Exploring Constructions of Compact NIZKs from Various Assumptions

Shuichi Katsumata[1,2], Ryo Nishimaki[3], Shota Yamada[1], Takashi Yamakawa[3]

[1]National Institute of Advanced Industrial Science and Technology (AIST), Tokyo, Japan
{shuichi.katsumata,yamada-shota}@aist.go.jp
[2]The University of Tokyo, Tokyo, Japan
[3]NTT Secure Platform Laboratories, Tokyo, Japan
{ryo.nishimaki.zk,takashi.yamakawa.ga}@hco.ntt.co.jp

June 2, 2019

Abstract

A non-interactive zero-knowledge (NIZK) protocol allows a prover to non-interactively convince a verifier of the truth of the statement without leaking any other information. In this study, we explore shorter NIZK proofs for all **NP** languages. Our primary interest is NIZK proofs from falsifiable pairing/pairing-free group-based assumptions. Thus far, NIZKs in the common reference string model (CRS-NIZKs) for **NP** based on falsifiable pairing-based assumptions all require a proof size at least as large as $O(|C|\kappa)$, where C is a circuit computing the **NP** relation and κ is the security parameter. This holds true even for the weaker designated-verifier NIZKs (DV-NIZKs). Notably, constructing a (CRS, DV)-NIZK with proof size achieving an *additive*-overhead $O(|C|) + \mathsf{poly}(\kappa)$, rather than a multiplicative-overhead $|C| \cdot \mathsf{poly}(\kappa)$, based on any falsifiable pairing-based assumptions is an open problem.

In this work, we present various techniques for constructing NIZKs with *compact* proofs, i.e., proofs smaller than $O(|C|) + \mathsf{poly}(\kappa)$, and make progress regarding the above situation. Our result is summarized below.

- We construct CRS-NIZK for all **NP** with proof size $|C| + \mathsf{poly}(\kappa)$ from a (non-static) falsifiable Diffie-Hellman (DH) type assumption over pairing groups. This is the first CRS-NIZK to achieve a compact proof without relying on either lattice-based assumptions or non-falsifiable assumptions. Moreover, a variant of our CRS-NIZK satisfies universal composability (UC) in the erasure-free adaptive setting. Although it is limited to **NP** relations in $\mathbf{NC}^1$, the proof size is $|w| \cdot \mathsf{poly}(\kappa)$ where w is the witness, and in particular, it matches the state-of-the-art UC-NIZK proposed by Cohen, shelat, and Wichs (CRYPTO'19) based on lattices.
- We construct (multi-theorem) DV-NIZKs for **NP** with proof size $|C| + \mathsf{poly}(\kappa)$ from the computational DH assumption over *pairing-free* groups. This is the first DV-NIZK that achieves a compact proof from a standard DH type assumption. Moreover, if we further assume the **NP** relation to be computable in $\mathbf{NC}^1$ and assume hardness of a (non-static) falsifiable DH type assumption over *pairing-free* groups, the proof size can be made as small as $|w| + \mathsf{poly}(\kappa)$.

Another related but independent issue is that all (CRS, DV)-NIZKs require the running time of the prover to be at least $|C| \cdot \mathsf{poly}(\kappa)$. Considering that there exists NIZKs with efficient verifiers whose running time is strictly smaller than $|C|$, it is an interesting problem whether we can construct *prover-efficient* NIZKs. To this end, we construct prover-efficient CRS-NIZKs for **NP** with compact proof through a generic construction using laconic functional evaluation schemes (Quach, Wee, and Wichs (FOCS'18)). This is the first NIZK in any model where the running time of the prover is strictly smaller than the time it takes to compute the circuit C computing the **NP** relation.

Finally, perhaps of an independent interest, we formalize the notion of *homomorphic equivocal commitments*, which we use as building blocks to obtain the first result, and show how to construct them from pairing-based assumptions.

1 Introduction

1.1 Background

Zero-knowledge (ZK) protocols, introduced by Goldwasser, Micali, and Rackoff [GMR89], allow a prover to convince a verifier of the truth of a statement without leaking any knowledge other than the fact that the statement is indeed true. A practically useful and theoretically alluring feature for a ZK protocol to have is *non-interactiveness*, where a prover simply outputs a single message (called a proof) and convinces the verifier of the truth of the statement. Unfortunately, it is known that non-interactive ZK (NIZK) for non-trivial languages do not exist in the plain model where there is no trusted setup [GO94]. However, Blum, Feldman, and Micali [BFM88] showed how to construct a NIZK in a setting where the prover and verifier have access to a shared *common reference string* (as known as CRS-NIZK). Since then, NIZKs have been used as a ubiquitous building block for cryptography ranging from the early chosen-ciphertext secure public key encryption schemes [NY90, DDN00, Sah99], advanced signature schemes [Cv91, RST01, BMW03], and multi-party computation [GMW87].

Compact NIZK. One of the important research topics for NIZK is making the proof size as small as possible. So far, CRS-NIZK for all of **NP** in the standard model is known to exist from (doubly-enhanced) trapdoor permutation [FLS99, BY96, Gol04], pairing [Gro10a, Gro10b, Lip12, GOS12, GS12, GGPR13], indistinguishability obfuscation (iO) [SW14, BP15, BPW16, CL18], or correlation intractable hash function [KRR17, CCRR18, CCH+19]. Among these, CRS-NIZKs that have proof size independent of the size of the circuit C computing the **NP** relation are limited to those based on either a knowledge assumption [Gro10b, Lip12, GGPR13] or iO [SW14]. There also exist generic conversions from standard CRS-NIZKs to CRS-NIZKs with proof size independent of $|C|$. However, they rely on fully homomorphic encryption (FHE) [Gen09, GGI+15] or homomorphic trapdoor functions (HTDF) [CsW19] whose existence is only implied from lattice-based assumptions. Put differently, the classical CRS-NIZKs based on trapdoor permutations or (falsifiable [Nao03, GW11]) pairing-based assumptions all require a large proof size that is polynomially related to the circuit size $|C|$. Notably, even the most well-known Groth-Ostrovsky-Sahai NIZK (GOS-NIZK) [GOS12] based on the decisional linear or subgroup decision assumptions over pairing groups requires the proof size to be as large as $O(|C|\kappa)$, where κ is the security parameter. In fact, the CRS-NIZK with the shortest proof that does not rely on any of the above strong tools is the NIZK of Groth [Gro10a] based on the security of Naccache-Stern public key encryption scheme [NS98] which achieves proof size $|C| \cdot \mathsf{polylog}(\kappa)$. Therefore, it remains an interesting open problem to construct CRS-NIZKs with proof size smaller than the current state-of-the-art while avoiding to rely on strong tools such as knowledge assumptions, iO, FHE, and HTDF. Specifically, in this paper, one of the primary interest is to obtain a CRS-NIZK with proof size achieving an *additive*-overhead $O(|C|) + \mathsf{poly}(\kappa)$, rather than a multiplicative-overhead $|C| \cdot \mathsf{poly}(\kappa)$ (or $|C| \cdot \mathsf{polylog}(\kappa)$), based on any falsifiable pairing-based assumptions. Hereafter, we call such NIZKs with proof size $O(|C|) + \mathsf{poly}(\kappa)$ as NIZKs with *compact* proofs for simplicity.

Designated Verifier NIZKs and Compact Proofs. A relaxation of CRS-NIZKs called the *designated verifier* NIZKs (DV-NIZKs) [PsV06, DFN06] retain most of the useful properties of CRS-NIZKs and in some applications can be used as a substitute for CRS-NIZKs. The main difference between CRS and DV-NIZKs is that the latter limits the proof to only be verifiable by a designated party in possession of a verification key; the proof can still be generated by anybody as in CRS-NIZKs. Due to this extra secret information possessed by the verifier, DV-NIZKs suffer from the so-called verifier rejection attack. Specifically, a prover may learn partial information of the secret verification key and break soundness if the verifier uses the same verification key for verifying multiple statements. In this paper, our primary interest is *multi-theorem* DV-NIZKs (also known as *reusable* or *unbounded-soundness* DV-NIZKs) where the verification key can be reused for multiple statements without compromising soundness. Surprisingly, most DV-NIZKs [PsV06, DFN06, VV09, CG15, Lip17, CC18] (that are not a simple downgrade of CRS-NIZKs) are known to either suffer from the verifier rejection attack or to be limited to specific **NP** languages. It was not until recently that the first multi-theorem DV-NIZK for all **NP** languages was (concurrently and independently) shown by Couteau and Hofheinz [CH19], Katsumata et al. [KNYY19], and Quach et al. [QRW19]. They proposed a tweak to the classical Feige-Lapidot-Shamir (FLS) NIZK protocol [FLS99] and showed for the first time how to construct DV-NIZKs from the computational Diffie-Hellman (CDH) assumption over *pairing-free* groups; an assumption which is not yet known to imply CRS-NIZKs. However, one drawback of their DV-NIZK is that the CRS size and proof size are huge, i.e., $\mathsf{poly}(\kappa, |C|)$. This is due to the fact that the FLS NIZK, which they base their construction on, is highly specific to the

NP-complete Hamiltonicity problem. It is unclear if we can make their scheme compact since all other (CRS-)NIZKs following the footsteps of FLS NIZK such as [Kil94, KP98, Gro10a] suffer from the same problem of having large CRS and proof size. Therefore, it is unclear whether such a weak assumption as CDH over pairing-free groups can be used to construct a DV-NIZK with compact proofs. In fact, constructing DV-NIZKs with compact proof from any *pairing/pairing-free* group assumptions remains open.

Prover-Efficient NIZKs. Continuing the line of NIZKs with compact proofs, it is very natural and appealing to consider NIZKs that enjoy *efficient provers*, i.e., the running time of the prover is small. We say the prover is efficient if its running time is strictly smaller than the time it takes to compute $C(x, w)$ for statement x and witness w, where recall C was the circuit computing the **NP** relation. As an example, we can imagine a case where a user (acting as a prover) is given some sort of credential w as a witness by a trusted authority and is required to prove in zero-knowledge the fact that it possesses a valid credential to make some action. More concretely, in group signatures [BMW03] a trusted authority will provide users with a credential which allows them to sign anonymously on behalf of the group. In such a case, it would be appealing if the user could generate a proof without requiring to invest computational time-dependent of $|C|$, since if zero-knowledge was not required, the prover could have simply output the credential w in the clear and completely outsourced the computation of $C(x, w)$ to the verifier. Since the authority is providing a valid credential w to the user, in principle, the user should never need to compute $C(x, w)$ to check whether w is valid.

As far as our knowledge goes, all NIZKs, regardless of CRS or DV, have a prover with running time at least $|C| \cdot \mathsf{poly}(\kappa)$ which is much larger than the time it takes to simply compute the circuit C. We emphasize that solutions to the counterpart notion of *efficient verifiers* are well known and studied. Specifically, NIZKs with compact proofs with the additional property of having efficient verifiers are known as ZK-succinct non-interactive arguments (ZK-SNARGs) or ZK-succinct non-interactive arguments of knowledge (ZK-SNARKs).[1] They have been the subject of extensive research, e.g., [Gro10a, Lip12, BCCT12, GGPR13, Lip13, DFGK14, PHGR16, Gro16], where constructions are known to exist either in the random oracle model or based on non-falsifiable assumptions. We also note that it would be impossible to construct a NIZK where both the prover *and* the verifier are efficient since the circuit C representing the **NP** relation must be computed by at least one of the parties to check the validity of the witness w. Therefore, it is an interesting question of whether there exists an opposite flavor of the current NIZKs where we have an efficient prover instead of an efficient verifier.

1.2 Our Contribution

In this paper, we provide new constructions of CRS-NIZK and DV-NIZK with compact proofs. The former is instantiated on a pairing group and the latter on a paring-free group. The tools and techniques which we use for our CRS-NIZK can be slightly modified to construct universally composable NIZK (UC-NIZK) [GOS12] with compact proofs over pairing groups. Finally, we provide a generic construction of a CRS-NIZK with an efficient prover using as a building block the recently proposed laconic functional evaluation (LFE) scheme of Quach, Wee, and Wichs [QWW18]. We summarize our results below and refer to Table 1, 2, and 3 for a comparison between prior works. We note that we only include multi-theorem NIZKs supporting all of **NP** based on falsifiable assumptions in the table.

1. We construct CRS-NIZKs for **NP** with compact proof from a (non-static) assumption over pairing groups, namely, the (n, m)-computational Diffie-Hellman exponent and ratio (CDHER) assumption introduced by [KNYY19]. This is the first CRS-NIZK to achieve a compact proof without relying on either lattice-based assumptions, knowledge assumptions, or indistinguishability obfuscation. The proof size has an additive-overhead $|C| + \mathsf{poly}(\kappa)$, rather than a multiplicative-overhead $|C| \cdot \mathsf{poly}(\kappa)$, where C is the circuit that computes the **NP** relation (See Table 1). Moreover, if we assume the **NP** relation to be computable in $\mathbf{NC}^1$, we can make the proof size as small as $|w| + \mathsf{poly}(\kappa)$, where w is the witness. This matches the proof size of the CRS-NIZK of Gentry et al. [GGI$^+$15] based on fully-homomorphic encryption.

2. We construct UC-NIZKs for **NP** relations in $\mathbf{NC}^1$ with compact proof from the (n, m)-CDHER assumption. Although it is limited to **NP** relations in $\mathbf{NC}^1$, it matches the smallest proof size among all the UC-NIZKs secure against adaptive corruptions in the erasure-free setting (See Table 2). The proof size is small as $|w| \cdot \mathsf{poly}(\kappa)$, and

[1] We note that in ZK-SNARG/SNARK, it is conventional to require an efficient verifier to have running time that is only poly-logarithmic dependent of $|C|$, rather than being just strictly smaller than $|C|$.

in particular, matches the recent UC-NIZK of Cohen, shelat, and Wichs [CsW19] based on lattice-assumptions. Here, note that for $\mathbf{NC}^1$ circuits, the dependence on the depth d they have can be ignored, since asymptotically d is smaller than κ.

3. We construct (multi-theorem) DV-NIZKs for **NP** with compact proof from the CDH assumption over *pairing-free* groups. This is the first DV-NIZK that achieves a compact proof from a weak and static Diffie-Hellman type assumption such as CDH. Specifically, similarly to the above CRS-NIZK, the proof size of our DV-NIZK is $|C| + \mathsf{poly}(\kappa)$, whereas all previous DV-NIZKs had proof size $\mathsf{poly}(|C|, \kappa)$ (See Table 3). Moreover, if we further assume the **NP** relation to be computable in $\mathbf{NC}^1$ and assume the hardness of the parameterized ℓ-computational Diffie-Hellman inversion (CDHI) assumption over *pairing-free* groups [MSK02, CF18], we can make the proof size as small as $|w| + \mathsf{poly}(\kappa)$.

4. Finally, we construct prover-efficient CRS-NIZKs for **NP** through a generic construction using LFE schemes [QWW18]. This is the first NIZK in any model (e.g., CRS, DV) where the running time of the prover is strictly smaller than the time it takes to compute the circuit C computing the **NP** relation. Using any non-prover-efficient CRS-NIZK, we generically construct a CRS-NIZK where the running time of the prover (and the proof size) is $\mathsf{poly}(\kappa, |x|, |w|, d)$, independent of the circuit size $|C|$, by instantiating the LFE scheme by the sub-exponential security of the learning with errors (LWE) assumption with sub-exponential modulus-to-noise ratio, where x is the statement and d is the depth of C. Moreover, if we use as building block a CRS-NIZK whose prover running time is smaller than $|C| \cdot \mathsf{poly}(\kappa)$ (e.g., [GOS12]), the running time and proof size can be made as small as $\tilde{O}(|x| + |w|) \cdot \mathsf{poly}(\kappa, d)$ by instantiating the LFE scheme by the *adaptive* LWE assumption with sub-exponential modulus-to-noise ratio introduced in [QWW18].

Along the way of obtaining our first and second results, we formalize a new tool called *homomorphic equivocal commitments* (HEC)[2], which may be of independent interest. An HEC is a commitment with two additional properties called *equivocality* and *homomorphism*. The equivocality enables one to generate a commitment that can be opened to any message by using a master secret key. The homomorphism for a circuit family $\mathcal{C} = \{C : \mathcal{X} \to \mathcal{Z}\}$ informally requires that one can commit to a message $\mathbf{x} \in \mathcal{X}$, where its commitment com can be further publicly modified to a commitment com_C on the message $C(\mathbf{x}) \in \mathcal{Z}$ for any circuit $C \in \mathcal{C}$. Here, a decommitment for com_C can be computed by the knowledge of the message $\mathbf{x}$, decommitment of com, and the circuit C. To the knowledgeable readers, we note that HEC is a strictly weaker primitive compared to homomorphic trapdoor functions [GVW15]. Previously, an HEC supporting the family of all polynomial-sized circuits were only (implicitly) known from lattice-based assumptions [GVW15]. Apart from their construction, known (implicit) constructions of HEC only support linear functions [Ped92] or group operations on a pairing group [AFG+16]. In this paper, we provide the first instantiation of HEC supporting $\mathbf{NC}^1$ based on any pairing-based assumptions, namely, the (n, m)-CDHER assumption introduced in [KNYY19]. The construction is inspired by the recent construction of compact homomorphic signatures of Katsumata et al. [KNYY19]. The proposed HEC enjoys a particular form of compactness which is especially useful for generically converting CRS-NIZKs with non-compact proofs to CRS-NIZKs with compact proofs. Concretely, for any polynomially-sized circuit C, the evaluated commitment com_C and its decommitment of our HEC are of size $\mathsf{poly}(\kappa)$ independent of $|C|$, and one can verify the validity of the decommitment in time $\mathsf{poly}(\kappa)$ independent of $|C|$. Somewhat surprisingly, we also construct another instantiation of HEC supporting $\mathbf{NC}^1$ based on the CDH assumption over pairing groups. Although this HEC does not enjoy compactness, and hence cannot be used for our compact CRS-NIZK conversion, we believe it to be an interesting primitive on its own since we achieve homomorphic computations in $\mathbf{NC}^1$ from such a weak assumption as CDH.

1.3 Technical Overview

Our results can be broken up into three parts. The first two results concerning CRS and UC-NIZKs with short proof are obtained through a generic conversion from NIZKs with non-compact proofs to NIZKs with compact proofs using homomorphic equivocal commitments (HEC); a primitive which we formalize and provide instantiations in this work.

[2] This primitive was already informally mentioned in [GVW15] and we do not take credit for proposing the concept of HEC. We note that Abe et al. [AFG+16] also introduced a similar primitive with the name *homomorphic trapdoor commitments*.

Table 1: Comparison of CRS-NIZKs for **NP**.

Reference	Soundness	ZK	CRS size	Proof size	Assumption	Misc
[FLS99]	stat.	comp.	$\mathsf{poly}(\kappa, \lvert C\rvert)$	$\mathsf{poly}(\kappa, \lvert C\rvert)$	trapdoor permutation†	
[Gro10a]	stat.	comp.	$\lvert C\rvert \cdot k_{\mathsf{tpm}} \cdot \mathsf{polylog}(\kappa) + \mathsf{poly}(\kappa)$	$\lvert C\rvert \cdot k_{\mathsf{tpm}} \cdot \mathsf{polylog}(\kappa) + \mathsf{poly}(\kappa)$	trapdoor permutation†	
[Gro10a]	stat.	comp.	$\lvert C\rvert \cdot \mathsf{polylog}(\kappa) + \mathsf{poly}(\kappa)$	$\lvert C\rvert \cdot \mathsf{polylog}(\kappa) + \mathsf{poly}(\kappa)$	Naccache-Stern PKE	
[GOS12]	perf.	comp.	$\mathsf{poly}(\kappa)$	$O(\lvert C\rvert\kappa)$	DLIN/SD	
[GOS12]	comp.	perf.	$\mathsf{poly}(\kappa)$	$O(\lvert C\rvert\kappa)$	DLIN/SD	
[CHK07, Abu13]	stat.	comp.	$\mathsf{poly}(\kappa, \lvert C\rvert)$	$\mathsf{poly}(\kappa, \lvert C\rvert)$	CDH	pairing group
[GGI+15]	stat.	comp.	$\mathsf{poly}(\kappa)$	$\lvert w\rvert + \mathsf{poly}(\kappa)$	FHE and CRS-NIZK	circular security
Section 4	comp.	comp.	$\mathsf{poly}(\kappa, \lvert C\rvert)$	$\lvert C\rvert + \mathsf{poly}(\kappa)$	(n, m)-CDHER	
Section 4	comp.	comp.	$\mathsf{poly}(\kappa, \lvert C\rvert)$	$\lvert w\rvert + \mathsf{poly}(\kappa)$	(n, m)-CDHER	limited to $\mathbf{NC}^1$ relation
Section 7	stat./comp.	comp.	$\mathsf{poly}(\kappa, \lvert x\rvert, \lvert w\rvert, d)$	$\mathsf{poly}(\kappa, \lvert x\rvert, \lvert w\rvert, d)$	LFE and CRS-NIZK	prover-efficient, implied by sub-exp. LWE
Section 7	stat./comp.	comp.	$(\lvert x\rvert + \lvert w\rvert) \cdot \mathsf{poly}(\kappa, d)$	$\tilde{O}(\lvert x\rvert + \lvert w\rvert) \cdot \mathsf{poly}(\kappa, d)$	LFE and CRS-NIZK‡	prover-efficient, implied by adaptive LWE

In column "Soundness" (resp. "ZK"), perf., stat., and comp. means perfect, statistical, and computational soundness (resp. zero-knowledge), respectively. In column "CRS size" and "Proof size", κ is the security parameter, $|x|$, $|w|$ is the statement and witness size, $|C|$ and d are the size and depth of the circuit computing the **NP** relation, and k_{tpm} is the length of the domain of the trapdoor permutation. In column "Assumption", (n, m)-CDHER stands for the (parameterized) computational DH exponent and ratio assumption, LFE stands for laconic functional evaluation, and sub-exp. LWE stands for sub-exponentially secure learning with errors (LWE).

† If the domain of the permutation is not $\{0, 1\}^n$, we further assume they are doubly enhanced [Gol04].

‡ We additionally require a mild assumption that the prover run time is linear in the size of the circuit computing the **NP** relation.

Table 2: Comparison of UC-NIZKs for **NP**.

Reference	Security (erasure-free)	CRS size	Proof size	Assumption	Misc
[GOS12]	adaptive (✓)	$\mathsf{poly}(\kappa)$	$O(\lvert C\rvert\kappa)$	DLIN/SD	
[GGI+15]	adaptive (✗)	$\mathsf{poly}(\kappa)$	$\lvert w\rvert + \mathsf{poly}(\kappa)$	FHE and UC-NIZK	circular security
[CsW19]	adaptive (✓)	$\mathsf{poly}(\kappa, d)$	$\lvert w\rvert \cdot \mathsf{poly}(\kappa, d)$	HTDF and UC-NIZK	
Section 5	adaptive (✓)	$\mathsf{poly}(\kappa, \lvert C\rvert)$	$\lvert w\rvert \cdot \mathsf{poly}(\kappa)$	(n, m)-CDHER	limited to $\mathbf{NC}^1$ relation

In column "CRS size" and "Proof size", κ is the security parameter, $|w|$ is the witness size, $|C|$ and d are the size and depth of circuit computing the **NP** relation. In column "Assumption", DLIN stands for the decisional linear assumption, SD stands for the subgroup decision assumption, HTDF stands for homomorphic trapdoor functions, and (n, m)-CDHER stands for the (parameterized) computational DH exponent and ratio assumption.

Table 3: Comparison of DV-NIZKs for **NP**.

Reference	Soundness	ZK	CRS size	Proof size	Verification key size	Assumption	Misc
[CH19, KNYY19, QRW19]	stat.	comp.	$\mathsf{poly}(\kappa, \lvert C\rvert)$	$\mathsf{poly}(\kappa, \lvert C\rvert)$	$\mathsf{poly}(\kappa, \lvert C\rvert)$	CDH	pairing-free group
Section 6	stat.	comp.	$\mathsf{poly}(\kappa)$	$\lvert C\rvert + \mathsf{poly}(\kappa)$	$\mathsf{poly}(\kappa)$	CDH	pairing-free group
Section 6	comp.	comp.	$2^d \cdot \mathsf{poly}(\kappa)$	$\lvert w\rvert + \mathsf{poly}(\kappa)$	$\mathsf{poly}(\kappa)$	ℓ-CDHI	pairing-free group, limited to $\mathbf{NC}^1$ relation

In column "Soundness" (resp. "ZK"), stat. and comp. means statistical and computational soundness (resp. zero-knowledge), respectively. In the columns concerning sizes, κ is the security parameter, $|w|$ is the witness-size, $|C|$ and d are the size and depth of the circuit computing the **NP** relation. In column "Assumption", ℓ-CDHI stands for the ℓ-computational Diffie-Hellman inversion assumption.

The third result concerning DV-NIZKs with short proof size based on pairing-free groups, that is, CDH and ℓ-CDHI, are obtained by extending the recent result of Katsumata et al. [KNYY19] which constructs the first NIZKs in the preprocessing model (PP-NIZKs) with short proof size from pairing-free groups. As explained later, PP-NIZK is a strictly weaker primitive compared to DV-NIZK. Finally, the fourth result concerning prover-efficient NIZK is obtained by a generic construction based on the recently developed laconic function evaluation scheme of Quach et al. [QWW18]. In the following, we explain these approaches in more detail.

