (\sigma_1, \ldots, \sigma_k)$, we write $\mathsf{Verify}(\overrightarrow{\mathsf{pk}'}, \mathsf{vk}, \vec{x}, \vec{\sigma})$ to denote applying the verification algorithm to each signature component-wise. In other words, $\mathsf{Verify}(\overrightarrow{\mathsf{pk}'}, \mathsf{vk}, \vec{x}, \vec{\sigma})$ 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 $\mathsf{VerifyFresh}$ 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 $\mathsf{VerifyHide}$ 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 [GVW15]. 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 $\mathsf{SigEval}$). The hide verification algorithm $\mathsf{VerifyHide}$ is used for verifying signatures output by the context-hiding transformation Hide, which may be structurally different from the signatures output by Sign or $\mathsf{SigEval}$. Finally, we have a special verification algorithm $\mathsf{VerifyFresh}$ that can be used to verify signatures output by Sign (before any homomorphic evaluation has taken place). While Verify subsumes $\mathsf{VerifyFresh}$, having a separate $\mathsf{VerifyFresh}$ algorithm is useful for formulating a strong version of evaluation correctness. We now state our correctness definitions. First, we have the standard correctness requirement of any signature scheme. Specifically, signatures output by the honest signing algorithm should verify according to both Verify and $\mathsf{VerifyFresh}$.

Definition 3.2 (Signing Correctness)**.** A homomorphic signature scheme $\Pi_{\mathsf{HS}} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrmsEval}, \mathsf{SigEval}, \mathsf{Hide}, \mathsf{Verify}, \mathsf{VerifyFresh}, \mathsf{VerifyHide})$ with message space $\mathcal{X}$, message length ℓ, and function class $\mathcal{C}$ satisfies *signing correctness* if for all $\lambda \in \mathbb{N}$, messages $\vec{x} \in \mathcal{X}^\ell$, and setting $\overrightarrow{\mathsf{pk}} \leftarrow \mathsf{PrmsGen}(1^\lambda, 1^\ell)$, $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, $\vec{\sigma} \leftarrow \mathsf{Sign}(\overrightarrow{\mathsf{pk}}, \mathsf{sk}, \vec{x})$, we have

$$\Pr[\mathsf{Verify}(\overrightarrow{\mathsf{pk}}, \mathsf{vk}, \vec{x}, \vec{\sigma}) = 1] = 1 \quad \text{and} \quad \Pr[\mathsf{VerifyFresh}(\overrightarrow{\mathsf{pk}}, \mathsf{vk}, \vec{x}, \vec{\sigma}) = 1] = 1.$$

Evaluation correctness. Next, we require that if one applies the honest signature-evaluation algorithm $\mathsf{SigEval}$ to valid *fresh* signatures (namely, signatures that are accepted by $\mathsf{VerifyFresh}$), then the resulting signature verifies according to Verify (with respect to the corresponding evaluated public key). This is a *stronger* definition than the usual notion of evaluation correctness from [GVW15], which only requires correctness to holds when $\mathsf{SigEval}$ is applied to signatures output by the *honest*

signing algorithm Sign. In our definition, correctness must hold against *all* signatures deemed valid by $\mathsf{VerifyFresh}$ (with respect to an arbitrary public key pk and verification key vk), which may be a larger set of signatures than those that could be output by Sign. This notion of correctness will be useful in our construction of (malicious-secure) blind homomorphic signatures in Section 5.

Definition 3.3 (Evaluation Correctness)**.** A homomorphic signature scheme $\Pi_{\mathsf{HS}} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrmsEval}, \mathsf{SigEval}, \mathsf{Hide}, \mathsf{Verify}, \mathsf{VerifyFresh}, \mathsf{VerifyHide})$ with message space $\mathcal{X}$, message length ℓ, 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 $\mathcal{X}$) satisfies *evaluation correctness* if for all $\lambda \in \mathbb{N}$, all public keys $\overrightarrow{\mathsf{pk}}$, all verification keys vk, and all messages $\vec{x} \in \mathcal{X}^\ell$, the following properties hold:

- **Single-Hop Correctness:** For all $C \in \mathcal{C}_\lambda$ and all signatures $\vec{\sigma}$ where $\mathsf{VerifyFresh}(\overrightarrow{\mathsf{pk}}, \mathsf{vk}, \vec{x}, \vec{\sigma}) = 1$, if we set $\mathsf{pk}_C \leftarrow \mathsf{PrmsEval}(C, \overrightarrow{\mathsf{pk}})$ and $\sigma \leftarrow \mathsf{SigEval}(C, \overrightarrow{\mathsf{pk}}, \vec{x}, \vec{\sigma})$, then
$$\Pr[\mathsf{Verify}(\mathsf{pk}_C, \mathsf{vk}, C(\vec{x}), \sigma) = 1] = 1.$$
- **Multi-Hop Correctness:** For any collection of functions $C_1, \ldots, C_\ell \in \mathcal{C}_\lambda$ and $C' \colon \mathcal{X}^\ell \to \mathcal{X}$, define the composition $(C' \circ \vec{C}) \colon \mathcal{X}^\ell \to \mathcal{X}$ to be the mapping $\vec{x} \mapsto C'(C_1(\vec{x}), \ldots, C_\ell(\vec{x}))$. If $(C' \circ \vec{C}) \in \mathcal{C}_\lambda$, then for any set of signatures $\vec{\sigma} = (\sigma_1, \ldots, \sigma_\ell)$ where $\mathsf{Verify}(\mathsf{pk}_{C_i}, \mathsf{vk}, x_i, \sigma_i) = 1$ and $\mathsf{pk}_{C_i} \leftarrow \mathsf{PrmsEval}(C_i, \overrightarrow{\mathsf{pk}})$ for all $i \in [\ell]$, we have that
$$\Pr[\mathsf{Verify}(\mathsf{PrmsEval}(C', (\mathsf{pk}_{C_1}, \ldots, \mathsf{pk}_{C_\ell})), \mathsf{vk}, \vec{x}, \mathsf{SigEval}(C', (\mathsf{pk}_{C_1}, \ldots, \mathsf{pk}_{C_\ell}), \vec{x}, \vec{\sigma})) = 1] = 1.$$

Hiding correctness. Finally, we require that the hide algorithm also produces valid signatures. Similar to the case of evaluation correctness, we require that correctness holds whenever Hide is applied to any valid signature accepted by Verify (which need not coincide with the set of signatures output by an honest execution of Sign or $\mathsf{SigEval}$). This is essentially the definition in [GVW15].

Definition 3.4 (Hiding Correctness)**.** A homomorphic signature scheme $\Pi_{\mathsf{HS}} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrmsEval}, \mathsf{SigEval}, \mathsf{Hide}, \mathsf{Verify}, \mathsf{VerifyFresh}, \mathsf{VerifyHide})$ with message space $\mathcal{X}$, message length ℓ, and function class $\mathcal{C}$ satisfies *hiding correctness* if for all $\lambda \in \mathbb{N}$, all verification keys vk, messages $x \in \mathcal{X}$, and all signatures σ where $\mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, x, \sigma) = 1$, we have that
$$\Pr[\mathsf{VerifyHide}\big(\mathsf{pk}, \mathsf{vk}, x, \mathsf{Hide}(\mathsf{vk}, x, \sigma)\big) = 1] = 1.$$

Unforgeability. We now formally define unforgeability for a homomorphic signature scheme. 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 $y \neq C(x_1, \ldots, x_\ell)$.

Definition 3.5 (Unforgeability)**.** Fix a security parameter λ. 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 $\mathcal{X}$, message length ℓ, and function class $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$, where each $\mathcal{C}_\lambda$ is a collection of functions on $\mathcal{X}^\ell$. Then, for an adversary $\mathcal{A}$, we define the unforgeability security experiment $\mathsf{Expt}^{\mathsf{uf}}_{\mathcal{A},\Pi_{\mathsf{HS}}}(\lambda, \ell)$ as follows:

1. The challenger begins by generating public keys $\overrightarrow{\mathsf{pk}} \leftarrow \mathsf{PrmsGen}(1^\lambda, 1^\ell)$, and a signing-verification key $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$. It gives $\overrightarrow{\mathsf{pk}}$ and vk to $\mathcal{A}$.

2. The adversary $\mathcal{A}$ submits a set of messages $\vec{x} \in \mathcal{X}^\ell$ to be signed.
3. The challenger signs the messages $\vec{\sigma} \leftarrow \mathsf{Sign}(\overrightarrow{\mathsf{pk}}, \mathsf{sk}, \vec{x})$ and sends the signatures $\vec{\sigma}$ to $\mathcal{A}$.
4. The adversary $\mathcal{A}$ outputs a circuit C, a message $\vec{x}^*$, and a signature $\vec{\sigma}^*$.
5. The output of the experiment is 1 if $C \in \mathcal{C}_\lambda$, $\vec{x}^* \neq C(\vec{x})$, and $\mathsf{VerifyHide}(\mathsf{pk}_C, \mathsf{vk}, \vec{x}^*, \vec{\sigma}^*) = 1$, where $\mathsf{pk}_C \leftarrow \mathsf{PrmsEval}(C, \overrightarrow{\mathsf{pk}})$. Otherwise, the output of the experiment is 0.

We say that a homomorphic signature scheme Π_{HS} satisfies unforgeability if for all efficient adversaries $\mathcal{A}$,

$$\Pr[\mathsf{Expt}^{\mathsf{uf}}_{\mathcal{A},\Pi_{\mathsf{HS}}}(\lambda, \ell) = 1] = \mathsf{negl}(\lambda).$$

Remark 3.6 (Selective Unforgeability)**.** We can also define a weaker notion of unforgeability called *selective unforgeability* where the adversary commits to the messages $\vec{x} \in \mathcal{X}^\ell$ at the start of the experiment *before* it sees the public keys $\overrightarrow{\mathsf{pk}}$ and the verification key vk. In Section 3.1, we describe a simplified variant of the [GVW15] construction that satisfies this weaker notion of selective unforgeability. In Appendix B, we give the full construction from [GVW15] that satisfies the definition of adaptive unforgeability from Definition 3.5.

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 $(\vec{x}, \vec{\sigma})$ to obtain a signature $\sigma_{C(\vec{x})}$, and then runs the hide algorithm on $\sigma_{C(\vec{x})}$, the resulting signature $\sigma^*_{C(\vec{x})}$ does not contain any information about $\vec{x}$ other than what is revealed by C and $C(\vec{x})$. Previous works such as [GVW15] captured this notion by requiring that there exists an efficient simulator that can simulate the signature $\sigma^*_{C(\vec{x})}$ given just the signing key sk,[5] the function C, and the value $C(\vec{x})$. Notably, the simulator does not see the original message $\vec{x}$ or the signature $\sigma_{C(\vec{x})}$

While this is a very natural notion of context-hiding, it can be difficult to satisfy. The homomorphic signature candidate by Gorbunov et al. [GVW15] satisfies *selective unforgeability* (Remark 3.6) and context-hiding. Gorbunov et al. also give a variant of their construction that achieves adaptive unforgeability; however, this scheme does *not* simultaneously satisfy the notion of context-hiding. Nonetheless, the adaptively-secure scheme from [GVW15] can be shown to satisfy a weaker notion of context-hiding that suffices for all of our applications (and still captures all of the intuitive properties we expect from context-hiding). Specifically, in our weaker notion of context-hiding, we allow the simulator to also take in some components of the original signatures $\sigma_{C(\vec{x})}$, provided that those components are *independent* of the value that is signed.[6]

To formalize this notion, we first define the notion of a *decomposable* homomorphic signature scheme. In a decomposable homomorphic signature scheme, any valid signature $\sigma = (\sigma^{\mathsf{pk}}, \sigma^{\mathsf{m}})$ can be decomposed into a message-independent component σ^{pk} that contains no information about the signed message, and a message-dependent component σ^{m}.

Definition 3.7 (Decomposable Homomorphic Signatures)**.** 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 $\mathcal{X}$, message length ℓ, and function class $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$. We say that Π_{HS} is *decomposable* if the signing and evaluation algorithms can be decomposed into a message-independent and a message-dependent algorithm as follows:

[5] Note that the simulator must take in some secret value (not known to the evaluator). Otherwise, the existence of such a simulator breaks unforgeability of the signature scheme.

[6] The construction in Appendix B combines the homomorphic signature scheme that satisfies (full) unforgeability but not context-hiding and the selectively unforgeable homomorphic signature scheme that satisfies context-hiding in [GVW15].

- The signing algorithm Sign splits into a pair of algorithms $(\mathsf{SignPK}, \mathsf{SignM})$:
  - $\mathsf{SignPK}(\mathsf{pk}_i, \mathsf{sk}) \to \sigma_i^{\mathsf{pk}}$: On input a public key pk_i and a signing key sk, the SignPK algorithm outputs a message-independent component σ_i^{pk}.
  - $\mathsf{SignM}(\mathsf{pk}_i, \mathsf{sk}, x_i, \sigma_i^{\mathsf{pk}}) \to \sigma_i^{\mathsf{m}}$: On input a public key pk_i, a signing key sk, a message $x_i \in \mathcal{X}$, and a message-independent component σ_i^{pk}, the SignM algorithm outputs a message-dependent component σ_i^{m}.

  The actual signing algorithm $\mathsf{Sign}(\mathsf{pk}_i, \mathsf{sk}, x_i)$ then computes $\sigma_i^{\mathsf{pk}} \leftarrow \mathsf{SignPK}(\mathsf{pk}_i, \mathsf{sk})$ and $\sigma_i^{\mathsf{m}} \leftarrow \mathsf{SignM}(\mathsf{pk}_i, \mathsf{sk}, x_i, \sigma_i^{\mathsf{pk}})$. The final signature is the pair $\sigma_i = (\sigma_i^{\mathsf{pk}}, \sigma_i^{\mathsf{m}})$.

- The evaluation algorithm $\mathsf{SigEval}$ splits into a pair of algorithms: $(\mathsf{SigEvalPK}, \mathsf{SigEvalM})$:
  - $\mathsf{SigEvalPK}(C, \overrightarrow{\mathsf{pk}'}, \vec{\sigma}^{\mathsf{pk}}) \to \sigma^{\mathsf{pk}}$: On input a circuit $C \in \mathcal{C}_\lambda$, public keys $\overrightarrow{\mathsf{pk}'} = (\mathsf{pk}'_1, \ldots, \mathsf{pk}'_\ell)$, and message-independent signature components $\vec{\sigma}^{\mathsf{pk}} = (\sigma_1^{\mathsf{pk}}, \ldots, \sigma_\ell^{\mathsf{pk}})$, the $\mathsf{SigEvalPK}$ algorithm outputs a message-independent component σ^{pk}.
  - $\mathsf{SigEvalM}(C, \overrightarrow{\mathsf{pk}'}, \vec{x}, \vec{\sigma}) \to \sigma^{\mathsf{m}}$: On input a circuit $C \in \mathcal{C}_\lambda$, public keys $\overrightarrow{\mathsf{pk}'} = (\mathsf{pk}'_1, \ldots, \mathsf{pk}'_\ell)$, messages $\vec{x} \in \mathcal{X}^\ell$, and signatures $\vec{\sigma}$, the $\mathsf{SigEvalM}$ algorithm outputs a message-dependent component σ^{m}.

  The signature evaluation algorithm $\mathsf{SigEval}(C, \overrightarrow{\mathsf{pk}'}, \vec{x}, \vec{\sigma})$ first parses $\vec{\sigma} = (\vec{\sigma}^{\mathsf{pk}}, \vec{\sigma}^{\mathsf{m}})$, computes $\sigma^{\mathsf{pk}} \leftarrow \mathsf{SigEvalPK}(C, \overrightarrow{\mathsf{pk}'}, \vec{\sigma}^{\mathsf{pk}})$, $\sigma^{\mathsf{m}} \leftarrow \mathsf{SigEvalM}(C, \overrightarrow{\mathsf{pk}'}, \vec{x}, \vec{\sigma})$, and returns $\sigma = (\sigma^{\mathsf{pk}}, \sigma^{\mathsf{m}})$.

To formalize context-hiding, we require that there exists a simulator that can simulate the output of the hide algorithm given only the secret signing key sk, the function C, the output $C(\vec{x})$, and the message-independent component of the signature $\sigma_{C(\vec{x})}^{\mathsf{pk}}$. We give the formal definition below:

Definition 3.8 (Context-Hiding Against Honest Signers)**.** Fix a security parameter λ. Let $\Pi_{\mathsf{HS}} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrmsEval}, \mathsf{SigEval}, \mathsf{Hide}, \mathsf{Verify}, \mathsf{VerifyFresh}, \mathsf{VerifyHide})$ be a decomposable homomorphic signature scheme (Definition 3.7) with message space $\mathcal{X}$, message length ℓ, 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 $\mathcal{X}$. For a bit $b \in \{0,1\}$, a simulator $\mathcal{S}$ and an adversary $\mathcal{A}$, we define the *weak context-hiding* security experiment against an honest signer $\mathsf{Expt}^{\mathsf{ch\text{-}honest}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda, b)$ as follows:

1. The challenger begins by generating a signing and verification key $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and sends $(\mathsf{vk}, \mathsf{sk})$ to $\mathcal{A}$.

2. The adversary $\mathcal{A}$ can then submit (adaptive) queries to the challenger where each query consists of a public key pk, a message $x \in \mathcal{X}$, and a signature $\sigma = (\sigma^{\mathsf{pk}}, \sigma^{\mathsf{m}})$. On each query, the challenger first checks that $\mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, x, \sigma) = 1$. If this is not the case, then the challenger ignores the query and replies with $\perp$. Otherwise, the challenger proceeds as follows:
   - If $b = 0$, the challenger evaluates $\sigma^* \leftarrow \mathsf{Hide}(\mathsf{vk}, x, \sigma)$, and sends σ^* to $\mathcal{A}$.
   - If $b = 1$, the challenger computes $\sigma^* \leftarrow \mathcal{S}(\mathsf{pk}, \mathsf{vk}, \mathsf{sk}, x, \sigma^{\mathsf{pk}})$. It sends σ^* to $\mathcal{A}$.

3. Finally, $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is also the output of the experiment.

We say that a homomorphic signature scheme Π_{HS} satisfies *statistical context-hiding against an honest signer* if there exists an efficient simulator $\mathcal{S}$ such that for all (computationally-unbounded) adversaries $\mathcal{A}$,

$$\left|\Pr[\mathsf{Expt}^{\mathsf{ch\text{-}honest}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda,0)=1]-\Pr[\mathsf{Expt}^{\mathsf{ch\text{-}honest}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda,1)=1]\right|=\mathsf{negl}(\lambda).$$

Context-hiding against malicious signers. Typically, context-hiding is defined with respect to an *honest* signer that generates the signing and verification keys using the honest key-generation algorithm KeyGen. However, when constructing *blind homomorphic signatures* (Section 5) with security against *malicious* signers, the assumption that the keys are correctly generated no longer makes sense. Hence, we need a stronger security property that context-hiding holds even if the signing and verification keys for the homomorphic signature scheme are maliciously constructed.

Definition 3.8 does not satisfy this stronger notion of context-hiding because the challenger samples $(\mathsf{vk},\mathsf{sk})$ using the honest KeyGen algorithm, and the simulator is provided the (honestly-generated) signing key sk. The natural way to extend Definition 3.8 to achieve security against malicious signers is to allow the adversary to choose the verification key vk and signing key sk. However, this is too restrictive because the adversary could potentially cook up signatures that verify under vk, and yet, there is no natural notion of a signing key. To circumvent this issue, we introduce a stronger notion of context-hiding that holds against any party with the *capability* to sign messages. More concretely, we require the existence of a simulator that can extract a *simulation trapdoor* td from any *admissible* set of valid message-signature pairs. This trapdoor information td replaces the signing key sk as input to the simulator.

In our construction of homomorphic signatures (Construction 3.11), we say that a pair of messages $(\tilde{x}_0,\tilde{\sigma}_0)$ and $(\tilde{x}_1,\tilde{\sigma}_1)$ is admissible if $\tilde{x}_0\neq\tilde{x}_1$ and $\tilde{\sigma}_0$ and $\tilde{\sigma}_1$ are valid signatures of $\tilde{x}_0$ and $\tilde{x}_1$, respectively (with the *same* public components). Intuitively, our definition captures the fact that context-hiding holds against any signer, as long as they are able to produce or *forge* valid signatures on distinct messages $\tilde{x}_0$ and $\tilde{x}_1$ under some verification key vk. Note that this definition subsumes Definition 3.8, since any signer with an honestly-generated signing key can sign arbitrary messages of its choosing.

Definition 3.9 (Context-Hiding)**.** Fix a security parameter λ. Let $\Pi_{\mathsf{HS}}=(\mathsf{PrmsGen},\mathsf{KeyGen},\mathsf{Sign},\mathsf{PrmsEval},\mathsf{SigEval},\mathsf{Hide},\mathsf{Verify},\mathsf{VerifyFresh},\mathsf{VerifyHide})$ be a decomposable homomorphic signature scheme (Definition 3.7) with message space $\mathcal{X}$, message length ℓ, 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 $\mathcal{X}$. For a bit $b\in\{0,1\}$, a simulator $\mathcal{S}=(\mathcal{S}^{\mathsf{Ext}},\mathcal{S}^{\mathsf{Gen}})$, and an adversary $\mathcal{A}$, we define the *context-hiding* security experiment $\mathsf{Expt}^{\mathsf{ch}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda,b)$ as follows:

1. At the start of the experiment, $\mathcal{A}$ submits a public key pk, a verification key vk, and two message-signature pairs $(\tilde{x}_0,\tilde{\sigma}_0)$, $(\tilde{x}_1,\tilde{\sigma}_1)$ where $\tilde{x}_0,\tilde{x}_1\in\mathcal{X}$ and $\tilde{x}_0\neq\tilde{x}_1$ to the challenger.

2. The challenger parses the signatures as $\tilde{\sigma}_0=(\tilde{\sigma}_0^{\mathsf{pk}},\tilde{\sigma}_0^{\mathsf{m}})$ and $\tilde{\sigma}_1=(\tilde{\sigma}_1^{\mathsf{pk}},\tilde{\sigma}_1^{\mathsf{m}})$, and checks that $\tilde{x}_0\neq\tilde{x}_1$, $\tilde{\sigma}_0^{\mathsf{pk}}=\tilde{\sigma}_1^{\mathsf{pk}}$, and that $\mathsf{Verify}(\mathsf{pk},\mathsf{vk},\tilde{x}_0,\tilde{\sigma}_0)=1=\mathsf{Verify}(\mathsf{pk},\mathsf{vk},\tilde{x}_1,\tilde{\sigma}_1)$. If this is not the case, then the experiment halts with output 0. Otherwise, the challenger invokes the simulator $\mathsf{td}\leftarrow\mathcal{S}^{\mathsf{Ext}}(\mathsf{pk},\mathsf{vk},(\tilde{x}_0,\tilde{\sigma}_0),(\tilde{x}_1,\tilde{\sigma}_1))$.

3. The adversary $\mathcal{A}$ can then submit (adaptive) queries to the challenger where each query consists of a public key pk', a message $x\in\mathcal{X}$, and a signature $\sigma=(\sigma^{\mathsf{pk}},\sigma^{\mathsf{m}})$. For each query,

the challenger checks that $\mathsf{Verify}(\mathsf{pk}', \mathsf{vk}, x, \sigma) = 1$. If this is not the case, then the challenger ignores the query and replies with $\perp$. Otherwise, it proceeds as follows:

- If $b = 0$, the challenger evaluates $\sigma^* \leftarrow \mathsf{Hide}\big(\mathsf{vk}, x, (\sigma^{\mathsf{pk}}, \sigma^{\mathsf{m}})\big)$, and sends σ^* to $\mathcal{A}$.
- If $b = 1$, the challenger computes $\sigma^* \leftarrow \mathcal{S}^{\mathsf{Gen}}(\mathsf{pk}', \mathsf{vk}, \mathsf{td}, x, \sigma^{\mathsf{pk}})$. It provides σ^* to $\mathcal{A}$.

4. Finally, $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$, which is also the output of the experiment.

We say that a homomorphic signature scheme Π_{HS} satisfies *statistical context-hiding* if there exists an efficient simulator $\mathcal{S} = (\mathcal{S}^{\mathsf{Ext}}, \mathcal{S}^{\mathsf{Gen}})$ such that for all (computationally-unbounded) adversaries $\mathcal{A}$,

$$\big|\Pr[\mathsf{Expt}^{\mathsf{ch}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda, 0) = 1] - \Pr[\mathsf{Expt}^{\mathsf{ch}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda, 1) = 1]\big| = \mathsf{negl}(\lambda).$$

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

Definition 3.10 (Compactness)**.** Fix a security parameter λ. 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 $\mathcal{X}$, message length ℓ, and function class $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$, where each $\mathcal{C}_\lambda$ is a collection of Boolean circuits from $\mathcal{X}^\ell$ to $\mathcal{X}$ of depth at most $d = d(\lambda)$. We say that Π_{HS} is *compact* if there exists a universal polynomial $\mathsf{poly}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, messages $\vec{x} \in \mathcal{X}^\ell$, and functions $C \in \mathcal{C}_\lambda$, and setting $\overrightarrow{\mathsf{pk}} \leftarrow \mathsf{PrmsGen}(1^\lambda, 1^\ell)$, $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, $\vec{\sigma} \leftarrow \mathsf{Sign}(\overrightarrow{\mathsf{pk}}, \mathsf{sk}, \vec{x})$, and $\sigma \leftarrow \mathsf{SigEval}(C, \overrightarrow{\mathsf{pk}}, \vec{x}, \vec{\sigma})$, we have that $|\sigma| \le \mathsf{poly}(\lambda, d)$. In particular, the size of the evaluated signature $|\sigma|$ depends only on the depth of the circuit C, and *not* on the message length ℓ.

3.1 Homomorphic Signatures Construction

In this section, we show that the [GVW15] homomorphic signature construction is decomposable in the sense of Definition 3.7 and in addition, satisfies our stronger notion of context-hiding (Definition 3.9). We start with a description of a simpler variant of the [GVW15] construction that satisfies *selective unforgeability* (Remark 3.6), and show that it satisfies context-hiding (against malicious signers). Although it is possible to directly construct a homomorphic signature scheme that satisfies adaptive security, the simpler variant better demonstrates the main ideas of the construction. In Appendix B, we modify the construction and show that it satisfies both adaptive unforgeability and strong context-hiding. (Corollary B.6).

Construction 3.11 (Selectively-Secure Homomorphic Signature [GVW15, adapted])**.** Fix a security parameter λ and a message length $\ell = \mathsf{poly}(\lambda)$. Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ be a function class where each $\mathcal{C}_\lambda$ is a collection of Boolean circuits of depth at most $d = d(\lambda)$ from $\{0,1\}^\ell$ to $\{0,1\}$. In our description, we use lattice trapdoors and the GSW homomorphic operations described in Section 2.1. For lattice parameters n, m, q, and norm bounds β_{ini}, β_{eval}, β_{hide} we construct a decomposable homomorphic signature scheme $\Pi_{\mathsf{HS}} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrmsEval}, \mathsf{SigEval}, \mathsf{Hide}, \mathsf{Verify}, \mathsf{VerifyFresh}, \mathsf{VerifyHide})$ with message space $\mathcal{X} = \{0,1\}$, message length ℓ, and function class $\mathcal{C}$ as follows:

- $\mathsf{PrmsGen}(1^\lambda, 1^\ell) \to \overrightarrow{\mathsf{pk}}$: On input the security parameter λ and the message length ℓ, the parameter-generation algorithm samples matrices $\mathbf{V}_1, \ldots, \mathbf{V}_\ell \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times m}$. It sets $\mathsf{pk}_i = \mathbf{V}_i$ for $i \in [\ell]$ and returns the 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 samples a lattice trapdoor $(\mathbf{A}, \mathsf{td}) \leftarrow \mathsf{TrapGen}(1^\lambda)$. It sets $\mathsf{vk} = \mathbf{A}$ and $\mathsf{sk} = (\mathbf{A}, \mathsf{td})$.

- $\mathsf{Sign}(\mathsf{pk}_i, \mathsf{sk}, x_i) \to \sigma_i$: The signing algorithm computes $\sigma_i^{\mathsf{pk}} \leftarrow \mathsf{SignPK}(\mathsf{pk}_i, \mathsf{sk})$ and $\sigma_i^{\mathsf{m}} \leftarrow \mathsf{SignM}(\mathsf{pk}_i, \mathsf{sk}, x_i, \sigma_i^{\mathsf{pk}})$ where the algorithms SignPK and SignM are defined as follows:

  - $\mathsf{SignPK}(\mathsf{pk}_i, \mathsf{sk}) \to \sigma_i^{\mathsf{pk}}$: The SignPK algorithm outputs the empty string $\sigma_i^{\mathsf{pk}} = \varepsilon$.
  - $\mathsf{SignM}(\mathsf{pk}_i, \mathsf{sk}, x_i, \sigma_i^{\mathsf{pk}}) \to \sigma_i^{\mathsf{m}}$: On input a public key $\mathsf{pk}_i = \mathbf{V}_i$, a signing key $\mathsf{sk} = (\mathbf{A}, \mathsf{td})$, a message $x \in \{0,1\}$, and the public signature component σ_i^{pk}, the SignM algorithm samples a preimage $\mathbf{U}_i \leftarrow \mathsf{SamplePre}(\mathbf{A}, \mathbf{V}_i - x_i \cdot \mathbf{G}, \mathsf{td})$ and outputs $\sigma_i^{\mathsf{m}} = \mathbf{U}_i$.

  Finally, the signing algorithm outputs the signature $\sigma_i = (\sigma_i^{\mathsf{pk}}, \sigma_i^{\mathsf{m}})$.

- $\mathsf{PrmsEval}(C, \overrightarrow{\mathsf{pk}'})) \to \mathsf{pk}_C$: On input a Boolean circuit $C\colon \{0,1\}^\ell \to \{0,1\}$ and a collection of public keys $\overrightarrow{\mathsf{pk}'} = (\mathsf{pk}'_1, \ldots, \mathsf{pk}'_i)$ where $\mathsf{pk}'_i = \mathbf{V}'_i$ for $i \in [\ell]$, the parameter-evaluation algorithm outputs the evaluated public key $\mathsf{pk}_C = \mathbf{V}_C \leftarrow \mathsf{EvalPK}(\mathbf{V}'_1, \ldots, \mathbf{V}'_\ell, C)$.