1.3.1 Generic Construction of Compact (CRS, UC)-NIZK from HEC

Here, we explain our construction of compact CRS-NIZK. Our starting point is the recent result by Katsumata et al. [KNYY19], who constructed a *designated prover* NIZK (DP-NIZK) with compact proof, where DP-NIZK is an analogue of DV-NIZK where the prover requires secret information to generate proofs and anybody can publicly verify the proofs. Since the construction of Katsumata et al. is an instantiation of the generic conversion from homomorphic signature to DP-NIZK proposed by Kim and Wu [KW18a], we first briefly review Kim and Wu's conversion. Recall that in homomorphic signature, a signature $\boldsymbol{\sigma}$ on a message $\mathbf{m} \in \{0,1\}^\ell$ generated by a secret key sk, can be homomorphically evaluated to a signature σ on $C(\mathbf{m})$ for a circuit $C : \{0,1\}^\ell \rightarrow \{0,1\}$. Anybody can verify the validity of the signature by using a public verification key vk and the circuit C. As for the security requirements, we need that given a verification key vk and a signature $\boldsymbol{\sigma}$ on $\mathbf{m}$, it is computationally hard to forge a signature σ^* on z such that $z \neq C(\mathbf{m})$ (unforgeability) and an honestly evaluated signature σ on z does not reveal information about $\mathbf{m}$ beyond the fact that it was derived from a signature on $\mathbf{m}$ such that $C(\mathbf{m}) = z$ (context-hiding). Furthermore, as an efficiency requirement, we need that the size of σ is independent of the size of the circuit C. In Kim and Wu's construction of DP-NIZK, the prover is given a signature $\boldsymbol{\sigma}$ on a secret key $\mathbf{k}$ of a secret key encryption (SKE) scheme as the secret proving key. When the designated prover proves that x is in some language $\mathcal{L}$ that is specified by a relation $\mathcal{R}$, it generates an encryption ct of the witness w such that $(x, w) \in \mathcal{R}$ and homomorphically evaluates the signature $\boldsymbol{\sigma}$ with respect to a circuit that computes $f_{x,\mathsf{ct}}$, where $f_{x,\mathsf{ct}}$ is a function that takes as input $\mathbf{k}'$ and outputs whether $(x, \mathsf{SKE.Dec}(\mathbf{k}', \mathsf{ct})) \in \mathcal{R}$. The proof for DP-NIZK is then set as ct and the homomorphically evaluated signature σ. The verifier prepares the function $f_{x,\mathsf{ct}}$ from ct and x, and simply checks σ is a correct signature on 1 with respect to the evaluated function $f_{x,\mathsf{ct}}$. The soundness of the protocol follows from the unforgeability of the homomorphic signature since $f_{x,\mathsf{ct}}(\mathbf{k}') = 0$ for any $\mathbf{k}'$ when $\mathbf{x}$ is not in the language induced by the relation $\mathcal{R}$. Furthermore, the zero-knowledge property of the protocol follows from the security of SKE and the context-hiding property of the homomorphic signature. Katsumata et al. [KNYY19] gave a new homomorphic signature scheme with short evaluated signature σ that supports the function class of $\mathbf{NC}^1$ circuits based on a newly introduced (non-static) pairing-based assumption called the (n, m)-computational Diffie-Hellman exponent and ratio (CDHER) assumption. Plugging this homomorphic signature into the Kim-Wu conversion, they obtained the first compact DP-NIZK for all **NP** based on any pairing-based assumptions.[3]

The aim of our work is to modify the Kim-Wu conversion and remove the necessity of the prover keeping secret information to generate a proof so that we can convert the compact DP-NIZK of Katsumata et al. into a compact CRS-NIZK. The main reason why their construction cannot be used as a CRS-NIZK is because the prover cannot generate the signature $\boldsymbol{\sigma}$ on the fly without knowing the signing key sk of the homomorphic signature. To this end, our first idea is to let the prover choose vk, sk, and $\mathbf{k}$ on its own. This would allow the prover to generate a proof as in the designated prover setting since it can generate the signature $\boldsymbol{\sigma}$ on $\mathbf{k}$ on its own by using the signing key sk. The proof for the CRS-NIZK will then consist of the verification key vk and a proof of the DP-NIZK. Unfortunately, there are multiple of problems with this naive approach. The first problem is that the size of the verification key vk used in Katsumata et al. [KNYY19] is polynomially dependent on the size of the circuit that computes the relation to be proven, and thus, this ruins the compactness property of the original DP-NIZK proof. The second problem is that we can no longer invoke the unforgeability of the homomorphic signature to prove soundness since unforgeability holds against adversaries who only has access to a verification key vk and a signature $\boldsymbol{\sigma}$. Indeed, in the specific case of Katsumata et al.'s homomorphic signature scheme, an adversary will be able to completely break the soundness of the resulting scheme if it is further given the signing key sk. Therefore, to resolve these problems, we make use of the special structure that the homomorphic signature scheme of Katsumata et al. has and abstract it to a primitive which we call homomorphic equivocal commitments (HEC).

Our key observation is that in the Katsumata et al.'s homomorphic signature scheme, the reverse direction of the signing procedure is possible *without* the knowledge of the secret signing key sk if we are allowed to program part of the verification key vk. Namely, the verification key vk can be divided into two parts vk_0 and vk_1 where the size of vk_1 is compact (i.e., independent of the size of the circuit), and for a fixed vk_0 and $\mathbf{k}$, one can sample a signature $\boldsymbol{\sigma}$ and efficiently compute the remaining part of the verification key vk_1 without knowledge of the secret signing key sk so that

[3] Note that any **NP** relation can be converted to an **NP** relation in $\mathbf{NC}^1$ by expanding the witness size as large as the circuit computing the original **NP** relation. Notably, a homomorphic signature scheme supporting the function class of $\mathbf{NC}^1$ circuits is sufficient for constructing DP-NIZK for all of **NP**.

$\boldsymbol{\sigma}$ is a valid signature on $\mathbf{k}$ with respect to the entire verification key $\mathsf{vk} = (\mathsf{vk}_0, \mathsf{vk}_1)$. We then modify our above idea using this reverse direction of computation. Namely, we put the non-compact part of the verification key vk_0 in the common reference string. The prover first choose $\mathbf{k}$, $\boldsymbol{\sigma}$ on its own and then computes the remaining compact part of the verification key vk_1 from them so that $\boldsymbol{\sigma}$ is a valid signature on $\mathbf{k}$ with respect to the verification key vk. Notably, the prover no longer requires knowledge of the secret signing key sk, and thus, the prover can generate a proof publicly. The resulting proof is the same as in the case for the above naive construction except that we now only append vk_1 to the underlying DP-NIZK proof, rather than vk_0 *and* vk_1. The first problem of having a large proof size we encountered in our above attempt is now resolved since we moved the non-compact part of the verification key vk_0 to the common reference string and the proof now only contains the compact vk_1 and the compact proof of the underlying DP-NIZK. At first glance, the second problem of losing soundness seems to be resolved as well, as the prover is choosing the signature $\boldsymbol{\sigma}$ without knowledge of the underlying secret signing key sk. However, we encounter a new problem. Namely, once again, we cannot directly use the unforgeability of the homomorphic signature to prove soundness, since this time the part of the verification key vk_1 that the adversary appends to the underlying DP-NIZK proof may be maliciously chosen in a way that deviates from the security setting of the homomorphic signature. However, luckily, the proof for unforgeability provided by Katsumata et al. can be adapted without much change to the setting where vk_1 follows an arbitrary distribution since their proof does not depend on the specific distribution which vk_1 is chosen from. In this work, to capture this special security requirement as well as the syntactic structure that we require for the homomorphic signature, we introduce a new primitive that we call *homomorphic equivocal commitment* (HEC) and instantiate it by mimicking the homomorphic signature scheme of Katsumata et al. [KNYY19]. Roughly speaking, in our formulation, we regard vk_1 as a commitment of a message $\mathbf{k}$ with respect to a randomness $\boldsymbol{\sigma}$.

While the above explanation conveys our main idea, we need some more modification to obtain our final construction. In the above construction, an honest prover outputs a "commitment" vk_1 of a secret key $\mathbf{k}$. However, a malicious prover may choose the commitment that does not correspond to any secret key. In this case, we can no longer argue soundness. To avoid the problem, we rely on a non-compact NIZK to prove the well-formedness of the commitment. Since the size of the circuit for checking the well-formedness is independent of the size of the circuit for computing the relation to be proven, this does not harm the compactness of the proof. We finally remark that the construction we explained so far is still slightly different from the one we give in Section 4.2. There, we change the scheme so that the prover provides the proof of knowledge of σ instead of sending σ as part of the proof in the clear. While our scheme is secure without this change, this makes it easier to extend our construction to the UC-secure setting in Section 5.

The proof size of the resulting CRS-NIZK is $|C| + \mathsf{poly}(\kappa)$ since our HEC only supports $\mathbf{NC}^1$ and thus we have to expand the witness to the concatenation of all values corresponding to each wire of the circuit verifying the relation to make the verification of the relation be done in $\mathbf{NC}^1$. On the other hand, if the relation can be verified in $\mathbf{NC}^1$ from the beginning, then the expansion is not needed and the proof size is as small as $|w| + \mathsf{poly}(\kappa)$.

Interestingly, our CRS-NIZK can also be seen as a variant of the UC-NIZK recently proposed by Cohen, shelat, and Wichs [CsW19]. The differences from their scheme are (1) an HTDF is replaced with an HEC, (2) a witness is encrypted by SKE of which key is committed by a HEC instead of the witness itself, and (3) one-time signatures are omitted. If we are to construct a UC-NIZK in the adaptive non-erasure setting as is done in [CsW19], the modifications (2) and (3) are no longer applicable, but (1) is still applicable. Based on this observation, we obtain a UC-NIZK for $\mathbf{NC}^1$ in the adaptive non-erasure setting with a similar proof size to that of [CsW19] based on a HEC instead of a HTDF. A caveat of our construction is that the scheme only supports NP languages verifiable in $\mathbf{NC}^1$ whereas their scheme supports all of NP (verifiable by a polynomial-size circuit). On the other hand, our abstraction as HEC instead of HTDF enables us to instantiate the scheme based on a pairing assumption instead of lattices. In particular, it seems difficult to construct HTDF based on a pairing assumption.

1.3.2 Compact DV-NIZKs based on Pairing-Free Groups

Here, we explain our constructions of compact DV-NIZKs. Actually, we give a generic compiler to convert any non-compact DV-NIZK to a compact one additionally assuming the existence of PKE and $\mathbf{NC}^1$-decryptable SKE with additive ciphertext overhead. In this overview, we discuss a specific instantiation based on the CDH assumption in pairing-free groups.

The starting point of our constructions is the recent construction of compact NIZKs in the preprocessing model (PP-

NIZKs) by Katsumata et al. [KNYY19] based on inner-product functional encryptions (IPFE) [ABDP15].[4] PP-NIZK is a relaxation of (CRS, DV, DP)-NIZK where both the prover and the verifier are given proving and verification keys, respectively, which should be hidden from each other. Katsumata et al. first constructed a context-hiding homomorphic MAC for arithmetic circuits by adding the context-hiding property to the non-context-hiding homomorphic MAC of Catalano and Fiore [CF18] by using an IPFE. They then plugged the context-hiding homomorphic MAC into the generic conversion by Kim and Wu [KW18a] to obtain PP-NIZKs.[5] Recall that in the PP-NIZK construction of Kim and Wu, a prover key consists of an SKE key $\mathbf{k}$ and a signature $\boldsymbol{\sigma}$ on $\mathbf{k}$, and a verification key consists of a verification key vk of a homomorphic MAC scheme. The reason why their scheme is PP-NIZK and not DV-NIZK is that a prover has to obtain a signature $\boldsymbol{\sigma}$ on $\mathbf{k}$ which should be generated by a trusted third party who has the corresponding signing key sk.[6] Similarly to the case of our CRS-NIZK explained in the previous section, we observe the following fact. If one can choose $\boldsymbol{\sigma}$ and vk in the reverse order, that is, if one can first choose the signature $\boldsymbol{\sigma}$, and then define vk so that $\boldsymbol{\sigma}$ is a valid signature on $\mathbf{k}$, then we could modify the scheme to be a DV-NIZK by letting the prover choose $\mathbf{k}$ and $\boldsymbol{\sigma}$ on its own. Below, we observe that the homomorphic MAC of Katsumata et al. [KNYY19] indeed has this property. To explain this, we first recall the structure of their homomorphic MAC.

In their homomorphic MAC scheme, a verification key vk (which is also a signing key) consists of $s \xleftarrow{\$} \mathbb{Z}_p^*$, $\mathbf{r} \xleftarrow{\$} \mathbb{Z}_p^\ell$ and a decryption key of an IPFE corresponding to the vector $(s, ..., s^D) \in \mathbb{Z}_p^D$ where p is a sufficiently large prime, ℓ is the message length, and D is the degree of the arithmetic circuits supported by the homomorphic MAC scheme.[7] A signature on $\mathbf{k}$ is defined to be $\boldsymbol{\sigma} := (\mathbf{r} - \mathbf{k}) \cdot s^{-1} \mod p$. From the form of $\boldsymbol{\sigma}$, we can see that for any fixed $\mathbf{k}$ and s, one can set $\boldsymbol{\sigma}$ and $\mathbf{r}$ in the reverse order, that is, one can first pick $\boldsymbol{\sigma}$ and then set $\mathbf{r} := \mathbf{k} + \boldsymbol{\sigma} \cdot s \mod p$.

Going back to the construction of NIZK, this structure enables us to get close to DV-NIZK. Namely, a prover can now choose $\mathbf{k}$ and $\boldsymbol{\sigma}$ by itself, and it no longer needs any proving key generated by a trusted third party. However, there is an important problem still remaining on how the verifier gets to know $\mathbf{r} = \mathbf{k} + \boldsymbol{\sigma} \cdot s \mod p$, which is required for verification. Recall that $\mathbf{r}$ was part of the private verification key of the PP-NIZK of Kim and Wu. If s is given to a prover, then we cannot rely on unforgeability of the homomorphic MAC to prove soundness, and if the prover sends $\mathbf{k}$ and $\boldsymbol{\sigma}$ in the clear, then we cannot rely on the security of SKE to prove zero-knowledge. Therefore the prover has to transmit $\mathbf{r} = \mathbf{k} + \boldsymbol{\sigma} \cdot s \mod p$ to the verifier without knowing s nor revealing $\mathbf{k}$ and $\boldsymbol{\sigma}$ to the verifier. We observe that this task can be done by using IPFE. Namely, we give a secret key corresponding to the vector $(1, s)$ of IPFE to the verifier as a part of his verification key, and a prover encrypts vectors (k_i, σ_i) for each $i \in [\ell]$ where k_i and σ_i are the i-th entry of $\mathbf{k}$ and $\boldsymbol{\sigma}$, respectively, and sends the ciphertexts as a part of the proof. Then a verifier can obtain $\mathbf{r} = \mathbf{k} + \boldsymbol{\sigma} \cdot s \mod p$ by simply decrypting the IPFE ciphertexts with his decryption key.

Though the above idea seems to work at first glance, there is a problem that was also addressed in [KNYY19]. Namely, since a standard security notion of IPFE does not consider a malicious encryptor, an adversary may generate a malformed ciphertext whose decryption result is perfectly under his control, which breaks soundness. To prevent such an attack, Katsumata et al. [KNYY19] required a property called an *extractability* for an IPFE, which means that one can extract a corresponding message from any possibly malformed ciphertext if it does not decrypt to $\perp$. They then showed that the DDH-based IPFE scheme of Agrawal, Libert, and Stehlé [ALS16] can be used as an extractable IPFE. However, unfortunately, we will not be able to simply plug in the extractable IPFE of Agrawal et al. into our DV-NIZK. This is because the IPFE of Agrawal et al. embeds the message into the exponent of a group element, and forces one to compute the discrete logarithm to decrypt. Therefore, unless we can be sure that the exponent will be small, the IPFE of Agrawal et al. is difficult to use. Here, the reason why the PP-NIZK of Katsumata et al. [KNYY19] did not face any issue with this somewhat awkward decryption algorithm was because the verification algorithm only consisted of checking whether the decryption result is equal to a certain value, which could be tested in the exponent, using the verification key $(s, \mathbf{r})$. However, in our case, the verifier must first decrypt $\mathbf{r}$ using the IPFE secret key corresponding to the vector $(1, s)$ to recover $\mathbf{r}$, and only then it can run the internal verification algorithm of [KNYY19]

[4]Actually, their construction is based on a variant of IPFE called IPFE on exponent (expIPFE). We note that their construction works with standard IPFE. They used the notion of expIPFE instead of IPFE for making it possible to instantiate the scheme based on the DDH-based scheme by Agrawal, Libert, and Stehlé [ALS16].

[5]Kim and Wu [KW18a] showed that if one uses their generic conversion on homomorphic MACs instead of homomorphic signatures, it would result in PP-NIZKs instead of DP-NIZKs.

[6]In a homomorphic MAC, we can let $\mathsf{sk} := \mathsf{vk}$ since both are kept private.

[7]We remark that we cannot include the master secret key of IPFE in vk since the context-hiding property should hold even against the verifier who sees vk.

using the pair $(s, \mathbf{r})$. Notably, the verifier would have to solve the discrete logarithm for a random value in $\mathbb{Z}_p$ to recover the piece $\mathbf{r}$ of the verification key used in the PP-NIZK of Katsumata et al. However, obviously, there is no way to compute this efficiently. Therefore, in this work, we must take a different approach. Concretely, instead of relying on the extractability of IPFE, we require a prover to provide a proof that he has honestly generated ciphertexts by using another (non-compact) DV-NIZK. Here, since the validity check of IPFE ciphertexts can be done with computational complexity independent of the size of the language the prover really wants to prove, we can use a non-compact DV-NIZK for this part while keeping the whole proof size compact. In summary, we can convert the PP-NIZK of [KNYY19] to a DV-NIZK by adding ℓ IPFE ciphertexts along with their validity proof whose sizes are $\mathsf{poly}(\kappa)$. Since the proof size of the PP-NIZK of [KNYY19] is $|C| + \mathsf{poly}(\kappa)$, the proof size of the resulting DV-NIZK is also $|C| + \mathsf{poly}(\kappa)$. Moreover, we note that single-key secure IPFE suffices for the above construction of DV-NIZK. Since single-key secure functional encryption for all polynomial-sized functions exist under the existence of PKE [GVW12] and DV-NIZK for all of **NP** exists under the CDH assumption on a pairing-free group [CH19, KNYY19, QRW19], we can instantiate the above DV-NIZK based on the CDH assumption on a pairing-free group.[8] Finally, we note that by using the idea of the compact homomorphic MAC based on the ℓ-CDHI assumption by Catalano and Fiore [CF18], we can further reduce the proof size to be $|w| + \mathsf{poly}(\kappa)$ in the case when the language to be proven is computable in $\mathbf{NC}^1$.

1.3.3 Generic Construction of Prover-Efficient NIZK from LFE

To achieve prover-efficient NIZKs, we use laconic function evaluation (LFE) recently introduced by Quach, Wee, and Wichs [QWW18]. LFE schemes are defined for a class of circuits $\mathcal{C}$. We can generate a short digest of circuit $C \in \mathcal{C}$ from a CRS and the circuit C. Anybody can then generate a ciphertext ct of a message m from the CRS, the digest, and m. Finally, anybody can decrypt the ciphertext to $C(m)$ using the ciphertext ct and the circuit C. Here, the security of LFE imposes that the ciphertext ct leaks no additional information other than the value $C(m)$. The attractive feature of LFE is that the size of the CRS, digest, ciphertext ct, and the running time of the encryption algorithm are all strictly smaller than the size of the circuits in $\mathcal{C}$.

Our design idea is to impose the computation of the circuit C computing the NP-relation on the verifier by using LFE. Specifically, we put a digest of C (and a CRS of LFE) in the CRS of our NIZK. The prover then computes an LFE ciphertext of message (x, w) where x is a statement and w is its witness using the digest of C. A verifier can check the validity of the statement by decrypting the ciphertext with C. By the security of LFE, the verifier obtains nothing beyond $C(x, w)$, hence, zero-knowledge of our NIZK follows naturally. Furthermore, by the efficiency property of LFE, the running time of the prover is smaller than the size of C. However, this basic idea is not yet sufficient. This is because a cheating prover may not honestly compute an LFE ciphertext of the message (x, w) and may possibly break soundness of our NIZK. To overcome this issue, a prover must generate not only an LFE ciphertext of (x, w) but also a NIZK proof to prove that the prover honestly generated the LFE ciphertext of (x, w) with the CRS of LFE and the digest of C. We point out that this additional NIZK proof does not harm prover efficiency since the additional statement which the prover must prove is independent of the size of the circuit C owing to the feature of LFE. In particular, we can check the validity of the ciphertext by computing the encryption circuit of LFE whose size is independent of the size of C.

Using any non-prover-efficient NIZK for **NP** as building block and instantiating the LFE scheme by the sub-exponential security of LWE assumption with sub-exponential modulus-to-noise ratio, we obtain a prover-efficient CRS-NIZK for **NP** whose prover running time is $\mathsf{poly}(\kappa, |x|, |w|, d)$, where d is the depth of the circuit C computing the **NP** relation. In particular, the prover running time is independent of $|C|$. In fact, we can further reduce the prover running time to be as small as $\widetilde{O}(|x| + |w|) \cdot \mathsf{poly}(\kappa, d)$ where the dependence of the statement x and witness w size is only quasi-linear if we further use the following two assumptions (1) the prover running time of the underlying NIZK is linear in the size of the circuit that computes the **NP** relation, that is, $|C| \cdot \mathsf{poly}(\kappa)$ (2) a natural variant of the above LWE assumption introduced by Quach et al. [QWW18], called the *adaptive* LWE assumption. Note that the assumption we make on the underlying NIZK is not that strong, and in particular, we can use the NIZK of Groth, Ostrovsky, and Sahai [GOS12].

[8]One may wonder why we only need CDH though [KNYY19] assumed DDH. Recall that the DDH in their construction comes from the necessity of an extractable expIPFE. We show that this can be replaced with any IPFE and DV-NIZK both of which exist under the CDH assumption based on the same idea as explained above.

1.4 Related Works

Other than CRS and DV-NIZKs, which have been the main interest of this paper, there are other variants of NIZKs. One is PP-NIZK and the other is DP-NIZK as we briefly mentioned in Section 1.3. Similarly to DV-NIZKs, due to the extra secret information shared by the prover and/or verifier, the soundness (resp. zero-knowledge) property of (PP, DP)-NIZKs may be compromised after verifying (resp. proving) multiple statements. In fact most of the PP or DP-NIZKs [DMP90, KMO90, LS91, Dam93, CD04, IKOS09] are known only to be secure for bounded statements. The first multi-theorem PP and DP-NIZKs (that are not a trivial downgrade of CRS-NIZKs) where given by Kim and Wu [KW18a] who proposed a generic construction of them via homomorphic MACs and homomorphic signatures, respectively. Since homomorphic signatures were implied by lattice-based assumptions [GVW15], this implied the first DP-NIZKs based on lattices. Subsequently, Katsumata et al. [KNYY19] constructed a homomorphic signature based on the CDHER assumption and a homomorphic MAC based on the DDH assumption over pairing-free groups, and thus constructed DP and PP-NIZKs relative to those assumptions. One attractive feature of the NIZKs of Kim and Wu [KW18a] and Katsumata et al. [KNYY19] is that the proof size are compact: the DP-NIZK of [KW18a] has proof size $|w| + \mathsf{poly}(\kappa, d)$ and the (PP, DP)-NIZK of [KNYY19] have proof size $|C| + \mathsf{poly}(\kappa)$, where d is the depth of the circuit C computing the **NP** relation.

2 Preliminaries

Notations. For a distribution or random variable X, we write $x \overset{\$}{\leftarrow} X$ to denote the operation of sampling a random x according to X. For a set S, we write $s \overset{\$}{\leftarrow} S$ to denote the operation of sampling a random s from the uniform distribution over S. For random variables X and Y, $X \overset{\text{stat}}{\approx} Y$ means that X and Y are statistically indistinguishable. For a probabilistice algorithm $\mathcal{A}$, $y \overset{\$}{\leftarrow} \mathcal{A}(x)$ means that $\mathcal{A}$ is run on input x and outputs y, and $y := \mathcal{A}(x; r)$ means that $\mathcal{A}$ is run on input x with input r and outputs y. For an algorithm A that takes as input x and randomness r, "$y \in \mathsf{A}(x)$" means $\Pr_r[y' = y : y' \leftarrow \mathsf{A}(x; r)] > 0$.

2.1 Symmetric Key Encryption

Let $\{\mathcal{M}_\kappa\}_{\kappa \in \mathbb{N}}$ be a family of message space. In the following, we occasionally drop the subscript and simply write $\mathcal{M}$ when the meaning is clear.

Definition 2.1 (Symmetric Key Encryption). *A symmetric key encryption (SKE) scheme* Π_{SKE} *for message space* $\mathcal{M}$ *consists of PPT algorithms* $(\mathsf{SKE.KeyGen}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$.

$\mathsf{SKE.KeyGen}(1^\kappa) \rightarrow \mathsf{K}_{\mathsf{SKE}}$*: The key generation algorithm takes as input the security parameter* 1^κ *and outputs a secret key* $\mathsf{K}_{\mathsf{SKE}}$.

$\mathsf{SKE.Enc}(\mathsf{K}_{\mathsf{SKE}}, \mathsf{M}) \rightarrow \mathsf{ct}$*: The encryption algorithm takes as input a secret key* $\mathsf{K}_{\mathsf{SKE}}$ *and a message* $\mathsf{M} \in \mathcal{M}$ *and outputs a ciphertext* ct.

$\mathsf{SKE.Dec}(\mathsf{K}_{\mathsf{SKE}}, \mathsf{ct}) \rightarrow \mathsf{M}$ *or* $\bot$*: The decryption algorithm takes as input a secret key* $\mathsf{K}_{\mathsf{SKE}}$ *and a ciphertext* ct *and outputs a message* $\mathsf{M} \in \mathcal{M}$ *or a special symbol* $\bot$ *indicating decryption failure.*

Correctness. *For all* $\kappa \in \mathbb{N}$, $\mathsf{M} \in \mathcal{M}$, *and* $\mathsf{K}_{\mathsf{SKE}} \in \mathsf{SKE.KeyGen}(1^\kappa)$, *we have* $\mathsf{SKE.Dec}(\mathsf{K}_{\mathsf{SKE}}, \mathsf{SKE.Enc}(\mathsf{K}_{\mathsf{SKE}}, \mathsf{M})) = \mathsf{M}$.