- $\mathsf{SigEval}(C, \overrightarrow{\mathsf{pk}'}, \vec{x}, \vec{\sigma}) \to \sigma$: The signature-evaluation algorithm first parses $\vec{\sigma} = (\vec{\sigma}^{\mathsf{pk}}, \vec{\sigma}^{\mathsf{m}})$. Then, it computes $\sigma^{\mathsf{pk}} \leftarrow \mathsf{SigEvalPK}(C, \overrightarrow{\mathsf{pk}'}, \vec{\sigma}^{\mathsf{pk}})$ and $\sigma^{\mathsf{m}} \leftarrow \mathsf{SigEvalM}(C, \overrightarrow{\mathsf{pk}'}, \vec{x}, \vec{\sigma})$, where the algorithms $\mathsf{SigEvalPK}$ and $\mathsf{SigEvalM}$ are defined as follows:

  - $\mathsf{SigEvalPK}(C, \overrightarrow{\mathsf{pk}'}, \vec{\sigma}^{\mathsf{pk}}) \to \sigma^{\mathsf{pk}}$: The $\mathsf{SigEvalPK}$ algorithm outputs the empty string $\sigma^{\mathsf{pk}} = \varepsilon$.
  - $\mathsf{SigEvalM}(C, \overrightarrow{\mathsf{pk}'}, \vec{x}, \vec{\sigma}) \to \sigma^{\mathsf{m}}$: On input a Boolean circuit $C\colon \{0,1\}^\ell \to \{0,1\}$, a set of public keys $\overrightarrow{\mathsf{pk}'} = (\mathsf{pk}'_1, \ldots, \mathsf{pk}'_\ell)$, messages $\vec{x} = (x_1, \ldots, x_\ell)$, and signatures $\vec{\sigma} = (\sigma_1, \ldots, \sigma_\ell)$, the $\mathsf{SigEvalM}$ algorithm first parses $\mathsf{pk}'_i = \mathbf{V}'_i$ and $\sigma_i = (\sigma_i^{\mathsf{pk}}, \sigma_i^{\mathsf{m}}) = (\varepsilon, \mathbf{U}_i)$ for all $i \in [\ell]$. Then, it outputs $\sigma^{\mathsf{m}} = \mathbf{U}_C \leftarrow \mathsf{EvalU}\big((\mathbf{V}'_1, x_1, \mathbf{U}_1), \ldots, (\mathbf{V}'_\ell, x_\ell, \mathbf{U}_\ell), C\big)$.

  Finally, it outputs the signature $\sigma = (\sigma^{\mathsf{pk}}, \sigma^{\mathsf{m}})$.

- $\mathsf{Hide}(\mathsf{vk}, x, \sigma) \to \sigma^*$: On input a verification key $\mathsf{vk} = \mathbf{A}$, a message $x \in \{0,1\}$, and a signature $\sigma = (\varepsilon, \mathbf{U})$, the hide algorithm samples and outputs a signature

$$\sigma^* = \mathbf{u} \leftarrow \mathsf{SampleRight}(\mathbf{A}, \mathbf{AU} + (2x-1) \cdot \mathbf{G}, \mathbf{U}, \mathbf{0}, \beta_{\mathsf{hide}}).$$

- $\mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, x, \sigma) \to \{0,1\}$: On input a public key $\mathsf{pk} = \mathbf{V}$, a verification key $\mathsf{vk} = \mathbf{A}$, a message $x \in \mathcal{X}$, and a signature $\sigma = (\varepsilon, \mathbf{U})$, the verification algorithm first checks if $\mathbf{A}$ is a rank-n matrix and outputs 0 if this is the case. Then, it outputs 1 if $\|\mathbf{U}\| \leq \beta_{\mathsf{eval}}$ and $\mathbf{AU} + x \cdot \mathbf{G} = \mathbf{V}$ and 0 otherwise.

- $\mathsf{VerifyFresh}(\mathsf{pk}, \mathsf{vk}, x, \sigma) \to \{0,1\}$: On input a public key $\mathsf{pk} = \mathbf{V}$, a verification key $\mathsf{vk} = \mathbf{A}$, a message $x \in \mathcal{X}$ and a signature $\sigma = (\varepsilon, \mathbf{U})$, the fresh verification algorithm first checks if $\mathbf{A}$ is a rank-n matrix and outputs 0 if this is the case. Then, it outputs 1 if $\|\mathbf{U}\| \leq \beta_{\mathsf{ini}}$ and $\mathbf{AU} + x \cdot \mathbf{G} = \mathbf{V}$ and 0 otherwise.

- $\mathsf{VerifyHide}(\mathsf{pk}_C, \mathsf{vk}, x, \sigma^*) \to \{0,1\}$: On input a public key $\mathsf{pk}_C = \mathbf{V}$, a verification key $\mathsf{vk} = \mathbf{A}$, a message $x \in \{0,1\}$, and a signature $\sigma^* = \mathbf{u}$, the hide-verification algorithm first checks if $\mathbf{A}$ is a rank-n matrix and outputs 0 if this is the case. Then, it checks that $\|\mathbf{u}\| \leq \beta_{\mathsf{hide}}$ and that $[\mathbf{A} \mid \mathbf{V} + (x-1) \cdot \mathbf{G}] \cdot \mathbf{u} = \mathbf{0}$, and accepts if both of these conditions hold. Otherwise, it rejects.

We now state and prove the correctness and security theorems for Construction 3.11.

Theorem 3.12 (Correctness)**.** *Fix a security parameter λ, lattice parameters n, m, q, norm bounds $\beta_{\mathsf{ini}}, \beta_{\mathsf{eval}}, \beta_{\mathsf{hide}}$, and a depth bound d. Suppose $m = O(n \log q)$, $\beta_{\mathsf{ini}} \geq O(n\sqrt{\log q})$, $\beta_{\mathsf{eval}} \geq \beta_{\mathsf{ini}} \cdot 2^{\tilde{O}(d)}$, $\beta_{\mathsf{hide}} \geq \beta_{\mathsf{eval}} \cdot \omega(m\sqrt{\log m})$, and $q \geq \beta_{\mathsf{hide}}$. Then, Π_{HS} from Construction 3.11 satisfies signing correctness (Definition 3.2), evaluation correctness (Definition 3.3), and hiding correctness (Definition 3.4).*

Proof. Signing correctness follows from Theorem 2.3, evaluation correctness follows from Theorem 2.5, and hiding correctness follows from Theorem 2.4. □

Theorem 3.13 (Unforgeability)**.** *Fix a security parameter λ, lattice parameters n, m, q, norm bounds $\beta_{\mathsf{ini}}, \beta_{\mathsf{eval}}, \beta_{\mathsf{hide}}$, and a depth bound d. Suppose $m = O(n \log q)$. Then, under the $\mathsf{SIS}(n, m, q, \beta_{\mathsf{eval}})$ assumption, Π_{HS} in Construction 3.11 satisfies selective unforgeability (Definition 3.5, Remark 3.6).*

Proof. Follows from [GVW15, §6]. □

Theorem 3.14 (Context-Hiding)**.** *Fix a security parameter λ, lattice parameters n, m, q, norm bounds $\beta_{\mathsf{ini}}, \beta_{\mathsf{eval}}, \beta_{\mathsf{hide}}$, and a depth bound d. Suppose $m = O(n \log q)$, $\beta_{\mathsf{hide}} \geq 2 \cdot \beta_{\mathsf{eval}} \cdot \omega(m\sqrt{\log m})$, and $q \geq \beta_{\mathsf{hide}}$. Then, Π_{HS} in Construction 3.11 satisfies context-hiding security (Definition 3.9).*

Proof of Theorem 3.14. We construct a simulator $\mathcal{S} = (\mathcal{S}^{\mathsf{Ext}}, \mathcal{S}^{\mathsf{Gen}})$ as follows:

- $\mathcal{S}^{\mathsf{Ext}}(\mathsf{pk}, \mathsf{vk}, (\tilde{x}_0, \tilde{\sigma}_0), (\tilde{x}_1, \tilde{\sigma}_1))$: On input a public key pk, a verification key vk, and two message-signature pairs $(\tilde{x}_0, \tilde{\sigma}_0)$, $(\tilde{x}_1, \tilde{\sigma}_1)$, the simulator first parses $\tilde{\sigma}_0 = (\varepsilon, \tilde{\mathbf{U}}_0)$, $\tilde{\sigma}_1 = (\varepsilon, \tilde{\mathbf{U}}_1)$, and then outputs the simulation trapdoor $\mathsf{td} = \tilde{\mathbf{U}}_0 - \tilde{\mathbf{U}}_1$.
- $\mathcal{S}^{\mathsf{Gen}}(\mathsf{pk}, \mathsf{vk}, \mathsf{td}, x, \sigma^{\mathsf{pk}})$: On input a public key $\mathsf{pk} = \mathbf{V}$, a verification key $\mathsf{vk} = \mathbf{A}$, a trapdoor $\mathsf{td} = \tilde{\mathbf{U}}$, a message $x \in \{0,1\}$, and a message-independent component σ^{pk}, the simulator computes $\mathbf{u} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{V} + (x-1) \cdot \mathbf{G}, \tilde{\mathbf{U}}, \mathbf{0}, \beta^*)$, and returns $\mathbf{u}$.

We now show that for any adversary $\mathcal{A}$, the experiments $\mathsf{Expt}^{\mathsf{ch}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda, 0)$ and $\mathsf{Expt}^{\mathsf{ch}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda, 1)$ are statistically indistinguishable. Consider the context-hiding experiment:

- Let $\mathsf{pk} = \mathbf{V}$, $\mathsf{vk} = \mathbf{A}$, and $(\tilde{x}_0, \tilde{\sigma}_0)$, $(\tilde{x}_1, \tilde{\sigma}_1)$ be the values that $\mathcal{A}$ sends to the challenger. Write $\tilde{\sigma}_0 = (\varepsilon, \tilde{\mathbf{U}}_0)$ and $\tilde{\sigma}_1 = (\varepsilon, \tilde{\mathbf{U}}_1)$. Without loss of generality, we can assume that $\mathbf{A}$ is a rank-n matrix, $\tilde{x}_0 \neq \tilde{x}_1$, and $\mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, \tilde{x}_0, \tilde{\sigma}_0) = 1 = \mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, \tilde{x}_1, \tilde{\sigma}_1)$. Otherwise, the output is always 0 in both experiments. Since $\tilde{x}_0, \tilde{x}_1 \in \{0,1\}$ and $\tilde{x}_0 \neq \tilde{x}_1$, we can assume without loss of generality that $\tilde{x}_0 = 0$ and $\tilde{x}_1 = 1$. Moreover, since $\tilde{\sigma}_0$ and $\tilde{\sigma}_1$ are valid signatures, $\|\tilde{\mathbf{U}}_0\|, \|\tilde{\mathbf{U}}_1\| \leq \beta_{\mathsf{eval}}$. This means that $\tilde{\mathbf{U}} = \tilde{\mathbf{U}}_0 - \tilde{\mathbf{U}}_1$ has bounded norm $\|\tilde{\mathbf{U}}\| \leq 2 \cdot \beta_{\mathsf{eval}}$, and moreover, that $\mathbf{A}\tilde{\mathbf{U}} = \mathbf{G}$, so $\mathsf{td} = \tilde{\mathbf{U}}$ is a $\mathbf{G}$-trapdoor for $\mathbf{A}$ (Theorem 2.4).
- Let $\mathsf{pk}' = \mathbf{V}'$, $x \in \{0,1\}$, $\sigma = (\varepsilon, \mathbf{U})$ be a query that $\mathcal{A}$ makes to the challenger. If $\mathbf{AU} + x \cdot \mathbf{G} \neq \mathbf{V}'$ or $\|\mathbf{U}\| > \beta_{\mathsf{eval}}$, then the challenger ignores the query (and replies with $\perp$) in both experiments. Therefore, assume that $\mathbf{AU} + x \cdot \mathbf{G} = \mathbf{V}'$ and $\|\mathbf{U}\| \leq \beta_{\mathsf{eval}}$. Then the challenger proceeds as follows:

– In $\mathsf{Expt}^{\mathsf{ch}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda, 0)$, the challenger's response is $\mathbf{u} \leftarrow \mathsf{SampleRight}(\mathbf{A}, \mathbf{AU} + (2x-1)\cdot \mathbf{G}, \mathbf{U}, \mathbf{0}, \beta_{\mathsf{hide}})$.

– In $\mathsf{Expt}^{\mathsf{ch}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda, 1)$, the challenger responds with $\mathbf{u} \leftarrow \mathcal{S}^{\mathsf{Gen}}(\mathsf{pk}', \mathsf{vk}, \mathsf{td}, x, \sigma^{\mathsf{pk}})$, which is equivalent to $\mathbf{u} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{V}' + (x-1)\cdot \mathbf{G}, \tilde{\mathbf{U}}, \mathbf{0}, \beta_{\mathsf{hide}})$.

Since $\mathbf{V}' + (x-1)\cdot\mathbf{G} = \mathbf{AU} + (2x-1)\cdot\mathbf{G}$, by Theorem 2.4, as long as $\max(\|\mathbf{U}\|, \|\tilde{\mathbf{U}}\|)\cdot\omega(m\sqrt{\log m}) \leq \beta_{\mathsf{hide}} \leq q$ the challenger's responses to all of the queries in the two experiments are statistically indistinguishable. From above, $\|\tilde{\mathbf{U}}\| \leq 2\cdot\beta_{\mathsf{eval}}$ and $\|\mathbf{U}\| \leq \beta_{\mathsf{eval}}$, and the claim follows. □

Remark 3.15 (Weak Context-Hiding)**.** Theorem 3.14 implies that the homomorphic signature scheme Π_{HS} in Construction 3.11 also satisfies context-hiding security against honest signers (Definition 3.8). Specifically, Theorem 3.14 guarantees the existence of a simulator $\mathcal{S} = (\mathcal{S}^{\mathsf{Ext}}, \mathcal{S}^{\mathsf{Gen}})$ that can be used to simulate the signatures generated by the Hide algorithm. In the context-hiding security game against honest signers, the signing key sk and verification key vk are generated honestly, and the context-hiding simulator $\mathcal{S}_{\mathsf{hon}}$ is given both vk and sk. Given sk, the simulator $\mathcal{S}_{\mathsf{hon}}$ can choose an arbitrary public key $\mathsf{pk} = \mathbf{V} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n\times m}$, and construct honest signatures $\tilde{\sigma}_0 \leftarrow \mathsf{Sign}(\mathsf{pk}, \mathsf{sk}, 0)$ and $\tilde{\sigma}_1 \leftarrow \mathsf{Sign}(\mathsf{pk}, \mathsf{sk}, 1)$. Simulator $\mathcal{S}_{\mathsf{hon}}$ can then invoke $\mathcal{S}^{\mathsf{Ext}}$ on $(\mathsf{pk}, \mathsf{vk}, (0, \tilde{\sigma}_0), (1, \tilde{\sigma}_1))$ to obtain the simulation trapdoor td, and then use $\mathcal{S}^{\mathsf{Gen}}$ to simulate the Hide algorithm. Thus, we can construct a simulator $\mathcal{S}_{\mathsf{hon}}$ for the weak context-hiding security game using the simulator $\mathcal{S}$ guaranteed by Theorem 3.14.

Theorem 3.16 (Compactness)**.** *Fix a security parameter λ, lattice parameters n, m, q, norm bounds $\beta_{\mathsf{ini}}, \beta_{\mathsf{eval}}, \beta_{\mathsf{hide}}$, and a depth bound d. Suppose $n = \mathsf{poly}(\lambda)$, $m = O(n \log q)$, and $q = 2^{\mathsf{poly}(\lambda, d)}$. Then, Π_{HS} in Construction 3.11 satisfies compactness (Definition 3.10).*

Proof. Follows from Theorem 2.5. Specifically, the signature output by $\mathsf{SigEval}$ is a matrix $\mathbf{U} \in \mathbb{Z}_q^{m\times m}$ which has size $m^2 \log q = \mathsf{poly}(\lambda, d)$. □

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. [DMP88], 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 [LS90, Dam92, IKOS09] or oblivious transfer [KMO89]. 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 4.11.

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 (Construction 3.11) with Fact 2.2, 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. Finally, in Remark 4.10, we describe how to obtain a NIZK *proof* system in the preprocessing model using a simple variant of the construction.

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 $f_{x,\mathsf{ct}}$.[7]

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 $\vec{\sigma}_k \leftarrow \mathsf{Sign}(\overrightarrow{\mathsf{pk}}_{\mathsf{HS}}, \mathsf{sk}_{\mathsf{HS}}, k_{\mathsf{SE}})$ and output
$$k_P = (k_{\mathsf{SE}}, \overrightarrow{\mathsf{pk}}_{\mathsf{HS}}, \mathsf{vk}_{\mathsf{HS}}, \vec{\sigma}_k) \quad \text{and} \quad k_V = (\overrightarrow{\mathsf{pk}}_{\mathsf{HS}}, \mathsf{vk}_{\mathsf{HS}}, \mathsf{sk}_{\mathsf{HS}}).$$
- $\mathsf{Prove}(k_P, x, w) \to \pi$: If $\mathcal{R}(x, w) = 0$, output $\perp$. Otherwise, parse $k_P = (k_{\mathsf{SE}}, \overrightarrow{\mathsf{pk}}_{\mathsf{HS}}, \mathsf{vk}_{\mathsf{HS}}, \vec{\sigma}_k)$. 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 $\sigma'_{x,\mathsf{ct}} \leftarrow \mathsf{SigEval}(C_{x,\mathsf{ct}}, \overrightarrow{\mathsf{pk}}_{\mathsf{HS}}, k_{\mathsf{SE}}, \vec{\sigma}_k)$ 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 (Definition 3.3), hiding correctness (Definition 3.4), selective unforgeability (Definition 3.5, Remark 3.6), and context-hiding against honest signers (Definition 3.8), then Π_{NIZK} is a NIZK argument for $\mathcal{R}$ in the preprocessing model.*

We give the proof of Theorem 4.4 in Appendix C. Combining Construction 4.3 with the homomorphic signature construction Π_{HS} from Construction 3.11 and any LWE-based CPA-secure encryption scheme (Fact 2.2), we have the following corollary.

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

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

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 (Argument Length Approaching the Witness Size)**.** The proofs in our preprocessing NIZK argument from Construction 4.3 consists of an encryption ct of the witness and a homomorphic signature σ with respect to a circuit C that implements the decryption function of the encryption scheme and the NP relation $\mathcal{R}$. Suppose the relation $\mathcal{R}$ can be implemented by a Boolean circuit of depth d. Using CPA-secure encryption with additive overhead (Fact 2.2), $|\mathsf{ct}| = |w| + \mathsf{poly}(\lambda)$, where $|w|$ is the length of a witness to $\mathcal{R}$. If the homomorphic signature is compact (Definition 3.10), then $|\sigma| = \mathsf{poly}(\lambda, d')$ where d' is a bound on the depth of the circuit C. Since the decryption function can be implemented by a circuit of depth $\mathsf{poly}(\lambda)$, we have that $d' = \mathsf{poly}(d, \lambda)$. This means that the overall size of the arguments in our candidate is $|w| + \mathsf{poly}(\lambda, d)$. The overhead (on top of the NP witness) is *additive* in the security parameter and the *depth* of the NP relation. This is asymptotically shorter than the length of the proofs in NIZK constructions based on trapdoor permutations or pairings [FLS90, DDO+01, GOS06, Gro10], where the dependence is on the *size* of the circuit computing $\mathcal{R}$, and the overhead is *multiplicative* in the security parameter. Thus, our NIZK candidate gives a construction where the argument size approaches the *witness length*. Previously, Gentry et al. [GGI+15] gave a generic way to achieve these asymptotics by combining NIZKs with FHE. The advantage of our approach is that we only rely on lattice assumptions, while the Gentry et al. [GGI+15] compiler additionally assumes the existence of a NIZK scheme (which prior to this work, did not follow from standard lattice assumptions).

Remark 4.8 (Arguments with Common Witness)**.** The proofs in our preprocessing NIZK arguments from Construction 4.3 consists of an encryption of the witness together with a signature. This means that if the prover uses the same *witness* to prove multiple (distinct) statements, then the prover does not need to include a fresh encryption of its witness with every proof. It can send the encrypted witness once and then give multiple signatures with respect to the *same* encrypted witness. In particular, if a prover uses the same witness w to prove m statements, the total size of the proof is $|w| + m \cdot \mathsf{poly}(\lambda, d)$, where d is a bound on the depth of the (possibly different) NP relation associated with the m statements. Effectively, the additional overhead of proving multiple statements using a common witness is *independent* of the witness size, and thus, the cost of transmitting the encrypted witness can be *amortized* across multiple proofs. We leverage this observation to implement a *succinct* version of the classic Goldreich-Micali-Wigderson compiler [GMW86, GMW87] in Section 6.1. We note that this amortization is also possible if we first apply the FHE-based transformation of

Gentry et al. [GGI+15] to any NIZK construction. In our case, our NIZK candidate naturally satisfies this property.

Remark 4.9 (Preprocessing NIZKs from Homomorphic MACs)**.** We note that we can also instantiate Construction 4.3 with a *homomorphic MAC* [GW13, CF13, CFGN14, Cat14] to obtain a multi-theorem preprocessing NIZK. Although the resulting NIZK will not be publicly verifiable (Remark 4.6), a homomorphic MAC is a simpler cryptographic primitive that may be easier to construct, and thus, enable new constructions of multi-theorem preprocessing NIZKs from weaker assumptions. Many existing constructions of homomorphic MACs do not satisfy all of the necessary properties: the lattice-based construction in [GW13] is only secure against adversaries that can make a *bounded* number of verification queries, while the construction based on one-way functions in [CF13] does not provide context-hiding. The construction based on the ℓ-Diffie-Hellman inversion assumption [CF13, CFGN14] gives a context-hiding homomorphic MAC for bounded-degree polynomials (which suffices for verifying NC^1 computations)—here, $\ell = \mathsf{poly}(\lambda)$ is a parameter that scales with the *degree* of the computation being verified. Together with a group-based PRF with evaluation in NC^1 (e.g., the Naor-Reingold PRF [NR97]), we can use Construction 4.3 to obtain a preprocessing NIZK for general NP languages from the ℓ-Diffie-Hellman inversion assumption.[8] We leave it as an interesting open problem to build context-hiding homomorphic MACs that suffice for preprocessing NIZKs from weaker (and static) cryptographic assumptions (e.g., the DDH assumption).

Remark 4.10 (Preprocessing NIZK Proofs from Extractable Homomorphic Commitments)**.** Construction 4.3 gives a NIZK *argument* in the preprocessing model. This is because in the proof of Theorem 4.4, soundness of the preprocessing NIZK reduces to *computational* unforgeability of the underlying homomorphic signature scheme. We can modify Construction 4.3 to obtain a NIZK *proof* by substituting a context-hiding *statistically-binding* homomorphic commitment in place of the homomorphic signature. This means that the homomorphic commitment scheme satisfies "statistical unforgeability:" a *computationally-unbounded* adversary cannot take a commitment to a message x and open it to a commitment on any value $y \neq f(x)$ with respect to the function f. Then, the resulting preprocessing NIZK achieves statistical soundness. We can instantiate the statistically-binding homomorphic commitment using the extractable homomorphic trapdoor function from Gorbunov et al. [GVW15, Appendix B]. The specific construction is a variant of the Gorbunov et al. homomorphic signature scheme (Construction 3.11), where the public verification key $\mathsf{vk} = \mathbf{A}$ is chosen to be a public key of the GSW fully homomorphic encryption [GSW13] scheme.

While homomorphic commitments enable a preprocessing NIZK proof system, it is unclear how to efficiently implement the preprocessing step without relying on general-purpose MPC. In contrast, instantiating Construction 4.3 using homomorphic signatures yields a scheme where the preprocessing can be implemented directly using oblivious transfer (and does not require non-black-box use of the homomorphic signature scheme). For this reason, we focus on preprocessing NIZK arguments from homomorphic signatures in the remainder of this paper.

Remark 4.11 (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

[8] For this construction, we require that the NP relation can be implemented by an NC^1 circuit. While any NP relation can be represented as a constant-depth circuit (by including the intermediate wires of the NP circuit as part of the witness), the length of the preprocessing NIZK is now proportional to the *circuit size*, rather than the size of the witness.

several main families of assumptions known to imply NIZKs: number-theoretic conjectures such as quadratic residuosity [BFM88, DMP87, BDMP91],[9] trapdoor permutations [FLS90, DDO+01, Gro10], pairings [GOS06], or indistinguishability obfuscation [SW14]. In the designated-verifier setting, constructions are also known from additively homomorphic encryption [CD04, DFN06, CG15]. A number of works have also studied NIZKs in the preprocessing model, and several constructions have been proposed from one-way functions [DMP88, LS90, Dam92, IKOS09] and oblivious transfer [KMO89]. Since lattice-based assumptions imply one-way functions [Ajt96, Reg05], oblivious transfer [PVW08], and homomorphic encryption [Reg05, Gen09], 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 [DMP88, LS90, Dam92] 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 [DMP88, LS90, Dam92] do not directly give a multi-theorem NIZK.

  A natural question to ask is whether we can use the transformation by Feige et al. [FLS90] 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 [FLS90] 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.[10] 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 [DMP88, LS90, Dam92], the proving key must be kept private for zero-knowledge. Consequently, the preprocessing NIZKs of [DMP88, LS90, Dam92] do not give a general multi-theorem NIZK in the preprocessing model.

- The (preprocessing) NIZK constructions based on oblivious transfer [KMO89], the "MPC-in-the-head" paradigm [IKOS09], and the ones based on homomorphic encryption [CD04, DFN06, CG15] 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 [KMO89], 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 [IKOS09].

[9] Some of these schemes [BFM88, DMP87] 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.

[10] At a high-level, the proof in [FLS90] 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.

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.

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., [LP07]) 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 [Can01]. 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 Appendix A 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 [Can04] and the ideal blind signature functionality $\mathcal{F}_{\text{BLSIG}}$ by Fischlin [Fis06]. 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 [Can04, Fis06], it suffices for the applications we describe in this work.

Ideal signature functionalities. The $\mathcal{F}_{\text{SIG}}$ functionality from [Can04] 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 [Fis06] 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.[11] 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 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.[12] 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.

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

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

Preventing selective failures. In our definition, the functionalities $\mathsf{IdealSign}$ and $\mathsf{IdealEval}$ 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 [GVW15] (which we leverage in our construction) do 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 [GVW15] 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.

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.

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 $\mathcal{L}$.

Figure 1: The $\mathcal{F}_{\text{BHS}}$ functionality.

Protocol Π_{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 $(\vec{w}_1, \ldots, \vec{w}_t) \mapsto \bigoplus_{i\in[t]} \vec{w}_i$. Let $\Pi_{\mathsf{HS}} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrmsEval}, \mathsf{SigEval}, \mathsf{Hide}, \mathsf{Verify}, \mathsf{VerifyFresh}, \mathsf{VerifyHide})$ 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 Π_{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 $(\mathsf{sid}, \mathsf{sign}, \mathsf{vk}, \vec{x})$, where $\mathsf{vk} = (\overrightarrow{\mathsf{pk}}, \mathsf{vk}')$ and $\vec{x} \in \{0,1\}^\ell$, the receiver chooses t shares $\vec{w}_1, \ldots, \vec{w}_t \xleftarrow{\text{R}} \{0,1\}^\ell$ where $\bigoplus_{i\in[t]} \vec{w}_i = \vec{x}$. Then, for each $i \in [t]$, it sends $\big((\mathsf{sid}, i), \mathsf{receiver}, \vec{w}_i\big)$ 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 $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \vec{x}), \bot)$.
- **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 $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \vec{x}), \bot)$. Otherwise, it evaluates $\vec{\sigma} \leftarrow \mathsf{SigEval}\big(f_{\mathsf{recon}}, \overrightarrow{\mathsf{pk}}, (\vec{w}_1, \ldots, \vec{w}_t), (\vec{\sigma}_1, \ldots, \vec{\sigma}_t)\big)$, where $\vec{\sigma}_i = (\tilde{\sigma}_{i,1}, \ldots, \tilde{\sigma}_{i,\ell})$ for all $i \in [t]$. The receiver also sets the ready flag and outputs $\big(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \vec{x}), \vec{\sigma}\big)$.

Signature Verification: Upon receiving an input $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}, (f, \vec{x}), \vec{\sigma})$ 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 $t \leftarrow \mathsf{Verify}(\mathsf{pk}_f, \mathsf{vk}', \vec{x}, \vec{\sigma})$, and if $f \neq f_{\mathsf{id}}$, it sets $t \leftarrow \mathsf{VerifyHide}(\mathsf{pk}_f, \mathsf{vk}', \vec{x}, \vec{\sigma})$. It outputs $(\mathsf{sid}, \mathsf{verified}, \vec{x}, \vec{\sigma}, t)$.

Signature Evaluation: If the ready flag has not been set, then ignore the request. Otherwise, upon receiving an input $(\mathsf{sid}, \mathsf{eval}, \mathsf{vk}, g, (f, \vec{x}), \vec{\sigma})$, 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 $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}, (f, \vec{x}), \vec{\sigma})$ (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 $\sigma' \leftarrow \mathsf{SigEval}(g, \mathsf{pk}_{\mathsf{recon}}, \vec{x}, \vec{\sigma})$, and $\sigma^* \leftarrow \mathsf{Hide}(\mathsf{vk}', g(\vec{x}), \sigma')$. It outputs $(\mathsf{sid}, \mathsf{signature}, (g, g(\vec{x})), \sigma^*)$.

Figure 2: The Π_{BHS} protocol.