CPA-Security. *For all* $\kappa \in \mathbb{N}$ *and all PPT adversaries* $\mathcal{A}$, *if we run* $\mathsf{K}_{\mathsf{SKE}} \overset{\$}{\leftarrow} \mathsf{SKE.KeyGen}(1^\kappa)$, *then we have*

$$\left| \Pr[\mathcal{A}^{\mathcal{O}_0(\mathsf{K}_{\mathsf{SKE}}, \cdot, \cdot)}(1^\kappa) = 1] - \Pr[\mathcal{A}^{\mathcal{O}_1(\mathsf{K}_{\mathsf{SKE}}, \cdot, \cdot)}(1^\kappa) = 1] \right| = \mathsf{negl}(\kappa),$$

where $\mathcal{O}_b(\mathsf{K}_{\mathsf{SKE}}, \mathsf{M}_0, \mathsf{M}_1)$ *outputs* $\mathsf{SKE.Enc}(\mathsf{K}_{\mathsf{SKE}}, \mathsf{M}_b)$ *for* $b \in \{0, 1\}$.

For our construction of NIZKs in the following sections, we require an SKE scheme whose ciphertext overhead (i.e., $|\mathsf{ct}| - |m|$) is $\mathsf{poly}(\kappa)$ and whose decryption algorithm can be represented as a circuit in $\mathbf{NC}^1$. As noted in [KNYY19], such an SKE exists under the CDH assumption over pairing-free groups.

2.2 Public Key Encryption

Let $\{\mathcal{M}_\kappa\}_{\kappa\in\mathbb{N}}$ be a family of message space. In the following, we occasionally drop the subscript and simply write $\mathcal{M}$ when the meaning is clear.

Definition 2.2 (Public Key Encryption). *A public key encryption (PKE) scheme* Π_{PKE} *for message space* $\mathcal{M}$ *consists of PPT algorithms* $(\mathsf{PKE.KeyGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$.

$\mathsf{PKE.KeyGen}(1^\kappa) \to (\mathsf{pk}, \mathsf{sk})$*: The key generation algorithm takes as input the security parameter* 1^κ *and outputs public key* pk *and a secret key* sk.

$\mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{M}) \to \mathsf{ct}$*: The encryption algorithm takes as input a secret key* $\mathsf{K}_{\mathsf{SKE}}$ *and a message* $\mathsf{M} \in \mathcal{M}$ *and outputs a ciphertext* ct.

$\mathsf{PKE.Dec}(\mathsf{sk}, \mathsf{ct}) \to \mathsf{M}$ *or* $\perp$*: The decryption algorithm takes as input a secret key* sk *and a ciphertext* ct *and outputs a message* $\mathsf{M} \in \mathcal{M}$ *or a special symbol* $\perp$ *indicating decryption failure.*

Correctness. *For all* $\kappa \in \mathbb{N}$, $\mathsf{M} \in \mathcal{M}$, *and* $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{PKE.KeyGen}(1^\kappa)$, *we have* $\mathsf{PKE.Dec}(\mathsf{sk}, \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{M})) = \mathsf{M}$.

CPA-Security. *For all* $\kappa \in \mathbb{N}$ *and all PPT adversaries* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, *if we run* $(\mathsf{pk}, \mathsf{sk}) \overset{\$}{\leftarrow} \mathsf{PKE.KeyGen}(1^\kappa)$, $(\mathsf{M}_0, \mathsf{M}_1, \mathsf{st}) \overset{\$}{\leftarrow} \mathcal{A}_0(\mathsf{pk})$, $\mathsf{coin} \overset{\$}{\leftarrow} \{0,1\}$, $\mathsf{ct}^* \overset{\$}{\leftarrow} \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{M}_{\mathsf{coin}})$, *and* $\mathsf{coin}' \overset{\$}{\leftarrow} \mathcal{A}_1(\mathsf{st}, \mathsf{ct}^*)$, *then we have*

$$|2 \cdot \Pr[\mathsf{coin}' = \mathsf{coin}] - 1/2| = \mathsf{negl}(\kappa).$$

2.3 One-Time Signature

Let $\{\mathcal{M}_\kappa\}_{\kappa\in\mathbb{N}}$ be a family of message space. In the following, we occasionally drop the subscript and simply write $\mathcal{M}$ when the meaning is clear.

Definition 2.3 (One-time Signature). *A one-time signature (OTS) scheme* Π_{OTS} *for message space* $\mathcal{M}$ *consists of PPT algorithms* $(\mathsf{OTS.KeyGen}, \mathsf{OTS.Sign}, \mathsf{OTS.Verify})$.

$\mathsf{OTS.KeyGen}(1^\kappa) \to (\mathsf{vk}, \mathsf{sigk})$*: The key generation algorithm takes as input the security parameter* 1^κ *and outputs a verification key* vk *and a signing key* sigk.

$\mathsf{OTS.Sign}(\mathsf{sigk}, \mathsf{M}) \to \sigma$*: The signing algorithm takes as input a signing key* sigk *and a message* $\mathsf{M} \in \mathcal{M}$ *and outputs a signature* σ.

$\mathsf{OTS.Verify}(\mathsf{vk}, \mathsf{M}, \sigma) \to \top$ *or* $\perp$*: The decryption algorithm takes as input a verification key* vk*, a message* M *and a signature* σ *and outputs* $\top$ *to indicate acceptance of the signature and* $\perp$ *otherwise.*

Correctness. *For all* $\kappa \in \mathbb{N}$, $\mathsf{M} \in \mathcal{M}$, $(\mathsf{vk}, \mathsf{sigk}) \in \mathsf{OTS.KeyGen}(1^\kappa)$, *and* $\sigma \in \mathsf{OTS.Sign}(\mathsf{sigk}, \mathsf{M})$, *we have* $\mathsf{OTS.Verify}(\mathsf{vk}, \mathsf{M}, \sigma) = \top$.

Strong One-Time Unforgeability. *For all* $\kappa \in \mathbb{N}$ *and all PPT adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$,*we have*

$$\Pr\left[\begin{array}{r|rl} & (\mathsf{vk}, \mathsf{sigk}) & \overset{\$}{\leftarrow} \mathsf{OTS.KeyGen}(1^\kappa) \\ \mathsf{OTS.Verify}(\mathsf{vk}, \mathsf{M}^*, \sigma^*) = \top & (\mathsf{M}, \mathsf{st}) & \overset{\$}{\leftarrow} \mathcal{A}_1(\mathsf{vk}), \\ (\mathsf{M}, \sigma) \neq (\mathsf{M}^*, \sigma^*) & \sigma & \overset{\$}{\leftarrow} \mathsf{OTS.Sign}(\mathsf{sigk}, \mathsf{M}) \\ & (\mathsf{M}^*, \sigma^*) & \overset{\$}{\leftarrow} \mathcal{A}_2(\mathsf{st}, \sigma) \end{array}\right] \leq \mathsf{negl}(\kappa).$$

2.4 Non-Interactive Zero-Knowledge Proofs (and Arguments)

Let $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ be a polynomial time recognizable binary relation. For $(x, w) \in \mathcal{R}$, we call x as the statement and w as the witness. Let $\mathcal{L}$ be the corresponding **NP** language $\mathcal{L} = \{x \mid \exists w \text{ s.t. } (x, w) \in \mathcal{R}\}$. Below, we define a non-interactive zero-knowledge proofs for **NP** languages. As we discuss briefly below, the following syntax also captures the designated-verifier setting.

Definition 2.4 (NIZK Proofs). *A non-interactive zero-knowledge (NIZK) proof Π_{NIZK} for the relation $\mathcal{R}$ consists of PPT algorithms* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$.

$\mathsf{Setup}(1^\kappa) \to (\mathsf{crs}, k_{\mathsf{V}})$*: The setup algorithm takes as input the security parameter 1^κ and outputs a common reference string* crs *and a verification key k_{V}.*

$\mathsf{Prove}(\mathsf{crs}, x, w) \to \pi$*: The prover's algorithm takes as input a common reference string* crs*, a statement x, and a witness w and outputs a proof π.*

$\mathsf{Verify}(\mathsf{crs}, k_{\mathsf{V}}, x, \pi) \to \top$ *or* $\bot$*: The verifier's algorithm takes as input a common reference string, a verification key k_{V}, a statement x, and a proof π and outputs $\top$ to indicate acceptance of the proof and $\bot$ otherwise.*

A NIZK proof Π_{NIZK} must satisfy the following requirements for all $\kappa \in \mathbb{N}$, where the probabilities are taken over the random choice of the algorithms.

Completeness. *For all pairs $(x, w) \in \mathcal{R}$, if we run $(\mathsf{crs}, k_{\mathsf{V}}) \xleftarrow{\$} \mathsf{Setup}(1^\kappa)$, then we have*

$$\Pr[\pi \xleftarrow{\$} \mathsf{Prove}(\mathsf{crs}, x, w) : \mathsf{Verify}(\mathsf{crs}, k_{\mathsf{V}}, x, \pi) = \top] = 1.$$

Soundness. *For all (possibly inefficient) adversaries $\mathcal{A}$, if we run $(\mathsf{crs}, k_{\mathsf{V}}) \xleftarrow{\$} \mathsf{Setup}(1^\kappa)$, then we have*

$$\Pr[(x, \pi) \xleftarrow{\$} \mathcal{A}^{\mathsf{Verify}(\mathsf{crs}, k_{\mathsf{V}}, \cdot, \cdot)}(1^\kappa, \mathsf{crs}) : x \notin \mathcal{L} \land \mathsf{Verify}(\mathsf{crs}, k_{\mathsf{V}}, x, \pi) = \top] = \mathsf{negl}(\kappa).$$

Here, in case soundness only holds for computationally bounded adversaries $\mathcal{A}$, we say it is a NIZK argument.

Zero-Knowledge. *For all PPT adversaries $\mathcal{A}$, there exists a PPT simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ such that if we run $(\mathsf{crs}, k_{\mathsf{V}}) \xleftarrow{\$} \mathsf{Setup}(1^\kappa)$ and $(\overline{\mathsf{crs}}, \bar{k}_{\mathsf{V}}, \bar{\tau}) \xleftarrow{\$} \mathcal{S}_1(1^\kappa)$, then we have*

$$\left|\Pr[\mathcal{A}^{\mathcal{O}_0(\mathsf{crs}, \cdot, \cdot)}(1^\kappa, \mathsf{crs}, k_{\mathsf{V}}) = 1] - \Pr[\mathcal{A}^{\mathcal{O}_1(\overline{\mathsf{crs}}, \bar{k}_{\mathsf{V}}, \bar{\tau}, \cdot, \cdot)}(1^\kappa, \overline{\mathsf{crs}}, \bar{k}_{\mathsf{V}}) = 1]\right| = \mathsf{negl}(\kappa),$$

where $\mathcal{O}_0(\mathsf{crs}, x, w)$ outputs $\mathsf{Prove}(\mathsf{crs}, x, w)$ if $(x, w) \in \mathcal{R}$ and $\bot$ otherwise, and $\mathcal{O}_1(\overline{\mathsf{crs}}, \bar{k}_{\mathsf{V}}, \bar{\tau}, x, w)$ outputs $\mathcal{S}_2(\overline{\mathsf{crs}}, \bar{k}_{\mathsf{V}}, \bar{\tau}, x)$ if $(x, w) \in \mathcal{R}$ and $\bot$ otherwise.

Remark 2.5 (On the verifier key k_{V}). In case $k_{\mathsf{V}} = \bot$ in the above syntax, i.e., the proof can be verified publicly, we refer to it as CRS-NIZKs. For notational simplicity, we omit k_{V} from the algorithm when we consider CRS-NIZKs. In case, $k_{\mathsf{V}} \neq \bot$ and the verification key must be kept private, we call it *designated verifier* NIZKs (DV-NIZKs) [PsV06, DFN06]. The above definition captures the so-called *multi-theorem* DV-NIZK where soundness is guaranteed even if an adversary can access to the verification oracle and zero-knowledge is guaranteed even after the prover observes an arbitrary number of proofs. Note that CRS-NIZKs by definition satisfies the multi-theorem soundness since verification can be done publicly.

The above definition does not directly capture any efficiency requirements on NIZKs. Our main goal of this paper is to obtain efficient NIZKs in the following respect:

Efficiency. Throughout the paper, when we discuss proof sizes and efficiency of NIZKs, $|x|$ and $|w|$ denotes the sizes of a statement and a witness, respectively, C denotes the circuit that computes the relation (i.e., $C(x, w) = 1$ if and only if $(x, w) \in \mathcal{R}$), and $|C|$ denotes the size of C (i.e., the number of gates of C).[9] We want to ensure that

[9]Though there are many circuits that compute the same relation, we assume a corresponding circuit that computes the relation is implicitly fixed whenever we consider a relation.

the NIZK is compact, that is, the size of the proof π should be as small. In particular, we aim $|\pi|$ to have at least an additive-overhead $|C| + \mathsf{poly}(\kappa)$, rather than a multiplicative-overhead $|C| \cdot \mathsf{poly}(\kappa)$; all pairing-based NIZKs based on falsifiable assumptions have multiplicative overhead. Moreover, in some cases we may also want the prover to be efficient, meaning that its running time is small as possible, and in particular, we want the running time to be at least smaller than the time to compute C, i.e., relation $\mathcal{R}$.

2.5 Computational Diffie-Hellman Assumption

Let GGen be a PPT algorithm that on input 1^κ returns a description $\mathcal{G} = (\mathbb{G}, p, g)$ where $\mathbb{G}$ is a cyclic group of prime order p and g is the generator of $\mathbb{G}$. Then the computational Diffie-Hellman assumption is defined as follows.

Definition 2.6 (Computational Diffie-Hellman Assumption). *We say that the computational Diffie-Hellman (CDH) assumption holds relative to* GGen *in group* $\mathbb{G}$ *if for all PPT adversaries* $\mathcal{A}$*,*

$$\Pr\left[\ \mathcal{G} = (\mathbb{G}, p, g) \xleftarrow{\$} \mathsf{GGen}(1^\kappa),\ \alpha, \beta \xleftarrow{\$} \mathbb{Z}_p :\ g^{\alpha\beta} \leftarrow \mathcal{A}(1^\kappa, \mathcal{G}, g^\alpha, g^\beta)\ \right] \leq \mathsf{negl}(\kappa).$$

Moreover, let BGGen be a PPT algorithm that on input 1^κ returns a description $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, g, e(\cdot, \cdot))$ of symmetric pairing groups where $\mathbb{G}$ and $\mathbb{G}_T$ are cyclic groups of prime order p, g is the generator of $\mathbb{G}$, and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is an efficiently computable (non-degenerate) bilinear map. Then the CDH assumption relative to BGGen is defined similarly.

2.6 Laconic Function Evaluation

We define here laconic function evaluation proposed by [QWW18] with a slightly simplified presentation. Let $\mathcal{C}_{k,d} = \{C : \{0,1\}^k \to \{0,1\}\}$ be a circuit class consisting of circuits with depth at most d. We typically omit the subscript from $\mathcal{C}$ for simplicity.

Definition 2.7 (Laconic Function Evaluation). *A laconic function evaluation (LFE) scheme for a class of circuits* $\mathcal{C}$ *consists of PPT algorithms* ($\mathsf{LFE.crsGen}, \mathsf{LFE.Compress}, \mathsf{LFE.Enc}, \mathsf{LFE.Dec}$).

$\mathsf{LFE.crsGen}(1^\kappa, 1^k, 1^d) \to \mathsf{crs}$: *The common random string generator algorithm takes as input the security parameter* 1^κ*, the input length* 1^k *and bound on the depth* 1^d *of the circuits in* $\mathcal{C}$*, and outputs a common random string* crs.

$\mathsf{LFE.Compress}(\mathsf{crs}, C) \to \mathsf{digest}_C$: *The deterministic compression algorithm takes as input the common random string* crs *and a circuit* $C \in \mathcal{C}$ *and outputs a digest* digest_C.

$\mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{digest}_C, x) \to \mathsf{ct}$: *The encryption algorithm takes as input the common random string* crs*, a digest* digest_C *and a message* $x \in \{0,1\}^k$*, and outputs a ciphertext* ct.

$\mathsf{LFE.Dec}(\mathsf{crs}, C, \mathsf{ct}) \to y$: *The decryption algorithm takes as input the common random string* crs*, a circuit* C *and a ciphertext* ct*, and outputs a message* y.

An LFE scheme must satisfy the following requirements.

Correctness. *For all* $\kappa \in \mathbb{N}$, $k, d \in \mathsf{poly}(\kappa)$, $C \in \mathcal{C}$ *and* $x \in \{0,1\}^k$*, if we run* $\mathsf{crs} \xleftarrow{\$} \mathsf{LFE.crsGen}(1^\kappa, 1^k, 1^d)$, $\mathsf{digest}_C = \mathsf{LFE.Compress}(\mathsf{crs}, C)$, $\mathsf{ct} \xleftarrow{\$} \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{digest}_C, x)$*, and* $y \xleftarrow{\$} \mathsf{LFE.Dec}(\mathsf{crs}, C, \mathsf{ct})$*, then we have* $\Pr[C(x) = y] = 1$*, where the probability is taken over the randomness of all the algorithms.*

(Adaptive) Security. *There exists a PPT simulator* $\mathsf{LFE.Sim}$ *such that for all (stateful) PPT adversary* $\mathcal{A}$*, we have*

$$\left|\Pr\left[\mathsf{Exp}^{\mathsf{Real}}_{\mathsf{LFE},\mathcal{A}}(1^\kappa) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{Ideal}}_{\mathsf{LFE},\mathcal{A}}(1^\kappa) = 1\right]\right| \leq \mathsf{negl}(\kappa),$$

where the two experiments $\mathsf{Exp}^{\mathsf{Real}}_{\mathsf{LFE},\mathcal{A}}$ *and* $\mathsf{Exp}^{\mathsf{Ideal}}_{\mathsf{LFE},\mathcal{A}}$ *are defined in Figure 1. Note that the adaptive security holds under the same digest* digest_C *even if* $\mathcal{A}$ *is given many ciphertexts* ct_i *of many messages* x_i *where each* x_i *may arbitrarily*

$\mathsf{Exp}^{\mathsf{Real}}_{\mathsf{LFE},\mathcal{A}}$	$\mathsf{Exp}^{\mathsf{Ideal}}_{\mathsf{LFE},\mathcal{A}}$
1. $(1^k, 1^d) \xleftarrow{\$} \mathcal{A}(1^\kappa)$	1. $(1^k, 1^d) \xleftarrow{\$} \mathcal{A}(1^\kappa)$
2. $\mathsf{crs} \xleftarrow{\$} \mathsf{LFE.crsGen}(1^\kappa, \mathsf{params})$	2. $\mathsf{crs} \xleftarrow{\$} \mathsf{LFE.crsGen}(1^\kappa, \mathsf{params})$
3. $(x^*, C) \xleftarrow{\$} \mathcal{A}(\mathsf{crs}) : C \in \mathcal{C}_{k,d}$	3. $(x^*, C) \xleftarrow{\$} \mathcal{A}(\mathsf{crs}) : C \in \mathcal{C}_{k,d}$
4. $\mathsf{digest}_C = \mathsf{LFE.Compress}(\mathsf{crs}, C)$	4. $\mathsf{digest}_C = \mathsf{LFE.Compress}(\mathsf{crs}, C)$
5. $\mathsf{ct} \xleftarrow{\$} \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{digest}_C, x^*)$	5. $\mathsf{ct} \xleftarrow{\$} \mathsf{LFE.Sim}(\mathsf{crs}, C, \mathsf{digest}_C, C(x^*))$
6. Output $\mathcal{A}(\mathsf{ct})$	6. Output $\mathcal{A}(\mathsf{ct})$

Figure 1: Security Experiment for LFE.

depend on the ciphertexts it has received because the simulator is given a honestly generated crs*. This was observed by Quach et al [QWW18]. We call this version* multi-challenge *adaptive security in this paper.*

Efficiency. *We say the LFE is* laconic *if the size of* crs*,* digest_C*, and* ct*, and the run-time of* LFE.Enc *is strictly smaller than the circuit size* $|C|$*.*

Quach et al. [QWW18] showed how to construct LFE based on the sub-exponential security of the learning with errors (LWE) assumption or the adaptive LWE assumption. We put definitions of these assumptions in Appendix A since we only use LFE in a black-box manner, and do not care about the underlying assumptions.

Lemma 2.8. *Under the sub-exponential security of the LWE assumption with subexponential modulus-to-noise ratio, there exists an adaptive LFE for circuit class* $\mathcal{C}_{k,d}$ *where the* crs *size is* $\mathsf{poly}(\kappa, k, d)$*, the* digest_C *size is* $\mathsf{poly}(\kappa)$*, the ciphertext* ct *size and the running time of the encryption algorithm is* $\mathsf{poly}(\kappa, k, d)$*, and the running time of the compression and decryption algorithm is* $\tilde{O}(|C|) \cdot \mathsf{poly}(\kappa, k, d)$*.*
Moreover, under the adaptive *LWE assumption with subexponential modulus-to-noise ratio, there exists an adaptive LFE where the* crs *size is* $k \cdot \mathsf{poly}(\kappa, d)$*, the* digest_C *size is* $\mathsf{poly}(\kappa)$*, the ciphertext* ct *size and the running time of the encryption algorithm is* $\tilde{O}(k) \cdot \mathsf{poly}(\kappa, d)$*, and the running time of the compression and decryption algorithm is* $\tilde{O}(|C|) \cdot \mathsf{poly}(\kappa, d)$*.*

3 Homomorphic Equivocal Commitment

3.1 Definition

We introduce a new primitive which we call homomorphic equivocal commitment (HEC), which can be seen as a relaxed variant of HTDF defined by Gorbunov et al. [GVW15]. (See Appendix D for the construction of HEC from HTDF.) A HEC scheme with message space $\mathcal{X}$, randomness space $\mathcal{R}$, and a randomness distribution $\mathcal{D}_{\mathcal{R}}$ over $\mathcal{R}$ for a circuit class $\mathcal{C} = \{C : \mathcal{X} \to \mathcal{Z}\}$ consists of PPT algorithms ($\mathsf{HEC.Setup}, \mathsf{HEC.Commit}, \mathsf{HEC.Open}, \mathsf{HEC.Eval}^{in}, \mathsf{HEC.Eval}^{out}, \mathsf{HEC.Verify}$).

$\mathsf{HEC.Setup}(1^\kappa)$: The setup algorithm takes as input the security parameter 1^κ and outputs a public parameter pp, an evaluation key ek, and a master secret key msk.

$\mathsf{HEC.Commit}(\mathsf{pp}, \mathbf{x}; R)$: The commit algorithm takes as input a public parameter pp and a message $\mathbf{x} \in \mathcal{X}$ along with a randomness $R \in \mathcal{R}$, and outputs a commitment com. When we omit R to denote $\mathsf{HEC.Commit}(\mathsf{pp}, \mathbf{x})$, we mean that R is chosen according to the distribution $\mathcal{D}_{\mathcal{R}}$.

$\mathsf{HEC.Open}(\mathsf{msk}, (\mathbf{x}, R), \mathbf{x}')$: The open algorithm takes as input a master secret key msk, a message $\mathbf{x} \in \mathcal{X}$, a randomness $R \in \mathcal{R}$, and a fake message $\mathbf{x}' \in \mathcal{X}$, and outputs a fake randomness $R' \in \mathcal{R}$.

$\mathsf{HEC.Eval}^{in}(\mathsf{ek}, C, \mathbf{x}, R)$: The inner evaluation algorithm takes as input an evaluation key ek, a circuit $C \in \mathcal{C}$, a message $\mathbf{x} \in \mathcal{X}$, and a randomness $R \in \mathcal{R}$, and outputs a proof π.

$\mathsf{HEC.Eval}^{out}(\mathsf{ek}, C, \mathsf{com})$: The outer evaluation algorithm is a deterministic algorithm that takes as input an evaluation key ek, a circuit $C \in \mathcal{C}$, and a commitment com, and outputs an evaluated commitment $\mathsf{com}_{\mathsf{eval}}$.

$\mathsf{HEC.Verify}(\mathsf{pp}, \mathsf{com}_{\mathsf{eval}}, z, \pi)$: The verification algorithm takes as input a public parameter pp, an evaluated commitment $\mathsf{com}_{\mathsf{eval}}$, a message $z \in \mathcal{Z}$, and a proof π, and outputs $\top$ if the proof is valid and $\bot$ otherwise.

Evaluation Correctness. For all $\kappa \in \mathbb{Z}$, $(\mathsf{pp}, \mathsf{ek}, \mathsf{msk}) \xleftarrow{\$} \mathsf{HEC.Setup}(1^\kappa), \mathbf{x} \in \mathcal{X}, R \in \mathcal{R}, \mathsf{com} := \mathsf{HEC.Commit}(\mathsf{pp}, \mathbf{x}; R)$, $C \in \mathcal{C}$, $\pi \xleftarrow{\$} \mathsf{HEC.Eval}^{in}(\mathsf{msk}, C, \mathbf{x}, R)$, and $\mathsf{com}_{\mathsf{eval}} := \mathsf{HEC.Eval}^{out}(\mathsf{ek}, C, \mathsf{com})$, we have

$$\Pr[\mathsf{HEC.Verify}(\mathsf{pp}, \mathsf{com}_{\mathsf{eval}}, C(\mathbf{x}), \pi) = \top] = 1.$$

Distributional Equivalence of Open. We have

$$\{(\mathsf{pp}, \mathsf{ek}, \mathsf{msk}, \mathbf{x}, R, \mathsf{com})\} \overset{\text{stat}}{\approx} \{(\mathsf{pp}, \mathsf{ek}, \mathsf{msk}, \mathbf{x}, R', \mathsf{com}')\}$$

where $(\mathsf{pp}, \mathsf{ek}, \mathsf{msk}) \xleftarrow{\$} \mathsf{HEC.Setup}(1^\kappa)$, $(\mathbf{x}, \overline{\mathbf{x}}) \in \mathcal{X}^2$ are arbitrary random variables that may depend on $(\mathsf{pp}, \mathsf{ek}, \mathsf{msk})$, $R \xleftarrow{\$} \mathcal{D}_{\mathcal{R}}$, $\mathsf{com} := \mathsf{HEC.Commit}(\mathsf{pp}, \mathbf{x}; R)$, $\overline{R} \xleftarrow{\$} \mathcal{D}_{\mathcal{R}}$, $\mathsf{com}' := \mathsf{HEC.Commit}(\mathsf{pp}, \overline{\mathbf{x}}; \overline{R})$, and $R' \xleftarrow{\$} \mathsf{HEC.Open}(\mathsf{msk}, (\overline{\mathbf{x}}, \overline{R}), \mathbf{x})$.