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 (Figure 6).[13] 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 [CLOS02]. 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 $\vec{x}$. 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 $\vec{x}$ into a number of random shares $\vec{w}_1, \ldots, \vec{w}_t$ where $\vec{x} = \bigoplus_{i \in [t]} \vec{w}_i$. Instead of asking for a signature on $\vec{x}$ directly, it instead asks for a signature on the shares $\vec{w}_1, \ldots, \vec{w}_t$. Since the signatures on the shares $\vec{w}_1, \ldots, \vec{w}_t$ are homomorphic, the receiver can still compute a signature on the original message $\vec{x}$ and hence, correctness of signing is preserved. Moreover, as we show in the proof of Theorem 5.1, unless the malicious signer correctly guesses *all* of the shares of $\vec{w}_1, \ldots, \vec{w}_t$ the receiver chose, the probability that the receiver aborts (due to receiving an invalid signature) is *independent* of $\vec{x}$ 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 Appendix D.

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 (Definitions 3.2, 3.3, and 3.4), unforgeability (Definition 3.5), and context-hiding (Definition 3.9). 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 from Construction B.1 (based on [GVW15]) with the lattice-based UC-secure oblivious transfer protocol from [PVW08], we obtain a blind homomorphic signature scheme from standard lattice assumptions. We describe our instantiation below.

Fact 5.2 (Oblivious Transfer from LWE [PVW08])**.** 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 $\mathcal{F}_{\text{OT}}^{\ell,s}$ (Figure 6) in the presence of malicious adversaries

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

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.

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 Π'_{BHS} that securely realizes $\mathcal{F}_{\mathrm{BHS}}$ 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 Π'_{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 Π_{BHS} be the protocol from Figure 2 instantiated with the homomorphic signature scheme from Construction B.1. By Theorem 5.1 and Corollary B.6,[14] protocol Π_{BHS} securely realizes $\mathcal{F}_{\mathrm{BHS}}$ in the $\mathcal{F}_{\mathrm{OT}}^{\ell,s}$-hybrid model, for some $\ell, s = \mathsf{poly}(\lambda)$. We let Π'_{BHS} be the protocol obtained by instantiating the functionality $\mathcal{F}_{\mathrm{OT}}^{\ell,s}$ in Π_{BHS} with the protocol from Fact 5.2. Security of Π'_{BHS} then follows from the universal composition theorem (Theorem A.2) [Can01]. Key generation, signature verification, and signature evaluation in Π'_{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 Π_{HS}. The blind signing protocol reduces to a single invocation of $\mathcal{F}_{\mathrm{OT}}^{\ell,s}$, which by Fact 5.2, can be implemented by just two rounds of interaction. □

Remark 5.4 (Size of CRS in Corollary 5.3)**.** In the lattice-based OT construction of [PVW08], a single CRS can only be used for a bounded number of OTs. The blind signing protocol in Π'_{BHS} from Corollary 5.3 requires $\ell \cdot \mathsf{poly}(\lambda)$ invocations of OT, where ℓ is the message length. Thus, instantiating Π'_{BHS} requires a CRS of length $\mathsf{poly}(\ell, \lambda)$. In our preprocessing UC-NIZK (Section 6), $\ell = \mathsf{poly}(\lambda)$, and so a CRS of size $\mathsf{poly}(\lambda)$ suffices to obtain a preprocessing UC-NIZK for general NP languages. It is an open problem to build a lattice-based UC-secure OT protocol in the CRS model with a *reusable* CRS.

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 Π_{ZK} in the $\mathcal{F}_{\mathrm{BHS}}$-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 Appendix E.

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 $\mathcal{F}_{\mathrm{BHS}}$-hybrid model.*

[14] Note that we are using the fact that hardness of LWE also implies hardness of SIS (with corresponding parameters).

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

Let λ be a security parameter and $\Pi_{\mathsf{SE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ 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 $\sigma_{\mathsf{sk}} = \bot$, 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.

Figure 3: Preprocessing ZK argument 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 Fact 2.2. 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 (Theorem A.2). 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 $m + \mathsf{poly}(\lambda, d)$. □

6.1 Applications to MPC

In this section, we describe several applications of our preprocessing UC-NIZKs to boosting the security of MPC protocols. First, we show that combining our construction with the round-optimal, semi-malicious MPC protocol of Mukherjee-Wichs [MW16] 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 (Remark 6.5). Then, we show that by leveraging the observation in Remark 4.8, we obtain a *succinct* version of the GMW [GMW86, GMW87] compiler from lattice assumptions.

Malicious-secure MPC from lattices. Previously, Mukherjee and Wichs showed how to construct a two-round MPC protocol with UC-security against semi-malicious adversaries from standard lattice assumptions [MW16]. Their protocol has several notable properties, including optimal round complexity and near-optimal communication complexity: namely, the total communication between the parties depends only on the *length* of the parties' inputs and outputs, and *not* on the complexity (i.e., circuit size) of the underlying computation. Achieving this latter property is often referred to as breaking the "circuit-size barrier" for secure computation [BGI16].

The Mukherjee-Wichs construction (as well as its predecessor [AJL+12]) achieve security against semi-malicious adversaries, and rely on general-purpose NIZKs to achieve full security against malicious adversaries without increasing the round complexity. However, since NIZKs are not known to follow from standard lattice assumptions in the CRS model, the security of the malicious-secure protocols cannot be reduced to a single set of hardness assumptions (for instance, we need to combine lattice assumptions with other number-theoretic assumptions).

Using our lattice-based preprocessing NIZKs, we can obtain malicious-secure MPC in a preprocessing model while basing security *exclusively* on standard lattice assumptions. Specifically, in the preprocessing step, the parties would execute the preprocessing protocol of our UC-NIZK construction (Figure 3, Corollary 6.2). In the online phase of the protocol, the parties essentially have access to an ideal zero-knowledge functionality, and so, we can apply the same semi-malicious to malicious boosting described in [AJL+12, MW16] to obtain a protocol with full malicious security. The round complexity and communication complexity of the online phase of the protocol is unchanged from that of the Mukherjee-Wichs construction. Moreover, our preprocessing protocol has several appealing properties: it is not only independent of the party's inputs, but it is also (almost) independent of the computation being performed (it depends only *polylogarithmically* on the depth of the online computation). This means that the same preprocessing can in fact be *reused* across

many protocol executions, provided that the computations have bounded depth. In fact we can make the preprocessing completely independent of the online computation if we make an additional circular security assumption (c.f., Corollary 6.4). We state our conclusions more precisely below:

Fact 6.3 (MPC from Multi-Key FHE [MW16])**.** Let λ be a security parameter, and $f\colon (\{0,1\}^{\ell_{\text{in}}})^n \to (\{0,1\}^{\ell_{\text{out}}})^n$ be an arbitrary n-input function. Let C_f be the circuit that computes f, and let d_f be its depth. Then, under the LWE assumption, there exists a protocol Π_f that securely realizes $\mathcal{F}_f$ in the presence of (static) *semi-malicious* adversaries in the CRS model and assuming the parties have access to an authenticated broadcast channel. Recall that $\mathcal{F}_f$ is the general UC functionality for computing the function f (Figure 8). Moreover, the protocol Π_f satisfies the following properties:

- **Optimal round complexity:** The protocol Π_f is a two-round protocol.
- **Low communication complexity:** The total communication complexity of the protocol is $(\ell_{\text{in}} + \ell_{\text{out}}) \cdot \mathsf{poly}(\lambda, n, d_f)$. In other words, the total communication depends only on the security parameter, the length of the inputs, the length of the outputs, and the *depth* of the computation (rather than the size $|C_f|$). Moreover, if we make an additional *circular security* assumption, then the total communication complexity becomes $(\ell_{\text{in}} + \ell_{\text{out}}) \cdot \mathsf{poly}(\lambda, n)$, which is completely *independent* of the complexity of the computation f. This is essentially the best we can hope for from an MPC protocol for $\mathcal{F}_f$.

Corollary 6.4 (Malicious-Secure MPC in the Preprocessing Model from Lattices)**.** *Let λ be a security parameter, and let $f\colon (\{0,1\}^{\ell_{\text{in}}})^n \to (\{0,1\}^{\ell_{\text{out}}})^n$ be an arbitrary n-input function. Let C_f be the circuit that computes f, and let d_f be its depth. Then, under the LWE assumption, there exists a protocol Π_f that securely realizes $\mathcal{F}_f$ in the presence of (static)* malicious *adversaries in the CRS model (and assuming the parties have access to an authenticated broadcast channel). The protocol Π_f splits into two sub-protocols: a preprocessing protocol $\Pi_f^{(\text{pre})}$ and an online protocol $\Pi_f^{(\text{online})}$ with the following properties:*

- ***Reusable preprocessing:*** *The total computational and communication complexity of the preprocessing protocol $\Pi_f^{(\text{pre})}$ is $\mathsf{poly}(n, \lambda, \log d_f)$. Notably, the preprocessing is* independent *of the size of each party's inputs and the overall size $|C_f|$ of the computation. Because the preprocessing only depends* logarithmically *on the depth of C_f (and not its size), the same precomputation can be* reused *across many* parallel *evaluations of C_f (which would increase the size of the computation, but not its depth). Moreover, if we make the additional circular security assumption from Fact 6.3, then the total computational and communication complexity is $\mathsf{poly}(n, \lambda)$, and completely independent of the function f.*
- ***Optimal online round complexity:*** *The online protocol $\Pi_f^{(\text{online})}$ consists of two rounds of communication.*
- ***Low online communication complexity:*** *The total communication complexity of the online protocol $\Pi_f^{(\text{online})}$ is $(\ell_{\text{in}} + \ell_{\text{out}}) \cdot \mathsf{poly}(\lambda, n, d_f)$. If we make the additional circular security assumption from Fact 6.3, then the total communication complexity of $\Pi_f^{(\text{online})}$ is again essentially optimal: $(\ell_{\text{in}} + \ell_{\text{out}}) \cdot \mathsf{poly}(\lambda, n)$.*

Proof. Follows by applying the generic semi-malicious-to-malicious compiler of [AJL+12, Appendix E] to the MPC protocol described in Fact 6.3 along with our UC-NIZKs in the preprocessing model from LWE. □

Remark 6.5 (Reusable Preprocessing)**.** A nice property satisfied by our MPC protocol in the preprocessing model is that the preprocessing is *reusable*. Namely, we only have to run the preprocessing protocol once, provided that all of the computations in the online phase can be implemented by circuits of some bounded depth. In fact, if we are willing to make an additional circular security assumption, the preprocessing is entirely independent of the computation. We note that many classic MPC protocols that leverage preprocessing for better online efficiency do not provide reusable preprocessing [Bea91, DPSZ12]. In these cases, the complexity of the preprocessing phase scales with the *size* of the circuit that is computed in the online phase as opposed to the depth (e.g., the classic technique of Beaver multiplication triples [Bea91] requires generating a single triple for every multiplication gate that will be evaluated during the online phase of the protocol). Having a reusable preprocessing protocol enables us to amortize the cost of the preprocessing across many different computations.

Remark 6.6 (Non-Reusable Preprocessing from Weaker Assumptions)**.** An alternative approach to boosting the Mukherjee-Wichs protocol to provide malicious security in the preprocessing model is to use a bounded-theorem preprocessing NIZK, which can in turn be instantiated from one-way functions [DMP88, LS90, Dam92] or oblivious transfer [KMO89]. One drawback of this approach is that the preprocessing is no longer reusable across multiple computations (since each NIZK system can only be used to prove an *a priori* bounded number of statements). As a result, the round complexity and the computational costs of the preprocessing protocol can no longer be amortized across multiple protocol executions. Moreover, it is unclear that the original bounded-theorem NIZK candidates satisfy the stronger property of universal composability. As such, they cannot be directly applied to achieve malicious security of the Mukherjee-Wichs construction in the UC model.

A succinct GMW compiler from lattices. As discussed in Remark 4.8, if a prover wants to prove m statements (each of which can be checked by a circuit of depth at most d) using the *same* witness w, then the total length of all of the arguments will be $|w| + m \cdot \mathsf{poly}(\lambda, d)$. In particular, the length of the common witness can be *amortized* across many statements. We can leverage this property to obtain a "succinct" version of the classic GMW compiler [GMW86, GMW87] that transforms any MPC protocol Π for some function f in the semi-honest model to a protocol Π' for the same function f in the malicious model. We begin by briefly recalling the "GMW compiler:"

- **Input commitment:** First, the parties commit to their (private) inputs.
- **Coin tossing:** The parties engage in a secure coin-tossing protocol to determine the (secret) randomness each party uses in the protocol execution. At the end of this step, each party has a (private) random string as well as a commitment to every other party's randomness.
- **Protocol emulation:** During the protocol execution, the parties run the semi-honest protocol Π. Whenever the parties send a message, they include a NIZK argument that their message was computed according to the specification of Π on inputs and randomness that are consistent with their committed inputs and randomness.

The NIZK arguments bind each user to following the semi-honest protocol as described. In the UC-model, Canetti et al. [CLOS02] showed an analog of the GMW compiler based on UC-NIZKs.

Our preprocessing NIZKs from lattices gives a new instantiation of the GMW compiler from standard lattice assumptions. Our construction has the appealing property that the communication

overhead of the compiler protocol Π' is essentially *independent* of the parties' computational complexity in the semi-honest protocol Π. We give a concrete comparison below:

- Using traditional NIZKs based on trapdoor permutations [FLS90, DDO+01] or pairing [GOS06], the total size of the NIZK proofs is proportional to the size of each party's computation. Thus, the communication overhead of Π' compared to the original protocol Π on each round r is $\mathsf{poly}(\lambda, n, |C_r|)$, where λ is the security parameter, n is the number of parties, and C_r is the circuit that checks whether a party's message on round r is consistent with the protocol specification Π as well as the party's committed inputs and randomness.

- In the GMW protocol, each party uses the *same* witness to construct their proofs in each round of the protocol (the witness is their private input and randomness). Thus, using the trick described in Remark 4.8, the parties only have to communicate an encryption of their input and randomness *once* at the beginning of the protocol. Thereafter, on each round r of the protocol execution, the size of each proof is $\mathsf{poly}(\lambda, n, d_r)$, where d_r is a bound on the *depth* of the consistency check circuit C_r defined above. Since d_r can be significantly smaller than C_r, the communication overhead of using our lattice-based preprocessing NIZK to instantiate the GMW compiler can lead to substantial asymptotic savings.

As was also noted in Remark 4.8, a similar savings in communication is also possible by first applying the FHE-based transformation from [GGI+15] to any NIZK construction to obtain a NIZK with the same proof size as that of the construction in Corollary 6.2, and then using the resulting construction to implement the GMW compiler. Compared to this alternative approach, our construction has the advantage that it can be instantiated directly from lattice assumptions (and does not additionally assume the existence of a NIZK). Moreover, our construction is likely more efficient since we do not have to incur the cost of composing FHE decryption with NIZK verification in addition to performing FHE evaluation.

Acknowledgments

We thank Hoeteck Wee for helpful discussions on the connection between NIZK proofs and homomorphic commitments. We thank Dan Boneh and Akshayaram Srinivasan for many insightful comments and discussions on this work, and 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

[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In *EUROCRYPT*, 2010.

[ABC+07] Giuseppe Ateniese, Randal C. Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary N. J. Peterson, and Dawn Xiaodong Song. Provable data possession at untrusted stores. In *ACM CCS*, 2007.

[ABC+15] Jae Hyun Ahn, Dan Boneh, Jan Camenisch, Susan Hohenberger, Abhi Shelat, and Brent Waters. Computing on authenticated data. *J. Cryptology*, 28(2), 2015.

[Abe01] Masayuki Abe. A secure three-move blind signature scheme for polynomially many signatures. In *EUROCRYPT*, 2001.

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *CRYPTO*, 2009.

[AFG+10] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-preserving signatures and commitments to group elements. In *CRYPTO*, 2010.

[AHO10] Masayuki Abe, Kristiyan Haralambiev, and Miyako Ohkubo. Signing on elements in bilinear groups for modular protocol design. *IACR Cryptology ePrint Archive*, 2010, 2010.

[AJL+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In *EUROCRYPT*, 2012.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems. In *STOC*, 1996.

[Ajt99] Miklós Ajtai. Generating hard instances of the short basis problem. In *ICALP*, 1999.

[AKK09] Giuseppe Ateniese, Seny Kamara, and Jonathan Katz. Proofs of storage from homomorphic identification protocols. In *ASIACRYPT*, 2009.

[AL11] Nuttapong Attrapadung and Benoît Libert. Homomorphic network coding signatures in the standard model. In *PKC*, 2011.

[AO09] Masayuki Abe and Miyako Ohkubo. A framework for universally composable non-committing blind signatures. In *ASIACRYPT*, 2009.

[AP09] Joël Alwen and Chris Peikert. Generating shorter bases for hard random lattices. In *STACS*, 2009.

[AP14] Jacob Alperin-Sheriff and Chris Peikert. Faster bootstrapping with polynomial error. In *CRYPTO*, 2014.

[APSD18] Navid Alamati, Chris Peikert, and Noah Stephens-Davidowitz. New (and old) proof systems for lattice problems. In *PKC*, 2018.

[BBDQ18] Fabrice Benhamouda, Olivier Blazy, Léo Ducas, and Willy Quach. Hash proof systems over lattices revisited. In *PKC*, 2018.

[BDMP91] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Noninteractive zero-knowledge. *SIAM J. Comput.*, 20(6), 1991.

[Bea91] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, 1991.

[BF11a] Dan Boneh and David Mandell Freeman. Homomorphic signatures for polynomial functions. In *EUROCRYPT*, 2011.

[BF11b] Dan Boneh and David Mandell Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In *PKC*, 2011.

[BFF+09] Christina Brzuska, Marc Fischlin, Tobias Freudenreich, Anja Lehmann, Marcus Page, Jakob Schelbert, Dominique Schröder, and Florian Volk. Security of sanitizable signatures revisited. In *PKC*, 2009.

[BFKW09] Dan Boneh, David Mandell Freeman, Jonathan Katz, and Brent Waters. Signing a linear subspace: Signature schemes for network coding. In *PKC*, 2009.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *STOC*, 1988.

[BFR13] Michael Backes, Dario Fiore, and Raphael M. Reischuk. Verifiable delegation of computation on outsourced data. In *ACM CCS*, 2013.

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *EUROCRYPT*, 2014.

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In *CRYPTO*, 2016.

[BL13] Foteini Baldimtsi and Anna Lysyanskaya. Anonymous credentials light. In *ACM CCS*, 2013.

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *STOC*, 2013.

[BNPS03] Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The one-more-rsa-inversion problems and the security of chaum's blind signature scheme. *J. Cryptology*, 16(3), 2003.

[Bol03] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *PKC*, 2003.

[Bra00] Stefan A. Brands. *Rethinking public key infrastructures and digital certificates: building in privacy.* MIT Press, 2000.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based FHE as secure as PKE. In *ITCS*, 2014.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, 2001.

[Can04] Ran Canetti. Universally composable signature, certification, and authentication. In *CSFW*, 2004.

[Cat14] Dario Catalano. Homomorphic signatures and message authentication codes. In *SCN*, 2014.

[CC17] Pyrros Chaidos and Geoffroy Couteau. Efficient designated-verifier non-interactive zero-knowledge proofs of knowledge. *IACR Cryptology ePrint Archive*, 2017, 2017.

[CCRR18] Ran Canetti, Yilei Chen, Leonid Reyzin, and Ron D. Rothblum. Fiat-shamir and correlation intractability from strong KDM-secure encryption. In *EUROCRYPT*, 2018.

[CD04] Ronald Cramer and Ivan Damgård. Secret-key zero-knowlegde and non-interactive verifiable exponentiation. In *TCC*, 2004.

[CF13] Dario Catalano and Dario Fiore. Practical homomorphic MACs for arithmetic circuits. In *EUROCRYPT*, 2013.

[CFGN14] Dario Catalano, Dario Fiore, Rosario Gennaro, and Luca Nizzardo. Generalizing homomorphic MACs for arithmetic circuits. In *PKC*, 2014.

[CFW12] Dario Catalano, Dario Fiore, and Bogdan Warinschi. Efficient network coding signatures in the standard model. In *PKC*, 2012.

[CFW14] Dario Catalano, Dario Fiore, and Bogdan Warinschi. Homomorphic signatures with efficient verification for polynomial functions. In *CRYPTO*, 2014.

[CG15] Pyrros Chaidos and Jens Groth. Making sigma-protocols non-interactive without random oracles. In *PKC*, 2015.

[Cha82] David Chaum. Blind signatures for untraceable payments. In *CRYPTO*, 1982.

[CHKP10] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In *EUROCRYPT*, 2010.

[CKW04] Jan Camenisch, Maciej Koprowski, and Bogdan Warinschi. Efficient blind signatures without random oracles. In *SCN*, 2004.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, 2002.

[CR03] Ran Canetti and Tal Rabin. Universal composition with joint state. In *CRYPTO*, 2003.

[CS02] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *EUROCRYPT*, 2002.

[Dam92] Ivan Damgård. Non-interactive circuit based proofs and non-interactive perfect zero-knowledge with proprocessing. In *EUROCRYPT*, 1992.

[DDO+01] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In *CRYPTO*, 2001.

[DFN06] Ivan Damgård, Nelly Fazio, and Antonio Nicolosi. Non-interactive zero-knowledge from homomorphic encryption. In *TCC*, 2006.

[DMP87] Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Non-interactive zero-knowledge proof systems. In *CRYPTO*, 1987.

[DMP88] Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Non-interactive zero-knowledge with preprocessing. In *CRYPTO*, 1988.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, 2012.

[DVW09] Yevgeniy Dodis, Salil P. Vadhan, and Daniel Wichs. Proofs of retrievability via hardness amplification. In *TCC*, 2009.

[FHKS16] Georg Fuchsbauer, Christian Hanser, Chethan Kamath, and Daniel Slamanig. Practical round-optimal blind signatures in the standard model from weaker assumptions. In *SCN*, 2016.

[FHS15] Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. Practical round-optimal blind signatures in the standard model. In *CRYPTO*, 2015.

[Fis06] Marc Fischlin. Round-optimal composable blind signatures in the common reference string model. In *CRYPTO*, 2006.

[FLS90] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string. In *FOCS*, 1990.

[FMNP16] Dario Fiore, Aikaterini Mitrokotsa, Luca Nizzardo, and Elena Pagnin. Multi-key homomorphic authenticators. In *ASIACRYPT*, 2016.

[Fre12] David Mandell Freeman. Improved security for linearly homomorphic signatures: A generic framework. In *PKC*, 2012.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, 1986.

[Fuc09] Georg Fuchsbauer. Automorphic signatures in bilinear groups and an application to round-optimal blind signatures. *IACR Cryptology ePrint Archive*, 2009, 2009.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, 2009.

[GGI+15] Craig Gentry, Jens Groth, Yuval Ishai, Chris Peikert, Amit Sahai, and Adam D. Smith. Using fully homomorphic hybrid encryption to minimize non-interative zero-knowledge proofs. *J. Cryptology*, 28(4), 2015.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. In *FOCS*, 1984.

[GKKR10] Rosario Gennaro, Jonathan Katz, Hugo Krawczyk, and Tal Rabin. Secure network coding over the integers. In *PKC*, 2010.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *STOC*, 1985.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all np-statements in zero-knowledge, and a methodology of cryptographic protocol design. In *CRYPTO*, 1986.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, 1987.

[GO94] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *J. Cryptology*, 7(1), 1994.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for NP. In *EUROCRYPT*, 2006.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, 2008.

[Gro06] Jens Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In *ASIACRYPT*, 2006.

[Gro10] Jens Groth. Short non-interactive zero-knowledge proofs. In *ASIACRYPT*, 2010.

[GRS+11] Sanjam Garg, Vanishree Rao, Amit Sahai, Dominique Schröder, and Dominique Unruh. Round optimal blind signatures. In *CRYPTO*, 2011.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT*, 2008.

[GS12] Essam Ghadafi and Nigel P. Smart. Efficient two-move blind signatures in the common reference string model. In *ISC*, 2012.

[GS18] Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. In *EUROCRYPT*, 2018.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO*, 2013.

[GV15] Sergey Gorbunov and Dhinakaran Vinayagamurthy. Riding on asymmetry: Efficient ABE for branching programs. In *ASIACRYPT*, 2015.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In *STOC*, 2015.

[GW13] Rosario Gennaro and Daniel Wichs. Fully homomorphic message authenticators. In *ASIACRYPT*, 2013.

[HK16] Lucjan Hanzlik and Kamil Kluczniak. A short paper on blind signatures from knowledge assumptions. In *Financial Cryptography*, 2016.

[IKOS09] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge proofs from secure multiparty computation. *SIAM J. Comput.*, 39(3), 2009.

[KMO89] Joe Kilian, Silvio Micali, and Rafail Ostrovsky. Minimum resource zero-knowledge proofs. In *CRYPTO*, 1989.

[KR06] Yael Tauman Kalai and Ran Raz. Succinct non-interactive zero-knowledge proofs with preprocessing for LOGSNP. In *FOCS*, 2006.

[KV09] Jonathan Katz and Vinod Vaikuntanathan. Smooth projective hashing and password-based authenticated key exchange from lattices. In *ASIACRYPT*, 2009.

[KZ06] Aggelos Kiayias and Hong-Sheng Zhou. Concurrent blind signatures without random oracles. In *SCN*, 2006.

[LNSW13] San Ling, Khoa Nguyen, Damien Stehlé, and Huaxiong Wang. Improved zero-knowledge proofs of knowledge for the ISIS problem, and applications. In *PKC*, 2013.

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT*, 2007.

[LS90] Dror Lapidot and Adi Shamir. Publicly verifiable non-interactive zero-knowledge proofs. In *CRYPTO*, 1990.

[LW15] Vadim Lyubashevsky and Daniel Wichs. Simple lattice trapdoor sampling from a broad class of distributions. In *PKC*, 2015.

[Mic04] Daniele Micciancio. Almost perfect lattices, the covering radius problem, and applications to ajtai's connection factor. *SIAM J. Comput.*, 34(1), 2004.

[MM11] Daniele Micciancio and Petros Mol. Pseudorandom knapsacks and the sample complexity of LWE search-to-decision reductions. In *CRYPTO*, 2011.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*, 2012.

[MP13] Daniele Micciancio and Chris Peikert. Hardness of SIS and LWE with small parameters. In *CRYPTO*, 2013.

[MR07] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on gaussian measures. *SIAM J. Comput.*, 37(1), 2007.

[MW16] Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In *EUROCRYPT*, 2016.

[NR97] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *FOCS*, 1997.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem. In *STOC*, 2009.

[PS96] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In *EUROCRYPT*, 1996.

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *J. Cryptology*, 13(3), 2000.

[PV08] Chris Peikert and Vinod Vaikuntanathan. Noninteractive statistical zero-knowledge proofs for lattice problems. In *CRYPTO*, 2008.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO*, 2008.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, 2005.

[RSS18] Ron D. Rothblum, Adam Sealfon, and Katerina Sotiraki. Towards non-interactive zero-knowledge for NP from LWE. *IACR Cryptology ePrint Archive*, 2018, 2018.

[Rüc10] Markus Rückert. Lattice-based blind signatures. In *ASIACRYPT*, 2010.

[Sch89] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO*, 1989.

[SP92] Alfredo De Santis and Giuseppe Persiano. Zero-knowledge proofs of knowledge without interaction. In *FOCS*, 1992.

[SW08] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In *ASIACRYPT*, 2008.

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

[XXW13] Xiang Xie, Rui Xue, and Minqian Wang. Zero knowledge proofs from ring-lwe. In *CANS*, 2013.

[ZY17] Jiang Zhang and Yu Yu. Two-round PAKE from approximate SPH and instantiations from lattices. In *ASIACRYPT*, 2017.

A The Universal Composability Framework

In this section, we briefly review the universal composability (UC) framework. We refer to [Can01] for the full details. The description here is adapted from the presentation in [MW16, Appendix A] and [GS18, Appendix A]. Readers familiar with UC security can safely skip this section, and we only include it for completeness.

The UC framework. We work in the standard universal composability framework with static corruptions. The UC framework defines an environment $\mathcal{Z}$ (modeled as an efficient algorithm) that is invoked on the security parameter 1^λ and an auxiliary input $z \in \{0,1\}^*$. The environment oversees the protocol execution in one of two possible experiments:

- The *ideal world execution* involves dummy parties $\tilde{P}_1, \ldots, \tilde{P}_n$, an ideal adversary $\mathcal{S}$ (also called a "simulator") who may corrupt some of the dummy parties, and an ideal functionality $\mathcal{F}$.

- The *real world execution* involves parties $P_1, \ldots, P_n$ (modeled as efficient algorithms) and a real-world adversary $\mathcal{A}$ who may corrupt some of the parties.

In both cases, the environment $\mathcal{Z}$ chooses the inputs for the parties, receives the outputs from the uncorrupted parties, and can interact with the real/ideal world adversaries during the protocol execution. At the end of the protocol execution, the environment outputs a bit, which is defined to be the output of the experiment. More precisely, we define the following random variables:

- Let $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^\lambda, z)$ be the random variable for the output of the environment $\mathcal{Z}$ after interacting with the ideal world execution with adversary $\mathcal{S}$, the functionality $\mathcal{F}$ on security parameter λ and input z. We write $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ to denote the ensemble $\{\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^\lambda, z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}$.
- Let $\text{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(1^\lambda, z)$ denote the random variable for the output of the environment $\mathcal{Z}$ after interacting with the real world execution with adversary $\mathcal{A}$ and parties running a protocol Π on security parameter λ and input z. We write $\text{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}$ to denote the ensemble $\{\text{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(1^\lambda, z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}$.

Definition A.1. Fix $n \in \mathbb{N}$, let $\mathcal{F}$ be an n-ary functionality, and Π be an n-party protocol. We say that the protocol Π securely realizes $\mathcal{F}$ if for all efficient adversaries $\mathcal{A}$, there exists an ideal adversary $\mathcal{S}$ such that for all efficient environments $\mathcal{Z}$, we have that

$$\text{REAL}_{\Pi,\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}.$$

Hybrid protocols. Hybrid protocols are protocols where, in addition to communicating as usual in the standard model of execution, the parties have access to (multiple copies of) an ideal functionality. More precisely, in a protocol execution in the $\mathcal{F}$-hybrid model (where $\mathcal{F}$ denotes an ideal functionality), the parties may give inputs and receive outputs from an unbounded number of copies of $\mathcal{F}$. The different copies of $\mathcal{F}$ are differentiated using a session ID (denoted sid). All of the inputs to each copy of $\mathcal{F}$ and the outputs from each copy of $\mathcal{F}$ have the same session ID. We can correspondingly extend Definition A.1 to define the notion of a protocol Π securely realizing a functionality $\mathcal{G}$ in the $\mathcal{F}$-hybrid model.

The universal composition operation. We now define the universal composition operation and state the universal composition theorem. Let ρ be an $\mathcal{F}$-hybrid protocol, and let Π be a protocol that securely realizes $\mathcal{F}$ (Definition A.1). The composed protocol ρ^Π is the protocol where each invocation of the ideal functionality $\mathcal{F}$ in ρ is replaced by a fresh invocation of the protocol Π. Specifically, the first message sent to each copy of $\mathcal{F}$ (from any party) is replaced with the first message of Π (generated with the same input and sid associated with the particular copy of $\mathcal{F}$). Each output value generated by a copy of Π is treated as a message received from the corresponding copy of $\mathcal{F}$. Note that if Π is a $\mathcal{G}$-hybrid protocol (where $\mathcal{G}$ is an arbitrary ideal functionality), then ρ^Π is also a $\mathcal{G}$-hybrid protocol.

The universal composition theorem. Let $\mathcal{F}$ be an ideal functionality. In its general form, the universal composition theorem [Can01] states that if Π is a protocol that securely realizes $\mathcal{F}$, then for any $\mathcal{F}$-hybrid protocol ρ that securely realizes $\mathcal{G}$, the composed protocol ρ^Π securely realizes $\mathcal{G}$. We state the formal theorem below:

Theorem A.2 (Universal Composition [Can01, Corollary 15])**.** *Let $\mathcal{F}, \mathcal{G}$ be ideal functionalities, and let Π be a protocol that securely realizes $\mathcal{F}$. If ρ securely realizes $\mathcal{G}$ in the $\mathcal{F}$-hybrid model, then the composed protocol ρ^Π securely realizes $\mathcal{G}$.*

A.1 UC Functionalities

In this section, we review the ideal common reference string (CRS), oblivious transfer (OT), and zero-knowledge (ZK) functionalities that we use in this work.

The CRS functionality. The common reference string (CRS) functionality $\mathcal{F}_{\text{CRS}}^{\mathcal{D}}$ (parameterized by an efficiently-sampleable distribution $\mathcal{D}$) samples and outputs a string from $\mathcal{D}$. The formal specification from [CR03] is as follows:

Functionality $\mathcal{F}_{\text{CRS}}^{\mathcal{D}}$

The ideal CRS functionality $\mathcal{F}_{\text{CRS}}^{\mathcal{D}}$ is parameterized by an efficiently-sampleable distribution $\mathcal{D}$ and runs with parties $P_1, \ldots, P_n$ and an ideal adversary $\mathcal{S}$. Its behavior is as follows:

- Upon activation with session ID sid, the functionality samples $\text{CRS} \leftarrow \mathcal{D}$ and sends $(\mathsf{sid}, \text{CRS})$ to the adversary $\mathcal{S}$.
- On receiving sid from a party P_i, send $(\mathsf{sid}, \text{CRS})$ to P_i.

Figure 4: The $\mathcal{F}_{\text{CRS}}^{\mathcal{D}}$ functionality [CR03]

The OT functionality. The oblivious transfer (OT) functionality $\mathcal{F}_{\text{OT}}^{s}$ (parameterized by the message length s) is a two-party functionality between a sender $\mathbf{S}$ and a receiver $\mathbf{R}$. The sender's input consists of two messages $x_0, x_1 \in \{0,1\}^s$ and the receiver's input consists of a bit $b \in \{0,1\}$. At the end of the protocol execution, the receiver learns x_b (and nothing else), and the sender learns nothing. These requirements are captured by the OT functionality $\mathcal{F}_{\text{OT}}^{s}$ from [CLOS02] defined as follows:

Functionality $\mathcal{F}_{\text{OT}}^{s}$

The ideal OT functionality $\mathcal{F}_{\text{OT}}^{s}$ is parameterized by a message length s and runs with a sender $\mathbf{S}$, a receiver $\mathbf{R}$, and an ideal adversary $\mathcal{S}$. Its behavior is as follows:

- Upon receiving a message $(\mathsf{sid}, \mathsf{sender}, x_0, x_1)$ from $\mathbf{S}$ where $x_0, x_1 \in \{0,1\}^s$, store the tuple (sid, x_0, x_1).
- Upon receiving a message $(\mathsf{sid}, \mathsf{receiver}, b)$ from $\mathbf{R}$, check if a tuple of the form $(\mathsf{sid}, \hat{x}_0, \hat{x}_1)$ has been stored for some pair of messages $\hat{x}_0, \hat{x}_1$. If so, send $(\mathsf{sid}, \hat{x}_b)$ to $\mathbf{R}$, sid to the adversary, and halt. If not, send nothing to $\mathbf{R}$, but continue running.

Figure 5: The $\mathcal{F}_{\text{OT}}^{s}$ functionality [CLOS02]

For simplicity of notation, we define a block-wise OT functionality $\mathcal{F}_{\text{OT}}^{\ell,s}$ where the sender's input consists of ℓ pairs of messages $\{(x_{i,0}, x_{i,1})\}_{i\in[\ell]}$ and the receiver's input consists of ℓ bits $b_1, \ldots, b_\ell$. At the end of the protocol execution, the receiver learns the messages $x_{1,b_1}, \ldots, x_{\ell,b_\ell}$ (and nothing else), and the sender learns nothing. The block-wise OT functionality can be securely realized from the standard OT functionality $\mathcal{F}_{\text{OT}}^{s}$ via the universal composition theorem [Can01].

Functionality $\mathcal{F}_{\text{OT}}^{\ell,s}$

The ideal OT functionality $\mathcal{F}_{\text{OT}}^{\ell,s}$ is parameterized by the number of messages ℓ and message length s, and runs with a sender $\mathbf{S}$, a receiver $\mathbf{R}$, and an ideal adversary $\mathcal{S}$. Its behavior is as follows:

- Upon receiving a message $(\mathsf{sid}, \mathsf{sender}, \{(x_{i,0}, x_{i,1})\}_{i\in[\ell]})$ from $\mathbf{S}$ where $x_{i,\beta} \in \{0,1\}^s$ for $i \in [\ell]$, $\beta \in \{0,1\}$, store $\big(\mathsf{sid}, \{(x_{i,0}, x_{i,1})\}_{i\in[\ell]}\big)$.
- Upon receiving a message $(\mathsf{sid}, \mathsf{receiver}, (b_1, \ldots, b_\ell))$ from $\mathbf{R}$ for $b_1, \ldots, b_\ell \in \{0,1\}$, check if a tuple of the form $\big(\mathsf{sid}, \{(\hat{x}_{i,0}, \hat{x}_{i,1})\}_{i\in[\ell]}\big)$ has been stored for some choice of $\hat{x}_{i,\beta} \in \{0,1\}^\ell$ where $i \in [\ell]$ and $\beta \in \{0,1\}$. If so, send $(\mathsf{sid}, \{\hat{x}_{i,b_i}\}_{i\in[\ell]})$ to $\mathbf{R}$, sid to the adversary, and halt. If not, send nothing to $\mathbf{R}$, but continue running.

Figure 6: The $\mathcal{F}_{\text{OT}}^{\ell,s}$ functionality

The ZK functionality. The zero-knowledge (ZK) functionality is a two-party functionality between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$. The prover is able to send the functionality a description of an NP relation $\mathcal{R}$, a statement x to be proven along with a witness w. The functionality forwards the relation and the statement x to the verifier if and only if $\mathcal{R}(x, w) = 1$. Our definition is inherently multi-theorem; namely, the prover can prove arbitrarily many statements (possibly with respect to different NP relations). We distinguish between different proof sub-sessions by associating a unique sub-session ID ssid with each sub-session. Our definition is adapted from the one given in [CLOS02].

Functionality $\mathcal{F}_{\mathsf{ZK}}$

The ideal ZK functionality runs with a prover $\mathcal{P}$, a verifier $\mathcal{V}$ and an ideal adversary $\mathcal{S}$. Its behavior is as follows:

- Upon receiving a message $(\mathsf{sid}, \mathsf{ssid}, \mathsf{prove}, \mathcal{R}, x, w)$ from $\mathcal{P}$ where $\mathcal{R}$ is an NP relation, if $\mathcal{R}(x, w) = 1$, then send $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathcal{R}, x)$ to $\mathcal{V}$ and $\mathcal{S}$. Otherwise, ignore the message.

Figure 7: The $\mathcal{F}_{\mathsf{ZK}}$ functionality.

The general UC functionality. Let $f\colon (\{0,1\}^{\ell_{\text{in}}})^n \to (\{0,1\}^{\ell_{\text{out}}})^n$ be an arbitrary n-input function. The general UC-functionality $\mathcal{F}_f$ is parameterized with a function f and described in Figure 8. Our presentation is adapted from that in [GS18].

Functionality $\mathcal{F}_f$

The general UC functionality $\mathcal{F}_f$ is parameterized by a (possibly randomized) function $f : (\{0,1\}^{\ell_{\text{in}}})^n \to (\{0,1\}^{\ell_{\text{out}}})^n$ on n inputs, and runs with parties $\mathcal{P} = (P_1, \ldots, P_n)$, and an ideal adversary $\mathcal{S}$, as follows:

- Each party P_i sends $(\mathsf{sid}, \mathsf{input}, \mathcal{P}, P_i, x_i)$ where $x_i \in \{0,1\}^{\ell_{\text{in}}}$ to $\mathcal{F}_f$.
- After receiving inputs from each of the parties, the functionality computes $(y_1, \ldots, y_n) \leftarrow f(x_1, \ldots, x_n)$. For every party P_i that is corrupted, the functionality sends $\mathcal{S}$ the message $(\mathsf{sid}, \mathsf{output}, \mathcal{P}, P_i, y_i)$.
- When the functionality receives a message $(\mathsf{sid}, \mathsf{finish}, \mathcal{P}, P_i)$ from $\mathcal{S}$, the ideal functionality sends $(\mathsf{sid}, \mathsf{output}, \mathcal{P}, P_i, y_i)$ to P_i. The functionality $\mathcal{F}$ ignores the message if inputs from all parties in $\mathcal{P}$ have not been received.

Figure 8: The general UC functionality $\mathcal{F}_f$.

B Adaptively-Secure Homomorphic Signatures

In this section, we show how to transform a homomorphic signature scheme that satisfies only *selective unforgeability* to full unforgeability (Definition 3.5). Although the transformation follows the construction of [GVW15], we give the full construction to show that the resulting construction still satisfies our strengthened notion of context-hiding (Definition 3.9).

Construction B.1 (Adaptively-Secure Homomorphic Signature [GVW15, adapted])**.** Fix a security parameter λ and a message length $\ell \in \mathbb{N}$. Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ be a function class where each $\mathcal{C}_\lambda$ is a collection of Boolean circuits (on ℓ-bit inputs). Then, define the following quantities:

- First, let $\Pi_{\mathsf{HS,in}} = (\mathsf{PrmsGen}_{\mathsf{in}}, \mathsf{KeyGen}_{\mathsf{in}}, \mathsf{Sign}_{\mathsf{in}}, \mathsf{PrmsEval}_{\mathsf{in}}, \mathsf{SigEval}_{\mathsf{in}}, \mathsf{Hide}_{\mathsf{in}}, \mathsf{Verify}_{\mathsf{in}}, \mathsf{VerifyFresh}_{\mathsf{in}}, \mathsf{VerifyHide}_{\mathsf{in}})$ be a *selectively-secure* decomposable homomorphic signature scheme with message space $\{0,1\}$, message length $\ell \in \mathbb{N}$, and function class $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$. This is the "inner" homomorphic signature scheme that will be used to sign messages $\vec{x} \in \{0,1\}^\ell$. For simplicity, assume also that the signatures in $\Pi_{\mathsf{HS,in}}$ have an "empty" message-independent component.[15]
- Let ρ be the length of the public keys in $\Pi_{\mathsf{HS,in}}$. For a circuit $C \in \mathcal{C}_\lambda$, let F_C be the function that maps $\overrightarrow{\mathsf{pk}}_{\mathsf{in}} \mapsto \mathsf{PrmsEval}_{\mathsf{in}}(C, \overrightarrow{\mathsf{pk}}_{\mathsf{in}})$, where $\overrightarrow{\mathsf{pk}}_{\mathsf{in}}$ are the public parameters output by $\mathsf{PrmsGen}_{\mathsf{in}}$. Let $\mathcal{C}' = \{\mathcal{C}'_\lambda\}_{\lambda \in \mathbb{N}}$ be a function class where each function class $\mathcal{C}'_\lambda$ contains all functions F_C for $C \in \mathcal{C}_\lambda$.
- Finally, let $\Pi_{\mathsf{HS,out}} = (\mathsf{PrmsGen}_{\mathsf{out}}, \mathsf{KeyGen}_{\mathsf{out}}, \mathsf{Sign}_{\mathsf{out}}, \mathsf{PrmsEval}_{\mathsf{out}}, \mathsf{SigEval}_{\mathsf{out}}, \mathsf{Hide}_{\mathsf{out}}, \mathsf{Verify}_{\mathsf{out}}, \mathsf{VerifyFresh}_{\mathsf{out}}, \mathsf{VerifyHide}_{\mathsf{out}})$ be a *selectively-secure* homomorphic signature scheme with message space $\{0,1\}$, message length $\rho \in \mathbb{N}$, and function class $\mathcal{C}' = \{\mathcal{C}'_\lambda\}_{\lambda \in \mathbb{N}}$. This is the "outer" homomorphic signature scheme that will be used to sign the public keys of $\Pi_{\mathsf{HS,in}}$.

[15] This restriction simplifies the presentation of our construction, and is satisfied by Construction 3.11. It is straightforward (but notationally cumbersome) to modify this generic construction to apply to the setting where the signatures in $\Pi_{\mathsf{HS,in}}$ have a non-empty message-independent component.

We construct a homomorphic signature scheme $\Pi_{\mathsf{HS}} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrmsEval}, \mathsf{SigEval}, \mathsf{Hide}, \mathsf{Verify}, \mathsf{VerifyFresh}, \mathsf{VerifyHide})$ for message space $\{0,1\}$, message length ℓ, and function class $\mathcal{C}$ as follows:

- $\mathsf{PrmsGen}(1^\lambda, 1^\ell) \to \overrightarrow{\mathsf{pk}}$: On input the security parameter λ and message length ℓ, the parameter-generation algorithm generates independent public parameters $\overrightarrow{\mathsf{pk}}_{\mathsf{out},i} \leftarrow \mathsf{PrmsGen}_{\mathsf{out}}(1^\lambda, 1^\rho)$ for $i \in [\ell]$. Then, it sets $\mathsf{pk}_i = \overrightarrow{\mathsf{pk}}_{\mathsf{out},i}$ for $i \in [\ell]$ and returns $\overrightarrow{\mathsf{pk}} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell)$.

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{vk}, \mathsf{sk})$: On input the security parameter λ, the key-generation algorithm generates two pairs of keys $(\mathsf{vk}_{\mathsf{in}}, \mathsf{sk}_{\mathsf{in}}) \leftarrow \mathsf{KeyGen}_{\mathsf{in}}(1^\lambda)$, $(\mathsf{vk}_{\mathsf{out}}, \mathsf{sk}_{\mathsf{out}}) \leftarrow \mathsf{KeyGen}_{\mathsf{out}}(1^\lambda)$, and sets $\mathsf{vk} = (\mathsf{vk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{out}})$ and $\mathsf{sk} = (\mathsf{sk}_{\mathsf{in}}, \mathsf{sk}_{\mathsf{out}})$.

- $\mathsf{Sign}(\mathsf{pk}_i, \mathsf{sk}, x_i) \to \sigma_i$: On input a public key $\mathsf{pk}_i = \overrightarrow{\mathsf{pk}}_{\mathsf{out},i}$, a signing key $\mathsf{sk} = (\mathsf{sk}_{\mathsf{in}}, \mathsf{sk}_{\mathsf{out}})$, and a message $x_i \in \{0,1\}$, the signing algorithm computes $\sigma_i^{\mathsf{pk}} \leftarrow \mathsf{SignPK}(\mathsf{pk}_i, \mathsf{sk})$ and $\sigma_i^{\mathsf{m}} \leftarrow \mathsf{SignM}(\mathsf{pk}_i, \mathsf{sk}, x_i, \sigma_i^{\mathsf{pk}})$ where the algorithms SignPK and SignM are defined as follows:

  - $\mathsf{SignPK}(\mathsf{pk}_i, \mathsf{sk})$: The message-independent signing algorithm first samples a fresh public key $\mathsf{pk}_{\mathsf{in},i} \leftarrow \mathsf{PrmsGen}_{\mathsf{in}}(1^\lambda, 1^1)$ for the inner homomorphic signature scheme.[16] By assumption, $\mathsf{pk}_{\mathsf{in},i}$ is a bit-string of length ρ. Then, the algorithm signs the public key $\mathsf{pk}_{\mathsf{in},i}$ using the outer signature scheme: $\vec{\sigma}_{\mathsf{out},i} \leftarrow \mathsf{Sign}_{\mathsf{out}}(\overrightarrow{\mathsf{pk}}_{\mathsf{out},i}, \mathsf{sk}_{\mathsf{out}}, \mathsf{pk}_{\mathsf{in},i})$. It returns $\sigma_i^{\mathsf{pk}} = (\vec{\sigma}_{\mathsf{out},i}, \mathsf{pk}_{\mathsf{in},i})$.
  - $\mathsf{SignM}(\mathsf{pk}_i, \mathsf{sk}, x_i, \sigma^{\mathsf{pk}})$: The message-dependent signing algorithm parses $\sigma^{\mathsf{pk}} = (\vec{\sigma}_{\mathsf{out},i}, \mathsf{pk}_{\mathsf{in},i})$, and signs the message using the inner signature scheme: $\sigma_{\mathsf{in},i} \leftarrow \mathsf{Sign}_{\mathsf{in}}(\mathsf{pk}_{\mathsf{in},i}, \mathsf{sk}_{\mathsf{in}}, x_i)$. It outputs $\sigma_i^{\mathsf{m}} = \sigma_{\mathsf{in},i}$.

  Finally, the signing algorithm outputs $\sigma_i = (\sigma_i^{\mathsf{pk}}, \sigma_i^{\mathsf{m}})$.

- $\mathsf{PrmsEval}(C, \overrightarrow{\mathsf{pk}}) \to \mathsf{pk}_C$: On input a circuit $C \in \mathcal{C}$ and public parameters $\overrightarrow{\mathsf{pk}} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell)$, the parameter-evaluation algorithm parses $\mathsf{pk}_i = \overrightarrow{\mathsf{pk}}_{\mathsf{out},i}$ for each $i \in [\ell]$. It outputs $\mathsf{pk}_C \leftarrow \mathsf{PrmsEval}_{\mathsf{out}}(F_C, (\overrightarrow{\mathsf{pk}}_{\mathsf{out},1}, \ldots, \overrightarrow{\mathsf{pk}}_{\mathsf{out},\ell}))$.

- $\mathsf{SigEval}(C, \overrightarrow{\mathsf{pk}}', \vec{x}, \vec{\sigma}) \to \sigma$: On input a circuit $C \in \mathcal{C}$, public parameters $\overrightarrow{\mathsf{pk}}' = (\mathsf{pk}'_1, \ldots, \mathsf{pk}'_\ell)$, a message $\vec{x} \in \{0,1\}^\ell$ and a signature $\vec{\sigma} = (\vec{\sigma}^{\mathsf{pk}}, \vec{\sigma}^{\mathsf{m}})$, the signature-evaluation algorithm parses $\mathsf{pk}'_i = \overrightarrow{\mathsf{pk}}'_{\mathsf{out},i}$ for all $i \in [\ell]$. Then, it computes $\sigma^{\mathsf{pk}} \leftarrow \mathsf{SigEvalPK}(C, \overrightarrow{\mathsf{pk}}', \vec{\sigma}^{\mathsf{pk}})$ and $\sigma^{\mathsf{m}} \leftarrow \mathsf{SigEvalM}(C, \overrightarrow{\mathsf{pk}}', \vec{x}, \vec{\sigma})$, where $\mathsf{SigEvalPK}$ and $\mathsf{SigEvalM}$ are defined as follows:

  - $\mathsf{SigEvalPK}(C, \overrightarrow{\mathsf{pk}}, \vec{\sigma}^{\mathsf{pk}})$: The message-independent signature-evaluation algorithm first parses $\vec{\sigma}^{\mathsf{pk}} = ((\vec{\sigma}_{\mathsf{out},1}, \mathsf{pk}_{\mathsf{in},1}), \ldots, (\vec{\sigma}_{\mathsf{out},\ell}, \mathsf{pk}_{\mathsf{in},\ell}))$. It then computes

    $$\vec{\sigma}_{\mathsf{out},C} \leftarrow \mathsf{SigEval}_{\mathsf{out}}(F_C, (\overrightarrow{\mathsf{pk}}'_{\mathsf{out},1}, \ldots, \overrightarrow{\mathsf{pk}}'_{\mathsf{out},\ell}), (\mathsf{pk}_{\mathsf{in},1}, \ldots, \mathsf{pk}_{\mathsf{in},\ell}), (\vec{\sigma}_{\mathsf{out},1}, \ldots, \vec{\sigma}_{\mathsf{out},\ell})),$$

    and $\mathsf{pk}_{\mathsf{in},C} \leftarrow \mathsf{PrmsEval}_{\mathsf{in}}(C, (\mathsf{pk}_{\mathsf{in},1}, \ldots, \mathsf{pk}_{\mathsf{in},\ell}))$. Finally, it returns $\sigma^{\mathsf{pk}} = (\vec{\sigma}_{\mathsf{out},C}, \mathsf{pk}_{\mathsf{in},C})$.

[16] Note that we are implicitly assuming here that the public keys $\mathsf{pk}_{\mathsf{in},i}$ for each $i \in [\ell]$ can be generated independently of one another: namely, that the output distribution of $\mathsf{PrmsGen}_{\mathsf{in}}(1^\lambda, 1^\ell)$ is identical to ℓ independent invocations of $\mathsf{PrmsGen}_{\mathsf{in}}(1^\lambda, 1^1)$. This property is satisfied by the homomorphic signature scheme in Construction 3.11.

– $\mathsf{SigEvalM}(C, \overrightarrow{\mathsf{pk}}', \vec{x}, \vec{\sigma})$: The message-dependent signature-evaluation algorithm writes $\vec{\sigma}$ as $(\vec{\sigma}^{\mathsf{pk}}, \vec{\sigma}^{\mathsf{m}})$, where $\vec{\sigma}^{\mathsf{pk}} = ((\vec{\sigma}_{\mathsf{out},1}, \mathsf{pk}_{\mathsf{in},1}), \ldots, (\vec{\sigma}_{\mathsf{out},\ell}, \mathsf{pk}_{\mathsf{in},\ell}))$, and $\vec{\sigma}^{\mathsf{m}} = (\sigma_{\mathsf{in},1}, \ldots, \sigma_{\mathsf{in},\ell})$. It outputs the signature $\sigma^{\mathsf{m}} \leftarrow \mathsf{SigEval}_{\mathsf{in}}(C, (\mathsf{pk}_{\mathsf{in},1}, \ldots, \mathsf{pk}_{\mathsf{in},\ell}), \vec{x}, (\sigma_{\mathsf{in},1}, \ldots, \sigma_{\mathsf{in},\ell}))$.

Finally, the signature-evaluation algorithm outputs $\sigma = (\sigma^{\mathsf{pk}}, \sigma^{\mathsf{m}})$.

- $\mathsf{Hide}(\mathsf{vk}, x, \sigma) \to \sigma^*$: On input a verification key $\mathsf{vk} = (\mathsf{vk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{out}})$, a message $x \in \{0,1\}$, and a signature $\sigma = (\sigma^{\mathsf{pk}}, \sigma^{\mathsf{m}})$, the hide algorithm parses $\sigma^{\mathsf{m}} = \sigma_{\mathsf{in}}$. It computes $\sigma^*_{\mathsf{in}} \leftarrow \mathsf{Hide}_{\mathsf{in}}(\mathsf{vk}_{\mathsf{in}}, x, \sigma_{\mathsf{in}})$, and returns $\sigma^* = (\sigma^{\mathsf{pk}}, \sigma^*_{\mathsf{in}})$.

- $\mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, x, \sigma) \to \{0,1\}$: On input a public key $\mathsf{pk} = \overrightarrow{\mathsf{pk}}_{\mathsf{out}}$, a verification key $\mathsf{vk} = (\mathsf{vk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{out}})$, a message $x \in \{0,1\}$, and a signature $\sigma = (\sigma^{\mathsf{pk}}, \sigma^{\mathsf{m}})$, the verification algorithm parses $\sigma^{\mathsf{pk}} = (\vec{\sigma}_{\mathsf{out}}, \mathsf{pk}_{\mathsf{in}})$, $\sigma^{\mathsf{m}} = \sigma_{\mathsf{in}}$, and accepts if

$$\mathsf{Verify}_{\mathsf{out}}(\overrightarrow{\mathsf{pk}}_{\mathsf{out}}, \mathsf{vk}_{\mathsf{out}}, \mathsf{pk}_{\mathsf{in}}, \vec{\sigma}_{\mathsf{out}}) = 1 \quad \text{and} \quad \mathsf{Verify}_{\mathsf{in}}(\mathsf{pk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{in}}, x, \sigma_{\mathsf{in}}) = 1.$$

Otherwise, it rejects.

- $\mathsf{VerifyFresh}(\mathsf{pk}, \mathsf{vk}, x, \sigma) \to \{0,1\}$: On input a public key $\mathsf{pk} = \overrightarrow{\mathsf{pk}}_{\mathsf{out}}$, a verification key $\mathsf{vk} = (\mathsf{vk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{out}})$, a message $x \in \{0,1\}$, and signature $\sigma = (\sigma^{\mathsf{pk}}, \sigma^{\mathsf{m}})$, the fresh verification algorithm parses $\sigma^{\mathsf{pk}} = (\vec{\sigma}_{\mathsf{out}}, \mathsf{pk}_{\mathsf{in}})$, $\sigma^{\mathsf{m}} = \sigma_{\mathsf{in}}$, and accepts if

$$\mathsf{VerifyFresh}_{\mathsf{out}}(\overrightarrow{\mathsf{pk}}_{\mathsf{out}}, \mathsf{vk}_{\mathsf{out}}, \mathsf{pk}_{\mathsf{in}}, \vec{\sigma}_{\mathsf{out}}) = 1 \quad \text{and} \quad \mathsf{VerifyFresh}_{\mathsf{in}}(\mathsf{pk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{in}}, x, \sigma_{\mathsf{in}}) = 1.$$

Otherwise, it rejects.

- $\mathsf{VerifyHide}(\mathsf{pk}, \mathsf{vk}, x, \sigma^*) \to \{0,1\}$: On input a public key $\mathsf{pk} = \overrightarrow{\mathsf{pk}}_{\mathsf{out}}$, a verification key $\mathsf{vk} = (\mathsf{vk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{out}})$, a message $x \in \{0,1\}$, and signature $\sigma^* = (\sigma^{\mathsf{pk}}, \sigma^{\mathsf{m}})$, the hide verification algorithm parses $\sigma^{\mathsf{pk}} = (\vec{\sigma}_{\mathsf{out}}, \mathsf{pk}_{\mathsf{in}})$, $\sigma^{\mathsf{m}} = \sigma_{\mathsf{in}}$, and accepts if

$$\mathsf{Verify}_{\mathsf{out}}(\overrightarrow{\mathsf{pk}}_{\mathsf{out}}, \mathsf{vk}_{\mathsf{out}}, \mathsf{pk}_{\mathsf{in}}, \vec{\sigma}_{\mathsf{out}}) = 1 \quad \text{and} \quad \mathsf{VerifyHide}_{\mathsf{in}}(\mathsf{pk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{in}}, x, \sigma_{\mathsf{in}}) = 1.$$

Otherwise, it rejects.

Theorem B.2 (Correctness)**.** *Suppose* $\Pi_{\mathsf{HS},\mathsf{in}}$ *and* $\Pi_{\mathsf{HS},\mathsf{out}}$ *satisfy signing correctness (Definition 3.2), evaluation correctness (Definition 3.3), and hiding correctness (Definition 3.4). Then, Construction B.1 satisfies signing correctness, evaluation correctness, and hiding correctness.*

Proof. Follows by construction. □

Theorem B.3 (Unforgeability)**.** *Suppose* $\Pi_{\mathsf{HS},\mathsf{in}}$ *and* $\Pi_{\mathsf{HS},\mathsf{out}}$ *satisfy selective-unforgeability (Definition 3.5). Then, Construction B.1 satisfies unforgeability (Definition 3.3).*

Proof. Follows from [GVW15, §4]. □

Theorem B.4 (Context-Hiding)**.** *Suppose* $\Pi_{\mathsf{HS},\mathsf{in}}$ *satisfies context-hiding (Definition 3.9). Then, Construction B.1 satisfies context-hiding.*

Proof. Let $\mathcal{S}_{\mathsf{in}} = (\mathcal{S}^{\mathsf{Ext}}_{\mathsf{in}}, \mathcal{S}^{\mathsf{Gen}}_{\mathsf{in}})$ be the context-hiding simulator for $\Pi_{\mathsf{HS},\mathsf{in}}$. We construct a context-hiding simulator $\mathcal{S} = (\mathcal{S}^{\mathsf{Ext}}, \mathcal{S}^{\mathsf{Gen}})$ for Π_{HS} as follows:

- $\mathcal{S}^{\mathsf{Ext}}(\mathsf{pk}, \mathsf{vk}, (\tilde{x}_0, \tilde{\sigma}_0), (\tilde{x}_1, \tilde{\sigma}_1))$: On input a public key pk, a verification key $\mathsf{vk} = (\mathsf{vk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{out}})$, and two message-signature pairs $(\tilde{x}_0, \tilde{\sigma}_0)$, $(\tilde{x}_1, \tilde{\sigma}_1)$, the simulator first parses $\tilde{\sigma}_0 = (\tilde{\sigma}_0^{\mathsf{pk}}, \tilde{\sigma}_0^{\mathsf{m}})$, and $\tilde{\sigma}_1 = (\tilde{\sigma}_1^{\mathsf{pk}}, \tilde{\sigma}_1^{\mathsf{m}})$. Then, it parses $\tilde{\sigma}_0^{\mathsf{pk}} = (\vec{\sigma}_{\mathsf{out}}, \mathsf{pk}_{\mathsf{in}}) = \tilde{\sigma}_1^{\mathsf{pk}}$, $\tilde{\sigma}_0^{\mathsf{m}} = \tilde{\sigma}_{\mathsf{in},0}$, and $\tilde{\sigma}_1^{\mathsf{m}} = \tilde{\sigma}_{\mathsf{in},1}$. Finally, it outputs the trapdoor $\mathsf{td} \leftarrow \mathcal{S}_{\mathsf{in}}^{\mathsf{Ext}}(\mathsf{pk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{in}}, (\tilde{x}_0, \tilde{\sigma}_{\mathsf{in},0}), (\tilde{x}_1, \tilde{\sigma}_{\mathsf{in},1}))$.
- $\mathcal{S}^{\mathsf{Gen}}(\mathsf{pk}, \mathsf{vk}, \mathsf{td}, x, \sigma^{\mathsf{pk}})$: On input a public key pk, a verification key $\mathsf{vk} = (\mathsf{vk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{out}})$, a trapdoor td, a message $x \in \{0,1\}$, and a message-independent signature component $\sigma^{\mathsf{pk}} = (\vec{\sigma}_{\mathsf{out}}, \mathsf{pk}_{\mathsf{in}})$, the simulator computes $\sigma_{\mathsf{in}}^* \leftarrow \mathcal{S}_{\mathsf{in}}^{\mathsf{Gen}}(\mathsf{pk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{in}}, \mathsf{td}, x, \varepsilon)$ and outputs $\sigma^* \leftarrow (\sigma^{\mathsf{pk}}, \sigma_{\mathsf{in}}^*)$. Here, we rely on the assumption that the signatures in $\Pi_{\mathsf{HS},\mathsf{in}}$ have an empty message-independent component.