Computational Binding for Evaluated Commitment. For all PPT adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{r|rl} \mathsf{HEC.Verify}(\mathsf{pp}, \mathsf{com}_{\mathsf{eval}}, z^*, \pi^*) = \top & (\mathsf{pp}, \mathsf{ek}, \mathsf{msk}) & \xleftarrow{\$} \mathsf{HEC.Setup}(1^\kappa), \\ z^* \neq C(\mathbf{x}) & (\mathbf{x}, R, C, z^*, \pi^*) & \xleftarrow{\$} \mathcal{A}(\mathsf{pp}, \mathsf{ek}), \\ & \mathsf{com} & := \mathsf{HEC.Commit}(\mathsf{pp}, \mathbf{x}; R) \\ & \mathsf{com}_{\mathsf{eval}} & := \mathsf{HEC.Eval}^{out}(\mathsf{ek}, C, \mathsf{com}) \end{array}\right] \leq \mathsf{negl}(\kappa).$$

Efficient Committing. There exists a polynomial poly such that for all $(\mathsf{pp}, \mathsf{ek}, \mathsf{msk}) \xleftarrow{\$} \mathsf{HEC.Setup}(1^\kappa)$, $\mathbf{x} \in \mathcal{X}$, $R \in \mathcal{R}$, the running time of $\mathsf{com} := \mathsf{HEC.Commit}(\mathsf{pp}, \mathbf{x}; R)$ is bounded by $|\mathbf{x}| \cdot \mathsf{poly}(\kappa)$.
Efficient Verification (optional). There exists a polynomial poly such that for all $(\mathsf{pp}, \mathsf{ek}, \mathsf{msk}) \xleftarrow{\$} \mathsf{HEC.Setup}(1^\kappa)$, $\mathbf{x} \in \mathcal{X}, R \in \mathcal{R}, \mathsf{com} := \mathsf{HEC.Commit}(\mathsf{pp}, \mathbf{x}; R), C \in \mathcal{C}, \pi \xleftarrow{\$} \mathsf{HEC.Eval}^{in}(\mathsf{ek}, C, \mathbf{x}, R), \mathsf{com}_{\mathsf{eval}} := \mathsf{HEC.Eval}^{out}(\mathsf{ek}, C, \mathsf{com})$, and $z \in \mathcal{Z}$, we have $|\pi| \leq \mathsf{poly}(\kappa)$ and $|\mathsf{com}_{\mathsf{eval}}| \leq \mathsf{poly}(\kappa)$ and the running time of $\mathsf{HEC.Verify}(\mathsf{pp}, \mathsf{com}_{\mathsf{eval}}, z, \pi)$ is at most $\mathsf{poly}(\kappa)$. We remark that poly does not depend on C.

Context-Hiding (optional). There exists a PPT simulator HEC.ProofSim such that for all $\kappa \in \mathbb{N}$, $(\mathsf{pp}, \mathsf{ek}, \mathsf{msk}) \xleftarrow{\$} \mathsf{HEC.Setup}(1^\kappa)$, $\mathbf{x} \in \mathcal{X}$, $C \in \mathcal{C}$, $R \in \mathcal{R}$, and $\mathsf{com} := \mathsf{HEC.Commit}(\mathsf{pp}, \mathbf{x}; R)$, we have

$$\{\pi \xleftarrow{\$} \mathsf{HEC.Eval}^{in}(\mathsf{ek}, C, \mathbf{x}, R))\} \overset{\text{stat}}{\approx} \{\pi' \xleftarrow{\$} \mathsf{HEC.ProofSim}(\mathsf{msk}, \mathsf{com}, C, C(\mathbf{x})))\}$$

where the probability is only over the randomness used by the algorithms $\mathsf{HEC.Eval}^{in}$ and HEC.ProofSim.

Remark 3.1. We can generically convert any HEC scheme to a context-hiding one by using any statistical CRS-NIZK scheme. Namely, instead of directly using π as an output of the inner evaluation algorithm, it outputs a NIZK proof for the statement that there exists π that passes the verification.

Remark 3.2. The following properties immediately follow from the distributional equivalence of open.
Equivocality. We have

$$\Pr[\mathsf{HEC.Commit}(\mathsf{pp}, \overline{\mathbf{x}}; \overline{R}) \neq \mathsf{HEC.Commit}(\mathsf{pp}, \mathbf{x}; R)] = \mathsf{negl}(\kappa)$$

where $(\mathsf{pp}, \mathsf{ek}, \mathsf{msk}) \xleftarrow{\$} \mathsf{HEC.Setup}(1^\kappa)$, $(\mathbf{x}, \overline{\mathbf{x}}) \in \mathcal{X}^2$ are arbitrary random variables that may depend on $(\mathsf{pp}, \mathsf{ek}, \mathsf{msk})$, $\overline{R} \xleftarrow{\$} \mathcal{D}_{\mathcal{R}}$, and $R \xleftarrow{\$} \mathsf{HEC.Open}(\mathsf{msk}, (\overline{\mathbf{x}}, \overline{R}), \mathbf{x})$.
Hiding. We have

$$\{\mathsf{pp}, \mathsf{ek}, \mathsf{com} \xleftarrow{\$} \mathsf{HEC.Commit}(\mathsf{pp}, \mathbf{x})\} \overset{\text{stat}}{\approx} \{\mathsf{pp}, \mathsf{ek}, \mathsf{com}' \xleftarrow{\$} \mathsf{HEC.Commit}(\mathsf{pp}, \mathbf{x}')\},$$

where $(\mathsf{pp}, \mathsf{ek}, \mathsf{msk}) \xleftarrow{\$} \mathsf{HEC.Setup}(1^\kappa)$ and $(\mathbf{x}, \mathbf{x}') \in \mathcal{X}^2$ are arbitrary random variables that may depend on $(\mathsf{pp}, \mathsf{ek}, \mathsf{msk})$. We say that a scheme is computationally hiding if the above two distributions are computationally indistinguishable.

Remark 3.3. If we require neither efficient verification nor context-hiding, then there is a trivial construction of HEC based on any equivocal commitment. Namely, we can just set $\mathsf{com}_{\mathsf{eval}} := C \| \mathsf{com}$ and $\pi := (\mathbf{x}, R)$. The verification algorithm can verify them by checking if com is a commitment of $\mathbf{x}$ with randomness R and $z = C(\mathbf{x})$ holds. On the other hand, if we require either of efficient verification or context hiding, then there does not seem to be such a trivial solution.[10] This is reminiscent of the similar situation for fully homomorphic encryption where a scheme without compactness nor function privacy is trivial to construct but a scheme with either of them is non-trivial [Gen09].

3.2 Construction of HEC with Efficient Verification from CDHER Assumption

Here we give a construction of HEC scheme with efficient verification based on the (n, m)-CDHER assumption. The construction and security proof of this HEC scheme is similar to those of homomorphic signature scheme by Katsumata et al. [KNYY19]. Though our HEC scheme can be obtained by a slight syntactic adaptation of their homomorphic signature scheme, we give the full description and security proof for completeness. We note that many parts in this section are taken verbatim from their paper.

Preparation: Bilinear Maps and Monotone Span Programs.

Before providing the concrete construction, we first prepare the hardness assumption and tools that will be used throughout this section.

Let BGGen be a PPT algorithm that on input 1^κ returns a description $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, g, e(\cdot, \cdot))$ of symmetric pairing groups where $\mathbb{G}$ and $\mathbb{G}_T$ are cyclic groups of prime order p, g is the generator of $\mathbb{G}$, and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is an efficiently computable (non-degenerate) bilinear map.

Definition 3.4 (**(n, m)-Computational Diffie-Hellman Exponent and Ratio Assumption**). *[KNYY19] Let* BGGen *be a group generator and $n := n(\kappa) = \mathsf{poly}(\kappa)$, $m := m(\kappa) = \mathsf{poly}(\kappa)$. We say that the (n, m)-*decisional Diffie-Hellman exponent and ratio (CDHER) assumption *holds with respect to* BGGen, *if for all PPT adversaries $\mathcal{A}$, we have*

$$\Pr\left[\mathcal{A}(\mathcal{G}, g, \Psi) \to e(g, g)^{sa^{m+1}}\right] = \mathsf{negl}(\kappa)$$

where $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, g, e(\cdot, \cdot)) \xleftarrow{\$} \mathsf{BGGen}(1^\lambda)$, $g \xleftarrow{\$} \mathbb{G}$, $s, a, b_1, \ldots, b_n, c_1, \ldots c_n \xleftarrow{\$} \mathbb{Z}_p^*$, and

$$\Phi := \begin{pmatrix} \left\{g^{a^j}\right\}_{j\in[m]}, & \left\{g^{c_i}\right\}_{i\in[n]}, & \left\{g^{a^j/b_i}\right\}_{\substack{i\in[n], j\in[2m]\\ j\neq m+1}}, & \left\{g^{a^{m+1}c_{i'}/b_i c_i}\right\}_{i,i'\in[n], i\neq i'}, \\ & \left\{g^{ac_i}\right\}_{i\in[n]}, & \left\{g^{a^j/b_i c_i}\right\}_{i\in[n], j\in[2m+1]}, & \left\{g^{a^j c_{i'}/b_i}\right\}_{i,i'\in[n], j\in[m]}, \\ g^s, & \left\{g^{sb_i}\right\}_{i\in[n]}, & \left\{g^{sa^{m+1}b_i/b_{i'}c_{i'}}\right\}_{i,i'\in[n], i\neq i'}, & \left\{g^{sa^j b_i/b_{i'}}\right\}_{\substack{i,i'\in[n], j\in[m]\\ i\neq i'}} \end{pmatrix}.$$

Katsumata et al. showed that the CDHER assumption holds in the generic group model introduced by Shoup [Sho97]. We define a slightly simplified version of (monotone) span programs below.

Definition 3.5 (Monotone Span Program). *A (monotone) span program for universe $[n]$ is a matrix $\mathbf{M}$, where $\mathbf{M}$ is an $n \times m$ matrix over $\mathbb{Z}_p$. Given $\mathbf{y} = (y_1, \ldots, y_n) \in \{0, 1\}^n$, we say that*

$$\mathbf{y} \textit{ satisfies } \mathbf{M} \textit{ iff } \mathbf{1} \in \text{span}\,\langle \mathbf{M}_I \rangle\,.$$

Here $\mathbf{1} = (1, 0, \ldots, 0) \in \mathbb{Z}_p^{1\times m}$ is a row vector; $\mathbf{M}_I$ denotes the matrix obtained by removing the j-th row of $\mathbf{M}$ for j such that $j \notin I$ for $I := \{i \in [n] \mid y_i = 0\}$[11]; and span *refers to $\mathbb{Z}_p$-linear span of row vectors.*

[10] As remarked in Remark 3.1, we can convert the trivial construction to a context-hiding one additionally assuming a statistical CRS-NIZK for all of **NP**. Though this is less interesting than schemes with efficient verification, we do not consider it a "trivial solution" since the existence of a statistical CRS-NIZK is an additional assumption to an equivocal commitment.

[11] We note that our definition of I here is somewhat non-standard. Usually, we define I as $I := \{i \in [n] \mid y_i = 1\}$. This change is introduced because it slightly simplifies our presentation and is not essential.

That is, $\mathbf{y}$ satisfies $\mathbf{M}$ iff there exist coefficients $\{w_i \in \mathbb{Z}_p\}_{i \in I}$ such that

$$\sum_{i \in I} w_i \mathbf{M}_i = \mathbf{1},$$

where $\mathbf{M}_i$ denotes the i-th row vector of $\mathbf{M}$. Observe that the coefficients $\{w_i \in \mathbb{Z}_p\}_{i \in I}$ can be computed in time polynomial in the size of $\mathbf{M}$ via Gaussian elimination.

Note that we adopt a slightly non-standard definition of the monotone span program, in that we do not allow the program to read the same input bit multiple times. This is for the sake of the brevity and this limitation can be removed by blowing up the matrices as well as inputs by a polynomial factor.

The following lemma is taken from [GPSW06a].

Lemma 3.6 ([GPSW06a, Proposition 1]). *If a vector $\mathbf{y} \in \{0,1\}^n$ does not satisfy a (monotone) span program $\mathbf{M} \in \mathbb{Z}_p^{n \times m}$, then there exists an efficiently computable vector $\mathbf{d} = (d_1, \ldots, d_m) \in \mathbb{Z}_p^m$ such that $\mathbf{M}_I \mathbf{d}^\top = \mathbf{0}$ and $d_1 = -1$, where $I := \{i \in [n] \mid y_i = 0\}$.*

It is well known that $\mathbf{NC}^1$ circuits can be represented as a polynomial-sized Boolean formulae. Furthermore, any polynomial-sized Boolean formulae can be converted into an equivalent monotone span program (See e.g., Appendix G of [LW11]). The span program we obtain is the standard one where the same input bit is read multiple times, but this can be converted into one with one-time read as we noted above. Combining them, we have the following lemma, which allows us to use (monotone) span programs as $\mathbf{NC}^1$ circuits instead.

Lemma 3.7. *Let $d = d(\kappa), \ell = \ell(\kappa)$, and $s = s(\kappa)$ be integers. There exist integer parameters $n = n(d, \ell, s)$ and $m = m(d, \ell, s)$ and* deterministic *algorithms* EncInp *and* EncCir *with the following properties.*

- $\mathsf{EncInp}(\mathbf{x}) \to \mathbf{y} \in \{0,1\}^n$, *where* $\mathbf{x} \in \{0,1\}^\ell$.
- $\mathsf{EncCir}(C) \to \mathbf{M} \in \{-1,0,1\}^{n \times m}$, *where $C : \{0,1\}^\ell \to \{0,1\}$ is a circuit with depth and size bounded by d and s, respectively.*

We have that $\mathbf{y}$ satisfies the span program $\mathbf{M}$ over $\mathbb{Z}_p$ if and only if $C(\mathbf{x}) = 0$ for any prime modulus $p > 3$. We also have that the running time of EncCir *is* $\mathsf{poly}(\ell, s, 2^d)$. *In particular, if C is a polynomial-sized circuit with logarithmic depth (i.e., if the circuit is in $\mathbf{NC}^1$),* EncCir *runs in polynomial time. In particular, we have $n = \mathsf{poly}(\kappa)$ and $m = \mathsf{poly}(\kappa)$ in this case. Furthermore, for $\mathbf{x} \in \{0,1\}^\ell$, we have*

$$\mathsf{EncInp}(\mathbf{x}) = (\neg x_1)^\eta \| x_1^\eta \| \cdots \| (\neg x_\ell)^\eta \| x_\ell^\eta \in \{0,1\}^n,$$

for some integer η such that $n = 2\ell\eta$. In the above, x_i denotes the i-th bit of $\mathbf{x}$ and x_i^η is the η-times repetition of the bit x_i.

Description of the Function Class. Let $d(\kappa) = O(\log \kappa)$, $\ell = \mathsf{poly}(\kappa)$, and $s = \mathsf{poly}(\kappa)$. The circuit class dealt with by our HEC scheme is denoted as $\mathcal{C}^{\mathbf{NC}^1} = \{\mathcal{C}^{\mathbf{NC}^1}_{\kappa,d,\ell,s}\}_{\kappa \in \mathbb{N}}$, where $\mathcal{C}^{\mathbf{NC}^1}_{\kappa,d,\ell,s}$ is a set of circuits whose input lengths are ℓ, and depths and sizes are bounded by d and s, respectively. We also define a circuit class $\tilde{\mathcal{C}}^{\mathbf{NC}^1} = \{\tilde{\mathcal{C}}^{\mathbf{NC}^1}_{\kappa,d,\ell,s}\}_{\kappa \in \mathbb{N}}$ associated with $\mathcal{C}^{\mathbf{NC}^1}$ as

$$\tilde{\mathcal{C}}^{\mathbf{NC}^1}_{\kappa,d,\ell,s} = \{\tilde{C}_z(\cdot) = (C(\cdot) \overset{?}{=} z) \mid \forall z \in \{0,1\}, \forall C \in \mathcal{C}^{\mathbf{NC}^1}_{\kappa,d,\ell,s}\}.$$

More specifically, $\tilde{\mathcal{C}}^{\mathbf{NC}^1}_{\kappa,d,\ell,s}$ is a set of circuits with input length ℓ such that a circuit $\tilde{C}_z \in \tilde{\mathcal{C}}^{\mathbf{NC}^1}_{\kappa,d,\ell,s}$ on input $\mathbf{x} \in \{0,1\}^\ell$ outputs 1 if and only if circuit $C \in \mathcal{C}^{\mathbf{NC}^1}_{\kappa,d,\ell,s}$ outputs z on input $\mathbf{x}$. Since $(z \overset{?}{=} z')$ for $z, z' \in \{0,1\}$ can be expressed by a constant-size circuit, every circuit in $\tilde{\mathcal{C}}^{\mathbf{NC}^1}_{\kappa,d,\ell,s}$ have input length ℓ, and the depths and sizes are bounded by $d + O(1)$ and $s + O(1)$, respectively. Then, by Lemma 3.7, there exist $n(\kappa) = \mathsf{poly}(\kappa)$ and $m(\kappa) = \mathsf{poly}(\kappa)$ such that $\mathsf{EncInp}(\mathbf{x}) \in \{0,1\}^n$ for any $\mathbf{x} \in \{0,1\}^\ell$ and $\mathsf{EncCir}(C) \in \{-1,0,1\}^{n \times m}$ for any $C \in \tilde{\mathcal{C}}^{\mathbf{NC}^1}_{\kappa,d,\ell,s}$. Furthermore, all of these values can be computed in time $\mathsf{poly}(\kappa)$. In the following, since we fix κ, d, ℓ, and s in the construction, we drop the subscript and denote $\mathcal{C}^{\mathbf{NC}^1}_{\kappa,d,\ell,s}, \tilde{\mathcal{C}}^{\mathbf{NC}^1}_{\kappa,d,\ell,s}$ as $\mathcal{C}^{\mathbf{NC}^1}, \tilde{\mathcal{C}}^{\mathbf{NC}^1}$ for notational convenience.

Construction. Our HEC scheme $\Pi_{\mathsf{HEC}} = (\mathsf{HEC.Setup}, \mathsf{HEC.Commit}, \mathsf{HEC.Open}, \mathsf{HEC.Eval}^{in}, \mathsf{HEC.Eval}^{out}, \mathsf{HEC.Verify})$ with message space $\{0,1\}^\ell$ and randomness space $\mathbb{Z}_p$ (on which the distribution $\mathcal{D}_\mathcal{R}$ is defined to be the uniform distribution) for the function class $\mathcal{C}^{\mathbf{NC}^1}$ is described below.

$\mathsf{HEC.Setup}(1^\kappa)$: On input the security parameter 1^κ, sample the group description $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, g, e(\cdot,\cdot)) \xleftarrow{\$} \mathsf{BGGen}(1^\kappa)$. Then, sample $a \leftarrow \mathbb{Z}_p$, and $b_i, c_i \leftarrow \mathbb{Z}_p^*$ for $i \in [n]$ and outputs

$$\mathsf{ek} := \left(\begin{array}{cccc} \left\{ g^{a^j} \right\}_{j\in[m]}, & \{g^{c_i}\}_{i\in[n]}, & \left\{ g^{a^j/b_i} \right\}_{\substack{i\in[n], j\in[2m],\\ j\neq m+1}}, & \left\{ g^{a^{m+1}c_{i'}/b_i c_i} \right\}_{i,i'\in[n], i\neq i'}, \\ & \{g^{ac_i}\}_{i\in[n]}, & \left\{ g^{a^j/b_i c_i} \right\}_{i\in[n], j\in[2m+1]}, & \left\{ g^{a^j c_{i'}/b_i} \right\}_{i,i'\in[n], j\in[m]} \end{array} \right), \quad (1)$$

$$\mathsf{pp} := \left(\left\{ V_{i,\beta} := \prod_{j\in[(2i+\beta-2)\eta+1, (2i+\beta-1)\eta]} g^{a^{m+1}/b_j c_j} \right\}_{i\in[\ell], \beta\in\{0,1\}} \right),$$

and

$$\mathsf{msk} := \big(a, \{b_i, c_i\}_{i\in[n]}\big).$$

$\mathsf{HEC.Commit}(\mathsf{pp}, \mathbf{x}; R)$: On input $\mathbf{x} \in \{0,1\}^\ell$ and a randomness $R \in \mathbb{Z}_p$, compute and output

$$\mathsf{com} := g^R \cdot \prod_{i\in[\ell]} V_{i,x_i}.$$

We note that we have $\mathsf{com} = g^{R + \sum_{i\in[n]} y_i \cdot (a^{m+1}/b_i c_i)}$ for $\mathbf{y} = \mathsf{EncInp}(\mathbf{x})$, since we have

$$\begin{aligned} & \sum_{i\in[\ell]} \left(\sum_{j\in[(2i+x_i-2)\eta+1, (2i+x_i-1)\eta]} a^{m+1}/b_j c_j \right) \\ = & \sum_{i\in[\ell]} \left(\sum_{j\in[(2i-2)\eta+1, (2i-1)\eta]} (\neg x_i)\cdot (a^{m+1}/b_j c_j) + \sum_{j\in[(2i-1)\eta+1, 2i\eta]} x_i \cdot (a^{m+1}/b_j c_j) \right) \\ = & \sum_{i\in[n]} y_i \cdot (a^{m+1}/b_i c_i), \end{aligned}$$

where the last equation follows from the definition of $\mathbf{y}$ (See Lemma 3.7).

Remark 3.8. Note that we could have set $\mathsf{pp} = \emptyset$ since pp can be efficiently computed from ek. However, if we do so, we can no longer have efficient committing property, since the computation of com would require time linear in n rather than ℓ. To ensure efficient committing property, we precompute some of the elements that are required and put them into pp.

$\mathsf{HEC.Open}(\mathsf{msk}, (\mathbf{x}, R), \mathbf{x}')$: Parse $\big(a, \{b_i, c_i\}_{i\in[n]}\big) \leftarrow \mathsf{msk}$, run $\mathsf{EncInp}(\mathbf{x}) = \mathbf{y} \in \{0,1\}^n$ and $\mathsf{EncInp}(\mathbf{x}') = \mathbf{y}' \in \{0,1\}^n$, compute

$$R' := R + \sum_{i\in[n]} (y_i - y_i') \cdot \big(a^{m+1}/b_i c_i\big)$$

and output R'.

$\mathsf{HEC.Eval}^{in}(\mathsf{ek}, C, \mathbf{x}, R)$: Compute $z = C(\mathbf{x}) \in \{0,1\}$ and construct the circuit $\tilde{C}_z \in \tilde{\mathcal{C}}^{\mathbf{NC}^1}$. Here, we have $\tilde{C}_z(\mathbf{x}) = 1$ by the definition. Then, run $\mathsf{EncInp}(\mathbf{x}) = \mathbf{y} \in \{0,1\}^n$ and $\mathsf{EncCir}(\tilde{C}_z) = \mathbf{M} \in \mathbb{Z}_p^{n\times m}$. By Lemma 3.7, $\mathbf{y}$ *does not* satisfy the span program $\mathbf{M}$ since $\tilde{C}_z(\mathbf{x}) = 1$. Then find a vector $\mathbf{d} = (d_1, \dots, d_m) \in \mathbb{Z}_p^m$ such that $d_1 = -1$

and $\langle \mathbf{M}_i, \mathbf{d} \rangle = 0$ for all $i \in [n]$ satisfying $y_i = 0$, where $\mathbf{M}_i$ is the i-th row of $\mathbf{M}$. Note that such a vector exists and can be found efficiently due to Lemma 3.6. Then pick $\tilde{r} \xleftarrow{\$} \mathbb{Z}_p$ and compute

$$K_1 = g^{\tilde{r}} \cdot \prod_{j\in[m]} \left(g^{a^{m+1-j}}\right)^{d_j} \cdot \prod_{i\in[n]} (g^{c_i})^{-\langle \mathbf{M}_i,\mathbf{d}\rangle} \quad \text{and} \tag{2}$$

$$K_2 = (g^a)^{\tilde{r}} \cdot \prod_{j\in[2,m]} (g^{a^{m+2-j}})^{d_j} \cdot \prod_{i\in[n]} (g^{ac_i})^{-\langle \mathbf{M}_i,\mathbf{d}\rangle}. \tag{3}$$

Note that the above terms can be efficiently computed as linear combinations of the group elements in the evaluation key ek. Then compute

$$L_1 := K_1^R \cdot \left(\prod_{i\in[n]} \left(g^{a^{m+1}/b_i c_i}\right)^{y_i} \cdot \prod_{i\in[n],j\in[m]} \left(g^{a^j/b_i}\right)^{M_{i,j}} \right)^{\tilde{r}},$$

$$L_2 := \prod_{i\in[n],j\in[m]} \left(g^{a^{2m+2-j}/b_i c_i}\right)^{d_j y_i} \cdot \prod_{i,i'\in[n],j\in[m]} \left(g^{a^j c_{i'}/b_i}\right)^{-\langle \mathbf{M}_{i'},\mathbf{d}\rangle M_{i,j}},$$

$$L_3 := \prod_{\substack{i,i'\in[n]\\ i\neq i'}} \left(g^{a^{m+1}c_{i'}/b_i c_i}\right)^{-\langle \mathbf{M}_{i'},\mathbf{d}\rangle y_i} \cdot \prod_{\substack{i\in[n],j,j'\in[m],\\ j\neq j'}} \left(g^{a^{m+1-j'+j}/b_i}\right)^{M_{i,j}d_{j'}}, \quad \text{and}$$

$$K_3 := L_1 \cdot L_2 \cdot L_3 \tag{4}$$

Note that all of them can be efficiently computed as linear combinations of the group elements in the evaluation key ek. Finally, output $\pi = (K_1, K_2, K_3)$.

$\mathsf{HEC.Eval}^{out}(\mathsf{ek}, C, \mathsf{com})$: Construct the circuit $\tilde{C}_z \in \tilde{\mathcal{C}}^{\mathbf{NC}^1}$ and run $\mathsf{EncCir}(\tilde{C}_z) = \mathbf{M}_z \in \mathbb{Z}_p^{n\times m}$ for $z \in \{0,1\}$. Then compute $e(g,g)^{a^{m+1}} = e(g^a, g^{a^m})$ and

$$W_z := \mathsf{com} \cdot \prod_{i\in[n],j\in[m]} \left(g^{a^j/b_i}\right)^{M_{z,i,j}}$$

for $z \in \{0,1\}$, where $M_{z,i,j}$ is the (i,j)-th entry of $\mathbf{M}_z$. Finally output $\mathsf{com}_{\mathsf{eval}} = (e(g,g)^{a^{m+1}}, W_0, W_1)$.