We now show that if $\Pi_{\mathsf{HS},\mathsf{in}}$ is context-hiding, then experiments $\mathsf{Expt}^{\mathsf{ch}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda, 0)$ and $\mathsf{Expt}^{\mathsf{ch}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda, 1)$ are indistinguishable for any unbounded adversary $\mathcal{A}$.

- Let pk be the public key, $\mathsf{vk} = (\mathsf{vk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{out}})$ be the verification key, and $(\tilde{x}_0, \tilde{\sigma}_0)$, $(\tilde{x}_1, \tilde{\sigma}_1)$ be the message-signature pairs that $\mathcal{A}$ submits to the context-hiding challenger at the beginning of the experiment. Write $\tilde{\sigma}_0 = (\tilde{\sigma}_0^{\mathsf{pk}}, \tilde{\sigma}_0^{\mathsf{m}})$ and $\tilde{\sigma}_1 = (\tilde{\sigma}_1^{\mathsf{pk}}, \tilde{\sigma}_1^{\mathsf{m}})$, where $\tilde{\sigma}_0^{\mathsf{m}} = \tilde{\sigma}_{\mathsf{in},0}$ and $\tilde{\sigma}_1^{\mathsf{m}} = \tilde{\sigma}_{\mathsf{in},1}$ Without loss of generality, we can assume that $\tilde{x}_0 \neq \tilde{x}_1$, $\tilde{\sigma}_0^{\mathsf{pk}} = \tilde{\sigma}_1^{\mathsf{pk}}$, and that $\mathsf{Verify}_{\mathsf{in}}(\mathsf{pk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{in}}, \tilde{x}_0, \tilde{\sigma}_{\mathsf{in},0}) = 1 = \mathsf{Verify}_{\mathsf{in}}(\mathsf{pk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{in}}, \tilde{x}_1, \tilde{\sigma}_{\mathsf{in},1})$. Otherwise, the output of the experiment is always 0, and the adversary's distinguishing advantage is correspondingly 0. Next, in $\mathsf{Expt}^{\mathsf{ch}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda, 1)$, the challenger constructs a trapdoor td by invoking $\mathsf{td} \leftarrow \mathcal{S}_{\mathsf{in}}^{\mathsf{Ext}}(\mathsf{pk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{in}}, (\tilde{x}_0, \tilde{\sigma}_{\mathsf{in},0}), (\tilde{x}_1, \tilde{\sigma}_{\mathsf{in},1}))$.
- Let pk', $x \in \{0,1\}$, $\sigma = (\sigma^{\mathsf{pk}}, \sigma^{\mathsf{m}})$ be a query $\mathcal{A}$ makes to the challenger. If $\mathsf{Verify}(\mathsf{pk}', \mathsf{vk}, x, \sigma) = 1$, then the challenger proceeds as follows:
  - In $\mathsf{Expt}^{\mathsf{ch}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda, 0)$, the challenger parses $\sigma^{\mathsf{m}} = \sigma_{\mathsf{in}}$, and computes $\sigma_{\mathsf{in}}^* \leftarrow \mathsf{Hide}_{\mathsf{in}}(\mathsf{vk}_{\mathsf{in}}, x, \sigma_{\mathsf{in}})$. It replies to the adversary with $\sigma^* \leftarrow (\sigma^{\mathsf{pk}}, \sigma_{\mathsf{in}}^*)$.
  - In $\mathsf{Expt}^{\mathsf{ch}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda, 1)$, the challenger parses $\sigma^{\mathsf{pk}} = (\vec{\sigma}_{\mathsf{out}}, \mathsf{pk}_{\mathsf{in}})$, and computes $\sigma_{\mathsf{in}}^* \leftarrow \mathcal{S}^{\mathsf{Gen}}(\mathsf{pk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{in}}, \mathsf{td}, x, \varepsilon)$. It returns $\sigma^* = (\sigma^{\mathsf{pk}}, \sigma_{\mathsf{in}}^*)$.

Since $\mathsf{Verify}(\mathsf{pk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{in}}, \tilde{x}_0, \tilde{\sigma}_{\mathsf{in},0}) = 1 = \mathsf{Verify}(\mathsf{pk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{in}}, \tilde{x}_1, \tilde{\sigma}_{\mathsf{in},1})$, and $\tilde{\sigma}_0^{\mathsf{pk}} = \tilde{\sigma}_1^{\mathsf{pk}}$, we have that td is a valid trapdoor for $\mathcal{S}_{\mathsf{in}}^{\mathsf{Ext}}$. Since $\Pi_{\mathsf{HS},\mathsf{in}}$ is context-hiding, the message-dependent component σ_{in}^* of the final signature σ^* generated by $\mathcal{S}^{\mathsf{Gen}}$ in $\mathsf{Expt}^{\mathsf{ch}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda, 1)$ is statistically indistinguishable from σ_{in}^* generated by the challenger in $\mathsf{Expt}^{\mathsf{ch}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda, 0)$. The claim follows. □

Theorem B.5 (Compactness)**.** *Fix a security parameter λ. Suppose $\Pi_{\mathsf{HS},\mathsf{in}}$ and $\Pi_{\mathsf{HS},\mathsf{out}}$ satisfy compactness (Definition 3.10), and moreover, the size of a homomorphically-evaluated public key output by $\mathsf{PrmsEval}_{\mathsf{out}}(C, \cdot)$ is $\mathsf{poly}(\lambda, d)$, where d is a bound on the depth of the circuit C. Then, Construction B.1 satisfies compactness.*

Proof. Follows immediately by construction. Specifically, the signature output by $\mathsf{SigEval}$ consists of compact signatures output by $\mathsf{SigEval}_{\mathsf{out}}$ and $\mathsf{SigEval}_{\mathsf{in}}$, and a homomorphically-evaluated public key output by $\mathsf{PrmsEval}_{\mathsf{out}}$. Therefore, the size of the signatures depend only on $|C(\vec{x})|$ and is independent of $|\vec{x}|$. □

Instantiating the construction. We note that both $\Pi_{\mathsf{HS},\mathsf{in}}$ and $\Pi_{\mathsf{HS},\mathsf{out}}$ can be instantiated by Construction 3.11 in Section 3. In particular, Construction 3.11 satisfies the additional compactness requirement on the size of the public keys needed in Theorem B.5. This yields the following corollary:

Corollary B.6 (Adaptively-Secure Homomorphic Signatures)**.** *Fix a security parameter λ and a message length $\ell = \mathsf{poly}(\lambda)$. Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ be a function class where $\mathcal{C}_\lambda$ consists of Boolean circuits of depth up to $d = d(\lambda)$ on ℓ-bit inputs. Then, under the SIS assumption, there exists a homomorphic signature scheme $\Pi_{\mathsf{HS}} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrmsEval}, \mathsf{SigEval}, \mathsf{Hide}, \mathsf{Verify}, \mathsf{VerifyFresh}, \mathsf{VerifyHide})$ with message space $\{0,1\}$, message length ℓ, and function class $\mathcal{C}$ that satisfies adaptive unforgeability (Definition 3.5), context-hiding (Definition 3.9), and compactness (Definition 3.10).*

C Proof of Theorem 4.4

We show completeness, soundness, zero-knowledge separately.

Completeness. Take any statement x and witness w where $\mathcal{R}(x,w) = 1$. Let $(k_P, k_V) \leftarrow \mathsf{Setup}(1^\lambda)$, where $k_P = (k_{\mathsf{SE}}, \overrightarrow{\mathsf{pk}}_{\mathsf{HS}}, \mathsf{vk}_{\mathsf{HS}}, \vec{\sigma}_k)$. Take $(\mathsf{ct}, \sigma^*_{x,\mathsf{ct}}) \leftarrow \mathsf{Prove}(k_P, x, w)$. By correctness of Π_{SE},

$$C_{x,\mathsf{ct}}(k_{\mathsf{SE}}) = \mathcal{R}(x, \mathsf{SE.Decrypt}(k_{\mathsf{SE}}, \mathsf{ct})) = \mathcal{R}(x,w) = 1.$$

Completeness of Π_{NIZK} then follows from evaluation correctness (Definition 3.3) and hiding correctness (Definition 3.4) of Π_{HS}.

Soundness. At a high-level, soundness follows from (selective) unforgeability of Π_{HS} (Definition 3.5, Remark 3.6). An adversary that succeeds in breaking soundness must produce a statement $x \notin \mathcal{L}$, a ciphertext ct and a signature $\sigma^*_{x,\mathsf{ct}}$ on the message 1 with respect to the function $C_{x,\mathsf{ct}}$. Since $x \notin \mathcal{L}$, there does not exist any witness $w \in \{0,1\}^m$ where $\mathcal{R}(x,w) = 1$, which means that there are no inputs to $C_{x,\mathsf{ct}}$ where the output is 1. More formally, suppose there is an adversary $\mathcal{A}$ that breaks soundness of Π_{NIZK} with advantage ε. We use $\mathcal{A}$ to construct an adversary that breaks selective unforgeability of $\mathcal{B}$. Algorithm $\mathcal{B}$ works as follows:

1. At the beginning of the selective unforgeability game, algorithm $\mathcal{B}$ generates a secret key $k_{\mathsf{SE}} \leftarrow \mathsf{SE.KeyGen}(1^\lambda)$, and sends k_{SE} to the challenger. The challenger replies with the public parameters $\overrightarrow{\mathsf{pk}}_{\mathsf{HS}}$, the verification key $\mathsf{vk}_{\mathsf{HS}}$ and a signature $\vec{\sigma}_k$.
2. Algorithm $\mathcal{B}$ sets $k_P = (k_{\mathsf{SE}}, \overrightarrow{\mathsf{pk}}_{\mathsf{HS}}, \mathsf{vk}_{\mathsf{HS}}, \vec{\sigma}_k)$ and gives k_P to $\mathcal{A}$.
3. Whenever $\mathcal{A}$ makes an oracle query to the verification oracle, algorithm $\mathcal{B}$ answers according to the specification in Construction 4.3. Note that the verification algorithm only depends on $\overrightarrow{\mathsf{pk}}_{\mathsf{HS}}$ and $\mathsf{vk}_{\mathsf{HS}}$, both of which are known to $\mathcal{B}$ (and in fact $\mathcal{A}$). Notably, the secret signing key $\mathsf{sk}_{\mathsf{HS}}$ is not needed to run Verify.
4. At the end of the game, when $\mathcal{A}$ outputs a statement x and a proof $\pi = (\mathsf{ct}, \sigma^*_{x,\mathsf{ct}})$, algorithm $\mathcal{B}$ gives the circuit $C_{x,\mathsf{ct}}$, the message 1, and the signature $\sigma^*_{x,\mathsf{ct}}$ to the challenger.

By construction, algorithm $\mathcal{B}$ *perfectly* simulates the prover key for $\mathcal{A}$. Thus, with probability ε, algorithm $\mathcal{A}$ outputs $x \notin \mathcal{L}$ such that $\sigma^*_{x,\mathsf{ct}}$ is a valid signature on the message 1 with respect to the function $C_{x,\mathsf{ct}}$. By definition, $C_{x,\mathsf{ct}}(k_{\mathsf{SE}}) = 0$, so $\sigma^*_{x,\mathsf{ct}}$ is a valid forgery. Soundness follows.

Zero-Knowledge. At a high-level, zero-knowledge follows by CPA-security of the encryption scheme and weak context-hiding of the homomorphic signature scheme. Since Π_{HS} is weak context-hiding (Definition 3.8), there exists an efficient simulator $\mathcal{S}_{\mathsf{ch}}$ that can simulate the signatures output by the Hide algorithm. We use $\mathcal{S}_{\mathsf{ch}}$ to construct the zero-knowledge simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$:

- On input the security parameter λ and the verification state $k_V = (\overrightarrow{\mathsf{pk}}_{\mathsf{HS}}, \mathsf{vk}_{\mathsf{HS}}, \mathsf{sk}_{\mathsf{HS}})$ where $\overrightarrow{\mathsf{pk}}_{\mathsf{HS}} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_\rho)$, algorithm $\mathcal{S}_1$ samples a secret key $k_{\mathsf{SE}} \leftarrow \mathsf{SE.KeyGen}(1^\lambda)$. Next, it computes $\vec{\sigma}_k^{\mathsf{pk}} \leftarrow \mathsf{SignPK}(\overrightarrow{\mathsf{pk}}, \mathsf{sk}_{\mathsf{HS}})$, and outputs the state $\tau_V = (k_{\mathsf{SE}}, \vec{\sigma}_k^{\mathsf{pk}})$.

- On input the verification state $k_V = (\overrightarrow{\mathsf{pk}}_{\mathsf{HS}}, \mathsf{vk}_{\mathsf{HS}}, \mathsf{sk}_{\mathsf{HS}})$, the simulation state $\tau_V = (k_{\mathsf{SE}}, \vec{\sigma}_k^{\mathsf{pk}})$, and a statement $x \in \{0,1\}^n$, the simulator algorithm $\mathcal{S}_2$ begins by constructing a ciphertext $\mathsf{ct} \leftarrow \mathsf{SE.Encrypt}(k_{\mathsf{SE}}, 0^m)$. Then, it computes $\mathsf{pk}_{x,\mathsf{ct}} \leftarrow \mathsf{PrmsEval}(C_{x,\mathsf{ct}}, \overrightarrow{\mathsf{pk}}_{\mathsf{HS}})$, $\sigma_{x,\mathsf{ct}}^{\mathsf{pk}} \leftarrow \mathsf{SigEvalPK}(C_{x,\mathsf{ct}}, \overrightarrow{\mathsf{pk}}_{\mathsf{HS}}, \vec{\sigma}_k^{\mathsf{pk}})$, and finally, it simulates the signature by computing $\sigma_{x,\mathsf{ct}}^{\mathsf{m}} \leftarrow \mathcal{S}_{\mathsf{ch}}(\mathsf{pk}_{x,\mathsf{ct}}, \mathsf{vk}_{\mathsf{HS}}, \mathsf{sk}_{\mathsf{HS}}, 1, \sigma_{x,\mathsf{ct}}^{\mathsf{pk}})$, and outputs the simulated proof $\pi = (\mathsf{ct}, \sigma_{x,\mathsf{ct}}^*)$, where $\sigma_{x,\mathsf{ct}}^* = (\sigma_{x,\mathsf{ct}}^{\mathsf{pk}}, \sigma_{x,\mathsf{ct}}^{\mathsf{m}})$.

To complete the proof, we use a hybrid argument:

- HYB_0: This is the experiment where the adversary has access to $\mathcal{O}_0$, where $\mathcal{O}_0(k_P, x, w) := \mathsf{Prove}(k_P, x, w)$.

- HYB_1: Same as HYB_0, except the $\mathsf{Prove}(k_P, x, w)$ queries are handled as follows:

  1. The challenger first computes $\mathsf{ct} \leftarrow \mathsf{SE.Encrypt}(k_{\mathsf{SE}}, w)$.
  2. Next, it computes the public key $\mathsf{pk}_{x,\mathsf{ct}} \leftarrow \mathsf{PrmsEval}(C_{x,\mathsf{ct}}, \overrightarrow{\mathsf{pk}}_{\mathsf{HS}})$, a public signature component $\sigma_{x,\mathsf{ct}}^{\mathsf{pk}} \leftarrow \mathsf{SigEvalPK}(C_{x,\mathsf{ct}}, \overrightarrow{\mathsf{pk}}_{\mathsf{HS}}, \vec{\sigma}_k^{\mathsf{pk}})$, and a simulated signature $\sigma_{x,\mathsf{ct}}^{\mathsf{m}} \leftarrow \mathcal{S}_{\mathsf{ch}}(\mathsf{pk}_{x,\mathsf{ct}}, \mathsf{vk}_{\mathsf{HS}}, \mathsf{sk}_{\mathsf{HS}}, 1, \sigma_{x,\mathsf{ct}}^{\mathsf{pk}})$. Here $\vec{\sigma}_k = (\vec{\sigma}_k^{\mathsf{pk}}, \vec{\sigma}_{\mathsf{sk}}^{\mathsf{m}})$ is the signature on k_{SE} the challenger generated from Setup (and is part of the proving key k_P).
  3. Finally, the challenger responds with $\pi = (\mathsf{ct}, \sigma_{x,\mathsf{ct}}^*)$, where $\sigma_{x,\mathsf{ct}}^* = (\sigma_{x,\mathsf{ct}}^{\mathsf{pk}}, \sigma_{x,\mathsf{ct}}^{\mathsf{m}})$.

- HYB_2: Same as HYB_1, except the challenger replaces the encryption of w with an encryption of 0^m when answering the $\mathsf{Prove}(k_P, x, w)$ queries.

- HYB_3: This is the experiment where the adversary has access to $\mathcal{O}_1$, where $\mathcal{O}_1(k_V, \tau_V, x, w) := \mathcal{S}_2(k_V, \tau_V, x)$.

We now briefly argue that each pair of hybrids are computationally indistinguishable:

- Hybrids HYB_0 and HYB_1 are computationally indistinguishable by weak context-hiding security of Π_{HS}. Specifically, if $\mathcal{A}$ is able to distinguish HYB_0 and HYB_1, then we can construct an adversary $\mathcal{B}$ that breaks context-hiding as follows:

  1. At the beginning of the game, algorithm $\mathcal{B}$ receives a signing and a verification key $(\mathsf{vk}_{\mathsf{HS}}, \mathsf{sk}_{\mathsf{HS}})$ from the challenger. It then samples parameters $\overrightarrow{\mathsf{pk}}_{\mathsf{HS}} \leftarrow \mathsf{PrmsGen}(1^\lambda, 1^\rho)$, a symmetric key $k_{\mathsf{SE}} \leftarrow \mathsf{SE.KeyGen}(1^\lambda)$ and a signature $\vec{\sigma}_k \leftarrow \mathsf{Sign}(\overrightarrow{\mathsf{pk}}_{\mathsf{HS}}, \mathsf{sk}_{\mathsf{HS}}, k_{\mathsf{SE}})$. Algorithm $\mathcal{B}$ constructs the verification key $k_V = (\overrightarrow{\mathsf{pk}}_{\mathsf{HS}}, \mathsf{vk}_{\mathsf{HS}}, \mathsf{sk}_{\mathsf{HS}})$ and sends it to $\mathcal{A}$.
  2. When $\mathcal{A}$ makes an oracle query on a pair (x, w) where $\mathcal{R}(x, w) = 1$, algorithm $\mathcal{B}$ simulates the response by first computing $\mathsf{ct} \leftarrow \mathsf{SE.Encrypt}(k_{\mathsf{SE}}, w)$. Next, it computes $\sigma_{x,\mathsf{ct}} \leftarrow \mathsf{SigEval}(C_{x,\mathsf{ct}}, \overrightarrow{\mathsf{pk}}_{\mathsf{HS}}, k_{\mathsf{SE}}, \vec{\sigma}_k)$ and parses the result as $\sigma_{x,\mathsf{ct}} = (\sigma_{x,\mathsf{ct}}^{\mathsf{pk}}, \sigma'_{x,\mathsf{ct}})$. It also computes $\mathsf{pk}_{x,\mathsf{ct}} \leftarrow \mathsf{PrmsEval}(C_{x,\mathsf{ct}}, \overrightarrow{\mathsf{pk}}_{\mathsf{HS}})$, and sends the public key $\mathsf{pk}_{x,\mathsf{ct}}$, the message 1, and the signature $(\sigma_{x,\mathsf{ct}}^{\mathsf{pk}}, \sigma'_{x,\mathsf{ct}})$ to the context-hiding challenger. The challenger replies with a refreshed signature $\sigma_{x,\mathsf{ct}}^*$. Algorithm $\mathcal{B}$ responds to the query with $(\mathsf{ct}, \sigma_{x,\mathsf{ct}}^*)$.

3. At the end of the experiment, $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

By construction, if the signatures returned by the context-hiding challenger are generated using the Hide algorithm, then $\mathcal{B}$ perfectly simulates HYB_0, while if the signatures are generated using the simulator, then $\mathcal{B}$ perfectly simulates HYB_1. Indistinguishability of the two hybrids thus follows by context-hiding.

- Hybrids HYB_1 and HYB_2 are computationally indistinguishable by CPA-security of Π_{SE}. Specifically, the challenger's logic in HYB_1 and HYB_2 does not depend on k_{SE}, so we can simulate the two hybrid experiments given access to an encryption oracle. Note that the signature component $\vec{\sigma}_k^{\mathsf{pk}}$ needed to respond to queries in HYB_1 and HYB_2 is only the *public* component of the signature (and can be generated without knowledge of the actual secret key k_{SE}).

- Hybrids HYB_2 and HYB_3 are identical experiments. Namely, the behavior of the challenger in HYB_2 precisely coincides with the behavior in the experiment where the adversary is given access to the oracle $\mathcal{O}_1(k_V, \tau_V, x, w) := \mathcal{S}_2(k_V, \tau_V, x)$.

Since each pair of hybrid experiments are computationally indistinguishable, we conclude that Π_{NIZK} provides zero-knowledge. □

D Proof of Theorem 5.1

Let $\mathcal{A}$ be a static adversary that interacts with the environment $\mathcal{Z}$, a signer $\mathbf{S}$, and receiver $\mathbf{R}$ running the real protocol Π_{BHS} (Figure 2). We construct an ideal world adversary (simulator) $\mathcal{S}$ that interacts with the environment $\mathcal{Z}$, the ideal functionality $\mathcal{F}_{\text{BHS}}$, and dummy parties $\tilde{\mathbf{S}}$, $\tilde{\mathbf{R}}$ such that no environment $\mathcal{Z}$ can distinguish an interaction with $\mathcal{A}$ in the real protocol from one with $\mathcal{S}$ in the ideal world.

We begin by describing the simulator $\mathcal{S}$. At the beginning of the protocol execution, the simulator $\mathcal{S}$ begins by simulating an execution of Π_{BHS} with adversary $\mathcal{A}$. In particular, $\mathcal{S}$ simulates the environment $\mathcal{Z}$, the behavior of the honest parties, as well as the ideal OT functionality $\mathcal{F}_{\text{OT}}^{\ell,s}$ in the simulated protocol execution with $\mathcal{A}$. Algorithm $\mathcal{A}$ begins by declaring which parties it wants to corrupt, and $\mathcal{S}$ corrupts the analogous set of dummy parties in the ideal execution (e.g., if $\mathcal{A}$ corrupts the signer $\mathbf{S}$, then $\mathcal{S}$ corrupts the dummy signer $\tilde{\mathbf{S}}$). The simulation then proceeds as follows.

Simulating the communication with the environment. Whenever the simulator $\mathcal{S}$ receives an input from the environment $\mathcal{Z}$, it forwards the input to $\mathcal{A}$ (as if it came from the environment in the simulated protocol execution). Whenever $\mathcal{A}$ writes a message on its output tape (in the simulated protocol execution), the simulator $\mathcal{S}$ writes the same output on its own output tape (to be read by the environment).

Simulating the key-generation phase. In the key-generation phase, the simulator $\mathcal{S}$ proceeds as follows, depending on whether the signer $\tilde{\mathbf{S}}$ is corrupt:

- *The signer is honest.* When $\mathcal{S}$ receives a value $(\mathsf{sid}, \mathsf{keygen})$ from $\mathcal{F}_{\text{BHS}}$, the simulator generates $\overrightarrow{\mathsf{pk}} \leftarrow \mathsf{PrmsGen}(1^\lambda, 1^{t\ell})$, $(\mathsf{sk}, \mathsf{vk}') \leftarrow \mathsf{KeyGen}(1^\lambda)$, and stores $(\mathsf{sid}, \mathsf{sk})$. It sets $\mathsf{vk} = (\overrightarrow{\mathsf{pk}}, \mathsf{vk}')$, and sends $(\mathsf{sid}, \mathsf{vkey}, \mathsf{vk})$ to $\mathcal{F}_{\text{BHS}}$.

- *The signer is corrupt.* When $\mathcal{Z}$ activates a corrupt signer $\tilde{\mathbf{S}}$ on input $(\mathsf{sid}, \mathsf{keygen})$, $\mathcal{S}$ activates the signer $\mathbf{S}$ with the same input $(\mathsf{sid}, \mathsf{keygen})$ in its simulated copy of Π_{BHS}. Let $(\mathsf{sid}, \mathsf{vkey}, \mathsf{vk})$ be the verification key output by $\mathbf{S}$ (as decided by $\mathcal{A}$). The simulator $\mathcal{S}$ then sends a request $(\mathsf{sid}, \mathsf{keygen})$ to $\mathcal{F}_{\text{BHS}}$ (on behalf of $\tilde{\mathbf{S}}$), and responds to the key-generation request from $\mathcal{F}_{\text{BHS}}$ with the tuple $(\mathsf{sid}, \mathsf{vkey}, \mathsf{vk})$.

Simulating the signature-generation phase. The simulator $\mathcal{S}$ simulates the signing protocol as follows, depending on whether the signer $\tilde{\mathbf{S}}$ is corrupt:

- *The signer is honest.* We first describe how the simulator $\mathcal{S}$ constructs the ideal algorithms $(\mathsf{IdealSign}, \mathsf{IdealEval})$ when it receives a query $(\mathsf{sid}, \mathsf{signature})$ from $\mathcal{F}_{\text{BHS}}$. Let $\mathsf{vk} = (\overrightarrow{\mathsf{pk}}, \mathsf{vk}')$ and sk be the parameters the simulator sampled in the key-generation phase (since $\tilde{\mathbf{S}}$ is honest, the simulator chose the secret signing key). The simulator then defines the $\mathsf{IdealSign}$ and $\mathsf{IdealEval}$ algorithms (with $\overrightarrow{\mathsf{pk}}, \mathsf{vk}', \mathsf{sk}$ hard-wired) as follows:
  - $\mathsf{IdealSign}(\vec{x})$: On input $\vec{x} \in \{0,1\}^\ell$:
    1. Sample shares $\vec{w}_1, \ldots, \vec{w}_t \xleftarrow{\text{R}} \{0,1\}^\ell$ such that $\bigoplus_{i\in[t]} \vec{w}_i = \vec{x}$.
    2. Generate $(\vec{\sigma}_1, \ldots, \vec{\sigma}_t) \leftarrow \mathsf{Sign}\big(\overrightarrow{\mathsf{pk}}, \mathsf{sk}, (\vec{w}_1, \ldots, \vec{w}_t)\big)$.
    3. Return $\mathsf{SigEval}\big(f_{\mathsf{recon}}, \overrightarrow{\mathsf{pk}}, (\vec{w}_1, \ldots, \vec{w}_t), (\vec{\sigma}_1, \ldots, \vec{\sigma}_t)\big)$.
  - $\mathsf{IdealEval}(g, x)$: On input a function $g \in \mathcal{H}$ and a value $x \in \{0,1\}$:
    1. Compute $\mathsf{pk}_g \leftarrow \mathsf{PrmsEval}(g \circ f_{\mathsf{recon}}, \overrightarrow{\mathsf{pk}})$.
    2. Sign $\sigma \leftarrow \mathsf{Sign}(\mathsf{pk}_g, \mathsf{sk}, x)$.
    3. Return $\mathsf{Hide}(\mathsf{vk}', x, \sigma)$.

  The simulator replies to $\mathcal{F}_{\text{BHS}}$ with $(\mathsf{IdealSign}, \mathsf{IdealEval})$. If the receiver is honest, then this completes the simulation for the signing request. Conversely, if the receiver is corrupt, then the simulator $\mathcal{S}$ proceeds as follows:
  - When $\mathcal{Z}$ activates the receiver $\tilde{\mathbf{R}}$ on input $(\mathsf{sid}, \mathsf{sign}, \mathsf{vk}, \vec{x})$, the simulator forwards $(\mathsf{sid}, \mathsf{sign}, \mathsf{vk}, \vec{x})$ to $\mathbf{R}$ (which is under the control of $\mathcal{A}$) in the simulated protocol execution (as if it came from $\mathcal{A}$'s environment).
  - After $\mathbf{R}$ sends inputs $\big((\mathsf{sid}, i), \mathsf{receiver}, \vec{w}_i\big)$ for all $i \in [t]$ to the ideal OT functionality $\mathcal{F}_{\text{OT}}^{\ell,s}$ in the simulated protocol execution, the simulator computes $\vec{x} \leftarrow \bigoplus_{i\in[t]} \vec{w}_i$. If this is *not* the first signing request from $\mathbf{R}$, then the simulator ignores the request. Otherwise, the simulator sends $(\mathsf{sid}, \mathsf{sign}, \mathsf{vk}, \vec{x})$ to $\mathcal{F}_{\text{BHS}}$.
  - When $\mathcal{F}_{\text{BHS}}$ sends $(\mathsf{sid}, \mathsf{sign}, x)$ to $\mathcal{S}$ to choose the signature on behalf of $\tilde{\mathbf{R}}$, the simulator constructs signatures $\vec{\sigma}_i \leftarrow \mathsf{Sign}(\overrightarrow{\mathsf{pk}_i}, \mathsf{sk}, \vec{w}_i)$ and sends $\big((\mathsf{sid}, i), \vec{\sigma}_i\big)$ to $\mathbf{R}$ for $i \in [t]$. For the message-independent components of the signatures, $\mathcal{S}$ parses $\vec{\sigma}_i = (\vec{\sigma}_i^{\mathsf{pk}}, \vec{\sigma}_i^{\mathsf{m}})$ for $i \in [t]$, and sends $\{\vec{\sigma}_i^{\mathsf{pk}}\}_{i\in[t]}$ to $\mathbf{R}$. The simulator also computes $\vec{\sigma} \leftarrow \mathsf{SigEval}(f_{\mathsf{recon}}, \overrightarrow{\mathsf{pk}}, (\vec{w}_1, \ldots, \vec{w}_t), (\vec{\sigma}_1, \ldots, \vec{\sigma}_t))$, and sends $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \vec{x}), \vec{\sigma})$, where $\vec{x} = \bigoplus_{i\in[t]} \vec{w}_i$, to $\mathcal{F}_{\text{BHS}}$.
- *The signer is corrupt.* If the receiver $\tilde{\mathbf{R}}$ is also corrupt, then $\mathcal{S}$ determines the behavior of $\tilde{\mathbf{S}}$ and $\tilde{\mathbf{R}}$ using $\mathcal{A}$ (who controls the behavior of $\mathbf{S}$ and $\mathbf{R}$ in the simulated protocol execution). Specifically, the simulator proceeds as follows:

– When the environment activates $\tilde{\mathbf{R}}$ with an input $(\mathsf{sid}, \mathsf{sign}, \mathsf{vk}, \vec{x})$, the simulator activates the receiver $\mathbf{R}$ in its simulated protocol execution with the same input.
– The simulator simulates the ideal OT functionality $\mathcal{F}_{\text{OT}}^{\ell,s}$ in its simulated protocol execution exactly according to the specification of $\mathcal{F}_{\text{OT}}^{\ell,s}$ in Figure 6.
– The simulator echoes any output of $\mathcal{A}$ (to the environment).

Note that in this case where the signer and receiver are both corrupt, the simulator $\mathcal{S}$ *never* interacts with the ideal functionality. Conversely, if the receiver $\tilde{\mathbf{R}}$ is honest, then the simulator proceeds as follows:

– When the ideal functionality sends a query $(\mathsf{sid}, \mathsf{signature})$ to $\mathcal{S}$, the simulator needs to respond with a specification of the ideal signing and evaluation functionalities $\mathsf{IdealSign}$ and $\mathsf{IdealEval}$. The simulator starts by performing several basic checks:
  1. The simulator begins by activating the signer $\mathbf{S}$ with the input $(\mathsf{sid}, \mathsf{signature})$ in its simulated execution of the protocol. Let $\big((\mathsf{sid}, i), \mathsf{sender}, \{(\sigma_{i,j,0}, \sigma_{i,j,1})\}_{j\in[\ell]}\big)$ for $i \in [t]$ be the inputs $\mathbf{S}$ sends to $\mathcal{F}_{\text{OT}}^{\ell,s}$, and let $\{\sigma_{i,j}^{\mathsf{pk}}\}_{i\in[t],j\in[\ell]}$ be the message-independent components $\mathbf{S}$ sends to $\mathbf{R}$ in the simulated protocol execution. Note that in the real protocol execution, the receiver $\mathbf{R}$ only interacts with $\mathcal{F}_{\text{OT}}^{\ell,s}$ and does not send any messages to $\mathbf{S}$ (so $\mathcal{S}$ does not need to simulate any messages on behalf of $\mathbf{R}$).
  2. Let vk be the verification key $\mathcal{S}$ chose during key-generation. The simulator parses the verification key as $\mathsf{vk} = (\overrightarrow{\mathsf{pk}}, \mathsf{vk}')$ where $\overrightarrow{\mathsf{pk}} = \{\mathsf{pk}_{i,j}\}_{i\in[t],j\in[\ell]}$. If the verification key does not have this structure, then the simulator defines the ideal signing and evaluation functions $\mathsf{IdealSign}$ and $\mathsf{IdealEval}$ to always output $\perp$.
  3. Otherwise, the simulator parses $\sigma_{i,j,b} = (\sigma_{i,j,b}^{\mathsf{pk}}, \sigma_{i,j,b}^{\mathsf{m}})$ for $i \in [t]$, $j \in [\ell]$, $b \in \{0,1\}$. We say that a signature $\sigma_{i,j,b}$ is "valid" if
  $$\sigma_{i,j,b}^{\mathsf{pk}} = \sigma_{i,j}^{\mathsf{pk}} \quad \text{and} \quad \mathsf{VerifyFresh}(\mathsf{pk}_{i,j}, \mathsf{vk}', b, \sigma_{i,j,b}) = 1, \tag{D.1}$$
  and otherwise, we say that $\sigma_{i,j,b}$ is "invalid." Then, if there exists indices $i \in [t]$ and $j \in [\ell]$ where $\sigma_{i,j,0}$ and $\sigma_{i,j,1}$ are both invalid, the simulator defines the signing and evaluation functions $\mathsf{IdealSign}$ and $\mathsf{IdealEval}$ to always output $\perp$.
  4. Finally, the simulator checks if for *all* $j \in [\ell]$, there exists $i \in [t]$ where $\sigma_{i,j,0}$ and $\sigma_{i,j,1}$ are both valid. If this is not the case, then $\mathcal{S}$ defines the ideal signing and evaluation functions $\mathsf{IdealSign}$ and $\mathsf{IdealEval}$ to always output $\perp$.

  If all of the checks pass, then there exists i^*, j^* where $\sigma_{i^*,j^*,0}$ and $\sigma_{i^*,j^*,1}$ are both valid. In this case, the simulator uses the context-hiding simulator $\mathcal{S}^{\mathsf{ch}} = (\mathcal{S}^{\mathsf{Ext}}, \mathcal{S}^{\mathsf{Gen}})$ from Definition 3.9 to extract a simulation trapdoor $\mathsf{td} \leftarrow \mathcal{S}^{\mathsf{Ext}}(\mathsf{pk}_{i^*,j^*}, \mathsf{vk}', (0, \sigma_{i^*,j^*,0}), (1, \sigma_{i^*,j^*,1}))$. Then, the simulator defines the functions $(\mathsf{IdealSign}, \mathsf{IdealEval})$ as follows. Note that the public keys $\overrightarrow{\mathsf{pk}}$, the simulation trapdoor td, and the message-independent signature components $\{\sigma_{i,j}^{\mathsf{pk}}\}_{i\in[t],j\in[\ell]}$ are hard-wired in the description of the algorithms.

- $\mathsf{IdealSign}(\vec{x})$: On input $\vec{x} \in \{0,1\}^\ell$:
  1. First, the ideal signing algorithm initializes $\vec{w}_1, \ldots, \vec{w}_t \leftarrow 0^\ell$.
  2. By assumption, for all $i \in [t]$ and $j \in [\ell]$, there is *at least* one $b \in \{0,1\}$ where $\sigma_{i,j,b}$ is valid. Now, for all $i \in [t]$ and $j \in [\ell]$, if there is *exactly* one bit $b \in \{0,1\}$ where $\sigma_{i,j,b}$ is valid, then the simulator sets $w_{i,j} = b$.

3. For all remaining indices $i \in [t]$ and $j \in [\ell]$ where both $\sigma_{i,j,0}$ and $\sigma_{i,j,1}$ are valid, the simulator samples $w_{i,j} \xleftarrow{R} \{0,1\}$, subject to the restriction that $\bigoplus_{i\in[t]} \vec{w}_i = \vec{x}$. Note that this constraint is always satisfiable since for all $j \in [\ell]$, there is at least one $i \in [t]$ where both $\sigma_{i,j,0}$ and $\sigma_{i,j,1}$ are valid by assumption.
4. Then, for all $i \in [t]$, the algorithm sets $\vec{\sigma}_i = (\sigma_{i,1,w_{i,1}}, \ldots, \sigma_{i,\ell,w_{i,\ell}})$, and outputs the signature $\mathsf{SigEval}\big(f_{\mathsf{recon}}, \overrightarrow{\mathsf{pk}}, (\vec{w}_1, \ldots, \vec{w}_t), (\vec{\sigma}_1, \ldots, \vec{\sigma}_t)\big)$.

- $\mathsf{IdealEval}(g, x)$: On input a function $g \in \mathcal{H}$, and a value $x \in \{0,1\}$:
  1. Compute $\mathsf{pk}_g \leftarrow \mathsf{PrmsEval}(g \circ f_{\mathsf{recon}}, \overrightarrow{\mathsf{pk}})$.
  2. Compute $\sigma_g^{\mathsf{pk}} \leftarrow \mathsf{SigEvalPK}\big(g \circ f_{\mathsf{recon}}, \overrightarrow{\mathsf{pk}}, (\vec{\sigma}_1^{\mathsf{pk}}, \ldots, \vec{\sigma}_t^{\mathsf{pk}})\big)$, where $\vec{\sigma}_i^{\mathsf{pk}} = (\sigma_{i,1}^{\mathsf{pk}}, \ldots, \sigma_{i,\ell}^{\mathsf{pk}})$.
  3. Return $\mathcal{S}^{\mathsf{Gen}}(\mathsf{pk}_g, \mathsf{vk}', \mathsf{td}, x, \sigma_g^{\mathsf{pk}})$

– When the ideal functionality sends $(\mathsf{sid}, \mathsf{sig\text{-}success})$ to $\mathcal{S}$, the simulator responds as follows. First, let $\{(\sigma_{i,j,0}, \sigma_{i,j,1})\}_{i\in[t],j\in[\ell]}$ be the set of signatures the signer provided to the ideal OT functionality and $\{\sigma_{i,j}^{\mathsf{pk}}\}_{i\in[t],j\in[\ell]}$ be the set of message-independent public components sent by $\mathbf{S}$ in the simulated protocol execution. As before, we say that $\sigma_{i,j,b}$ is valid if and only if Eq. (D.1) holds. First, if the simulator previously defined $\mathsf{IdealSign}$ and $\mathsf{IdealEval}$ to $\perp$, then it replies with $(\mathsf{sid}, 0)$. Otherwise, let n be the number of indices $i \in [t]$, $j \in [\ell]$, and $b \in \{0,1\}$ where $\sigma_{i,j,b}$ is invalid. Then, with probability $1 - 2^{-n}$, the simulator responds with $(\mathsf{sid}, 0)$. With probability 2^{-n}, the simulator responds with $(\mathsf{sid}, 1)$.

Simulating the signature-verification phase. When the environment activates $\tilde{\mathbf{P}} \in \{\tilde{\mathbf{S}}, \tilde{\mathbf{R}}\}$ on input $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}', (f, \vec{x}), \vec{\sigma})$, the simulator $\mathcal{S}$ proceeds as follows:

- If $\tilde{\mathbf{P}}$ is honest and the simulator $\mathcal{S}$ receives a query $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}', (f, \vec{x}), \vec{\sigma})$ from $\mathcal{F}_{\text{BHS}}$, the simulator first parses $\mathsf{vk}' = (\overrightarrow{\mathsf{pk}'}, \mathsf{vk}'')$. It then computes $\mathsf{pk}'_f \leftarrow \mathsf{PrmsEval}(f \circ f_{\mathsf{recon}}, \overrightarrow{\mathsf{pk}'})$ and sets $t \leftarrow \mathsf{VerifyHide}(\mathsf{pk}'_f, \mathsf{vk}'', \vec{x}, \vec{\sigma})$ if $f \neq f_{\mathsf{id}}$, and $t \leftarrow \mathsf{Verify}(\mathsf{pk}'_f, \mathsf{vk}'', \vec{x}, \vec{\sigma})$ if $f = f_{\mathsf{id}}$. It returns $(\mathsf{sid}, \mathsf{verified}, \vec{x}, \vec{\sigma}, t)$ to $\mathcal{F}_{\text{BHS}}$.

- If $\tilde{\mathbf{P}}$ is corrupted, then $\mathcal{S}$ activates the party $\mathbf{P}$ with the input $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}', (f, \vec{x}), \vec{\sigma})$ in its simulated copy of Π_{BHS}. Let $(\mathsf{sid}, \mathsf{verified}, \vec{x}, \vec{\sigma}, t)$ be the output by $\mathbf{P}$. The simulator forwards $(\mathsf{sid}, \mathsf{verified}, \vec{x}, \vec{\sigma}, t)$ to the environment. Note that the simulator does not interact with the ideal functionality $\mathcal{F}_{\text{BHS}}$ in this case.

Simulating the signature-evaluation phase. When the environment activates $\tilde{\mathbf{P}} \in \{\tilde{\mathbf{S}}, \tilde{\mathbf{R}}\}$ on an input $(\mathsf{sid}, \mathsf{eval}, \mathsf{vk}, g, (f, \vec{x}), \vec{\sigma})$, where $f = f_{\mathsf{id}}$, the simulator $\mathcal{S}$ proceeds as follows:

- If $\tilde{\mathbf{P}}$ is honest, then $\mathcal{S}$ only needs to simulate the verification request (if asked by the ideal functionality). The simulator responds to the verification request using the procedure described above (for simulating the verification queries).

- If $\tilde{\mathbf{P}}$ is corrupt, then $\mathcal{S}$ activates party $\mathbf{P}$ with the input $(\mathsf{sid}, \mathsf{eval}, \mathsf{vk}, g, (f, \vec{x}), \vec{\sigma})$ in its simulated copy of Π_{BHS}. Let $(\mathsf{sid}, \mathsf{signature}, (g, g(\vec{x})), \sigma')$ be the output by $\mathbf{P}$. The simulator forwards $(\mathsf{sid}, \mathsf{signature}, (g, g(\vec{x})), \sigma')$ to the environment. Note that the simulator does *not* interact with the ideal functionality $\mathcal{F}_{\text{BHS}}$ in this case.

To complete the proof, we show that no efficient environment $\mathcal{Z}$ can distinguish the output of the real execution with the adversary $\mathcal{A}$ from the output of the ideal execution with the simulator $\mathcal{S}$.

Our argument considers several distinct cases, depending on whether the signer and receiver are honest or corrupt.

Lemma D.1. *If both the signer and the receiver are honest, then for all efficient environments $\mathcal{Z}$, we have that* $\text{IDEAL}_{\mathcal{F}_{\text{BHS}},\mathcal{S},\mathcal{Z}} \overset{c}{\approx} \text{REAL}_{\Pi_{\text{BHS}},\mathcal{A},\mathcal{Z}}$.

Proof. We proceed via a hybrid argument:

- HYB_0: This is the real distribution $\text{REAL}_{\Pi_{\text{BHS}},\mathcal{A},\mathcal{Z}}$.
- HYB_1: Same as HYB_0, except we modify the honest parties' behavior as follows:
  - At the beginning of the experiment, initialize $\vec{x}^* \leftarrow \perp$.
  - At the end of a signing request, let $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \vec{x}), \vec{\sigma})$ be the signature output by the receiver. Update $\vec{x}^* \leftarrow \vec{x}$. If any party issued a verification request of the form $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \vec{x}), \vec{\sigma})$ prior to the signing request, then the experiment aborts with output $\perp$.
  - Let vk be the verification key generated by the signer in the key-generation phase. When the environment activates a party on a verification request $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}', (f, \vec{x}), \vec{\sigma})$ where $\mathsf{vk}' = \mathsf{vk}$ and $\vec{x} \neq f(\vec{x}^*)$, then the party outputs $(\mathsf{sid}, \mathsf{verified}, (f, \vec{x}), \vec{\sigma}, 0)$. Otherwise, the output is determined as in HYB_0.
- HYB_2: This is the ideal distribution $\text{IDEAL}_{\mathcal{F}_{\text{BHS}},\mathcal{S},\mathcal{Z}}$.

We now show that the outputs of each pair of consecutive hybrid experiments are computationally indistinguishable.

Claim D.2. *Suppose Π_{HS} satisfies unforgeability (Definition 3.5). Then, the outputs of* HYB_0 *and* HYB_1 *are computationally indistinguishable.*

Proof. Suppose there exists an environment $\mathcal{Z}$ (and an adversary $\mathcal{A}$) such that the outputs of HYB_0 and HYB_1 are distinguishable. We use $\mathcal{Z}$ and $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks unforgeability (Definition 3.5) of Π_{HS}. Algorithm $\mathcal{B}$ operates according to the specification of the unforgeability security experiment $\mathsf{Expt}^{\mathsf{uf}}_{\mathcal{A},\Pi_{\mathsf{HS}}}(\lambda)$, and simulates an execution of HYB_0 or HYB_1 for the environment $\mathcal{Z}$ (and adversary $\mathcal{A}$). Specifically, $\mathcal{B}$ simulates the behavior of the honest signer and receiver in the protocol execution experiment:

- At the beginning of the unforgeability security game, algorithm $\mathcal{B}$ receives public keys $\overrightarrow{\mathsf{pk}}$ and a verification key vk' from the challenger. It also initializes $\vec{x}^* \leftarrow \perp$.
- When $\mathcal{Z}$ activates the signer $\mathbf{S}$ to run the key-generation protocol with a query $(\mathsf{sid}, \mathsf{keygen})$, algorithm $\mathcal{B}$ simulates the honest signer's behavior by outputting $(\mathsf{sid}, \mathsf{vkey}, (\overrightarrow{\mathsf{pk}}, \mathsf{vk}'))$.

  By definition of the unforgeability experiment $\mathsf{Expt}^{\mathsf{uf}}_{\mathcal{A},\Pi_{\mathsf{HS}}}(\lambda)$, the unforgeability challenger samples $\overrightarrow{\mathsf{pk}} \leftarrow \mathsf{PrmsGen}(1^\lambda, 1^{t\ell})$, and $(\mathsf{sk}, \mathsf{vk}') \leftarrow \mathsf{KeyGen}(1^\lambda)$. Thus, algorithm $\mathcal{B}$ perfectly simulates the signer's behavior in HYB_0 and HYB_1.

- For signing queries, after $\mathcal{Z}$ activates the receiver $\mathbf{R}$ with a tuple $(\mathsf{sid}, \mathsf{sign}, \mathsf{vk}, \vec{x})$ and the signer $\mathbf{S}$ with a tuple $(\mathsf{sid}, \mathsf{signature})$, algorithm $\mathcal{B}$ samples $\vec{w}_1, \ldots, \vec{w}_t \xleftarrow{R} \{0,1\}^\ell$ such that $\bigoplus_{i\in[t]} \vec{w}_i = \vec{x}$ and submits $(\vec{w}_1, \ldots, \vec{w}_t)$ to the unforgeability challenger to receive $(\vec{\sigma}_1, \ldots, \vec{\sigma}_t)$. It computes $\vec{\sigma} \leftarrow \mathsf{SigEval}(f_{\mathsf{recon}}, \overrightarrow{\mathsf{pk}}, (\vec{w}_1, \ldots, \vec{w}_t), (\vec{\sigma}_1, \ldots, \vec{\sigma}_t))$ and simulates the receiver's output as $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \vec{x}), \vec{\sigma})$. In addition, $\mathcal{B}$ sets $\vec{x}^* \leftarrow \vec{x}$.

  In $\mathsf{Expt}^{\mathsf{uf}}_{\mathcal{A},\Pi_{\mathsf{HS}}}(\lambda)$, the challenger computes $(\vec{\sigma}_1, \ldots, \vec{\sigma}_t) \leftarrow \mathsf{Sign}\big(\overrightarrow{\mathsf{pk}}, \mathsf{sk}, (\vec{w}_1, \ldots, \vec{w}_t)\big)$, exactly as in HYB_0 and HYB_1. Thus, $\mathcal{B}$ perfectly simulates the signing queries in HYB_0 and HYB_1.

- For verification and evaluation queries, $\mathcal{B}$ implements the same procedure as in HYB_0 and HYB_1. None of these queries require knowledge of the secret signing key sk, and thus, can be perfectly simulated by $\mathcal{B}$.

- At any point during the simulation, if $\mathcal{Z}$ activates a party on a verification request of the form $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}, (f, \vec{x}), \vec{\sigma})$ where $f(\vec{x}^*) \neq \vec{x}$ and $\vec{\sigma}$ is a valid signature on $(f, \vec{x})$, then $\mathcal{B}$ does the following:

  - If $f = f_{\mathsf{id}}$, then $\mathcal{B}$ computes $\vec{\sigma}^* \leftarrow \mathsf{Hide}(\mathsf{vk}', \vec{x}, \vec{\sigma})$ and sends the tuple $(f_{\mathsf{recon}}, \vec{x}, \vec{\sigma}^*)$ to the unforgeability challenger as its forgery.
  - Otherwise, $\mathcal{B}$ sends the tuple $(f \circ f_{\mathsf{recon}}, \vec{x}, \vec{\sigma})$ to the unforgeability challenger as its forgery.

Since the only difference between HYB_0 and HYB_1 is the additional checks in the signing and verification protocols, if the outputs of HYB_0 and HYB_1 are distinguishable with non-negligible advantage ε, then one of the following conditions must hold with probability ε:

- The receiver's output in the signing request is a tuple $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \vec{x}), \vec{\sigma})$ and a party was activated to run a verification request on the tuple $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \vec{x}), \vec{\sigma})$ before the signing request. Since $\vec{\sigma}$ was output by an honest signing request, this means that $\vec{\sigma}$ is a valid signature on $\vec{x}$: namely, that $\mathsf{Verify}(\mathsf{pk}_{\mathsf{recon}}, \mathsf{vk}', \vec{x}, \vec{\sigma}) = 1$, where $\mathsf{pk}_{\mathsf{recon}} \leftarrow \mathsf{PrmsEval}(f_{\mathsf{recon}}, \overrightarrow{\mathsf{pk}})$. Moreover, since the verification request occurred before the signing request, algorithm $\mathcal{B}$ would have submitted the tuple $(f_{\mathsf{recon}}, \vec{x}, \vec{\sigma}^*)$ to the unforgeability challenger where $\vec{\sigma}^* \leftarrow \mathsf{Hide}(\mathsf{vk}', \vec{x}, \vec{\sigma})$ *before* it made any signing queries to the unforgeability challenger. By hiding correctness, $\vec{\sigma}^*$ is a valid signature on $\vec{x}$ with respect to f_{recon}, and so $\mathcal{B}$ wins the unforgeability game.

- Otherwise, the environment must have activated a party on a verification query of the form $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}, (f, \vec{x}), \vec{\sigma})$ the successfully verifies in HYB_0 but not in HYB_1. First, since the signature $\vec{\sigma}$ verifies in HYB_0, this means that $f \circ f_{\mathsf{recon}} \in \mathcal{H}'$ and moreover, that $\mathsf{VerifyHide}(\mathsf{pk}_{f\circ f_{\mathsf{recon}}}, \mathsf{vk}', \vec{x}, \vec{\sigma}) = 1$ where $\overrightarrow{\mathsf{pk}}_{f\circ f_{\mathsf{recon}}} \leftarrow \mathsf{PrmsEval}(f \circ f_{\mathsf{recon}}, \overrightarrow{\mathsf{pk}})$. Now, if the adversary $\mathcal{B}$ made a signing request to the unforgeability challenger on the message $(\vec{w}_1, \ldots, \vec{w}_t)$, then it would have also set $\vec{x}^* = \bigoplus_{i\in[t]} \vec{w}_i$. Since $\vec{\sigma}$ verifies in HYB_0 but not in HYB_1, the special condition in HYB_1 must be satisfied which means

  $$(f \circ f_{\mathsf{recon}})(\vec{w}_1, \ldots, \vec{w}_t) = f(\vec{x}^*) \neq \vec{x}.$$

  This means that $\vec{\sigma}$ is a valid signature on $\vec{x}$ with respect to the function $f \circ f_{\mathsf{recon}}$, and thus, is a valid forgery. Alternatively, if $\mathcal{B}$ never made a signing request to the unforgeability challenger, then $\vec{\sigma}$ is trivially a valid forgery.

In both cases, algorithm $\mathcal{B}$ breaks unforgeability of Π_{HS}, so we conclude that $\mathcal{B}$ has advantage ε in the unforgeability game. □

Claim D.3. *The outputs of hybrids* HYB_1 *and* HYB_2 *are identically distributed.*

Proof. We consider the view of the environment $\mathcal{Z}$ in HYB_1 and HYB_2 during each phase of the protocol.

- *Key-generation*: For the key-generation phase, the simulator $\mathcal{S}$ in HYB_2 exactly emulates the generation of $\overrightarrow{\mathsf{pk}}$ and $\mathsf{vk} = (\mathsf{sk}, \mathsf{vk}')$ as defined in HYB_1. Thus, the outputs of the honest parties in the key-generation phase of HYB_1 and HYB_2 are identically distributed.

- *Signature-generation*: In HYB_2, since both $\mathbf{S}$ and $\mathbf{R}$ are honest, the signatures that the receiver obtains from $\mathcal{F}_{\text{BHS}}$ are determined by the ideal algorithm $\mathsf{IdealSign}$ that $\mathcal{S}$ provides to the functionality $\mathcal{F}_{\text{BHS}}$. Since $\mathcal{S}$ defines these algorithms exactly as in the protocol specification of Π_{BHS} using the identically-distributed signing key sk and verification key vk, the resulting signatures in HYB_1 and HYB_2 are identically distributed. Moreover, the same abort condition is present in both HYB_1 and HYB_2, so whenever an environment issues a query that causes the ideal functionality to abort in HYB_2, the experiment also aborts in HYB_1.

- *Signature-verification*: In HYB_2, the ideal functionality $\mathcal{F}_{\text{BHS}}$ handles the signature verification queries $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}', (f, \vec{x}), \vec{\sigma})$. We consider the different possibilities:

  - If $f \notin \mathcal{H}$, then $\mathcal{F}_{\text{BHS}}$ always sets the verification bit $t = 0$. In this case, the honest parties in HYB_1 also sets $t = 0$ according to the protocol specification.
  - Otherwise, if $\mathsf{vk} = \mathsf{vk}'$ and $(\mathsf{vk}, (f, \vec{x}), \vec{\sigma}, 1) \in \mathcal{L}$, then $\mathcal{F}_{\text{BHS}}$ sets $t = 1$. We consider several scenarios depending on how the entry $(\mathsf{vk}, (f, \vec{x}), \vec{\sigma}, 1) \in \mathcal{L}$ was added to $\mathcal{L}$. If $\vec{\sigma}$ was generated as the result of a signing or a evaluation request, then by correctness of Π_{HS}, the honest party in HYB_1 also outputs 1. If the entry was added as a result of a previous verification request (which successfully verified), then because the honest party's verification algorithm in Π_{HS} is *deterministic* (and the signature verified previously), the party also outputs 1 in HYB_1.
  - Otherwise, if $\mathsf{vk} = \mathsf{vk}'$, and there does not exist $(\mathsf{vk}, (f_{\mathsf{id}}, \vec{x}'), \vec{\sigma}', 1) \in \mathcal{L}$ for some $\vec{x}', \vec{\sigma}'$ where $\vec{x} = f(\vec{x}')$, then $\mathcal{F}_{\text{BHS}}$ sets $t = 0$. This corresponds to a setting where the receiver never makes a signing request on any $\vec{x}^* \in \{0,1\}^\ell$ where $\vec{x} = f(\vec{x}^*)$. This means the condition in HYB_1 is satisfied, in which case the party's output is $(\mathsf{sid}, \mathsf{verified}, (f, \vec{x}'), \vec{\sigma}', 0)$. This matches the behavior in HYB_2.
  - Otherwise, if there is already an entry $(\mathsf{vk}', (f, \vec{x}), \vec{\sigma}, t') \in \mathcal{L}$ for some t', the ideal functionality sets $\mathcal{F}_{\text{BHS}}$ sets $t = t'$. In the real protocol execution in HYB_1, the honest verifier's decision algorithm is deterministic. Hence, if a signature previously verified (resp., failed to verify), it will continue to verify (resp., fail to verify).
  - Finally, if none of the above criterion apply, then the ideal functionality allows the simulator $\mathcal{S}$ to decide the verification response in HYB_2. By construction, for an honest party, the simulator implements the same logic as that in the actual protocol Π_{BHS}.

  We conclude that the outputs of the honest parties in response to verification queries are identically distributed in HYB_1 and HYB_2.

- *Signature-evaluation*: In HYB_2, since both parties $\mathbf{S}$ and $\mathbf{R}$ are honest, the resulting signatures that a party receives from $\mathcal{F}_{\text{BHS}}$ are fully determined by the ideal algorithm $\mathsf{IdealEval}$ that $\mathcal{S}$ provides to the functionality $\mathcal{F}_{\text{BHS}}$. Since $\mathcal{S}$ implements these algorithms exactly as in the protocol specification of Π_{BHS} using the identically-distributed signing key sk and verification key vk, the signatures output by the evaluation algorithm in HYB_1 and HYB_2 are identically distributed. Moreover, by correctness of Π_{HS} and construction of $\mathcal{S}$, the abort condition in $\mathcal{F}_{\text{BHS}}$ for evaluation queries is never triggered. □

Lemma D.1 now follows by combining Claims D.2 and D.3. □

Lemma D.4. *If the signer is honest and the receiver is corrupt, then for all efficient environments $\mathcal{Z}$, we have that* $\text{IDEAL}_{\mathcal{F}_{\text{BHS}},\mathcal{S},\mathcal{Z}} \overset{c}{\approx} \text{REAL}_{\Pi_{\text{BHS}},\mathcal{A},\mathcal{Z}}$.