$\mathsf{HEC.Verify}(\mathsf{pp}, \mathsf{com}_{\mathsf{eval}}, z, \pi)$: Parse $\mathsf{com}_{\mathsf{eval}} = (e(g,g)^{a^{m+1}}, W_0, W_1)$ and $\pi = (K_1, K_2, K_3)$. Then, check the following conditions:

$$e\left(K_1,\ W_z\right) \overset{?}{=} e(K_3, g), \qquad e(g, K_2)\cdot e(g^a, K_1)^{-1} \overset{?}{=} e(g,g)^{a^{m+1}}.$$

If the above equations hold, output $\top$. Otherwise output $\perp$.

Context Hiding. Before proving the correctness, we prove that the scheme satisfies context-hiding since this is useful for proving the correctness.

Theorem 3.9. Π_{HEC} *satisfies context-hiding.*

Proof. This proof is almost entirely taken from the proof of context hiding of homomorphic signatures by Katsumata et al. [KNYY19] except syntactical adaptation to fit in the HEC setting. We include the proof for completeness.

To show the context-hiding property, we first construct the simulator HEC.ProofSim as follows:

$\mathsf{HEC.ProofSim}(\mathsf{msk}, \mathsf{com}, C, z)$: On input a circuit $C \in \mathcal{C}^{\mathbf{NC}^1}$ and a message $z \in \{0,1\}$, it first constructs the circuit $\tilde{C}_z \in \tilde{\mathcal{C}}^{\mathbf{NC}^1}$ associated to circuit C. Then, it computes $\mathsf{EncCir}(\tilde{C}_z) = \mathbf{M} \in \mathbb{Z}_p^{n\times m}$. It then picks $r \xleftarrow{\$} \mathbb{Z}_p$ and computes

$$K_1 := g^r, \quad K_2 := g^{a^{m+1}} \cdot (g^a)^r, \quad K_3 := \left(\mathsf{com} \cdot \prod_{i\in[n],j\in[m]} \left(g^{a^j/b_i}\right)^{M_{i,j}}\right)^r \tag{5}$$

and outputs $\pi = (K_1, K_2, K_3)$.

We now proceed to show that this simulator HEC.Sim satisfies the required conditions. Let $\mathbf{x} \in \{0,1\}^\ell$ be an arbitrary input such that $\tilde{C}_z(\mathbf{x}) = 1$, let $y_i \in \{0,1\}$ be the i-th bit of $\mathbf{y} = \mathsf{EncInp}(\mathbf{x})$, and let $\mathbf{d} \in \mathbb{Z}_p^m$ be as defined in $\mathsf{HEC.Eval}^{in}$. Let $\tilde{r} \in \mathbb{Z}_p$ be the unique element such that we have

$$r = \tilde{r} + \sum_{j\in[m]} d_j a^{m+1-j} - \sum_{i\in[n]} \langle \mathbf{M}_i, \mathbf{d}\rangle c_i$$

where $\mathbf{M}_i$ is the i-th row of $\mathbf{M}$, and rewrite K_1, K_2, and K_3 output by the simulator by using $\tilde{r}$. We note that $\tilde{r}$ is uniform over $\mathbb{Z}_p$ if r is uniform over $\mathbb{Z}_p$. Therefore what is left is to prove that K_1, K_2, and K_3 are written by the form of Eq. (2), (3), and (4) by using $\tilde{r}$. First, it is clear that K_1 is written by the form of Eq. (2) by the definition of $\tilde{r}$. Next, K_2 can be written by the form of (3) since we have

$$\begin{aligned}\log_g K_2 &= a^{m+1} + ar \\ &= a^{m+1} + a\left(\tilde{r} + \sum_{j\in[m]} d_j a^{m+1-j} - \sum_{i\in[n]} \langle \mathbf{M}_i, \mathbf{d}\rangle c_i\right) \\ &= \tilde{r}a + \sum_{j\in[2,m]} d_j a^{m+2-j} - \sum_{i\in[n]} \langle \mathbf{M}_i, \mathbf{d}\rangle a c_i,\end{aligned}$$

where the last equation follows from $d_1 = -1$. Note that the term a^{m+1} cancels out here.

We need some more work for K_3. We have

$$\begin{aligned}\log_g K_3 &= r\left(R + \sum_{i\in[n]} y_i a^{m+1}/b_i c_i + \sum_{i\in[n], j\in[m]} M_{i,j} a^j / b_i\right) \\ &= \left(\tilde{r} + \underbrace{\sum_{j\in[m]} d_j a^{m+1-j} - \sum_{i\in[n]} \langle \mathbf{M}_i, \mathbf{d}\rangle c_i}_{:=\Phi_1}\right) \cdot \left(R + \underbrace{\sum_{i\in[n]} y_i a^{m+1}/b_i c_i + \sum_{i\in[n], j\in[m]} M_{i,j} a^j / b_i}_{:=\Phi_2}\right) \\ &= R(\tilde{r} + \Phi_1) + \tilde{r}\Phi_2 + \Phi_1\Phi_2 \\ &= \underbrace{Rr + \tilde{r}\Phi_2}_{:=\Phi_3} + \Phi_1\Phi_2\end{aligned}$$

We can observe that $g^{\Phi_3} = L_1$. We next expand $\Phi_1\Phi_2$ and show that $g^{\Phi_1\Phi_2} = L_2 L_3$, which concludes the proof. Before doing so, we define

$$\Phi_{1,1} := \sum_{j\in[m]} d_j a^{m+1-j}, \qquad \Phi_{1,2} := -\sum_{i\in[n]} \langle \mathbf{M}_i, \mathbf{d}\rangle c_i,$$
$$\Phi_{2,1} := \sum_{i\in[n]} y_i a^{m+1}/b_i c_i, \qquad \Phi_{2,2} := \sum_{i\in[n], j\in[m]} M_{i,j} a^j / b_i.$$

It is readily seen that $\Phi_i = \Phi_{i,1} + \Phi_{i,2}$ for $i = 1,2$ and thus $\Phi_1\Phi_2 = \Phi_{1,1}\Phi_{2,1} + \Phi_{1,1}\Phi_{2,2} + \Phi_{1,2}\Phi_{2,1} + \Phi_{1,2}\Phi_{2,2}$. We have

$$\Phi_{1,1}\Phi_{2,1} + \Phi_{1,2}\Phi_{2,2} = \sum_{i\in[n], j\in[m]} d_j y_i \left(a^{2m+2-j}/b_i c_i\right) - \sum_{i,i'\in[n], j\in[m]} \langle \mathbf{M}_{i'}, \mathbf{d}\rangle M_{i,j} \left(a^j c_{i'}/b_i\right),$$

$$\Phi_{1,2}\Phi_{2,1} = \left(-\sum_{i'\in[n]} \langle \mathbf{M}_{i'}, \mathbf{d}\rangle c_{i'}\right)\left(\sum_{i\in[n]} y_i a^{m+1}/b_i c_i\right)$$

$$= -\sum_{\substack{i,i' \in [n] \\ i \neq i'}} \langle \mathbf{M}_{i'}, \mathbf{d} \rangle y_i \left(a^{m+1} c_{i'}/b_i c_i\right) - \sum_{i \in [n]} \langle \mathbf{M}_i, \mathbf{d} \rangle y_i \left(a^{m+1}/b_i\right)$$

and

$$\begin{aligned}
\Phi_{1,1}\Phi_{2,2} &= \left(\sum_{j' \in [m]} d_{j'} a^{m+1-j'}\right) \cdot \left(\sum_{i \in [n], j \in [m]} M_{i,j} a^j / b_i\right) \\
&= \sum_{\substack{i \in [n], j,j' \in [m], \\ j \neq j'}} M_{i,j} d_{j'} \left(a^{m+1-j'+j}/b_i\right) + \sum_{i \in [n], j \in [m]} M_{i,j} d_j \left(a^{m+1}/b_i\right) \\
&= \sum_{\substack{i \in [n], j,j' \in [m], \\ j \neq j'}} M_{i,j} d_{j'} \left(a^{m+1-j'+j}/b_i\right) + \sum_{i \in [n]} \langle \mathbf{M}_i, \mathbf{d} \rangle \left(a^{m+1}/b_i\right) \\
&= \sum_{\substack{i \in [n], j,j' \in [m], \\ j \neq j'}} M_{i,j} d_{j'} \left(a^{m+1-j'+j}/b_i\right) + \sum_{i \in [n]} \langle \mathbf{M}_i, \mathbf{d} \rangle y_i \left(a^{m+1}/b_i\right),
\end{aligned}$$

where in the last equation we used $y_i \in \{0,1\}$ and $\langle \mathbf{M}_i, \mathbf{d} \rangle = 0$ if $y_i = 0$. These imply that

$$\Phi_{1,2}\Phi_{2,1} + \Phi_{1,1}\Phi_{2,2} = -\sum_{\substack{i,i' \in [n] \\ i \neq i'}} \langle \mathbf{M}_{i'}, \mathbf{d} \rangle y_i \left(a^{m+1} c_{i'}/b_i c_i\right) + \sum_{\substack{i \in [n], j,j' \in [m], \\ j \neq j'}} M_{i,j} d_{j'} \left(a^{m+1-j'+j}/b_i\right),$$

where the terms of the form a^{m+1}/b_i all cancel out here. We can see that $g^{\Phi_{1,1}\Phi_{2,1}+\Phi_{1,2}\Phi_{2,2}} = L_2$ and $g^{\Phi_{1,2}\Phi_{2,1}+\Phi_{1,1}\Phi_{2,2}} = L_3$, which imply $g^{\Phi_1\Phi_2} = L_2 L_3$. This concludes the proof of the theorem. ☐

Correctness.

Theorem 3.10. Π_{HEC} *satisfies correctness.*

Proof. As seen in Theorem 3.9, for any $\mathbf{x} \in \{0,1\}^\ell$ and C, if we generate $(\mathsf{pp}, \mathsf{ek}, \mathsf{msk}) \xleftarrow{\$} \mathsf{HEC.Setup}(1^\kappa)$, $\mathsf{com} \xleftarrow{\$} \mathsf{HEC}(\mathsf{pp}, \mathbf{x}; R)$ where $R \xleftarrow{\$} \mathbb{Z}_p$, and $\pi \xleftarrow{\$} \mathsf{HEC.Eval}^{in}(\mathsf{ek}, C, \mathbf{x}, R)$, then $\pi = (K_1, K_2, K_3)$ can be written by the form as below:

$$K_1 := g^r, \quad K_2 := g^{a^{m+1}} \cdot (g^a)^r, \quad K_3 := \left(\mathsf{com} \cdot \prod_{i \in [n], j \in [m]} \left(g^{a^j/b_i}\right)^{M_{i,j}}\right)^r \tag{6}$$

where $z = C(\mathbf{x})$, $\mathbf{M} := \mathsf{EncCir}(\tilde{C}_z)$, and $M_{i,j}$ is the (i,j)-th entry of $\mathbf{M}$. Then it is straightforward to verify that $\pi = (K_1, K_2, K_3)$ satisfies the equations checked by $\mathsf{HEC.Verify}$. ☐

Distributional equivalence of open.

Theorem 3.11. Π_{HEC} *satisfies distributional equivalence of open.*

Proof. Let $(\mathsf{pp}, \mathsf{ek}, \mathsf{msk}) \xleftarrow{\$} \mathsf{HEC.Setup}(1^\kappa)$ and $\mathbf{x}, \overline{\mathbf{x}} \in \{0,1\}^\ell$ be arbitrary (that may depend on $(\mathsf{pp}, \mathsf{ek}, \mathsf{msk})$). Suppose that we generate $\mathsf{com} \xleftarrow{\$} \mathsf{HEC.Commit}(\mathsf{pp}, \mathbf{x})$ where we use a uniform randomness $R \xleftarrow{\$} \mathbb{Z}_p$. Then it is clear that $\mathsf{com} = g^{R + \sum_{i \in [n]} y_i \cdot \left(a^{m+1}/b_i c_i\right)}$ is uniformly distributed over $\mathbb{G}$ where $\mathbf{y} := \mathsf{EncInp}(\mathbf{x})$. On the other hand, if we pick $\overline{R}$ to generate $\mathsf{com}' := \mathsf{HEC.Commit}(\mathsf{pp}, \overline{\mathbf{x}}; \overline{R})$, then $\mathsf{com}' = g^{\overline{R} + \sum_{i \in [n]} \overline{y}_i \cdot \left(a^{m+1}/b_i c_i\right)}$ is also uniformly distributed on $\mathbb{G}$ where $\overline{\mathbf{y}} := \mathsf{EncInp}(\overline{\mathbf{x}})$. Moreover, if we generate $R' \xleftarrow{\$} \mathsf{HEC.Open}(\mathsf{msk}, (\overline{\mathbf{x}}, \overline{R}), \mathbf{x})$, then we have $R' = \overline{R} + \sum_{i \in [n]} (\overline{y}_i - y_i) \cdot \left(a^{m+1}/b_i c_i\right)$ and $g^{R' + \sum_{i \in [n]} y_i \cdot \left(a^{m+1}/b_i c_i\right)} = g^{\overline{R} + \sum_{i \in [n]} \overline{y}_i \cdot \left(a^{m+1}/b_i c_i\right)} = \mathsf{com}'$, since

$R' \in \mathbb{Z}_p$ is the unique element that satisfies this equation if we fix $(\mathsf{pp}, \mathsf{ek}, \mathsf{msk})$, $\mathbf{x}$, and com'. Thus we can conclude that we have

$$\{(\mathsf{pp}, \mathsf{ek}, \mathsf{msk}, \mathbf{x}, R, \mathsf{com})\} \overset{\text{stat}}{\approx} \{(\mathsf{pp}, \mathsf{ek}, \mathsf{msk}, \mathbf{x}, R', \mathsf{com}')\}$$

□

Computational binding for evaluated commitments.

Theorem 3.12. *If the (n, m)-CDHER assumption holds, then Π_{HEC} satisfies computational binding for evaluated commitments.*

Proof. This proof is almost entirely taken from the proof of unforgeability of the HS scheme by Katsumata et al. [KNYY19] except syntactical adaptation to fit in the HEC setting. We include the proof for completeness.

Suppose that a PPT adversary $\mathcal{A}$ breaks the computational binding for evaluated commitments of Π_{HEC}. Then we construct a PPT algorithm $\mathcal{B}$ that breaks the (n, m)-CDHER assumption.

$\mathcal{B}$ is given the problem instance Ψ of the (n, m)-CDHER problem. It sets pp and ek by using the corresponding part of Ψ, and gives pp to $\mathcal{A}$. Then $\mathcal{A}$ outputs $(\mathbf{x}, R, C, z^*, \pi^*)$. $\mathcal{B}$ aborts and outputs $\perp$ if $\mathcal{A}$ did not win the game. Otherwise, $\mathcal{B}$ parses the proof as $\pi^* = (K_1^*, K_2^*, K_3^*)$, constructs the circuit $\tilde{C}_{z^*} \in \tilde{\mathcal{C}}^{\mathbf{NC}^1}$ from C and z^*, and computes $\mathbf{M}^* \overset{\$}{\leftarrow} \mathsf{EncCir}(\tilde{C}_{z^*})$. Since the proof passes the verification, we have

$$e\left(K_1^*,\ W_{z^*}\right)^{-1} = e(K_3^*, g), \qquad e(g, K_2^*) \cdot e(g^a, K_1^*)^{-1} = e(g, g)^{a^{m+1}}$$

where

$$W_{z^*} = \mathsf{com} \cdot \prod_{i\in[n], j\in[m]} \left(g^{a^j/b_i}\right)^{M_{i,j}^*}, \qquad \mathsf{com} = g^{R + \sum_{i\in[n]} y_i \cdot \left(a^{m+1}/b_i c_i\right)}.$$

Then if we let $\widetilde{R} := R + \sum_{i\in[n]} y_i \cdot \left(a^{m+1}/b_i c_i\right) \mod p$ and $r^* := \log_g K_1^*$, then we have

$$K_1^* := g^{r^*}, \quad K_2^* := g^{a^{m+1}} \cdot (g^a)^{r^*}, \quad K_3^* := \left(g^{\widetilde{R}} \cdot \prod_{i\in[n], j\in[m]} \left(g^{a^j/b_i}\right)^{M_{i,j}^*}\right)^{r^*}, \tag{7}$$

where $M_{i,j}^*$ is the (i, j)-th entry of $\mathbf{M}^*$.

We then extract the answer for the CDHER problem from it. Let us define $I := \{i \in [n] : y_i = 0\}$. We first observe that we can compute $(g^{\widetilde{R}})^{sb_i}$ for $i \in I$ because we have

$$(g^{\widetilde{R}})^{sb_i} = \left(g^R \cdot \prod_{i'\in[n]} \left(g^{a^{m+1}/b_{i'} c_{i'}}\right)^{y_{i'}}\right)^{sb_i} = \left(g^{sb_i}\right)^R \cdot \prod_{i'\in[n]\setminus\{i\}} \left(g^{sa^{m+1}b_i/b_{i'}c_{i'}}\right)^{y_{i'}},$$

where the second equality above follows from $y_i = 0$. We then define

$$s_i^* := \sum_{j\in[m]} M_{i,j}^* s a^{j-1}$$

for $j \in [m]$. We then define G_i for $i \in [n]$ as follows:

$$\begin{aligned} G_i &:= (g^a)^{-s_i^*} \cdot \left(g^{\widetilde{R}} \cdot \prod_{i'\in[n], j\in[m]} \left(g^{a^j/b_{i'}}\right)^{M_{i',j}^*}\right)^{sb_i} \\ &= \prod_{j\in[m]} \left(g^{sa^j}\right)^{-M_{i,j}^*} \cdot \left(g^{\widetilde{R}}\right)^{sb_i} \cdot \prod_{i'\in[n], j\in[m]} \left(g^{sa^j b_i/b_{i'}}\right)^{M_{i',j}^*} \end{aligned}$$

$$= (g^{\widetilde{R}})^{sb_i} \cdot \prod_{i' \in [n]\setminus\{i\}, j \in [m]} \left(g^{sa^j b_i / b_{i'}} \right)^{M^*_{i',j}},$$

where the terms $\{g^{sa^j}\}$ all cancel out in the third equality. We can observe that G_i for $i \in I$ can be efficiently computed as a linear combination of the terms in the problem instance of the CDHER. We therefore can compute

$$e(K_1^*, G_i) \cdot e(K_3^*, g^{sb_i})^{-1} = e(g,g)^{-r^* a s_i^*}$$

for $i \in I$. Next, since $\mathcal{A}$ succeeds in breaking the binding property, we must have $C(\mathbf{x}) \neq z^*$. Specifically, we have $\tilde{C}_{z^*}(\mathbf{x}) = 0$. By Lemma 3.7, this implies that $\mathbf{y}$ satisfies the span program $\mathbf{M}^*$. Hence, by Definition 3.5, there exists an efficiently computable coefficients $\{w_i^*\}_{i \in I}$ such that $\sum_{i \in I} w_i^* \mathbf{M}_i^* = \mathbf{1}$. We observe that

$$\sum_{i \in I} w_i^* s_i^* = \sum_{i \in I} w_i^* \left(\sum_{j \in [m]} M^*_{i,j} s a^{j-1} \right) = \left(\sum_{i \in I} w_i^* \mathbf{M}_i^* \right) \cdot (s, sa, \ldots, sa^{m-1})^\top = s$$

holds for such $\{w_i^*\}_{i \in I}$. $\mathcal{B}$ then computes

$$\begin{aligned} e(K_2^*, g^s) \cdot \prod_{i \in I} \left(e(g,g)^{-r^* a s_i^*} \right)^{w_i^*} &= e(g,g)^{sa^{m+1}} \cdot e(g,g)^{sar^*} \cdot e(g,g)^{-r^* a \sum_{i \in I} w_i^* s_i^*} \\ &= e(g,g)^{sa^{m+1}}, \end{aligned}$$

where g^s is taken from the problem instance. Finally, $\mathcal{B}$ outputs $e(g,g)^{sa^{m+1}}$ as the answer to the (n,m)-CDHER problem. $\square$

Efficiency of Committing. Since com is computed by a product of $\ell = |\mathbf{x}|$ group elements, it is clear that the running time HEC.Commit is bounded by $|\mathbf{x}| \cdot \mathsf{poly}(\kappa)$.

Efficiency of Verification. It is easy to see that the sizes of π and $\mathsf{com}_{\mathsf{eval}}$ are $\mathsf{poly}(\kappa)$ since they consist of constant number of group elements, and HEC.Verify runs in time $\mathsf{poly}(\kappa)$ since it just verifies two pairing equations.

3.3 Construction of HEC without Efficient Verification from CDH

Preparations. To describe our HEC scheme from the CDH assumption, we use the following notation.

For a group element $g \in \mathbb{G}$ and vectors $\mathbf{v} = (v_1, \ldots, v_m)^\top$ and $\mathbf{w} = (w_1, \ldots, w_m)^\top$, $g^{\mathbf{v}}$ denotes a vector of group elements $(g^{v_1}, \ldots, g^{v_m})^\top$ and $\mathbf{v} \odot \mathbf{w}$ denotes a vector $(v_1 w_1, \ldots, v_m w_m)^\top$. Given $g^{\mathbf{v}}$ and $\mathbf{w}$, one can efficiently compute $g^{\mathbf{v} \odot \mathbf{w}}$ and $g^{\mathbf{v}^\top \mathbf{w}}$. Similarly, given $g^{\mathbf{v}}$ and $g^{\mathbf{w}}$, one can efficiently compute $e(g,g)^{\mathbf{v} \odot \mathbf{w}}$ and $g^{\mathbf{v}+\mathbf{w}} = g^{\mathbf{v}} \cdot g^{\mathbf{w}}$, where "$\cdot$" denotes component-wise multiplication in $\mathbb{G}$. One can also efficiently compute $g^{\mathbf{M}\mathbf{v}}$ given $g^{\mathbf{v}}$ and $\mathbf{M} \in \mathbb{Z}_p^{n \times m}$.

The following lemma will be useful when describing and analyzing our construction of HEC for $\mathbf{NC}^1$ circuits from the CDH assumption.

Lemma 3.13. *Let us define* GPSWKeyGen *as follows.*

- $\mathsf{GPSWKeyGen}(g^\gamma, g^{\mathbf{u}}, \mathbf{M})$: *It takes as input $g^\gamma \in \mathbb{G}$, $g^{\mathbf{u}} \in \mathbb{G}^n$, and $\mathbf{M} \in \mathbb{Z}_p^{n \times m}$. It then picks $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_p^{m-1}$ and $\mathbf{t} \xleftarrow{\$} \mathbb{Z}_p^n$ and outputs*
$$\vec{K} = \left(\vec{K}_0 = g^{\mathbf{M}\binom{\gamma}{\mathbf{s}} + \mathbf{t} \odot \mathbf{u}},\ \vec{K}_1 = g^{\mathbf{t}} \right) \in \mathbb{G}^n \times \mathbb{G}^n.$$

Then, there exists a PPT algorithm GPSWSim *that takes as input $g^\alpha, g^\beta \in \mathbb{G}$, $\mathbf{r} \in \mathbb{Z}_p^n$, $\mathbf{y} \in \{0,1\}^n$, and $\mathbf{M} \in \mathbb{Z}_p^{n \times m}$ and outputs $\vec{K} \in \mathbb{G}^n \times \mathbb{G}^n$ with the following property:*

For all $\alpha, \beta \in \mathbb{Z}_p$, $\mathbf{r} \in \mathbb{Z}_p^n$, $\mathbf{y} \in \{0,1\}^n$, and $\mathbf{M} \in \mathbb{Z}_p^{n \times m}$ such that $\mathbf{y}$ does not *satisfy* $\mathbf{M}$ *(when seeing the latter as a span program), the following distributions are equivalent:*

$$\{\vec{K} \leftarrow \mathsf{GPSWKeyGen}(g^{\alpha\beta}, g^{\mathbf{u}}, \mathbf{M})\} \approx \{\vec{K} \leftarrow \mathsf{GPSWSim}(g^\alpha, g^\beta, \mathbf{r}, \mathbf{y}, \mathbf{M})\},$$

where $\mathbf{u}$ *is defined as*

$$\mathbf{u} = \mathbf{r} + \alpha \cdot \mathbf{y}.$$

Here, $\mathbf{y} \in \{0,1\}^n$ *is regarded as a (column) vector in* $\mathbb{Z}_p^n$*. Furthermore, there exists an efficient deterministic algorithm* $\mathsf{GPSWKeyCheck}$ *that takes as input* $g^{\mathbf{u}} \in \mathbb{G}^n$*,* $e(g,g)^\gamma \in \mathbb{G}_T$*,* $\mathbf{M} \in \mathbb{Z}_p^{n\times m}$*, and* $\vec{K} \in \mathbb{G}^n \times \mathbb{G}^n$ *and outputs* $\top$ *if* $\vec{K} \in \mathsf{GPSWKeyGen}(g^\gamma, g^{\mathbf{u}}, \mathbf{M})$ *and otherwise* $\perp$*.*

Remark 3.14. The lemma extracts the essential components of the selectively-secure ABE scheme of [GPSW06b]. At a high level, $\mathsf{GPSWKeyGen}$ corresponds to the "real" key generation algorithm and $\mathsf{GPSWSim}$ corresponds to the "simulated" key generation algorithm. Informally, $\mathbf{y}$ is the challenge attribute chosen by the adversary at the outset of the selective security game and the reduction algorithm runs $\mathsf{GPSWSim}$ instead of $\mathsf{GPSWKeyGen}$ to answer the key generation query. One additional algorithm that is not present in the GPSW-ABE scheme is the $\mathsf{GPSWKeyCheck}$ algorithm. This algorithm checks consistency of the simulated key output by $\mathsf{GPSWSim}$ and is crucial for verification of proofs later on in the HEC construction.

Proof. We first prove the statement about $\mathsf{GPSWSim}$. We define $\mathsf{GPSWSim}$ as follows.