Proof. We use a similar hybrid structure as that used in the proof of Lemma D.1:

- HYB_0: This is the real distribution $\text{REAL}_{\Pi_{\text{BHS}},\mathcal{A},\mathcal{Z}}$.
- HYB_1: Same as HYB_0, except we modify the honest signer's behavior as follows:
  - At the beginning of the experiment, initialize $\vec{x}^* \leftarrow \perp$.
  - During a signing request, let $\vec{w}_1, \ldots, \vec{w}_t$ be the messages $\mathbf{R}$ submits to $\mathcal{F}_{\text{OT}}^{\ell,s}$. Update $\vec{x}^* \leftarrow \bigoplus_{i\in[t]} \vec{w}_i$. If the environment activated the signer to make a verification request of the form $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \vec{x}^*), \vec{\sigma})$ where $\vec{\sigma}$ is a valid signature on $(f_{\mathsf{id}}, \vec{x}^*)$ prior to the signing request, then the experiment aborts with output $\perp$.
  - Let vk be the verification key generated by the signer in the key-generation phase. If the environment activates the honest signer on a verification request of the form $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}', (f, \vec{x}), \vec{\sigma})$ where $\mathsf{vk}' = \mathsf{vk}$ and $\vec{x} \neq f(\vec{x}^*)$, then the signer's output is set to $(\mathsf{sid}, \mathsf{verified}, (f, \vec{x}), \vec{\sigma}, 0)$. Otherwise, the output is determined as in HYB_0.
- HYB_2: This is the ideal distribution $\text{IDEAL}_{\mathcal{F}_{\text{BHS}},\mathcal{S},\mathcal{Z}}$.

Claim D.5. *Suppose Π_{HS} satisfies unforgeability (Definition 3.5). Then, the outputs of* HYB_0 *and* HYB_1 *are computationally indistinguishable.*

Proof. Suppose there exists an environment $\mathcal{Z}$ and adversary $\mathcal{A}$ (that corrupts the receiver $\mathbf{R}$) such that the outputs of HYB_0 and HYB_1 are distinguishable. We use $\mathcal{Z}$ and $\mathcal{A}$ to construct an algorithm $\mathcal{B}$ that breaks unforgeability of Π_{HS}. In the reduction, algorithm $\mathcal{B}$ simulates the behavior of the honest signer for $\mathcal{Z}$ and $\mathcal{A}$ according to the protocol specification in HYB_0 and HYB_1. The overall argument follows a very similar structure as the proof of Claim D.2, so we only give a sketch of how $\mathcal{B}$ simulates the execution of HYB_0 and HYB_1 below:

- As in the proof of Claim D.2, algorithm $\mathcal{B}$ uses the public keys $\overrightarrow{\mathsf{pk}}$ and the verification key vk' from the unforgeability challenger as the signer's verification key $\mathsf{vk} = (\overrightarrow{\mathsf{pk}}, \mathsf{vk}')$.
- To simulate a signing protocol, after the receiver $\mathbf{R}$ (under the direction of $\mathcal{A}$) submits shares $\vec{w}_1, \ldots, \vec{w}_t \in \{0,1\}^\ell$ to $\mathcal{F}_{\text{OT}}^{\ell,s}$, algorithm $\mathcal{B}$ submits $(\vec{w}_1, \ldots, \vec{w}_t)$ to the unforgeability challenger to obtain the signatures $(\vec{\sigma}_1, \ldots, \vec{\sigma}_t)$, which it uses to simulate the response from $\mathcal{F}_{\text{OT}}^{\ell,s}$.
- Finally, algorithm $\mathcal{B}$ simulates the verification and evaluation queries to the honest signer as described in HYB_0 and HYB_1, since these operations only depend on the public parameters.

By an analogous argument to that in the proof of Claim D.2, algorithm $\mathcal{B}$ correctly simulates the behavior of the honest signer in a protocol execution with $\mathcal{Z}$ and $\mathcal{A}$. Thus, with non-negligible probability, the environment will activate the honest signer on a signing or verification query whose behavior differs between HYB_0 and HYB_1. As in the proof of Claim D.2, if either condition is satisfied, the environment's query enables $\mathcal{B}$ to break unforgeability of the signature scheme. □

Claim D.6. *The outputs of* HYB_1 *and* HYB_2 *are identically distributed.*

Proof. We argue that the view of the environment $\mathcal{Z}$ is identically distributed in HYB_1 and HYB_2. The argument follows similarly to that in the proof of Claim D.3. We sketch the key details below:

- *Key-generation*: The simulator $\mathcal{S}$ (in HYB_2) implements the key-generation phase exactly according to the specification of the real protocol Π_{BHS} (in HYB_1).
- *Signature-generation*: In HYB_1, when the receiver $\mathbf{R}$ (under the direction of $\mathcal{A}$) submits shares $\vec{w}_1, \ldots, \vec{w}_t \in \{0,1\}^\ell$ to $\mathcal{F}_{\text{OT}}^{\ell,s}$, it receives in response from the $\mathcal{F}_{\text{OT}}^{\ell,s}$ functionality signatures $\vec{\sigma}_1, \ldots, \vec{\sigma}_t$ where $(\vec{\sigma}_1, \ldots, \vec{\sigma}_t) \leftarrow \mathsf{Sign}(\overrightarrow{\mathsf{pk}}, \mathsf{sk}, (\vec{w}_1, \ldots, \vec{w}_t))$. This is precisely how $\mathcal{S}$ simulates the signing request for $\mathcal{A}$ in HYB_2. Let $((f_{\mathsf{id}}, \vec{x}), \vec{\sigma})$ be the message-signature pair that the simulator $\mathcal{S}$ registers with the ideal functionality $\mathcal{F}_{\text{BHS}}$ at the end of the signing request in HYB_2. If this pair is already registered with $\mathcal{F}_{\text{BHS}}$ as an invalid signature, then $\mathcal{F}_{\text{BHS}}$ aborts and the protocol execution halts in HYB_2. By definition of HYB_2 and $\mathcal{S}$, this is only possible if the environment activates the honest signer to make a verification request on the message-signature pair $((f_{\mathsf{id}}, \vec{x}), \vec{\sigma})$ prior to the signing request. This coincides with the abort condition in HYB_1, and so we conclude that the output of the signature-generation phase in HYB_1 and HYB_2 is identically distributed.
- *Signature-verification*: Signature verification is a non-interactive procedure, so it suffices to argue that the outputs of the honest signer in response to the environment's queries are identically distributed in HYB_1 and HYB_2. By construction of $\mathcal{S}$, only verification and evaluation queries involving an *honest* party requires interacting with the ideal functionality. The argument then proceeds as in the proof of Claim D.3.
- *Signature-evaluation*: Similar to the case of signature verification, signature evaluation is non-interactive, so it suffices to argue that the outputs of the honest signer in response to the environment's queries are identically distributed. This argument then proceeds as in the proof of Claim D.3. □

Combining Claims D.5 and D.6, the lemma follows. □

Lemma D.7. *If the signer is corrupt and the receiver is honest, then for all efficient environments* $\mathcal{Z}$*, we have that* $\text{IDEAL}_{\mathcal{F}_{\text{BHS}},\mathcal{S},\mathcal{Z}} \overset{c}{\approx} \text{REAL}_{\Pi_{\text{BHS}},\mathcal{A},\mathcal{Z}}$.

Proof. We proceed via a hybrid argument:

- HYB_0: This is the real distribution $\text{REAL}_{\Pi_{\text{BHS}},\mathcal{A},\mathcal{Z}}$.
- HYB_1: Same as HYB_0 except we modify the honest receiver's behavior in the signature-generation protocol as follows. Let vk be the verification key chosen by the signer in the key-generation phase. Let $\big((\mathsf{sid}, i), \mathsf{sender}, \{(\sigma_{i,j,0}, \sigma_{i,j,1})\}_{i\in[t],j\in[\ell]}\big)$ be the set of signatures the signer

submits to the ideal OT functionality $\mathcal{F}_{\mathrm{OT}}^{\ell,s}$, and let $\{\sigma_{i,j}^{\mathsf{pk}}\}_{i\in[t],j\in[\ell]}$ be the message-independent signature components $\mathbf{S}$ sends to $\mathbf{R}$. The receiver *always* outputs $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \vec{x}), \bot)$ if any of the following conditions hold:

- The signer's verification key vk cannot be written as $(\overrightarrow{\mathsf{pk}}, \mathsf{vk}')$ where $\overrightarrow{\mathsf{pk}} = \{\mathsf{pk}_{i,j}\}_{i\in[t],j\in[\ell]}$.
- If there exists indices $i \in [t]$ and $j \in [\ell]$ where both $\sigma_{i,j,0}$ and $\sigma_{i,j,1}$ are invalid. We say that a signature $\sigma_{i,j,b}$ is valid if it satisfies Eq. (D.1).
- If there exists $j \in [\ell]$ such that for all $i \in [t]$, at least one of $\sigma_{i,j,0}$ and $\sigma_{i,j,1}$ is invalid.

Otherwise, the honest receiver implements the verification protocol as in the real scheme.

- HYB_2: Same as HYB_1 except we use the context-hiding simulator $\mathcal{S} = (\mathcal{S}^{\mathsf{Ext}}, \mathcal{S}^{\mathsf{Gen}})$ to generate the signatures the honest receiver $\mathbf{R}$ outputs on evaluation queries. Here, we assume that none of the conditions from HYB_1 are satisfied (otherwise, the honest receiver outputs $\bot$ in the signing protocol and ignores all evaluation requests). In particular, we have the following:
  - Let $\big((\mathsf{sid}, i), \mathsf{sender}, \{(\sigma_{i,j,0}, \sigma_{i,j,1})\}_{i\in[t],j\in[\ell]}\big)$ be the set of signatures the signer submits to the ideal OT functionality $\mathcal{F}_{\mathrm{OT}}^{\ell,s}$, and let $\{\sigma_{i,j}^{\mathsf{pk}}\}_{i\in[t],j\in[\ell]}$ be the message-independent signature components $\mathbf{S}$ sends to $\mathbf{R}$.
  - Since none of the conditions in HYB_1 are satisfied, there exist indices i^*, j^* where $\sigma_{i^*,j^*,0}$ and $\sigma_{i^*,j^*,1}$ are both valid. Moreover, the verification key vk can be written as $\mathsf{vk} = (\overrightarrow{\mathsf{pk}}, \mathsf{vk}')$ where $\overrightarrow{\mathsf{pk}} = \{\mathsf{pk}_{i,j}\}_{i\in[t],j\in[\ell]}$. The experiment invokes the context-hiding simulator $\mathcal{S}^{\mathsf{Ext}}$ to extract a simulation trapdoor $\mathsf{td} \leftarrow \mathcal{S}^{\mathsf{Ext}}(\mathsf{vk}', (0, \sigma_{i^*,j^*,0}), (1, \sigma_{i^*,j^*,1}))$, and stores td. The receiver's signature is constructed using the same procedure from HYB_1.
  - During signature evaluation, on input $(\mathsf{sid}, \mathsf{eval}, \mathsf{vk}, g, (f, \vec{x}), \vec{\sigma})$, $\mathbf{R}$ first applies the signature verification procedure on input $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}, (f, \vec{x}), \vec{\sigma})$. If the signature verifies, the receiver's signature is generated by computing $\mathsf{pk}_g \leftarrow \mathsf{PrmsEval}(g \circ f_{\mathsf{recon}}, \overrightarrow{\mathsf{pk}})$, $\sigma_g^{\mathsf{pk}} \leftarrow \mathsf{SigEvalPK}\big(g \circ f_{\mathsf{recon}}, \overrightarrow{\mathsf{pk}}, (\vec{\sigma}_1^{\mathsf{pk}}, \ldots, \vec{\sigma}_t^{\mathsf{pk}})\big)$, where $\vec{\sigma}_i^{\mathsf{pk}} = (\sigma_{i,1}^{\mathsf{pk}}, \ldots, \sigma_{i,\ell}^{\mathsf{pk}})$, and finally $\sigma^* \leftarrow \mathcal{S}^{\mathsf{Gen}}(\mathsf{pk}_g, \mathsf{vk}', \mathsf{td}, g(\vec{x}), \sigma_g^{\mathsf{pk}})$. The receiver's output is the tuple $(\mathsf{sid}, \mathsf{signature}, (g, g(\vec{x})), \sigma^*)$.
- HYB_3: This is the ideal distribution $\mathrm{IDEAL}_{\mathcal{F}_{\mathrm{BHS}},\mathcal{S},\mathcal{Z}}$.

Claim D.8. *Suppose $t = \omega(\log\lambda)$. Then, the outputs of hybrids HYB_0 and HYB_1 are statistically indistinguishable.*

Proof. The only difference between the two experiments is the additional checks in HYB_1 which affects the honest receiver's output on signing queries. We consider each of the conditions separately, and argue that for each of them, the receiver's output in HYB_1 is the same as that in HYB_0, except with probability at most $2^{-(t-1)} = 2^{-\omega(\log\lambda)} = \mathsf{negl}(\lambda)$.

- Suppose that the signer's verification key is not well-formed: namely, that $\mathsf{vk} \neq (\overrightarrow{\mathsf{pk}}, \mathsf{vk}')$ where $\overrightarrow{\mathsf{pk}} = \{\mathsf{pk}_{i,j}\}_{i\in[t],j\in[\ell]}$. In this case, the receiver's signature is $\bot$ in both HYB_0 and HYB_1.
- Suppose there exists $i \in [t]$ and $j \in [\ell]$ where both $\sigma_{i,j,0}$ and $\sigma_{i,j,1}$ are invalid. In this case, the honest receiver in HYB_0 outputs $\bot$ as its signature, which matches the behavior in HYB_1.

- Suppose there exists $j \in [\ell]$ such that for all $i \in [t]$, at least one of $\sigma_{i,j,0}$ and $\sigma_{i,j,1}$ is invalid. We argue that in this case, the receiver outputs $\perp$ with probability at least $1 - 2^{-(t-1)}$ in HYB_0. Without loss of generality, we can assume that exactly one of $\sigma_{i,j,0}$ and $\sigma_{i,j,1}$ for all $i \in [t]$ is invalid (the case where both are invalid is captured by the previous case). Let $b_1, \ldots, b_t \in \{0,1\}$ be such that σ_{i,j,b_i} is invalid, and let $\vec{x} = (x_1, \ldots, x_\ell) \in \{0,1\}^\ell$ be the receiver's message in the signing protocol. In the real protocol, the honest receiver samples $w_{i,j} \xleftarrow{\text{R}} \{0,1\}$ for all $i \in [t]$ such that $x_j = \bigoplus_{i \in [t]} w_{i,j}$. We consider two possibilities:

  - Suppose $x_j \neq \bigoplus_{i \in [t]} b_i$. This means that there exists $i \in [t]$ where $w_{i,j} \neq b_i$. In the real protocol, this means that the receiver obtains signature $\sigma_{i,j,w_{i,j}}$, which by assumption is invalid. In this case, the receiver in HYB_0 outputs $\perp$ as its signature.
  - Suppose $x_j = \bigoplus_{i \in [t]} b_i$. Since $w_{i,j}$ are sampled uniformly at random subject to the constraint, with probability $2^{-(t-1)}$, it is the case that $w_{i,j} = b_i$ for all $i \in [t]$. In this case, the receiver in HYB_0 does not output $\perp$ (since every signature it obtains is valid). With probability $1 - 2^{-(t-1)}$, there is an index $i \in [t]$ where $w_{i,j} \neq b_i$. In this case, the receiver obtains signature $\sigma_{i,j,w_{i,j}}$, which by assumption is invalid. Thus, we conclude that the receiver in HYB_0 aborts with probability $1 - 2^{-(t-1)}$.

  In this case, the honest receiver in HYB_0 outputs $\perp$ with probability at least $1 - 2^{-(t-1)}$, while in HYB_1, the receiver outputs $\perp$ with probability 1. In both cases, the probability is taken over the receiver's random coins. Since $t = \omega(\log \lambda)$, we conclude that the statistical distance between the output distributions of HYB_0 and HYB_1 is negligible. □

Claim D.9. *Suppose Π_{HS} satisfies context-hiding (Definition 3.9). Then, the outputs of hybrids HYB_1 and HYB_2 are computationally indistinguishable.*

Proof. Suppose there exists an environment $\mathcal{Z}$ and adversary $\mathcal{A}$ (that corrupts the signer $\mathbf{S}$) such that the outputs of HYB_1 and HYB_2 are distinguishable. We use $\mathcal{Z}$ and $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks context-hiding security (Definition 3.9) of Π_{HS}. Algorithm $\mathcal{B}$ begins by simulating the protocol execution in HYB_1 and HYB_2 for $\mathcal{Z}$ and $\mathcal{A}$. In the simulation, $\mathcal{B}$ is responsible for simulating the behavior of the honest receiver $\mathbf{R}$ and the ideal OT functionality $\mathcal{F}_{\text{OT}}^{\ell,s}$.

- *Key-generation*: The key-generation protocol only involves $\mathcal{Z}$ and $\mathcal{A}$, so $\mathcal{B}$ does not need to simulate anything.
- *Signature-generation*: On a signature-generation query $(\mathsf{sid}, \mathsf{sign}, \mathsf{vk}, \vec{x})$, let $\{(\sigma_{i,j,0}, \sigma_{i,j,1})\}_{i \in [t], j \in [\ell]}$ be the signatures the signer $\mathbf{S}$ submits to the ideal OT functionality $\mathcal{F}_{\text{OT}}^{\ell,s}$ in the simulated protocol execution (as directed by $\mathcal{Z}$ and $\mathcal{A}$). Additionally, let $\{\sigma_{i,j}^{\mathsf{pk}}\}_{i \in [t], j \in [\ell]}$ be the message-independent signature components the signer sends to the receiver. Algorithm $\mathcal{B}$ checks the three conditions in HYB_1 and HYB_2, and if any condition is satisfied, it defines the receiver's output to be $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \vec{x}), \perp)$.

  Otherwise, the verification key vk has the form $\mathsf{vk} = (\overrightarrow{\mathsf{pk}}, \mathsf{vk}')$ where $\overrightarrow{\mathsf{pk}} = \{\mathsf{pk}_{i,j}\}_{i \in [t], j \in [\ell]}$, and moreover, there exists indices i^*, j^* where $\sigma_{i^*,j^*,0}$ and $\sigma_{i^*,j^*,1}$ are both valid. Algorithm $\mathcal{B}$ submits the public key pk_{i^*,j^*}, the verification key vk', and the message-signature pairs $(0, \sigma_{i^*,j^*,0})$ and $(1, \sigma_{i^*,j^*,1})$ to the context-hiding challenger. Finally, algorithm $\mathcal{B}$ simulates the receiver's output according to the specification in HYB_1 and HYB_2.

If the receiver's output is not $\perp$, algorithm $\mathcal{B}$ does the following. Let $\vec{w}_1, \ldots, \vec{w}_t$ where $\bigoplus_{i \in [t]} \vec{w}_i = \vec{x}$ be the bit-strings $\mathcal{B}$ chose when simulating the honest receiver. For $i \in [t]$, algorithm $\mathcal{B}$ defines $\vec{\sigma}_i = (\sigma_{i,1,w_{i,1}}, \ldots, \sigma_{i,\ell,w_{i,\ell}})$.

- *Signature-verification*: Algorithm $\mathcal{B}$ simulates the verification queries involving the receiver $\mathbf{R}$ as described in HYB_1 and HYB_2. Note that because signature verification is non-interactive, the environment $\mathcal{Z}$ and the adversary $\mathcal{A}$ completely dictate the behavior of verification queries to the corrupt signer.

- *Signature-evaluation*: Whenever $\mathcal{Z}$ activates the receiver on a signature-evaluation query $(\mathsf{sid}, \mathsf{eval}, \mathsf{vk}, g, (f, \vec{x}), \vec{\sigma})$, where $\mathsf{vk} = (\overrightarrow{\mathsf{pk}}, \mathsf{vk}')$, algorithm $\mathcal{B}$ ignores the request if the receiver's signature in the signing protocol was $\perp$. Otherwise, it proceeds as follows:

  - As in HYB_1 and HYB_2, algorithm $\mathcal{B}$ checks that $f = f_{\mathsf{id}}$ and that σ is a valid signature on $(f, \vec{x})$. If the signature verifies, then $\mathcal{B}$ first computes $\mathsf{pk}_g \leftarrow \mathsf{PrmsEval}(g \circ f_{\mathsf{recon}}, \overrightarrow{\mathsf{pk}})$. Then, it computes $\sigma' \leftarrow \mathsf{SigEval}(g \circ f_{\mathsf{recon}}, (\vec{\sigma}_1, \ldots, \vec{\sigma}_t))$ where $\vec{\sigma}_i$ is defined as in the signing protocol.
  - Algorithm $\mathcal{B}$ submits the public key pk_g, the message $g(\vec{x})$, and the signature σ' to the context-hiding challenger, and receives in response a signature σ^*. Algorithm $\mathcal{B}$ simulates the output of the honest receiver as $(\mathsf{sid}, \mathsf{signature}, (g, g(\vec{x})), \sigma^*)$.

  Note that signature evaluation is non-interactive, the environment $\mathcal{Z}$ and the adversary $\mathcal{A}$ completely dictate the behavior of evaluation queries to the corrupt signer.

- At the end of the experiment, when $\mathcal{Z}$ outputs a bit, $\mathcal{B}$ outputs the same bit.

We now show that $\mathcal{B}$ breaks context-hiding security with the same advantage as $\mathcal{Z}$. By construction, the only difference between the two hybrid experiments HYB_1 and HYB_2 is the way the honest receiver's signatures are generated on evaluation queries. Note that if the honest receiver outputs $\perp$ in response to a signing query, then the honest receiver in HYB_1 and HYB_2 ignores all evaluation queries. In this case, the two experiments are identical. Thus, without loss of generality, we assume that the signing protocol succeeds. In this case, algorithm $\mathcal{B}$ submits a valid key, verification key, and message-signature pairs to the context-hiding challenger.

Now, assume that $\mathcal{B}$ does not abort during signature generation. Then, if the context-hiding challenger implements the hide algorithm using Hide, then the signatures output by $\mathcal{B}$ when simulating the honest evaluation queries are distributed according to HYB_1, and $\mathcal{B}$ perfectly simulates an execution of HYB_1 for $\mathcal{Z}$ and $\mathcal{A}$. Alternatively, if the context-hiding challenger implements the hide algorithm using $\mathcal{S}^{\mathsf{Gen}}$, then the signatures output by $\mathcal{B}$ when simulating the honest evaluation queries are distributed according to HYB_2, and $\mathcal{B}$ perfectly simulates an execution of HYB_2 for $\mathcal{Z}$ and $\mathcal{A}$. Thus, if $\mathcal{Z}$ is able to distinguish experiments HYB_1 and HYB_2, algorithm $\mathcal{B}$ breaks context-hiding of Π_{HS} with the same advantage. □

Claim D.10. *The outputs of hybrids* HYB_2 *and* HYB_3 *are identically distributed.*

Proof. We now show that the view of the environment $\mathcal{Z}$ when interacting with an adversary $\mathcal{A}$ in HYB_2 is distributed identically with its view when interacting with the simulator $\mathcal{S}$ in HYB_3.

- *Key-generation*: When the environment $\mathcal{Z}$ activates the signer on a key-generation query, algorithm $\mathcal{S}$ simply forwards the query to its simulated protocol execution with adversary $\mathcal{A}$ (as if it came from $\mathcal{A}$'s environment). Thus, the output of $\mathcal{S}$ in HYB_3 is distributed identically to the output of $\mathcal{A}$ in HYB_2.

- *Signature-generation*: By construction, $\mathcal{S}$ perfectly simulates the behavior of the ideal OT functionality $\mathcal{F}_{\text{OT}}^{\ell,s}$, so it perfectly simulates the view of $\mathcal{A}$ in its simulated protocol execution (since $\mathcal{A}$ only interacts with $\mathcal{F}_{\text{OT}}^{\ell,s}$). Thus, $\mathcal{S}$ perfectly simulates any interaction between the adversary and the environment that can occur during this phase.

  It suffices to argue that the output of the honest receiver in HYB_2 and HYB_3 is identically distributed on a query $(\mathsf{sid}, \mathsf{sign}, \mathsf{vk}, \vec{x})$, where vk can be parsed as $\mathsf{vk} = (\overrightarrow{\mathsf{pk}}, \mathsf{vk}')$. Let $\{(\sigma_{i,j,0}, \sigma_{i,j,1})\}_{i\in[t],j\in[\ell]}$ be the signatures the signer submits to the ideal OT functionality $\mathcal{F}_{\text{OT}}^{\ell,s}$, and let $\{\sigma_{i,j}^{\mathsf{pk}}\}_{i\in[t],j\in[\ell]}$ be the message-independent signature components the signer sends to the receiver. First, if any of the verification conditions in HYB_2 (defined in the description of HYB_1) are satisfied, then the output of the honest receiver in HYB_2 is $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \vec{x}), \bot)$. By construction, the simulator $\mathcal{S}$ implements an identical set of checks. If any of the conditions are satisfied, then the simulator defines the $\mathsf{IdealSign}$ function to output $\bot$ on *all* inputs. This means that in HYB_3, the honest receiver also outputs $\bot$ as its signature in response to the signing request. We conclude that the behavior in HYB_2 and HYB_3 is identical whenever any of the conditions in HYB_2 is triggered.

  Now, consider the case where none of the conditions in HYB_2 are satisfied. This means that for all $j \in [\ell]$, there is at least one $i_j \in [t]$ where *both* $\sigma_{i_j,j,0}$ and $\sigma_{i_j,j,1}$ are valid (according to the criterion in Eq. (D.1)). We consider the receiver's output in HYB_2 and HYB_3.

  - In HYB_2, the honest receiver chooses $\vec{w}_i \xleftarrow{\text{R}} \{0,1\}^\ell$ for all $i \in [t]$ such that $\vec{x} = \bigoplus_{i\in[t]} \vec{w}_i$. This is equivalent to first sampling $w_{i,j} \xleftarrow{\text{R}} \{0,1\}$ for all $j \in [\ell]$ and $i \neq i_j$ and setting $w_{i_j,j} \in \{0,1\}$ such that $\vec{x} = \bigoplus_{i\in[t]} \vec{w}_i$. Next, we say that an index $i \in [t]$ and $j \in [\ell]$ is "bad" if either $\sigma_{i,j,0}$ or $\sigma_{i,j,1}$ is invalid. For all bad indices i, j, define $b_{i,j} \in \{0,1\}$ so that $\sigma_{i,j,b_{i,j}}$ is valid. We consider two possibilities.
    - The receiver in HYB_2 outputs $\bot$ as its signature if there is a bad index $i \in [t]$, $j \in [\ell]$ where $w_{i,j} \neq b_{i,j}$. Suppose there are n such bad indices. Since both $\sigma_{i_j,j,0}$ and $\sigma_{i_j,j,1}$ are valid for all $j \in [\ell]$, and all of the $w_{i,j}$'s are sampled uniformly at random for $i \neq i_j$, it follows that with probability $1 - 2^{-n}$ (over the randomness used to sample the $w_{i,j}$'s), there is at least one $i \neq i_j$ and $j \in [\ell]$ where $w_{i,j} \neq b_{i,j}$.
    - With probability 2^{-n}, for all bad indices $i \in [t]$, $j \in [\ell]$, we have that $w_{i,j} = b_{i,j}$. In this case, the honest receiver in HYB_2 obtains valid signatures $\sigma_{i,j,w_{i,j}}$ from $\mathcal{F}_{\text{OT}}^{\ell,s}$ and constructs $\vec{\sigma}$ according to the specification of HYB_2. Here, $w_{i,j} = b_{i,j}$ for all bad indices, and for all remaining indices $i \neq i_j$ and $j \in [\ell]$, the choice bit $w_{i,j}$ is uniformly random.
  - In HYB_3, the ideal signing algorithm $\mathsf{IdealSign}$ is used to generate the honest receiver's signature, and the simulator $\mathcal{S}$ decides whether the honest receiver outputs $\bot$ or the output of $\mathsf{IdealSign}$. By construction, if n is the number of bad indices, then $\mathcal{S}$ causes the honest receiver to output $\bot$ with probability $1-2^{-n}$, which is precisely the probability that the honest receiver outputs $\bot$ in HYB_2. With probability 2^{-n}, the honest receiver outputs

the signature computed by $\mathsf{IdealSign}$. We argue that in this case, the signature output by $\mathsf{IdealSign}$ is distributed identically to the signature that would have been constructed by the honest receiver in HYB_2. By construction, $\mathsf{IdealSign}$ sets $\vec{\sigma}_i = (\sigma_{i,1,w_{i,1}}, \ldots, \sigma_{i,\ell,w_{i,\ell}})$ for all $i \in [t]$ where $w_{i,j} = b_{i,j}$ for all bad indices $i \in [t]$ and $j \in [\ell]$. For the remaining indices, $w_{i,j}$ is uniformly random subject to the restriction that $\bigoplus \vec{w}_i = \vec{x}$ where $\vec{w}_i = (w_{i,1}, \ldots, \vec{w}_{i,\ell})$. Observe that this is the *same* distribution from which the $w_{i,j}$ are sampled in HYB_2. Finally, $\mathsf{IdealSign}$ constructs the final signature σ' by computing $\vec{\sigma} \leftarrow \mathsf{SigEval}(f_{\mathsf{recon}}, \overrightarrow{\mathsf{pk}}, (\vec{w}_1, \ldots, \vec{w}_t), (\vec{\sigma}_1, \ldots, \vec{\sigma}_t))$. This is precisely the behavior of the honest receiver in HYB_2. In addition, by correctness of Π_{HS}, it will never be the case that $(\mathsf{vk}, (f_{\mathsf{id}}, \vec{x}), \vec{\sigma}, 0) \in \mathcal{L}$. Specifically, $\vec{\sigma}$ is a valid signature on $\vec{x}$ under vk, so the simulator $\mathcal{S}$ would never register it as an invalid signature in $\mathcal{F}_{\text{BHS}}$. (Because the signer is corrupt, the unforgeability criterion in signature verification is ignored).

We conclude that the output of the honest receiver in response to a signing query is identically distributed in HYB_2 and HYB_3.

- *Signature-verification*: If the environment $\mathcal{Z}$ activates the corrupt signer $\tilde{\mathbf{S}}$ on a verification query $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}', (f, \vec{x}), \vec{\sigma})$, the simulator $\mathcal{S}$ activates the real signer $\mathbf{S}$ (under the control of $\mathcal{A}$) in its simulated version of Π_{BHS}. Since the simulator $\mathcal{S}$ forwards $\mathbf{S}$'s output to $\mathcal{Z}$, the responses to these queries in HYB_2 and HYB_3 are identically distributed.

  Next, suppose the environment $\mathcal{Z}$ activates the honest receiver $\mathbf{R}$ on a signature verification query $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}', (f, \vec{x}), \vec{\sigma})$. In HYB_3, the ideal functionality $\mathcal{F}_{\text{BHS}}$ handles the signature-verification queries. We consider the different possibilities below. Note that because the signer is assumed to be corrupt, the unforgeability condition is ignored.

  - If $f \notin \mathcal{H}$, then $\mathcal{F}_{\text{BHS}}$ sets the verification bit $t = 0$. This is the behavior in HYB_2.
  - Otherwise, if $\mathsf{vk} = \mathsf{vk}'$ and $(\mathsf{vk}, (f, \vec{x}), \vec{\sigma}, 1) \in \mathcal{L}$, then $\mathcal{F}_{\text{BHS}}$ sets $t = 1$. We consider several scenarios depending on how the entry $(\mathsf{vk}, (f, \vec{x}), \vec{\sigma}, 1)$ was added to $\mathcal{L}$. If $\vec{\sigma}$ was generated as a result of a signing or a evaluation request involving the honest receiver, then by (evaluation and hiding) correctness of Π_{HS}, $\vec{\sigma}$ is a valid signature on $\vec{x}$, and the honest receiver in HYB_2 would also accept the signature. If the entry was added as a result of a previous verification request (which successfully verified), then because the honest party's verification algorithm in Π_{HS} is *deterministic* and since the signature previously verified, then the honest receiver would also output 1 in HYB_2.
  - Otherwise, if there is already an entry $(\mathsf{vk}', (f, \vec{x}), \vec{\sigma}, t') \in \mathcal{L}$ for some t', $\mathcal{F}_{\text{BHS}}$ sets $t = t'$. In the real protocol in HYB_2, the honest verifier's algorithm is deterministic. Hence, if a signature previously verified (resp., failed to verify), it will continue to verify (resp., fail to verify).
  - Finally, if none of the above criterion apply, then the ideal functionality allows the simulator $\mathcal{S}$ to decide the verification response in HYB_3. By construction, for the honest receiver, the simulator implements the same logic as in the real protocol in HYB_2.

  We conclude that the output of the honest receiver in response to verification queries is identically distributed in HYB_2 and HYB_3.

- *Signature-evaluation*: By definition, for any signature evaluation query made by $\mathcal{Z}$ to $\tilde{\mathbf{S}}$ in HYB_3, the simulator $\mathcal{S}$ invokes $\mathbf{S}$ (under the control of $\mathcal{A}$) in its simulated copy of Π_{BHS} and forwards $\mathbf{S}$'s output to $\mathcal{Z}$. Therefore, $\mathcal{Z}$'s views in HYB_2 and HYB_3 are identical.

  Next, suppose that the environment $\mathcal{Z}$ activates the honest receiver $\tilde{\mathbf{R}}$ on an evaluation query in HYB_3. In this case, the ideal functionality first verifies the signature (as argued above, the outcome of the signature verification procedure is identically distributed in HYB_2 and HYB_3), and then invokes the $\mathsf{IdealEval}$ algorithm provided by $\mathcal{S}$ to construct the signature. By construction, $\mathsf{IdealEval}$ is precisely the algorithm used in HYB_2 to generate the signatures (specifically, both $\mathsf{IdealEval}$ and the procedure in HYB_2 simulate the signatures using the context-hiding simulator for Π_{HS}). We conclude that the output of the honest receiver in response to an evaluation query is identically distributed in HYB_2 and HYB_3.

We conclude that on all queries, the view of the environment $\mathcal{Z}$ in HYB_2 and HYB_3 is identically distributed. □

Lemma D.7 now follows by combining Claims D.8, D.9, and D.10. □

Lemma D.11. *If both the signer and the receiver are corrupt, then* $\text{IDEAL}_{\mathcal{F}_{\text{BHS}},\mathcal{S},\mathcal{Z}} \equiv \text{REAL}_{\Pi_{\text{BHS}},\mathcal{A},\mathcal{Z}}$.

Proof. When both parties are corrupt, the simulator $\mathcal{S}$ only needs to simulate the behavior of the ideal OT functionality $\mathcal{F}_{\text{OT}}^{\ell,s}$ when simulating the protocol execution for adversary $\mathcal{A}$. Since $\mathcal{S}$ forwards all of the queries from $\mathcal{Z}$ to $\mathcal{A}$ (as if it came from $\mathcal{A}$'s environment in the simulated protocol execution), and moreover, $\mathcal{S}$ perfectly simulates the behavior of the $\mathcal{F}_{\text{OT}}^{\ell,s}$ functionality, the output of $\mathcal{S}$ in the ideal execution is distributed identically to the output of $\mathcal{A}$ in the real execution. □

Theorem 5.1 now follows by combining Lemmas D.1, D.4, D.7, and D.11. □

E Proof of Theorem 6.1

Let $\mathcal{A}$ be a static adversary that interacts with the environment $\mathcal{Z}$, a prover $\mathcal{P}$, and a verifier $\mathcal{V}$ running the real protocol Π_{ZK} (Figure 3). We construct an ideal world adversary (simulator) $\mathcal{S}$ that interacts with the environment $\mathcal{Z}$, a dummy prover $\tilde{\mathcal{P}}$, a dummy verifier $\tilde{\mathcal{V}}$, and ideal functionality $\mathcal{F}_{\mathsf{ZK}}$ such that no environment $\mathcal{Z}$ can distinguish an interaction with $\mathcal{A}$ in the real execution from one with $\mathcal{S}$ in the ideal execution.

We begin by describing the simulator $\mathcal{S}$. At the beginning of the protocol execution, the simulator $\mathcal{S}$ begins by invoking the adversary $\mathcal{A}$. Algorithm $\mathcal{A}$ begins by declaring which parties it would like to corrupt, and $\mathcal{S}$ corrupts the corresponding set of dummy parties. The simulation then proceeds as follows.

Simulating the communication with the environment. Whenever the simulator $\mathcal{S}$ receives an input from the environment $\mathcal{Z}$, it forwards the input to $\mathcal{A}$ (as if it came from the environment in the simulated protocol execution). Whenever $\mathcal{A}$ writes a message on its output tape (in the simulated protocol execution), the simulator $\mathcal{S}$ writes the same output on its own output tape (to be read by the environment).

Simulating the ideal BHS functionality. At the beginning of the protocol execution, the simulator $\mathcal{S}$ initializes an empty list $\mathcal{L}$ to keep track of the signatures in the simulated instance of $\mathcal{F}_{\text{BHS}}$. The simulator $\mathcal{S}$ simulates the ideal BHS functionality exactly as described in Figure 1.

Whenever the specification of $\mathcal{F}_{\text{BHS}}$ needs to interact with the ideal adversary, the simulator $\mathcal{S}$ forwards the request to $\mathcal{A}$ (as if it came from $\mathcal{F}_{\text{BHS}}$ in the simulated protocol execution), and uses the response from $\mathcal{A}$ to continue the simulation.

Simulating the preprocessing phase. In the preprocessing phase, the verifier and the prover never exchange any messages with each other. They only interact with the $\mathcal{F}_{\text{BHS}}$ functionality. As stated above, the simulator simulates the behavior of $\mathcal{F}_{\text{BHS}}$ exactly as described in Figure 1. If a party is corrupt, then the simulator uses $\mathcal{A}$ to determine the messages it sends to $\mathcal{F}_{\text{BHS}}$. If a party is honest, then $\mathcal{S}$ simulates the behavior of the honest party exactly as in the real protocol. Let $\widetilde{\mathsf{vk}}$ be the verification key the verifier sends to the prover in the simulated execution.

Simulating the proofs. After simulating the preprocessing phase, the simulator $\mathcal{S}$ proceeds as follows, depending on which parties are corrupt:

- *The prover is honest*: If the prover is honest, then the prover (in both the real and ideal executions) does nothing until it is activated by the environment. In the ideal execution, whenever the environment activates the prover on an input $(\mathsf{sid}, \mathsf{ssid}, \mathsf{prove}, \mathcal{R}, x, w)$ where $\mathcal{R}(x, w) = 1$, then $\mathcal{S}$ receives a tuple $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathcal{R}, x)$ from $\mathcal{F}_{\mathsf{ZK}}$. When this occurs, $\mathcal{S}$ simulates the request as follows. First, let $\widetilde{\mathsf{sk}}$ be the secret key the simulator $\mathcal{S}$ chose for the prover when simulating the preprocessing phase (since the prover is honest, $\mathcal{S}$ chooses the secret key). Then, $\mathcal{S}$ constructs a ciphertext $\widetilde{\mathsf{ct}} \leftarrow \mathsf{Encrypt}(\mathsf{sk}, 0^\tau)$, where τ denotes the length of the witness for relation $\mathcal{R}$. Next, $\mathcal{S}$ constructs a signature $\tilde{\sigma}^* \leftarrow \mathsf{IdealEval}(\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},x}, 1)$, where $\mathsf{IdealEval}$ is the ideal signature evaluation functionality that $\mathcal{A}$ chooses for $\mathcal{F}_{\text{BHS}}$. The simulator $\mathcal{S}$ constructs the simulated proof $\tilde{\pi} = (\widetilde{\mathsf{ct}}, \tilde{\sigma}^*)$, adds the signature $(\mathsf{vk}, (\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},x}, 1), \tilde{\sigma}^*, 1)$ to $\mathcal{L}$ (if an entry does not already exist), and sends $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, x, \tilde{\pi})$ to the verifier in the simulated protocol execution.

- *The prover is corrupt*: First, if the verifier is also corrupt, then the simulator $\mathcal{S}$ only needs to simulate the BHS functionality $\mathcal{F}_{\text{BHS}}$. Specifically, whenever the environment activates the prover on an input in the ideal execution, the simulator simply forwards the input to the (corrupt) prover in the simulated execution.

  On the other hand, if the verifier is honest, then $\mathcal{S}$ proceeds as follows:

  - At the beginning of the simulation, $\mathcal{S}$ initializes $\widetilde{\mathsf{sk}}$ to $\perp$. At any point in the simulated protocol execution, if the prover (as dictated by $\mathcal{A}$) makes a successful signing request to the $\mathcal{F}_{\text{BHS}}$ functionality, the simulator $\mathcal{S}$ updates $\widetilde{\mathsf{sk}}$ to be the message the prover submitted to the signing functionality. By definition of the $\mathcal{F}_{\text{BHS}}$ functionality, the prover can make at most one successful signing request to the $\mathcal{F}_{\text{BHS}}$.
  - Whenever the environment activates the prover in the ideal execution on an input $(\mathsf{sid}, \mathsf{ssid}, \mathsf{prove}, \mathcal{R}, x, w)$, the simulator $\mathcal{S}$ activates the prover in the simulated protocol execution on the same input.
  - Whenever the prover in the simulated execution sends a message $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathcal{R}, x, \pi)$ to the verifier, the simulator parses $\pi = (\mathsf{ct}, \sigma^*)$. If π does not have this form or if $\widetilde{\mathsf{sk}} = \perp$, then $\mathcal{S}$ ignores the message. Otherwise, the simulator $\mathcal{S}$ submits the request $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}, (\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},x}, 1), \sigma^*)$ to its (simulated) ideal functionality $\mathcal{F}_{\text{BHS}}$. If the signature does not verify, then $\mathcal{S}$ ignores the request. Otherwise, it computes $w \leftarrow \mathsf{Decrypt}(\widetilde{\mathsf{sk}}, \mathsf{ct})$ and outputs $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathcal{R}, x)$ for the honest verifier in the

simulated execution. In addition, $\mathcal{S}$ submits $(\mathsf{sid}, \mathsf{ssid}, \mathsf{prove}, \mathcal{R}, x, w)$ to Π_{ZK} (on behalf of the prover $\tilde{\mathcal{P}}$).

To conclude the proof, we show that the environment cannot distinguish the output of the real execution with adversary $\mathcal{A}$ from an ideal execution with the simulator $\mathcal{S}$. We consider the two cases separately.

Lemma E.1. *If the prover is honest, and Π_{SE} is a CPA-secure encryption scheme, then in the $\mathcal{F}_{\text{BHS}}$-hybrid model, $\text{REAL}_{\Pi_{\mathsf{ZK}},\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \text{IDEAL}_{\mathcal{F}_{\mathsf{ZK}},\mathcal{S},\mathcal{Z}}$.*

Proof. Our proof proceeds via a hybrid argument:

- HYB_0: This is the real distribution $\text{REAL}_{\Pi_{\mathsf{ZK}},\mathcal{A},\mathcal{Z}}$.
- HYB_1: Same as HYB_0, except when constructing proofs, the honest prover does not 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 the signature σ^*. Instead, the signature is constructed as $\sigma^* \leftarrow \mathsf{IdealEval}(\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},x}, 1)$. Afterwards, the entry $(\mathsf{vk}, (\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},x}, 1), \sigma^*, 1)$ is added to $\mathcal{F}_{\text{BHS}}$ (if an entry does not already exist.)
- HYB_2: Same as HYB_1, except during the preprocessing phase, the honest prover sends $(\mathsf{sid}, \mathsf{sign}, \mathsf{vk}, \mathsf{sk}')$ to $\mathcal{F}_{\text{BHS}}$ where $\mathsf{sk}' \leftarrow \mathsf{KeyGen}(1^\lambda)$ is generated independently of sk. The ciphertexts in the encryption step are still generated using sk.
- HYB_3: Same as HYB_2, except the honest prover encrypts the all-zeroes string 0^τ (where τ is the bit-length of the witness) when constructing the proofs.
- HYB_4: Same as HYB_3, except the honest prover requests the signature on sk in the preprocessing step (instead of the dummy key sk').
- HYB_5: This is the ideal distribution $\text{IDEAL}_{\mathcal{F}_{\mathsf{ZK}},\mathcal{S},\mathcal{Z}}$.

We now show that assuming Π_{SE} is CPA-secure, the outputs of each pair of consecutive hybrid experiments are computationally indistinguishable.

- Hybrids HYB_0 and HYB_1 are identical experiments according to the specification of $\mathcal{F}_{\text{BHS}}$. Specifically, since the prover is honest, the ideal functionality $\mathcal{F}_{\text{BHS}}$ answers the signature-evaluation queries using the ideal evaluation function $\mathsf{IdealEval}$, which is precisely the procedure described in HYB_1.
- Hybrids HYB_1 and HYB_2 are computationally indistinguishable if the encryption scheme $(\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ is CPA-secure. First, the only difference in HYB_1 and HYB_2 is that in HYB_2, the entry $(\mathsf{sid}, \mathsf{vk}, (f_{\mathsf{id}}, \mathsf{sk}), \sigma_{\mathsf{sk}})$ in $\mathcal{F}_{\text{BHS}}$ is replaced with the entry $(\mathsf{sid}, \mathsf{vk}, (f_{\mathsf{id}}, \mathsf{sk}'), \sigma_{\mathsf{sk}})$. We consider two cases, depending on whether the verifier is honest or corrupt.

  The verifier is honest: If the verifier is honest, then these two experiments are identically distributed. Specifically, the only queries the honest verifier makes to $\mathcal{F}_{\text{BHS}}$ are on (computed) signatures σ^* that are registered with $\mathcal{F}_{\text{BHS}}$.

  The verifier is corrupt: In this case, the adversary $\mathcal{A}$ can make arbitrary queries (on behalf of the verifier) to the $\mathcal{F}_{\text{BHS}}$ functionality. In addition, since the verifier is the signer (with respect

to the $\mathcal{F}_{\text{BHS}}$ functionality), during signature verification, only the correctness and consistency conditions are checked (and in particular, *not* the unforgeability condition). This means that the view of adversary $\mathcal{A}$ is identically distributed in HYB_1 and HYB_2 unless $\mathcal{A}$ makes an evaluation or a verification query to $\mathcal{F}_{\text{BHS}}$ on a message of the form $(f_{\mathsf{id}}, \mathsf{sk})$ or $(f_{\mathsf{id}}, \mathsf{sk}')$. We first show that in HYB_2, the probability that $\mathcal{A}$ makes a verification query to $\mathcal{F}_{\text{BHS}}$ on a message of the form $(f_{\mathsf{id}}, \mathsf{sk}')$ is negligible. By construction, in HYB_2, the adversary's view is *completely* independent of sk', so we can effectively defer the sampling of sk' until *after* the adversary has made all of its verification queries. Since the adversary makes $\mathsf{poly}(\lambda)$ verification queries, and sk' is drawn from a distribution with min-entropy at least $\omega(\log \lambda)$,[17] the probability (taken over the randomness of the key-generation algorithm) that sk' coincides with a message in the adversary's query is negligible.

We now show that if there exists an adversary $\mathcal{A}$ and an environment $\mathcal{Z}$ such that the outputs of HYB_1 and HYB_2 are not computationally indistinguishable, then we can construct an adversary $\mathcal{B}$ that breaks CPA-security of Π_{SE}. Based on the above analysis, to achieve non-negligible distinguishing advantage, algorithm $\mathcal{A}$ has to issue a verification query on the message $(f_{\mathsf{id}}, \mathsf{sk})$ to the ideal functionality $\mathcal{F}_{\text{BHS}}$ with non-negligible probability. Algorithm $\mathcal{B}$ simulates an instance of the protocol execution environment according to HYB_2 as follows:

- Then, $\mathcal{B}$ starts the protocol execution experiment by activating the environment $\mathcal{Z}$. Algorithm $\mathcal{B}$ now simulates the protocol execution experiment as described in HYB_2.
- To simulate the honest prover during the preprocessing phase, $\mathcal{B}$ leaves the secret key sk unspecified (since it is not needed in the simulation). It samples $\mathsf{sk}' \leftarrow \mathsf{KeyGen}(1^\lambda)$ for the honest prover and simulates the rest of the preprocessing as described in HYB_2.
- Whenever the environment activates the prover to construct a proof on a statement-witness pair (x, w) for a relation $\mathcal{R}$, algorithm $\mathcal{B}$ simulates the honest prover in HYB_2 by first checking that $\mathcal{R}(x, w) = 1$. If so, then $\mathcal{B}$ submits an encryption query on the pair (w, w) to the encryption oracle to obtain a ciphertext ct. Algorithm $\mathcal{B}$ simulates the rest of the protocol exactly as described in HYB_2.
- Algorithm $\mathcal{B}$ simulates the $\mathcal{F}_{\text{BHS}}$ functionality according to the specification of HYB_1 and HYB_2 (the behavior of $\mathcal{F}_{\text{BHS}}$ is identical in the two hybrids, and does not depend on sk).
- At the end of the protocol execution experiment, algorithm $\mathcal{B}$ chooses a random bit-string $\xi \xleftarrow{R} \{0,1\}^\lambda$. It makes ξ chosen-message queries to the encryption oracle on pairs $(\xi_1, 0), \ldots, (\xi_\lambda, 1)$ to obtain ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_\lambda$. Then, for each verification query made by adversary $\mathcal{A}$ to $\mathcal{F}_{\text{BHS}}$ on a message of the form $(f_{\mathsf{id}}, \widehat{\mathsf{sk}})$ for some $\widehat{\mathsf{sk}}$, algorithm $\mathcal{B}$ checks to see if for all $i \in [\lambda]$, $\mathsf{Decrypt}(\widehat{\mathsf{sk}}, \mathsf{ct}_i) = \xi_i$. If this holds for all $i \in [\lambda]$, then $\mathcal{B}$ outputs 1. Otherwise, it outputs 0.

By definition, we see that $\mathcal{B}$ perfectly simulates the protocol execution according to the specification in HYB_2. By assumption, with non-negligible probability ε, algorithm $\mathcal{A}$ will issue a query to $\mathcal{F}_{\text{BHS}}$ on the message $(f_{\mathsf{id}}, \mathsf{sk})$. This means that with probability ε, there is some $\widehat{\mathsf{sk}}$ where $\widehat{\mathsf{sk}} = \mathsf{sk}$.

[17] This is implied by CPA-security of the encryption scheme. Otherwise, the adversary has a noticeable probability of guessing the key for the encryption scheme, which trivially breaks CPA-security.

– Suppose $\mathcal{B}$ is interacting with the encryption oracle $\mathcal{O}_0$ in the CPA-security game. In this case, if there is a message of the form $(f_{\mathsf{id}}, \widehat{\mathsf{sk}})$ where $\widehat{\mathsf{sk}} = \mathsf{sk}$, then $\mathcal{B}$ outputs 1 by correctness of the encryption scheme. In this case, $\mathcal{B}$ outputs 1 with probability at least ε.

– Suppose instead that $\mathcal{B}$ is interacting with the encryption oracle $\mathcal{O}_1$. In this case, the ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_\lambda$ and keys $\widehat{\mathsf{sk}}$ are all independent of ξ_i. Thus, union bounding over all of the messages of the form $(f_{\mathsf{id}}, \widehat{\mathsf{sk}})$ appearing in the verification queries to $\mathcal{F}_{\text{BHS}}$, the probability that $\mathcal{B}$ outputs 1 (taken over the choice of ξ_i's) is at most $\mathsf{poly}(\lambda)/2^\lambda = \mathsf{negl}(\lambda)$.

We conclude that $\mathcal{B}$ is able to break CPA-security with non-negligible probability $\varepsilon - \mathsf{negl}(\lambda)$. Thus, if the encryption scheme is CPA-secure, the outputs of hybrids HYB_1 and HYB_2 are computationally indistinguishable.

- Hybrids HYB_2 and HYB_3 are computationally indistinguishable if Π_{SE} is CPA-secure. Observe that none of the logic in HYB_2 and HYB_3 depend on the secret key sk, and all of the messages can be simulated given access to an encryption oracle $\mathsf{Encrypt}(\mathsf{sk}, \cdot)$. By CPA-security of Π_{SE}, we conclude that the outputs of these two hybrid experiments are computationally indistinguishable.

- Hybrids HYB_3 and HYB_4 are computationally indistinguishable if Π_{SE} is CPA-secure. The argument follows by the same logic as that used to argue computational indistinguishability of HYB_1 and HYB_2.

- Hybrids HYB_4 and HYB_5 are identically distributed. By construction, the honest prover's behavior in HYB_4 precisely coincides with the behavior of the simulated prover in HYB_5. Thus, the outputs of $\mathcal{A}$ in HYB_4 are distributed exactly as the outputs of $\mathcal{S}$ in HYB_5. Moreover, if the verifier is honest, then the outputs of the honest verifier in HYB_4 are distributed identically to the outputs in HYB_5; this follows by appealing to the correctness property of the ideal $\mathcal{F}_{\text{BHS}}$ functionality. We conclude that the output distribution of HYB_4 is identical to that of the ideal execution.

Since each pair of hybrid arguments is computationally indistinguishable, the lemma follows. □

Lemma E.2. *If the prover is corrupt, then in the $\mathcal{F}_{\text{BHS}}$-hybrid model, we have that* $\text{REAL}_{\Pi_{\mathsf{ZK}}, \mathcal{A}, \mathcal{Z}} \equiv \text{IDEAL}_{\mathcal{F}_{\mathsf{ZK}}, \mathcal{S}, \mathcal{Z}}$.

Proof. In the case where the prover is corrupt, we show that the output of the real and ideal protocol executions are identically distributed. We consider two cases.

The verifier is corrupt: If the verifier is also corrupt, then the simulator $\mathcal{S}$ is only responsible for simulating the $\mathcal{F}_{\text{BHS}}$ functionality. Since $\mathcal{S}$ simulates the ideal BHS functionality exactly as described in Figure 1, the output of $\mathcal{S}$ is identically distributed as the output of $\mathcal{A}$ in the real execution, and the claim follows.

The verifier is honest: If the verifier is honest, we show that $\mathcal{S}$ perfectly simulates the behavior of the honest verifier in the simulated protocol execution. By construction, $\mathcal{S}$ perfectly simulates the behavior of the honest verifier in the preprocessing phase. Next, in the real execution, the honest verifier only responds when it receives a tuple of the form $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathcal{R}, x, \pi)$ from the prover. We show that the simulation is correct:

- Suppose in the real scheme, the verifier has not set the ready flag. This corresponds to the setting where the prover has never made a signing request to $\mathcal{F}_{\text{BHS}}$. In this case, the verifier ignores the request. In the simulated protocol execution, if the prover never makes a signing request to $\mathcal{F}_{\text{BHS}}$, then $\widetilde{\mathsf{sk}} = \perp$, and the verifier also ignores the request.

- Suppose in the real scheme, the proof π does not have the form (ct, σ). In this case, the verifier also ignores the request. This is precisely how $\mathcal{S}$ simulates the honest verifier's behavior in the simulated protocol execution.

- Otherwise, in the real scheme, the honest verifier parses the proof as $\pi = (\mathsf{ct}, \sigma)$, and submits $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}, (\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},x}, 1), \sigma)$ to $\mathcal{F}_{\text{BHS}}$. We consider several cases:

  Case 1: Suppose that $(\mathsf{vk}, (\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},x}, 1), \sigma, 1) \in \mathcal{L}$, where $\mathcal{L}$ is the list of signatures maintained by $\mathcal{F}_{\text{BHS}}$. In this case, $\mathcal{F}_{\text{BHS}}$ declares the signature valid, and the honest verifier in the real scheme accepts the proof by outputting $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathcal{R}, x)$. According to the specification of $\mathcal{F}_{\text{BHS}}$, there are two possible ways for $(\mathsf{vk}, (\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},x}, 1), \sigma, 1)$ to be added to $\mathcal{L}$:

  - The prover made a successful evaluation query with function $\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},x}$ on some input sk where $\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},x}(\mathsf{sk}) = 1$, and moreover, there is an entry $(\mathsf{vk}, (f_{\mathsf{id}}, \mathsf{sk}), \sigma', 1) \in \mathcal{L}$ for some σ'.
  - The prover previously made a verification query on $(\mathsf{vk}, (\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},x}, 1), \sigma, 1)$ and the adversary decided the verification result. According to the $\mathcal{F}_{\text{BHS}}$ specification, the adversary chooses the verification output only if there exists $(\mathsf{vk}, (f_{\mathsf{id}}, \mathsf{sk}), \sigma', 1) \in \mathcal{L}$ for some σ' where $\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},x}(\mathsf{sk}) = 1$.

  We conclude that in this case, there exist sk and σ' where $(\mathsf{vk}, (f_{\mathsf{id}}, \mathsf{sk}), \sigma', 1) \in \mathcal{L}$ and $\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},x}(\mathsf{sk}) = 1$. Since the verifier is honest, by the specification of $\mathcal{F}_{\text{BHS}}$, this is possible only if the prover has previously made a successful signing request on sk. This means that in the simulated protocol execution, the prover must have submitted a signing request to $\mathcal{F}_{\text{BHS}}$ on message sk. By construction of the simulator, $\widetilde{\mathsf{sk}} = \mathsf{sk}$. Now, in the simulation, $\mathcal{S}$ computes $\mathsf{Decrypt}(\widetilde{\mathsf{sk}}, \mathsf{ct})$ to obtain w. Since $\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},x}(\mathsf{sk}) = 1$, this means that $\mathcal{R}(x, w) = 1$. In the ideal execution, the simulator sends $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathcal{R}, x, w)$ to Π_{ZK}, which by definition forwards the output $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathcal{R}, x)$ to the dummy verifier. Thus, in this case, the honest verifier's behavior in both the real and ideal executions is identical.

  Case 2: Suppose that $(\mathsf{vk}, (\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},x}, 1), \sigma, 1) \notin \mathcal{L}$. We consider two possibilities.

  - If there does not exist an entry $(\mathsf{vk}, (f_{\mathsf{id}}, \mathsf{sk}), \sigma', 1)$ where $\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},x}(\mathsf{sk}) = 1$ in the list $\mathcal{L}$ for some sk and σ', then by the unforgeability condition, the ideal functionality $\mathcal{F}_{\text{BHS}}$ declares the signature invalid, and the honest verifier in the real scheme ignores the message. Since $\mathcal{S}$ simulates the ideal functionality $\mathcal{F}_{\text{BHS}}$ perfectly, the simulator $\mathcal{S}$ also ignores the message in the simulated execution.
  - On the other hand, if $\mathcal{L}$ does contain an entry $(\mathsf{vk}, (f_{\mathsf{id}}, \mathsf{sk}), \sigma', 1)$ for some sk and σ' where $\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},x}(\mathsf{sk}) = 1$, then $\mathcal{F}_{\text{BHS}}$ allows the adversary to decide whether the signature is valid or not. If the adversary declares the signature invalid, then in both

the real and the simulated executions, the verifier ignores the message. If the adversary declares the signature valid, then in the real execution, the verifier accepts the proof and outputs $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathcal{R}, x)$. In the simulated execution, because there does exist $(\mathsf{vk}, (f_{\mathsf{id}}, \mathsf{sk}), \sigma', 1) \in \mathcal{L}$ where $\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},x}(\mathsf{sk}) = 1$, we can apply the same analysis from Case 1 to argue that in the ideal execution, the simulated verifier also accepts the proof and outputs $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathcal{R}, x)$. Moreover, in this case, $\mathcal{S}$ also forwards $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathcal{R}, x, w)$ where $w \leftarrow \mathsf{Decrypt}(\widetilde{\mathsf{sk}}, \mathsf{ct})$ and $\mathcal{R}(x, w) = 1$ to Π_{ZK}. This means that the honest verifier in the ideal execution also outputs $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathcal{R}, x)$.

From the above analysis, we see that in all cases, the behavior of the honest verifier in both the real execution and the ideal execution is identical. Moreover, algorithm $\mathcal{S}$ perfectly simulates the view of $\mathcal{A}$ in the simulated protocol execution. The lemma follows. □

Combining Lemmas E.1 and E.2, we conclude that the Π_{ZK} protocol securely realizes $\mathcal{F}_{\mathsf{ZK}}$ in the presence of malicious adversaries in the $\mathcal{F}_{\text{BHS}}$-hybrid model. □