$\mathsf{GPSWSim}(g^\alpha, g^\beta, \mathbf{r}, \mathbf{y}, \mathbf{M})$: It first computes a vector $\mathbf{v} \in \mathbb{Z}_p^{m-1}$ such that $\mathbf{M}_I \left(\begin{smallmatrix}1\\ \mathbf{v}\end{smallmatrix}\right) = \mathbf{0}$ for $I := \{i \in [n] \mid y_i = 0\}$. Such a vector exists because $\mathbf{y}$ does not satisfy $\mathbf{M}$ by Lemma 3.6. Furthermore, it can be efficiently computed. For such $\mathbf{v}$, we have $\mathbf{M}\left(\begin{smallmatrix}1\\ \mathbf{v}\end{smallmatrix}\right) = \mathbf{y} \odot \mathbf{w}$ for some $\mathbf{w} \in \mathbb{Z}_p^n$. It then samples $\tilde{\mathbf{t}} \xleftarrow{\$} \mathbb{Z}_p^n$ and $\tilde{\mathbf{s}} \xleftarrow{\$} \mathbb{Z}_p^{m-1}$ and outputs

$$\vec{K}_0 = g^{\mathbf{M}\left(\begin{smallmatrix}0\\ \tilde{\mathbf{s}}\end{smallmatrix}\right) + \tilde{\mathbf{t}}\odot\mathbf{r}} \cdot (g^\alpha)^{\tilde{\mathbf{t}}\odot\mathbf{y}} \cdot (g^\beta)^{-\mathbf{w}\odot\tilde{\mathbf{u}}}, \qquad \vec{K}_1 = g^{\tilde{\mathbf{t}}} \cdot (g^\beta)^{-\mathbf{w}}.$$

We claim that if we set $\mathbf{s} = \alpha\beta\mathbf{v} + \tilde{\mathbf{s}}$, $\mathbf{t} = \tilde{\mathbf{t}} - \beta\mathbf{w}$, and $\mathbf{u} = \mathbf{r} + \alpha\mathbf{y}$, we have

$$\vec{K}_0 = g^{\mathbf{M}\left(\begin{smallmatrix}\alpha\beta\\ \mathbf{s}\end{smallmatrix}\right) + \mathbf{t}\odot\mathbf{u}}, \qquad \vec{K}_1 = g^{\mathbf{t}}.$$

It is easy to check that $\vec{K}_1 = g^{\mathbf{t}}$. We have

$$\begin{aligned}
\mathbf{M}\left(\begin{smallmatrix}\alpha\beta\\ \mathbf{s}\end{smallmatrix}\right) + \mathbf{t}\odot\mathbf{u} &= \mathbf{M}\begin{pmatrix}\alpha\beta\\ \alpha\beta\mathbf{v}+\tilde{\mathbf{s}}\end{pmatrix} + (\tilde{\mathbf{t}} - \beta\mathbf{w}) \odot (\mathbf{r} + \alpha\mathbf{y}) \\
&= \alpha\beta\mathbf{M}\left(\begin{smallmatrix}1\\ \mathbf{v}\end{smallmatrix}\right) + \mathbf{M}\left(\begin{smallmatrix}0\\ \tilde{\mathbf{s}}\end{smallmatrix}\right) + (\tilde{\mathbf{t}} - \beta\mathbf{w}) \odot (\mathbf{r} + \alpha\mathbf{y}) \\
&= \alpha\beta\mathbf{y}\odot\mathbf{w} + \mathbf{M}\left(\begin{smallmatrix}0\\ \tilde{\mathbf{s}}\end{smallmatrix}\right) + \tilde{\mathbf{t}}\odot\mathbf{r} + \alpha\tilde{\mathbf{t}}\odot\mathbf{y} - \beta\mathbf{w}\odot\mathbf{r} - \alpha\beta\mathbf{w}\odot\mathbf{y} \\
&= \left(\mathbf{M}\left(\begin{smallmatrix}0\\ \tilde{\mathbf{s}}\end{smallmatrix}\right) + \tilde{\mathbf{t}}\odot\mathbf{r}\right) + \alpha\cdot\left(\tilde{\mathbf{t}}\odot\mathbf{y}\right) - \beta\cdot(\mathbf{w}\odot\mathbf{r}).
\end{aligned}$$

We therefore have $\vec{K}_0 = g^{\mathbf{M}\left(\begin{smallmatrix}\alpha\beta\\ \mathbf{s}\end{smallmatrix}\right) + \mathbf{t}\odot\mathbf{u}}$ as well. Furthermore, $\mathbf{s}$ and $\mathbf{t}$ are distributed uniformly at random over $\mathbb{Z}_p^{m-1}$ and $\mathbb{Z}_p^n$ respectively. Therefore, output distribution of $\mathsf{GPSWKeyGen}(g^{\alpha\beta}, g^{\mathbf{u}}, \mathbf{M})$ and $\mathsf{GPSWSim}(g^\alpha, g^\beta, \mathbf{r}, \mathbf{y}, \mathbf{M})$ are the same as required.

We then prove the statement about $\mathsf{GPSWKeyCheck}$. We define $\mathsf{GPSWKeyCheck}$ as follows.

$\mathsf{GPSWKeyCheck}(g^{\mathbf{u}}, e(g,g)^\gamma, \mathbf{M}, \vec{K})$: It first checks $\vec{K} \in \mathbb{G}^n \times \mathbb{G}^n$ and outputs $\perp$ otherwise. If it holds, it parses $\vec{K} \to (\vec{K}_0 = g^{\mathbf{k}}, \vec{K}_1 = g^{\mathbf{t}})$, where $\mathbf{k}$ and $\mathbf{t}$ are vectors in $\mathbb{Z}_p^n$ that are not known to the algorithm. It then computes

$$e(g,g)^{\mathbf{k}'} := e(g,g)^{\mathbf{k}} \cdot e(g,g)^{-\mathbf{t}\odot\mathbf{u}} \cdot e(g,g)^{-\mathbf{M}\left(\begin{smallmatrix}\gamma\\ \mathbf{0}\end{smallmatrix}\right)},$$

where the first term in the right-hand side can be computed from $g^{\mathbf{k}}$, the second term can be computed from $g^{\mathbf{t}}$ and $g^{\mathbf{u}}$, and the third term can be computed from $e(g,g)^\gamma$ and $\mathbf{M}$. Let $\mathbf{M}_{-1} \in \mathbb{Z}_p^{n\times(m-1)}$ be the matrix obtained by removing the first column of $\mathbf{M}$. If the columns of $\mathbf{M}_{-1}$ spans $\mathbb{Z}_p^n$, it outputs $\top$. Otherwise, it computes $\mathbf{M}_{-1}^\perp \in \mathbb{Z}_p^{n\times n'}$ such that the columns of $\mathbf{M}_{-1}^\perp$ span the space $\{\mathbf{v} \in \mathbb{Z}_p^n : \mathbf{v}^\top\mathbf{M}_{-1} = \mathbf{0}\}$. It then checks

$$e(g,g)^{(\mathbf{M}_{-1}^\perp)^\top\mathbf{k}'} = e(g,g)^{\mathbf{0}}$$

and outputs $\top$ if it holds. Otherwise, it outputs $\perp$.

We then prove that $\mathsf{GPSWKeyCheck}(g^{\mathbf{u}}, e(g,g)^{\gamma}, \mathbf{M}, \vec{K})$ outputs $\top$ if and only if $\vec{K} \in \mathsf{GPSWKeyGen}(g^{\gamma}, g^{\mathbf{u}}, \mathbf{M})$.

We first prove the "if" direction. If $\vec{K} \in \mathsf{GPSWKeyGen}(g^{\gamma}, g^{\mathbf{u}}, \mathbf{M})$, we have $\mathbf{k} = \mathbf{M}\binom{\gamma}{\mathbf{s}} + \mathbf{t} \odot \mathbf{u}$ for some $\mathbf{s} \in \mathbb{Z}_p^{m-1}$ and $\mathbf{t} \in \mathbb{Z}_p^n$. Therefore, $\mathbf{k}' = \mathbf{M}\binom{0}{\mathbf{s}} = \mathbf{M}_{-1}\mathbf{s}$ and thus $(\mathbf{M}_{-1}^{\perp})^{\top}\mathbf{k}' = (\mathbf{M}_{-1}^{\perp})^{\top}\mathbf{M}_{-1}\mathbf{s} = \mathbf{0}$. Therefore, the output of $\mathsf{GPSWKeyCheck}$ is $\top$.

We then prove the "only if" direction. If the output of $\mathsf{GPSWKeyCheck}$ is $\top$, we have $(\mathbf{M}_{-1}^{\perp})^{\top}\mathbf{k}' = \mathbf{0}$. Thus, there exists $\mathbf{s} \in \mathbb{Z}_p^{m-1}$ such that $\mathbf{k}' = \mathbf{M}_{-1}\mathbf{s} = \mathbf{M}\binom{0}{\mathbf{s}}$. Then, we have $\mathbf{k} = \mathbf{t} \odot \mathbf{u} + \mathbf{M}\binom{\gamma}{\mathbf{0}} + \mathbf{k}' = \mathbf{t} \odot \mathbf{u} + \mathbf{M}\binom{\gamma}{\mathbf{s}}$. Therefore, we can conclude $\vec{K} = (g^{\mathbf{k}}, g^{\mathbf{t}}) \in \mathsf{GPSWKeyGen}(g^{\gamma}, g^{\mathbf{u}}, \mathbf{M})$. This completes the proof of the lemma. $\square$

Construction.

In this section, we provide a construction of context-hiding HEC for $\mathbf{NC}^1$ circuits without efficient verification from the CDH assumption. Let us define $\mathcal{C}^{\mathbf{NC}^1} = \{\mathcal{C}^{\mathbf{NC}^1}_{\kappa,d,\ell,s}\}_{\kappa\in\mathbb{N}}$ and $\tilde{\mathcal{C}}^{\mathbf{NC}^1} = \{\tilde{\mathcal{C}}^{\mathbf{NC}^1}_{\kappa,d,\ell,s}\}_{\kappa\in\mathbb{N}}$ as in Section 3.2. The circuit class dealt with by our scheme here is exactly the same as that in Section 3.2, i.e., $\mathcal{C}^{\mathbf{NC}^1}$. Our HEC scheme $\Pi'_{\mathsf{HEC}} = (\mathsf{HEC.Setup}, \mathsf{HEC.Commit}, \mathsf{HEC.Open}, \mathsf{HEC.Eval}^{in}, \mathsf{HEC.Eval}^{out}, \mathsf{HEC.Verify})$ with message space $\{0,1\}^{\ell}$ and randomness space $\mathbb{Z}_p^n$ (on which the distribution $\mathcal{D}_{\mathcal{R}}$ is defined to be the uniform distribution) for the function class $\mathcal{C}^{\mathbf{NC}^1}$ is described below.

$\mathsf{HEC.Setup}(1^{\kappa})$: On input the security parameter 1^{κ} in unary representation, sample the group description $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, g, e(\cdot,\cdot)) \xleftarrow{\$} \mathsf{BGGen}(1^{\kappa})$. Then sample $\alpha, \beta \leftarrow \mathbb{Z}_p^*$ and output

$$\mathsf{pp} = \mathsf{ek} = \left(g^{\alpha}, g^{\beta}\right) \quad \text{and} \quad \mathsf{msk} = \left(\alpha, \beta\right).$$

For notational simplicity, we assume the group description $\mathcal{G}$ is implicitly included in pp, ek, and msk.

$\mathsf{HEC.Commit}(\mathsf{pp}, \mathbf{x} = (x_1, \cdots, x_{\ell}); R)$: On input $\mathbf{x} \in \{0,1\}^{\ell}$ and randomness $R = \mathbf{r} \in \mathbb{Z}_p^n$, run $\mathsf{EncInp}(\mathbf{x}) = \mathbf{y} \in \{0,1\}^n$, compute

$$\mathsf{com} = g^{\mathbf{r}+\alpha\cdot\mathbf{y}},$$

and output com.

$\mathsf{HEC.Open}(\mathsf{msk}, (\mathbf{x}, R), \mathbf{x}')$: Parse $\mathsf{msk} \to \left(\alpha, \beta\right)$ and $R = \mathbf{r} \in \mathbb{Z}_p^n$, run $\mathsf{EncInp}(\mathbf{x}) = \mathbf{y} \in \{0,1\}^n$ and $\mathsf{EncInp}(\mathbf{x}') = \mathbf{y}' \in \{0,1\}^n$, compute

$$\mathbf{r}' := \mathbf{r} + \alpha \cdot (\mathbf{y} - \mathbf{y}')$$

and output $R' := \mathbf{r}'$.

$\mathsf{HEC.Eval}^{in}(\mathsf{ek}, C, \mathbf{x}, R)$: Compute $z = C(\mathbf{x}) \in \{0,1\}$ and construct the circuit $\tilde{C}_z \in \tilde{\mathcal{C}}^{\mathbf{NC}^1}$. Here, we have $\tilde{C}_z(\mathbf{x}) = 1$ by the construction. Then, run $\mathsf{EncInp}(\mathbf{x}) = \mathbf{y} \in \{0,1\}^n$ and $\mathsf{EncCir}(\tilde{C}_z) = \mathbf{M} \in \mathbb{Z}_p^{n\times m}$. By Lemma 3.7, $\mathbf{y}$ *does not* satisfy the span program $\mathbf{M}$ since $\tilde{C}_z(\mathbf{x}) = 1$. Then, parse $R = \mathbf{r} \in \mathbb{Z}_p^n$ and run

$$\vec{K} \xleftarrow{\$} \mathsf{GPSWSim}(g^{\alpha}, g^{\beta}, \mathbf{r}, \mathbf{y}, \mathbf{M})$$

using g^{α}, g^{β} included in the public parameter pp. Finally, output $\pi := \vec{K}$.

$\mathsf{HEC.Eval}^{out}(\mathsf{ek}, C, \mathsf{com})$: Output $\mathsf{com}_{\mathsf{eval}} := (C, \mathsf{com})$.

$\mathsf{HEC.Verify}(\mathsf{pp}, \mathsf{com}_{\mathsf{eval}}, z, \pi)$: Parse $\mathsf{com}_{\mathsf{eval}} \to (C, \mathsf{com})$ and $\pi = \vec{K}$, and construct the circuit $\tilde{C}_z \in \tilde{\mathcal{C}}^{\mathbf{NC}^1}$. Then, run $\mathsf{EncCir}(\tilde{C}_z) = \mathbf{M} \in \mathbb{Z}_p^{n\times m}$ and check the following condition:

$$\mathsf{GPSWKeyCheck}(\mathsf{com}, e(g^{\alpha}, g^{\beta}), \mathbf{M}, \vec{K}) = \top.$$

If it holds output $\top$, otherwise output $\bot$.

Correctness.

Theorem 3.15. Π'_{HEC} *satisfies correctness.*

Proof. For any $(\mathsf{pp}, \mathsf{ek}, \mathsf{msk}) \xleftarrow{\$} \mathsf{HEC.Setup}(1^\kappa)$, $\mathbf{x} \in \{0,1\}^\ell$, $R = \mathbf{r} \in \mathbb{Z}_p^n$, $\mathsf{com} := \mathsf{HEC.Commit}(\mathsf{pp}, \mathbf{x}; R)$, $C \in \mathcal{C}^{\mathbf{NC}^1}$, $\pi = \vec{K} \xleftarrow{\$} \mathsf{HEC.Eval}^{in}(\mathsf{pp}, C, \mathbf{x}, R)$, we have $\vec{K} \in \mathsf{GPSWKeyGen}(g^{\alpha\beta}, \mathsf{com}, \mathbf{M})$ by Lemma 3.13 where $\mathbf{M} = \mathsf{EncCir}(\tilde{C}_{C(\mathbf{x})})$. Then we have $\mathsf{GPSWKeyCheck}(\mathsf{com}, e(g^\alpha, g^\beta), \mathbf{M}, \vec{K}) = \top$ again by Lemma 3.13. Thus $\mathsf{HEC.Verify}(\mathsf{pp}, \mathsf{com}_{\mathsf{eval}} = (C, \mathsf{com}), C(\mathbf{x}), \pi)$ always returns $\top$. □

Distributional Equivalence of Open.

Theorem 3.16. Π'_{HEC} *satisfies distributional equivalence of open.*

Proof. Let $(\mathsf{pp}, \mathsf{ek}, \mathsf{msk}) \xleftarrow{\$} \mathsf{HEC.Setup}(1^\kappa)$ and $\mathbf{x}, \overline{\mathbf{x}} \in \{0,1\}^\ell$ be arbitrary (that may depend on $(\mathsf{pp}, \mathsf{msk})$). Suppose that we generate $\mathsf{com} \xleftarrow{\$} \mathsf{HEC.Commit}(\mathsf{pp}, \mathbf{x})$ where we use a uniform randomness $R := \mathbf{r} \xleftarrow{\$} \mathbb{Z}_p^n$. Then it is clear that $\mathsf{com} = g^{\mathbf{r}+\alpha\cdot\mathbf{y}}$ is uniformly distributed over $\mathbb{G}^n$ where $\mathbf{y} := \mathsf{EncInp}(\mathbf{x})$. On the other hand, if we pick $\overline{R} := \overline{\mathbf{r}} \xleftarrow{\$} \mathbb{Z}_p^n$ to generate $\mathsf{com}' := \mathsf{HEC.Commit}(\mathsf{pp}, \overline{\mathbf{x}}; \overline{R})$, then $\mathsf{com}' = g^{\overline{\mathbf{r}}+\alpha\cdot\overline{\mathbf{y}}}$ is also uniformly distributed on $\mathbb{G}^n$ where $\overline{\mathbf{y}} := \mathsf{EncInp}(\overline{\mathbf{x}})$. Moreover, if we generate $R' := \mathbf{r}' \xleftarrow{\$} \mathsf{HEC.Open}(\mathsf{msk}, (\overline{\mathbf{x}}, \overline{R}), \mathbf{x})$, then we have $\mathbf{r}' = \overline{\mathbf{r}} + \alpha \cdot (\overline{\mathbf{y}} - \mathbf{y})$ and $g^{\mathbf{r}'+\alpha\cdot\mathbf{y}} = g^{\overline{\mathbf{r}}+\alpha\cdot\overline{\mathbf{y}}} = \mathsf{com}'$. Since $\mathbf{r}' \in \mathbb{Z}_p^n$ is the unique element that satisfies this equation if we fix $(\mathsf{pp}, \mathsf{ek}, \mathsf{msk})$, $\mathbf{x}$ and com'. Thus we can conclude that we have

$$\{(\mathsf{pp}, \mathsf{ek}, \mathsf{msk}, \mathbf{x}, R, \mathsf{com})\} \overset{\text{stat}}{\approx} \{(\mathsf{pp}, \mathsf{ek}, \mathsf{msk}, \mathbf{x}, R', \mathsf{com}')\}$$

as desired. □

Context-Hiding.

Theorem 3.17. Π'_{HEC} *satisfies context-hiding.*

Proof. To show the context-hiding property, we first construct the proof simulator $\mathsf{HEC.ProofSim}$ as follows:

$\mathsf{HEC.ProofSim}(\mathsf{msk}, \mathsf{com}, C, z)$: On input a master secret key $\mathsf{msk} = (\alpha, \beta)$, a commitment $\mathsf{com} \in \mathbb{Z}_p^n$, a circuit $C \in \mathcal{C}^{\mathbf{NC}^1}$, and a message $z \in \{0,1\}$, it first constructs the circuit $\tilde{C}_z \in \tilde{\mathcal{C}}^{\mathbf{NC}^1}$ associated to circuit C. Then, it computes $\mathsf{EncCir}(\tilde{C}_z) = \mathbf{M} \in \mathbb{Z}_p^{n\times m}$. Using msk, it then computes $g^{\alpha\beta}$. Finally, it runs

$$\vec{K} \xleftarrow{\$} \mathsf{GPSWKeyGen}(g^{\alpha\beta}, \mathsf{com}, \mathbf{M}) \tag{8}$$

and outputs $\pi = \vec{K}$.

We now proceed to show that this simulator $\mathsf{HEC.ProofSim}$ satisfies the required conditions. Let us fix pp, ek, msk, $\mathbf{x}$, com, and C. By construction, the proof output by $\mathsf{HEC.Eval}^{in}(\mathsf{ek}, C, \mathbf{x}, R = \mathbf{r})$ is

$$\vec{K} \xleftarrow{\$} \mathsf{GPSWSim}(g^\alpha, g^\beta, \mathbf{r}, \mathbf{y}, \mathbf{M}). \tag{9}$$

Furthermore, by the way we convert the circuit C to $\tilde{C}_z$ where $z = C(\mathbf{x})$, $\mathbf{y} = \mathsf{EncInp}(\mathbf{x})$ does not satisfy the span program $\mathbf{M}$. Finally, we observe that the output distributions of $\mathsf{HEC.ProofSim}$ for fixed com is $\mathsf{GPSWKeyGen}(g^{\alpha\beta}, \mathsf{com}, \mathbf{M})$. Therefore, by Lemma 3.13, we have that the output distributions of $\mathsf{HEC.ProofSim}$ and $\mathsf{HEC.Eval}^{in}$ are exactly the same. This concludes the proof of the theorem. □

Computational binding for evaluated commitments.

Theorem 3.18. *If the CDH assumption holds, then* Π'_{HEC} *satisfies computational binding for evaluated commitments.*

Proof. To prove the theorem, it suffices to show that there exists a PPT algorithm $\mathcal{B}$ that solves the CDH problem with non-negligible probability assuming a PPT adversary $\mathcal{A}$ against the computational binding for evaluated commitments of Π'_{HEC} with non-negligible advantage ϵ. We give the description of $\mathcal{B}$ in the following.

The reduction algorithm $\mathcal{B}$ is given $(1^\kappa, \mathcal{G}, g^\alpha, g^\beta)$ as the problem instance of the CDH problem. Then, $\mathcal{B}$ runs $\mathcal{A}$ on input $\mathsf{pp} := (g^\alpha, g^\beta)$ to obtain $(\mathbf{x}, R = \mathbf{r}, C, z^*, \pi^* = \vec{K}^*)$. $\mathcal{B}$ aborts and outputs $\perp$ if $\mathcal{A}$ did not win the game. Otherwise $\mathcal{B}$ constructs the circuit $\tilde{C}_{z^*} \in \tilde{\mathcal{C}}^{\mathbf{NC}^1}$, and computes $\mathbf{M}^* \xleftarrow{\$} \mathsf{EncCir}(\tilde{C}_{z^*})$ and $\mathbf{y} := \mathsf{EncInp}(\mathbf{x})$. We note that we have $\mathsf{GPSWKeyCheck}(\mathsf{com}, e(g^\alpha, g^\beta), \mathbf{M}^*, \vec{K}^*) = \top$ where $\mathsf{com} = g^{\mathbf{r}+\alpha\cdot\mathbf{y}}$ since $\mathcal{A}$ wins the game. By Lemma 3.13, this implies that there exist $\mathbf{s}^* \in \mathbb{Z}_p^{m-1}$ and $\mathbf{t}^* \in \mathbb{Z}_p^n$ such that

$$\vec{K}^* = \left(\vec{K}_0^* = g^{\mathbf{M}^* \binom{\gamma}{\mathbf{s}^*} + \mathbf{t}^* \odot \mathbf{u}},\ \vec{K}_1^* = g^{\mathbf{t}^*} \right) \in \mathbb{G}^n \times \mathbb{G}^n,$$

where $\gamma = \alpha\beta$ and $\mathbf{u} = \mathbf{r} + \alpha \mathbf{y}$. Then we can rewrite

$$\vec{K}_0^* = g^{\mathbf{M}^* \binom{\gamma}{\mathbf{s}^*} + \mathbf{t}^* \odot \mathbf{u}} = g^{\mathbf{M}^* \binom{\gamma}{\mathbf{s}^*} + \mathbf{t}^* \odot \mathbf{r} + \alpha \cdot \mathbf{t}^* \odot \mathbf{y}}.$$

Since $\mathcal{B}$ knows $\mathbf{r}$, $\mathcal{B}$ can compute $g^{\mathbf{t}^* \odot \mathbf{r}}$ using $\vec{K}_1^* = g^{\mathbf{t}^*}$. Therefore, from $\vec{K}$, $\mathcal{B}$ can compute

$$g^{\mathbf{M}^* \binom{\gamma}{\mathbf{s}^*} + \alpha \cdot \mathbf{t}^* \odot \mathbf{y}}. \tag{10}$$

By denoting $\mathbf{M}_I^*$ as the matrix obtained by removing the j-th row of $\mathbf{M}^*$ for j such that the j-th coefficient of $\mathbf{y}$ is equal to 1, we can extract the following from Eq. (10):

$$g^{\mathbf{M}_I^* \binom{\gamma}{\mathbf{s}^*}}.$$

Next, since $\mathcal{A}$ succeeds in breaking the binding, we must have $C(\mathbf{x}) \neq z^*$. Specifically, we have $\tilde{C}_{z^*}(\mathbf{x}) = 0$. By Lemma 3.7, this implies that $\mathbf{y}$ satisfies the span program $\mathbf{M}^*$. Hence, by Definition 3.5, there exists an efficiently computable row vector $\mathbf{w}^*$ such that $\mathbf{w}^* \mathbf{M}_I^* = \mathbf{1}$. Therefore, $\mathcal{B}$ can retrieve g^γ by computing

$$g^{\mathbf{w}^* \mathbf{M}_I^* \binom{\gamma}{\mathbf{s}^*}} = g^{\mathbf{1} \binom{\gamma}{\mathbf{s}^*}} = g^\gamma.$$

Finally, $\mathcal{B}$ outputs g^γ as its answer to the CDH problem.

It can be seen that the simulation by $\mathcal{B}$ is perfect. Furthermore, as we have seen, $\mathcal{B}$ can extract the solution to the CDH problem whenever $\mathcal{A}$ wins the game. This concludes the proof of the theorem. □

4 Compact CRS-NIZK from HEC

Here, we give a construction of a compact CRS-NIZK scheme based on any non-compact CRS-NIZK scheme and HEC with efficient verification. If we instantiate the construction with the HEC given in Section 3.2, then the proof size of the resulting CRS-NIZK scheme is $|C| + \mathsf{poly}(\kappa)$. Moreover, if the relation supported by the scheme is verifiable in $\mathbf{NC}^1$, then the proof size is $|w| + \mathsf{poly}(\kappa)$.

4.1 Extractable CRS-NIZK

First, we define extractability for CRS-NIZK, which is needed for our construction of compact CRS-NIZK scheme. We note that the extractability defined here is a mild property, and we can convert any CRS-NIZK scheme to the one with extractability if we additionally assume the existence of PKE as shown in Lemma 4.1.

An extractable CRS-NIZK is a CRS-NIZK with an additional deterministic algorithm Extract which takes as input a randomness r_{Setup} used in Setup and a proof π, and outputs a witness w that satisfies the following.

Extractability. For all PPT adversary $\mathcal{A}$, we have

$$\Pr\left[\begin{array}{r|r} \mathsf{Verify}(\mathsf{crs}, x, \pi) = \top & \mathsf{crs} \xleftarrow{\$} \mathsf{Setup}(1^\kappa) \\ (x, w) \notin \mathcal{R} & (x, \pi) \xleftarrow{\$} \mathcal{A}(\mathsf{crs}) \\ & w \xleftarrow{\$} \mathsf{Extract}(r_{\mathsf{Setup}}, \pi) \end{array}\right] \leq \mathsf{negl}(\kappa).$$

where r_{Setup} is the randomness used in Setup to generate crs.

The following lemma is easy to prove.

Lemma 4.1. *If there exist CRS-NIZK for all of* ***NP*** *and a CPA-secure PKE scheme, then there exists CRS-NIZK for all of* ***NP*** *with extractability.*

Proof. Let $\mathcal{R}$ be any relation, $\Pi_{\mathsf{PKE}} = (\mathsf{PKE.KeyGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ be a CPA secure PKE scheme, and $\Pi_{\mathsf{CRSNIZK}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ be CRS-NIZK for the language $\widetilde{\mathcal{L}}$ corresponding to the relation $\widetilde{\mathcal{R}}$ defined below:

$$((x, \mathsf{pk}, \mathsf{ct}), (w, r_{\mathsf{Enc}})) \in \widetilde{\mathcal{R}} \iff (x, w) \in \mathcal{R} \wedge \mathsf{ct} = \mathsf{PKE.Enc}(\mathsf{pk}, w; r_{\mathsf{Enc}}).$$

Then we construct an extractable CRS-NIZK $\Pi'_{\mathsf{CRSNIZK}} = (\mathsf{Setup}', \mathsf{Prove}', \mathsf{Verify}')$ as follows.

$\mathsf{Setup}'(1^\kappa; r_{\mathsf{Setup}'} = (r_{\mathsf{Setup}}, r_{\mathsf{KeyGen}}))$**:** This algorithm generates $\mathsf{crs} := \mathsf{Setup}(1^\kappa; r_{\mathsf{Setup}})$ and $(\mathsf{pk}, \mathsf{sk}) := \mathsf{PKE.KeyGen}(1^\kappa; r_{\mathsf{KeyGen}})$, and outputs a common reference string $\mathsf{crs}' := (\mathsf{crs}, \mathsf{pk})$.

$\mathsf{Prove}'(\mathsf{crs}', x, w)$: This algorithm aborts if $(x, w) \notin \mathcal{R}$. Otherwise it parses $(\mathsf{crs}, \mathsf{pk}) \leftarrow \mathsf{crs}'$, picks r_{Enc} uniformly from the randomness space of Π_{PKE}, generates $\mathsf{ct} := \mathsf{PKE.Enc}(\mathsf{pk}, w; r_{\mathsf{Enc}})$ and $\pi \xleftarrow{\$} \mathsf{Prove}((x, \mathsf{pk}, \mathsf{ct}), (w, r_{\mathsf{Enc}}))$, and outputs $\pi' := (\pi, \mathsf{ct})$.

$\mathsf{Verify}'(\mathsf{crs}', x, \pi)$**:** This algorithm parses $(\mathsf{crs}, \mathsf{pk}) \leftarrow \mathsf{crs}'$ and $(\pi, \mathsf{ct}) \leftarrow \pi'$, and outputs $\top$ if $\mathsf{Prove}((x, \mathsf{pk}, \mathsf{ct}), \pi) = \top$ and outputs $\perp$ otherwise.

The correctness of Π'_{CRSNIZK} easily follows from correctness of Π_{CRSNIZK} and Π_{PKE}. We describe the extractor Extract as follows:

$\mathsf{Extract}(r'_{\mathsf{Setup}} = (r_{\mathsf{Setup}}, r_{\mathsf{KeyGen}}), \pi' = (\pi, \mathsf{ct}))$**:** It generates $(\mathsf{pk}, \mathsf{sk}) := \mathsf{PKE.KeyGen}(1^\kappa; r_{\mathsf{KeyGen}})$ and outputs $w := \mathsf{PKE.Dec}(\mathsf{sk}, \mathsf{ct})$.

We prove that this extractor works correctly. Suppose that we have $\mathsf{Verify}'(\mathsf{crs}, x, \pi') = \top$ and $(x, w) \notin \mathcal{R}$ where $\mathsf{crs}' = (\mathsf{crs}, \mathsf{pk}) \xleftarrow{\$} \mathsf{Setup}(1^\kappa)$, $(x, \pi' = (\pi, \mathsf{ct})) \xleftarrow{\$} \mathcal{A}(\mathsf{crs})$ and $w \xleftarrow{\$} \mathsf{Extract}(r_{\mathsf{Setup}}, \pi)$. In this case, ct is not a valid encryption of any w' such that $(x, w') \in \mathcal{R}$ under the public key pk, and thus $(x, \mathsf{pk}, \mathsf{ct}) \notin \widetilde{\mathcal{R}}$. This happens with negligible probability by the soundness of Π_{CRSNIZK}. Thus Π'_{CRSNIZK} satisfies the extractability. It is easy to see that the extractability implies computational soundness.

Finally, we prove that Π'_{CRSNIZK} satisfies zero-knowledge. Let $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ be a simulator for zero-knowledge property of Π_{CRSNIZK}. Then we construct a simulator $\mathcal{S}' = (\mathcal{S}'_1, \mathcal{S}'_2)$ for Π'_{CRSNIZK} as follows.

$\mathcal{S}'_1(1^\kappa)$**:** It generates $(\mathsf{crs}, \tau) \xleftarrow{\$} \mathcal{S}_1(1^\kappa)$ and $(\mathsf{pk}, \mathsf{sk}) := \mathsf{PKE.KeyGen}(1^\kappa)$ and outputs $\mathsf{crs}' = (\mathsf{crs}, \mathsf{pk})$ and $\tau' = \tau$.

$\mathcal{S}'_2(\mathsf{crs}' = (\mathsf{crs}, \mathsf{pk}), \tau' = \tau, x)$**:** It computes $\mathsf{ct} \xleftarrow{\$} \mathsf{PKE.Enc}(\mathsf{pk}, 0^{|w|})$ and $\pi \xleftarrow{\$} \mathcal{S}_2(\mathsf{crs}, \tau, (x, \mathsf{pk}, \mathsf{ct}))$, and outputs $\pi' = (\pi, \mathsf{ct})$.

It is easy to prove that proofs generated by the above simulator is computationally indistinguishable if Π_{CRSNIZK} satisfies zero-knowledge w.r.t. $\mathcal{S}$ and Π_{PKE} is CPA-secure. $\square$

4.2 Construction of Compact CRS-NIZK

Before describing the construction, we prepare some building blocks and notations.

- Let $\mathcal{L}$ be an **NP** language defined by a relation $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$. Let $n(\kappa)$ and $m(\kappa)$ be any fixed polynomials. Let C be a circuit that computes the relation $\mathcal{R}$ on $\{0,1\}^n \times \{0,1\}^m$, i.e., for $(x,w) \in \{0,1\}^n \times \{0,1\}^m$, we have $C(x,w) = 1$ if and only if $(x,w) \in \mathcal{R}$.
- Let $\Pi_{\mathsf{SKE}} = (\mathsf{SKE.KeyGen}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ be a symmetric key encryption (SKE) scheme with ciphertext space $\mathcal{CT}$ and key space $\{0,1\}^\ell$.

In the following, for $x \in \{0,1\}^n$ and $\mathsf{ct} \in \mathcal{CT}$, we define the function

$$f_{x,\mathsf{ct}}(K) := C(x, \mathsf{SKE.Dec}(K, \mathsf{ct})).$$

- Let $\Pi_{\mathsf{HEC}} = (\mathsf{HEC.Setup}, \mathsf{HEC.Commit}, \mathsf{HEC.Open}, \mathsf{HEC.Eval}^{in}, \mathsf{HEC.Eval}^{out}, \mathsf{HEC.Verify})$ be a HEC scheme with the message space that contains $\{0,1\}^\ell$ and randomness space $\mathcal{R}$ on which a distribution $\mathcal{D}_{\mathcal{R}}$ is defined. We need the HEC scheme to support a function class containing $\{f_{x,\mathsf{ct}}\}_{x \in \{0,1\}^n, \mathsf{ct} \in \mathcal{CT}}$.
- Let $\Pi_{\mathsf{CRSNIZK}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ be an extractable CRS-NIZK for the language corresponding to the relation $\widetilde{\mathcal{R}}$ defined below:
  $((\mathsf{pp}, \mathsf{com}, \mathsf{com}_{\mathsf{eval}}), (K, R, \pi_{\mathsf{HEC}})) \in \widetilde{\mathcal{R}}$ if and only if the followings are satisfied:
  1. $K \in \{0,1\}^\ell$,
  2. $\mathsf{HEC.Commit}(\mathsf{pp}, K; R) = \mathsf{com}$,
  3. $\mathsf{HEC.Verify}(\mathsf{pp}, \mathsf{com}_{\mathsf{eval}}, 1, \pi_{\mathsf{HEC}}) = \top$.

  We note that extractable CRS-NIZK for all of **NP** exists assuming (non-extractable) CRS-NIZK for all of **NP** and CPA secure PKE as shown in Lemma 4.1.

The CRS-NIZK $\Pi'_{\mathsf{CRSNIZK}} = (\mathsf{Setup}', \mathsf{Prove}', \mathsf{Verify}')$ for $\mathcal{L}$ is described as follows.

$\mathsf{Setup}'(1^\kappa)$**:** This algorithm generates $\mathsf{crs} \xleftarrow{\$} \mathsf{Setup}(1^\kappa)$ and $(\mathsf{pp}, \mathsf{ek}, \mathsf{msk}), \xleftarrow{\$} \mathsf{HEC.Setup}(1^\kappa)$. It outputs a common reference string $\mathsf{crs}' = (\mathsf{crs}, \mathsf{pp}, \mathsf{ek})$.

$\mathsf{Prove}'(\mathsf{crs}', x, w)$**:** This algorithm aborts if $\mathcal{R}(x,w) = 0$. Otherwise it parses $(\mathsf{crs}, \mathsf{pp}, \mathsf{ek}) \leftarrow \mathsf{crs}'$, picks $K \xleftarrow{\$} \mathsf{SKE.KeyGen}(1^\kappa)$ and $R \xleftarrow{\$} \mathcal{D}_{\mathcal{R}}$, computes $\mathsf{ct} \xleftarrow{\$} \mathsf{SKE.Enc}(K, w)$, generates $\mathsf{com} := \mathsf{HEC.Commit}(\mathsf{pp}, K; R)$, $\pi_{\mathsf{HEC}} \xleftarrow{\$} \mathsf{HEC.Eval}^{in}(\mathsf{ek}, f_{x,\mathsf{ct}}, K, R)$, $\mathsf{com}_{\mathsf{eval}} := \mathsf{HEC.Eval}^{out}(\mathsf{ek}, f_{x,\mathsf{ct}}, \mathsf{com})$, and $\pi_{\mathsf{NIZK}} \xleftarrow{\$} \mathsf{Prove}(\mathsf{crs}, (\mathsf{pp}, \mathsf{com}, \mathsf{com}_{\mathsf{eval}}), (K, R, \pi_{\mathsf{HEC}}))$, and outputs a proof $\pi' := (\mathsf{ct}, \mathsf{com}, \pi_{\mathsf{NIZK}})$.

$\mathsf{Verify}'(\mathsf{crs}', x, \pi')$**:** This algorithm parses $(\mathsf{crs}, \mathsf{pp}, \mathsf{ek}) \leftarrow \mathsf{crs}'$ and $(\mathsf{ct}, \mathsf{com}, \pi_{\mathsf{NIZK}}) \leftarrow \pi'$, computes $\mathsf{com}_{\mathsf{eval}} := \mathsf{HEC.Eval}^{out}(\mathsf{ek}, f_{x,\mathsf{ct}}, \mathsf{com})$, and outputs $\top$ if $\mathsf{Verify}(\mathsf{crs}, (\mathsf{pp}, \mathsf{com}, \mathsf{com}_{\mathsf{eval}}), \pi_{\mathsf{NIZK}}) = \top$, and outputs $\bot$ otherwise.

Correctness. Suppose that $(\mathsf{ct}, \mathsf{com}, \pi_{\mathsf{NIZK}})$ is an honestly generated proof on $(x,w) \in \mathcal{R}$. Then we have $\mathsf{ct} \xleftarrow{\$} \mathsf{SKE.Enc}(K, w)$ and $\mathsf{com} = \mathsf{HEC.Commit}(\mathsf{pp}, K; R)$ with some K and R. By the correctness of Π_{SKE}, we have $f_{x,\mathsf{ct}}(K) = 1$, and by the correctness of Π_{HEC}, we have $\mathsf{HEC.Verify}(\mathsf{pp}, \mathsf{com}_{\mathsf{eval}}, 1, \pi_{\mathsf{HEC}}) = \top$ where we generate $\mathsf{com}_{\mathsf{eval}} := \mathsf{HEC.Eval}^{out}(\mathsf{ek}, f_{x,\mathsf{ct}}, \mathsf{com})$ and $\pi_{\mathsf{HEC}} \xleftarrow{\$} \mathsf{HEC.Eval}^{in}(\mathsf{ek}, f_{x,\mathsf{ct}}, K, R)$. Since we have $((\mathsf{pp}, \mathsf{com}, \mathsf{com}_{\mathsf{eval}}), (K, R, \pi_{\mathsf{HEC}})) \in \widetilde{\mathcal{R}}$, if we generate $\pi_{\mathsf{NIZK}} \xleftarrow{\$} \mathsf{Prove}(\mathsf{crs}, (\mathsf{pp}, \mathsf{com}, \mathsf{com}_{\mathsf{eval}}), (K, R, \pi_{\mathsf{HEC}}))$, then we have $\mathsf{Verify}(\mathsf{crs}, (\mathsf{pp}, \mathsf{com}, \mathsf{com}_{\mathsf{eval}}), \pi_{\mathsf{NIZK}}) = \top$ by the correctness of Π_{CRSNIZK}.

Security. The security of NIZK' is stated as follows.

Theorem 4.2 (Soundness.). *If* Π_{CRSNIZK} *satisfies extractability and* HEC *satisfies computational binding for evaluated commitment, then* Π'_{CRSNIZK} *satisfies computational soundness.*

Proof. Suppose that there is a PPT adversary $\mathcal{A}$ that breaks soundness. Then we construct a PPT adversary $\mathcal{B}$ that breaks the computational binding for evaluated commitment of HEC as follows.

$\mathcal{B}(\mathsf{pp}, \mathsf{ek})$**:** It generates $\mathsf{crs} \xleftarrow{\$} \mathsf{Setup}(1^\kappa; r_{\mathsf{Setup}})$, runs $\mathcal{A}(\mathsf{crs}')$ to obtain $(x^*, \pi'^* = (\mathsf{ct}, \mathsf{com}, \pi_{\mathsf{NIZK}}))$ where $\mathsf{crs}' := (\mathsf{crs}, \mathsf{pp}, \mathsf{ek})$. Then it computes $\mathsf{com}_{\mathsf{eval}} := \mathsf{HEC.Eval}^{out}(\mathsf{ek}, f_{x^*,\mathsf{ct}}, \mathsf{com})$, and $(K, R, \pi_{\mathsf{HEC}}) \xleftarrow{\$} \mathsf{Extract}(r_{\mathsf{Setup}}, \pi_{\mathsf{NIZK}})$, and outputs $(K, R, f_{x^*,\mathsf{ct}}, \mathsf{com}, 1, \pi_{\mathsf{HEC}})$.

This completes the description of $\mathcal{B}$. In the following, we show that $\mathcal{B}$ breaks the computational binding for evaluated commitments of HEC. Since we assume $\mathcal{A}$ breaks the soundness of Π'_{CRSNIZK},

$$\Pr[x^* \notin \mathcal{L} \wedge \mathsf{Verify}(\mathsf{crs}, (\mathsf{pp}, \mathsf{com}, \mathsf{com}_{\mathsf{eval}}), \pi_{\mathsf{NIZK}}) = \top]$$

is non-negligible. On the other hand, by the extractability of Π_{CRSNIZK},

$$\Pr[\mathsf{Verify}(\mathsf{crs}, (\mathsf{pp}, \mathsf{com}, \mathsf{com}_{\mathsf{eval}}), \pi_{\mathsf{NIZK}}) = \top \wedge ((\mathsf{pp}, \mathsf{com}, \mathsf{com}_{\mathsf{eval}}), (K, R, \pi_{\mathsf{HEC}})) \notin \widetilde{\mathcal{R}}]$$

is negligible. Therefore

$$\Pr[x^* \notin \mathcal{L} \wedge \mathsf{Verify}(\mathsf{crs}, (\mathsf{pp}, \mathsf{com}, \mathsf{com}_{\mathsf{eval}}), \pi_{\mathsf{NIZK}}) = \top \wedge ((\mathsf{pp}, \mathsf{com}, \mathsf{com}_{\mathsf{eval}}), (K, R, \pi_{\mathsf{HEC}})) \in \widetilde{\mathcal{R}}]$$

is non-negligible. Suppose that this event happens. Since we have $x^* \notin \mathcal{L}$, we have $f_{x^*,\mathsf{ct}}(K') = 0$ for all $K' \in \{0,1\}^\ell$. On the other hand, $((\mathsf{pp}, \mathsf{com}, \mathsf{com}_{\mathsf{eval}}), (K, R, \pi_{\mathsf{HEC}})) \in \widetilde{\mathcal{R}}$ implies $K \in \{0,1\}^\ell \wedge \mathsf{com} := \mathsf{HEC.Commit}(\mathsf{pp}, K; R) \wedge \mathsf{HEC.Verify}(\mathsf{pp}, \mathsf{com}_{\mathsf{eval}}, 1, \pi_{\mathsf{HEC}}) = \top$. This means that $\mathcal{B}$ succeeds in breaking the computational binding for evaluated commitment of HEC. □

Theorem 4.3 (Zero-knowledge.). *If* Π_{CRSNIZK} *satisfies zero-knowledge,* HEC *is computationally hiding,*[12] *and* SKE *is CPA secure, then* Π'_{CRSNIZK} *satisfies zero-knowledge.*

Proof. Let $(\mathcal{S}_1, \mathcal{S}_2)$ be the simulator for Π_{CRSNIZK}. We describe the simulator $(\mathcal{S}_1, \mathcal{S}_2)$ for Π'_{CRSNIZK} below.

$\mathcal{S}'_1(1^\kappa)$**:** It generates $(\mathsf{crs}, \tau) \xleftarrow{\$} \mathcal{S}_1(1^\kappa)$ and $(\mathsf{pp}, \mathsf{msk}) \xleftarrow{\$} \mathsf{HEC.Setup}(1^\kappa)$, and outputs $\mathsf{crs}' := (\mathsf{crs}, \mathsf{pp})$ and $\tau' := \tau$.

$\mathcal{S}'_2(\mathsf{crs}' = (\mathsf{crs}, \mathsf{pp}), \tau' = \tau, x)$**:** It picks $K \xleftarrow{\$} \mathsf{SKE.KeyGen}(1^\kappa)$, computes $\mathsf{ct} \xleftarrow{\$} \mathsf{SKE.Enc}(K, 0^m)$, $\mathsf{com} \xleftarrow{\$} \mathsf{HEC.Commit}(\mathsf{pp}, 0^\ell)$, $\mathsf{com}_{\mathsf{eval}} \xleftarrow{\$} \mathsf{HEC.Eval}^{out}(\mathsf{pp}, f_{x,\mathsf{ct}}, \mathsf{com})$, and $\pi_{\mathsf{NIZK}} \xleftarrow{\$} \mathcal{S}_2(\mathsf{crs}, \tau, (\mathsf{pp}, \mathsf{com}, \mathsf{com}_{\mathsf{eval}}))$, and outputs $\pi' := (\mathsf{ct}, \mathsf{com}, \pi_{\mathsf{NIZK}})$.

This completes the description of the simulator. We prove that proofs simulated by the above simulator are computationally indistinguishable from the honestly generated proofs. To prove this, we consider the following sequence of games between a PPT adversary $\mathcal{A}$ and a challenger.

Game_0**:** In this game, proofs are generated honestly. Namely,

1. The challenger generates $\mathsf{crs} \xleftarrow{\$} \mathsf{Setup}(1^\kappa)$ and $(\mathsf{pp}, \mathsf{ek}, \mathsf{msk}), \xleftarrow{\$} \mathsf{HEC.Setup}(1^\kappa)$, and gives $\mathsf{crs}' := (\mathsf{crs}, \mathsf{pp}, \mathsf{ek})$ to $\mathcal{A}$.
2. $\mathcal{A}$ is given $(1^\kappa, \mathsf{crs}')$, and allowed to query $\mathcal{O}(\mathsf{crs}, \cdot, \cdot)$, which works as follows. When $\mathcal{A}$ queries (x, w), if $(x, w) \notin \mathcal{R}$, then the oracle returns $\perp$. Otherwise, it picks $K \xleftarrow{\$} \mathsf{SKE.KeyGen}(1^\kappa)$ and $R \xleftarrow{\$} \mathcal{D}_{\mathcal{R}}$, computes $\mathsf{ct} := \mathsf{SKE.Enc}(K, w)$, generates $\mathsf{com} := \mathsf{HEC.Commit}(\mathsf{pp}, K; R)$, $\pi_{\mathsf{HEC}} \xleftarrow{\$} \mathsf{HEC.Eval}^{in}(\mathsf{ek}, f_{x,\mathsf{ct}}, K, R)$, $\mathsf{com}_{\mathsf{eval}} \xleftarrow{\$} \mathsf{HEC.Eval}^{out}(\mathsf{ek}, f_{x,\mathsf{ct}}, \mathsf{com})$, and $\pi_{\mathsf{NIZK}} \xleftarrow{\$} \mathsf{Prove}(\mathsf{crs}, (\mathsf{pp}, \mathsf{com}, \mathsf{com}_{\mathsf{eval}}), (K, R, \pi_{\mathsf{HEC}}))$, and returns a proof $\pi' := (\mathsf{ct}, \mathsf{com}, \pi_{\mathsf{NIZK}})$.
3. Finally, $\mathcal{A}$ returns a bit β.

Game_1**:** This game is identical to the previous game except that crs and π_{NIZK} are generated differently. Namely, the challenger generates $(\mathsf{crs}, \tau) \xleftarrow{\$} \mathcal{S}_1(1^\kappa)$ at the beginning of the game, and π_{NIZK} is generated as $\pi_{\mathsf{NIZK}} \xleftarrow{\$} \mathcal{S}_2(\mathsf{crs}, \tau, (\mathsf{pp}, \mathsf{com}, \mathsf{com}_{\mathsf{eval}}))$ for each oracle query.

[12]Recall that the computational hiding (or even statistical soundness) follows from the distributional equivalence of open.

Game_2: This game is identical to the previous game except that com is generated as $\mathsf{com} \xleftarrow{\$} \mathsf{HEC.Commit}(\mathsf{pp}, 0^{\ell})$ for each oracle query.

Game_3: This game is identical to the previous game except that ct is generated as $\mathsf{ct} \xleftarrow{\$} \mathsf{SKE.Enc}(K, 0^m)$.

Let T_i be the event that $\mathcal{A}$ returns 1 in Game_i for $i = 0, 1, 2, 3$. It is easy to see that proofs are generated by $\mathcal{S}' = (\mathcal{S}'_1, \mathcal{S}'_2)$ in Game_3. Thus we have to prove that $|\Pr[\mathsf{T}_0] - \Pr[\mathsf{T}_3]|$ is negligible. The following lemmas are straightforward to prove.

Lemma 4.4. *If Π_{CRSNIZK} satisfies the computational zero-knowledge w.r.t. the simulator $\mathcal{S}$, then* $|\Pr[\mathsf{T}_0] - \Pr[\mathsf{T}_1]| = \mathsf{negl}(\kappa)$.

Proof. We observe that every proof π_{NIZK} given to $\mathcal{A}$ is created for a correct statement in both games. Therefore, the indistinguishability of the games can be reduced to the zero-knowledge property of Π_{CRSNIZK}. □

Lemma 4.5. *If* HEC *satisfies the computational hiding property, then* $|\Pr[\mathsf{T}_1] - \Pr[\mathsf{T}_2]| = \mathsf{negl}(\kappa)$.

Proof. Because of the change we introduced in Game_1, the randomness R used to generate com is not used anywhere else in both games. Therefore, the indistinguishability of these games can be reduced to the hiding property of HEC. □

Lemma 4.6. *If* SKE *is CPA-secure, then* $|\Pr[\mathsf{T}_2] - \Pr[\mathsf{T}_3]| = \mathsf{negl}(\kappa)$.

Proof. Because of the change we introduced in Game_2, the secret key K of SKE that is used to generate ct is not used anywhere else in both games. therefore, the indistinguishability of these games can be reduced to the CPA security of SKE. □

This completes the proof of Theorem 4.3. □

4.3 Instantiations

Here, we discuss that by appropriately instantiating Π_{CRSNIZK}, we can achieve compact proof size. In particular, we consider instantiating the HEC scheme with our construction in Section 3.2. Since our HEC scheme only supports $\mathbf{NC}^1$ circuits, we have to ensure that $f_{x,\mathsf{ct}}$ is computable in $\mathbf{NC}^1$. For ensuring this, we use the fact that any efficiently verifiable relation can be verified in $\mathbf{NC}^1$ at the cost of making the witness size as large as the size of a circuit that verifies the relation (e.g., [GGH+16]). This is done by considering all values corresponding to all gates when computing the circuit on input (x, w) to be the new witness. In addition, we use an SKE scheme whose decryption circuit is in $\mathbf{NC}^1$ with additive ciphertext overhead (i.e., the ciphertext length is the message length plus $\mathsf{poly}(\kappa)$) and the key size $\ell = \kappa$, which exists under the CDH assumption [KNYY19]. Then $f_{x,\mathsf{ct}}$ is computable in $\mathbf{NC}^1$ for every x and ct. In this case, we have that $|\mathsf{ct}| \leq |C| + \mathsf{poly}(\kappa)$. In order to bound the length of the proof π', we also bound $|\mathsf{com}|$ and $|\pi_{\mathsf{NIZK}}|$. By the efficient committing property of HEC, $|\mathsf{com}|$ and the size of the circuit computing HEC.Commit is bounded by $|K| \cdot \mathsf{poly}(\kappa) \leq \mathsf{poly}(\kappa)$. Furthermore, by the efficient verification property of HEC, the size of the circuit computing HEC.Verify is bounded by $\mathsf{poly}(\kappa)$. Therefore, the size of the circuit computing $\widetilde{\mathcal{R}}$ is bounded by $\mathsf{poly}(\kappa)$, which implies that $|\pi_{\mathsf{NIZK}}|$ is bounded by $\mathsf{poly}(\kappa)$ as well (even if Π_{CRSNIZK} is non-compact). To sum up, we have that the proof size of Π_{CRSNIZK} is $|C| + \mathsf{poly}(\kappa)$. Moreover, if we only consider a relation computable in $\mathbf{NC}^1$ in the first place, then we need not expand the witness, and the proof size can be further reduced to be $|w| + \mathsf{poly}(\kappa)$. Finally, we remark that (non-compact) CRS-NIZK for all of **NP** exists under the CDH assumption on a pairing group [CHK07, Abu13], which in particular holds under the CDHER assumption. In summary, we obtain the following corollary.

Corollary 4.7. *If the CDHER assumption holds on a pairing group, then there exists CRS-NIZK for all of* **NP** *with proof size* $|C| + \mathsf{poly}(\kappa)$. *Moreover, if the corresponding relation is computable in* $\mathbf{NC}^1$*, then the proof size is* $|w| + \mathsf{poly}(\kappa)$.

Variant with Sublinear Proof Size. Katsumata et al. [KNYY19] showed that their DP-NIZK achieves sublinear proof size i.e., $|w| + |C|/\log\kappa + \mathsf{poly}(\kappa)$ if C is a leveled circuit [BGI16] whose gates are divided into L levels, and all incoming wires to a gate of level $i + 1$ come from gates of level i. Exactly the same idea can be applied to our CRS-NIZK to achieve sublinear proof size. More detailed explanation can be found in Appendix C. Namely, we obtain the following corollary:

Corollary 4.8. *If the CDHER assumption holds on a pairing group, then there exists CRS-NIZK for all* ***NP*** *languages whose corresponding relation is computable by a leveled circuit with proof size* $|w| + |C|/\log\kappa + \mathsf{poly}(\kappa)$.

5 Compact UC-NIZK for NC^1 from HEC

5.1 UC Framework

Here, we briefly recall the UC framework. We refer to [Can01] for the full descriptions. The following description is partially based on [KW18b, Appendix. A]. Readers familiar with the UC framework can safely skip this section.

The UC framework. The UC-security is formalized by indistinguishability of real and ideal worlds. In the real world, parties $P_1, ..., P_N$ execute a protocol Π, and an adversary $\mathcal{A}$ may corrupt some of them. We say that $\mathcal{A}$ is adaptive if it adaptively decides which party to corrupt. In the ideal world, dummy parties $\tilde{P}_1, ..., \tilde{P}_N$ execute an ideal functionality $\mathcal{F}$, and a simulator $\mathcal{S}$ may corrupt some of them. In addition to (dummy) parties and an adversary/simulator, we consider another entity $\mathcal{Z}$ which tries to distinguish these two worlds. In the real (resp. ideal) world, an environment $\mathcal{Z}$, which takes the security parameter 1^κ and an auxiliary input z as input, can arbitrarily interact with $\mathcal{A}$ (resp. $\mathcal{S}$), and it can also feed any input to any uncorrupted party (resp. dummy party) to let it honestly run the protocol Π (resp. the ideal functionality) on the input and report the output to $\mathcal{Z}$. Finally, $\mathcal{Z}$ outputs a bit as its guess of in which world it is. We denote the output distribution of $\mathcal{Z}$ with auxiliary input z in the real world by $\mathtt{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(1^\kappa, z)$, and denote the ensemble $\{\mathtt{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(1^\kappa, z)\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}$ by $\mathtt{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}$ Similarly, we denote the output distribution of $\mathcal{Z}$ with auxiliary input z in the ideal world by $\mathtt{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^\kappa, z)$, and denote the ensemble $\{\mathtt{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^\kappa, z)\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}$ by $\mathtt{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$.

Definition 5.1. *Let* $\mathcal{X} = \{X(1^\kappa, z)\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}$ *and* $\mathcal{Y} = \{Y(1^\kappa, z)\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}$ *be two distribution ensembles over* $\{0,1\}$. *We say that* $\mathcal{X}$ *and* $\mathcal{Y}$ *are indistinguishable (denoted by* $\mathcal{X} \approx \mathcal{Y}$*) if for any* $c, d \in \mathbb{N}$*, there exists* $\kappa_0 \in \mathbb{N}$ *such that we have* $|\Pr[X(1^\kappa, z) = 1] - \Pr[Y(1^\kappa, z) = 1]| < \kappa^{-c}$ *for all* $\kappa > \kappa_0$ *and all* $z \in \{0,1\}^{\le\kappa^d}$.

Definition 5.2. *We say that* Π *UC-realizes* $\mathcal{F}$ *if for all PPT adversaries* $\mathcal{A}$*, there exists a PPT simulator* $\mathcal{S}$ *such that for all PPT environment* $\mathcal{Z}$[13]*, we have* $\mathtt{REAL}_{\Pi,\mathcal{A},\mathcal{Z}} \approx \mathtt{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$.

Hybrid models. We often construct a protocol in a setting where (multiple copies of) an ideal functionality $\mathcal{F}$ is available for each party. We call such a model $\mathcal{F}$-hybrid model. Definition 5.2 can be extended to define the notion of a protocol Π securely realizing a functionality $\mathcal{G}$ in the $\mathcal{F}$-hybrid model.

Compositions and universal composition theorem. For a protocol ρ in the $\mathcal{F}$-hybrid model, and a protocol Π that realizes $\mathcal{F}$ (in the standard model), we can naturally define a composed protocol ρ^Π in the standard model, in which calls for $\mathcal{F}$ by ρ are responded by Π instead of $\mathcal{F}$. Canetti [Can01] proved the following *universal composition theorem*.

Theorem 5.3 ([Can01]). *If* ρ *UC-realizes* $\mathcal{G}$ *in the* $\mathcal{F}$*-hybrid model and* Π *UC-realizes* $\mathcal{F}$*, then* ρ^Π *UC-realizes* $\mathcal{G}$.

Remark 5.4 (Static/Adaptive Security and Erasure). In some works of the UC-security, one considers a weaker security called the *static* security, which only considers adversaries that declare which party to corrupt at the beginning of the experiment. Also, one often considers the adaptive UC-security in the setting where parties can securely erase their internal state information so that an adversary that later corrupts the party cannot see the erased information. In this paper, we consider the strongest UC-security against *adaptive* adversaries *without assuming secure erasures*.

5.2 Ideal Functionalities.

Here, we recall ideal functionalities of CRS and NIZK. The ideal functionality of CRS denoted by $\mathcal{F}_{\mathsf{crs}}^{\mathcal{D}}$ is given in Figure 2, and that of NIZK denoted by $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}}$ is given in Figure 3. The descriptions here are taken verbatim from [CsW19]. The ideal functionality $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}}$ captures correctness, soundness, and zero-knowledge as roughly explained below:

[13]Strictly speaking, $\mathcal{Z}$ is limited to be a *balanced* one, which roughly means that $\mathcal{Z}$ should not give too much inputs to the adversary compared to inputs given to the other parties. See [Can00] for more details.

Ideal Functionality $\mathcal{F}_{\mathsf{crs}}^{\mathcal{D}}$

$\mathcal{F}_{\mathsf{crs}}^{\mathcal{D}}$ proceeds as follows, running with parties $P_1, ..., P_N$ and an adversary $\mathcal{S}$, and parametrized by a distribution $\mathcal{D}$.

- Upon receiving a message $(\mathsf{init}, \mathsf{sid})$ from party P_i, do:
    1. If there is no value $(\mathsf{sid}, \mathsf{crs})$ recorded, then sample $\mathsf{crs} \xleftarrow{\$} \mathcal{D}$ and record it.
    2. Send $(\mathsf{sid}, \mathsf{crs})$ as a public delayed output to P_i.[a]

[a] "Public delayed output" means that the value is first sent to $\mathcal{S}$, and then sent to P_i after the permission by $\mathcal{S}$. See [Can01] for details.

Figure 2: The common reference string functionality

Ideal Functionality $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}}$

$\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}}$ proceeds as follows, running with parties $P_1, ..., P_N$ and an adversary $\mathcal{S}$, and parametrized by a relation $\mathcal{R}$.

Proof: On input $(\mathsf{prove}, \mathsf{sid}, x, w)$ from a party P_i, ignore if $(x, w) \notin \mathcal{R}$. Send $(\mathsf{prove}, P_i, \mathsf{sid}, x)$ to $\mathcal{S}$ and wait for $(\mathsf{proof}, \mathsf{sid}, \pi)$. Upon receiving the answer store (sid, x, π) and send $(\mathsf{proof}, \mathsf{sid}, \pi)$ to P_i.

Verification: On input $(\mathsf{verify}, \mathsf{sid}, x, \pi)$ from a party P_j check whether (sid, x, π) is stored. If not send $(\mathsf{verify}, P_j, \mathsf{sid}, x, \pi)$ to $\mathcal{S}$ and wait for an answer $(\mathsf{witness}, w)$. Upon receiving the answer, check whether $(x, w) \in \mathcal{R}$ and in that case, store (sid, x, π). If (sid, x, π) has been stored return $(\mathsf{verification}, \mathsf{sid}, x, \pi, \top)$ to P_j, else return $(\mathsf{verification}, \mathsf{sid}, x, \pi, \bot)$ to P_j.

Figure 3: The NIZK functionality

- Correctness is captured since all proofs generated by $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}}$ are stored and thus accepted.
- Soundness is captured since if a proof π on a statement x that has not been generated by $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}}$ passes the verification, then $\mathcal{S}$ should succeed in extracting a valid witness w for the statement x, which is possible only when $x \in \mathcal{L}$ where $\mathcal{L} = \{x | \exists w \text{ s.t. } (x, w) \in \mathcal{R}\}$.
- Zero-knowledge is captured since $\mathcal{S}$ generates a proof π without knowing a witness w.

5.3 Construction

Here, we give our construction of adaptive UC-NIZK scheme. The construction is very similar to UC-NIZK scheme from HTDF by Cohen et al. [CsW19]. The construction can also be seen as a variant of the CRS-NIZK scheme given in Section 4 with the following two modifications:

- The proving algorithm directly commit to the witness w by HEC instead of committing to the SKE key K by which the witness w is encrypted. Though this increases the proof size (i.e., from $|w| + \mathsf{poly}(\kappa)$ to $|w| \cdot \mathsf{poly}(\kappa)$), this is needed for achieving the adaptive security.
- A verification key and a signature by a OTS scheme are added as a part of a proof. Accordingly, the relation proven by the underlying non-compact NIZK scheme is augmented to include a OTS verification key vk into a statement. Roughly speaking, this is to add "non-malleability" for the scheme.

In the following, we describe our UC-NIZK scheme in more details. Let $\mathcal{R}$ be any relation over $\{0,1\}^n \times \{0,1\}^m$. For each $x \in \{0,1\}^n$, C_x denotes a circuit that takes $w \in \{0,1\}^m$ as input, and returns 1 if $(x, w) \in \mathcal{R}$ and 0 otherwise. We construct a UC-NIZK protocol in the $(\mathcal{F}_{\mathsf{NIZK}}^{\tilde{\mathcal{R}}}, \mathcal{F}_{\mathsf{crs}}^{\mathcal{D}_{\mathsf{HEC}}})$-hybrid model based on the following building blocks where $\tilde{\mathcal{R}}$ and $\mathcal{D}_{\mathsf{HEC}}$ are defined below:

- A HEC scheme $\mathsf{HEC} = (\mathsf{HEC.Setup}, \mathsf{HEC.Commit}, \mathsf{HEC.Open}, \mathsf{HEC.Eval}^{in}, \mathsf{HEC.Eval}^{out}, \mathsf{HEC.Verify})$ with the message space $\{0,1\}^m$ and randomness space $\mathcal{R}$ on which the randomness distribution $\mathcal{D}_{\mathcal{R}}$ is defined that supports a function class that contains $\{C_x\}_{x \in \{0,1\}^n}$.

Protocol Π_{UCNIZK}

- Common reference string: a public parameter pp and an evaluation key ek generated as $(\mathsf{pp}, \mathsf{ek}, \mathsf{msk}) \overset{\$}{\leftarrow} \mathsf{HEC.Setup}(1^\kappa)$.

- Upon receiving $(\mathsf{prove}, \mathsf{sid}, x, w)$, a party proceeds as follows:

  1. Choose $R \overset{\$}{\leftarrow} \mathcal{D}_{\mathcal{R}}$.
  2. Compute $\mathsf{com} := \mathsf{HEC.Commit}(\mathsf{pp}, w; R)$.
  3. Compute $\pi_{\mathsf{HEC}} := \mathsf{HEC.Eval}^{in}(\mathsf{ek}, C_x, w, R)$.
  4. Compute $\mathsf{com}_{\mathsf{eval}} := \mathsf{HEC.Eval}^{out}(\mathsf{ek}, C_x, \mathsf{com})$.
  5. Generate $(\mathsf{vk}, \mathsf{sigk}) \overset{\$}{\leftarrow} \mathsf{OTS.KeyGen}(1^\kappa)$.
  6. Send $(\mathsf{prove}, \mathsf{sid}, (\mathsf{pp}, \mathsf{com}, \mathsf{com}_{\mathsf{eval}}, \mathsf{vk}), (w, R, \pi_{\mathsf{HEC}}))$ to $\mathcal{F}_{\mathsf{NIZK}}^{\widetilde{\mathcal{R}}}$.
  7. Wait for the answer $(\mathsf{proof}, \mathsf{sid}, (\mathsf{pp}, \mathsf{com}, \mathsf{com}_{\mathsf{eval}}, \mathsf{vk}), \pi_{\mathsf{NIZK}})$.
  8. Compute $\sigma \overset{\$}{\leftarrow} \mathsf{OTS.Sign}(\mathsf{sigk}, (x, \mathsf{com}, \pi_{\mathsf{NIZK}}, \mathsf{vk}))$.
  9. Return $(\mathsf{proof}, \mathsf{sid}, x, (\mathsf{com}, \pi_{\mathsf{NIZK}}, \mathsf{vk}, \sigma))$.

- Upon receiving $(\mathsf{prove}, \mathsf{sid}, x, \pi)$, a party proceeds as follows:

  1. Parse π as $(\mathsf{com}, \pi_{\mathsf{NIZK}}, \mathsf{vk}, \sigma)$.
  2. Verify that $\mathsf{OTS.Verify}(\mathsf{vk}, (x, \mathsf{com}, \pi_{\mathsf{NIZK}}, \mathsf{vk}), \sigma) = \top$. If not return $(\mathsf{verification}, \mathsf{sid}, x, \pi, \bot)$.
  3. Compute $\mathsf{com}_{\mathsf{eval}} := \mathsf{HEC.Eval}^{out}(\mathsf{ek}, C_x, \mathsf{com})$.
  4. Send $(\mathsf{verify}, \mathsf{sid}, (\mathsf{pp}, \mathsf{com}, \mathsf{com}_{\mathsf{eval}}, \mathsf{vk}), \pi_{\mathsf{NIZK}})$ to $\mathcal{F}_{\mathsf{NIZK}}^{\widetilde{\mathcal{R}}}$.
  5. Wait for the answer $(\mathsf{verification}, \mathsf{sid}, (\mathsf{pp}, \mathsf{com}, \mathsf{com}_{\mathsf{eval}}, \mathsf{vk}), \pi_{\mathsf{NIZK}}, b)$.
  6. Return $(\mathsf{verification}, \mathsf{sid}, x, \pi, b)$.

Figure 4: Adaptively secure UC-NIZK protocol

- A strongly unforgeable OTS scheme $\mathsf{OTS} = (\mathsf{OTS.KeyGen}, \mathsf{OTS.Sign}, \mathsf{OTS.Verify})$.
- The relation $\widetilde{R}$ is defined as follows:
  $((\mathsf{pp}, \mathsf{com}, \mathsf{com}_{\mathsf{eval}}, \mathsf{vk}), (w, R, \pi_{\mathsf{HEC}})) \in \widetilde{\mathcal{R}}$ if and only if the followings are satisfied:
  1. $\mathsf{HEC.Commit}(\mathsf{pp}, w; R) = \mathsf{com}$,
  2. $\mathsf{HEC.Verify}(\mathsf{pp}, \mathsf{com}_{\mathsf{eval}}, 1, \pi_{\mathsf{HEC}}) = \top$.
- The distribution $\mathcal{D}_{\mathsf{HEC}}$ is the distribution of pp and ek where $(\mathsf{pp}, \mathsf{ek}, \mathsf{msk}) \xleftarrow{\$} \mathsf{HEC.Setup}(1^{\kappa})$.

Our UC-NIZK scheme Π_{UCNIZK} in the $(\mathcal{F}_{\mathsf{NIZK}}^{\widetilde{\mathcal{R}}}, \mathcal{F}_{\mathsf{crs}}^{\mathcal{D}_{\mathsf{HEC}}})$-hybrid model is described in Figure 4. Here, we remark that we describe the scheme as if there is a common reference string available for every party. Strictly speaking, we should implement it by using the ideal functionality $\mathcal{F}_{\mathsf{crs}}^{\mathcal{D}_{\mathsf{HEC}}}$ (i.e., each party accesses to $\mathcal{F}_{\mathsf{crs}}^{\mathcal{D}_{\mathsf{HEC}}}$ whenever it needs the common reference string). For notational simplicity, we omit this, and just think that a common reference string is chosen at the beginning of the protocol execution and published for every party.
Security.

Theorem 5.5. *If Π_{HEC} satisfies distributional equivalence of open and computational binding for evaluated commitments, then Π_{UCNIZK} UC-realizes $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}}$ in the $(\mathcal{F}_{\mathsf{NIZK}}^{\widetilde{\mathcal{R}}}, \mathcal{F}_{\mathsf{crs}}^{\mathcal{D}})$-hybrid model tolerating adaptive, malicious adversaries.*

Proof. The proof is basically identical to the security proof for UC-NIZK based on HTDF by Cohen et al. [CsW19] except that HTDF is replaced with HEC. Let $\mathcal{A}$ be any PPT adaptive adversary. What we have to do is to construct a simulator that interacts with dummy parties $\tilde{P}_1, ..., \tilde{P}_N$ and the ideal functionality $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}}$ such that no environment $\mathcal{Z}$ can distinguish the simulated execution from the real execution of $\mathcal{A}$ that interacts with real parties $P_1, ..., P_N$ who run the real protocol Π_{UCNIZK}. Following [CsW19], we first consider a simulator $\mathcal{S}_{\mathsf{Real}}$ that perfectly simulates the real execution by using an extended capabilities that it can know inputs to the ideal functionality $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}}$ and control the output of it. (Note that this is not allowed in the ideal world. We consider the execution of $\mathcal{S}_{\mathsf{Real}}$ just as a mental experiment.) Then we gradually modify the simulator without letting the environment notice it with a non-negligible advantage, and finally present a legitimate simulator $\mathcal{S}_{\mathsf{Sim}}$ in the ideal world. We consider the following sequence of simulators.

$\mathcal{S}_{\mathsf{Real}}$**:** As noted above, $\mathcal{S}_{\mathsf{Real}}$ can know inputs to the ideal functionality $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}}$ and control the output of it. The simulator $\mathcal{S}_{\mathsf{Real}}$ along with dummy parties $\tilde{P}_1, ..., \tilde{P}_N$ perfectly simulates the execution between $\mathcal{A}$ and $P_1, ...P_N$ who run Π_{UCNIZK} by using these capabilities. Specifically, it works as follows:

- To simulate the common reference string, $\mathcal{S}_{\mathsf{Real}}$ first generates $(\mathsf{pp}, \mathsf{ek}, \mathsf{msk}) \xleftarrow{\$} \mathsf{HEC.Setup}(1^{\kappa})$ and sets pp as a common reference string used throughout the execution.
- When $\mathcal{S}_{\mathsf{Real}}$ receives $(\mathsf{prove}, P_i, \mathsf{sid}, x)$ from the ideal functionality $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}}$, it must be the case that honest $\tilde{P}_i$ has sent $(\mathsf{prove}, \mathsf{sid}, x, w)$ to $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}}$ such that $(x, w) \in \mathcal{R}$. By using the capability to see the input $(\mathsf{prove}, \mathsf{sid}, x, w)$, $\mathcal{S}_{\mathsf{Real}}$ honestly runs the proving algorithm of Π_{UCNIZK} on input $(\mathsf{prove}, \mathsf{sid}, x, w)$ to generate a proof π, and returns π to $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}}$.
- When $\mathcal{S}_{\mathsf{Real}}$ receives $(\mathsf{verify}, P_j, \mathsf{sid}, x, \pi)$ from the ideal functionality $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}}$, it must be the case that $\tilde{P}_j$ has sent $(\mathsf{prove}, \mathsf{sid}, x, \pi)$ to $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}}$ and π is not a proof that has been generated by the ideal functionality $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}}$. $\mathcal{S}_{\mathsf{Real}}$ honestly runs the protocol Π_{UCNIZK} on input $(\mathsf{verify}, \mathsf{sid}, x, \pi)$ to generate $(\mathsf{verification}, \mathsf{sid}, x, \pi, b)$, and instructs $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}}$ to return $(\mathsf{verification}, \mathsf{sid}, x, \pi, b)$ to $\tilde{P}_j$ by using the capability to control the output of $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}}$. Note that $\mathcal{S}_{\mathsf{Real}}$ does not give a witness w to $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}}$. This is not needed since $\mathcal{S}_{\mathsf{Real}}$ has the capability to control the behavior of $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}}$.
- To simulate the interaction between $\mathcal{A}$ and $\mathcal{Z}$, $\mathcal{S}_{\mathsf{Real}}$ just internally simulates $\mathcal{A}$, and forwards all communications between $\mathcal{A}$ and $\mathcal{Z}$. Whenever $\mathcal{A}$ corrupts a party P_i, $\mathcal{S}_{\mathsf{Real}}$ corrupts the corresponding dummy party $\tilde{P}_i$, and simulate the communication between $\mathcal{A}$ and P_i. We note that since all proofs output by $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}}$ are actually generated by $\mathcal{S}_{\mathsf{Real}}$, it knows all internal coins P_i is supposed to know. Therefore it can perfectly simulate internal states of corrupted parties that are needed for simulating the communication between $\mathcal{A}$ and P_i.

Lemma 5.6. *If Π_{HEC} and Π_{OTS} are correct, then we have* $\texttt{REAL}_{\Pi_{\mathsf{UCNIZK}},\mathcal{A},\mathcal{Z}} \approx \texttt{IDEAL}_{\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}},\mathcal{S}_{\mathsf{Real}},\mathcal{Z}}$.

Proof. The only difference between the real execution of $\mathcal{A}$ interacting with $P_1, ..., P_N$ and the execution of $\mathcal{S}_{\mathsf{Real}}$ interacting with $\tilde{P}_1, ..., \tilde{P}_N$ from the view of $\mathcal{Z}$ is that in the latter, a proof generated through $\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}$ are always accepted by $\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}$ (i.e., it returns $(\mathsf{verification}, \star, \star, \star, \top)$) without checking the validity of the proofs by using the actual verification protocol of Π_{UCNIZK}. On the other hand, it is easy to see that an honestly generated proof is accepted with probability 1 in Π_{UCNIZK} by the correctness of Π_{HEC} and Π_{OTS} and the functionality of $\mathcal{F}^{\widetilde{\mathcal{R}}}_{\mathsf{NIZK}}$. Therefore these two distributions are perfectly indistinguishable. □

$\mathcal{S}_{\mathsf{Ext}}$**:** This simulator works similarly to $\mathcal{S}_{\mathsf{Real}}$ except the way of simulating verification when it receives $(\mathsf{verify}, P_j, \mathsf{sid}, x, \pi)$ from the ideal functionality $\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}$. Unlike $\mathcal{S}_{\mathsf{Real}}$, $\mathcal{S}_{\mathsf{Ext}}$ does not use the power of controlling the behavior of $\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}$, and returns some $(\mathsf{witness}, w)$ whenever $(\mathsf{verify}, P_j, \mathsf{sid}, x, \pi)$ is sent from $\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}$ as supposed to do in the description of $\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}$ as follows: Upon receiving $(\mathsf{verify}, P_j, \mathsf{sid}, x, \pi)$, $\mathcal{S}_{\mathsf{Ext}}$ honestly runs Π_{UCNIZK} on input $(\mathsf{verify}, \mathsf{sid}, x, \pi)$ to obtain $(\mathsf{verification}, \mathsf{sid}, x, \pi, b)$. If $b = \bot$, then it returns $(\mathsf{witness}, \bot)$ to $\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}$. Otherwise, it parses $\pi = (\mathsf{com}, \pi_{\mathsf{NIZK}}, \mathsf{vk}, \sigma)$, computes $\mathsf{com}_{\mathsf{eval}} := \mathsf{HEC.Eval}^{out}(\mathsf{ek}, C_x, \mathsf{com})$, gives $(\mathsf{verification}, \mathsf{sid}, (\mathsf{pp}, \mathsf{com}, \mathsf{com}_{\mathsf{eval}}, \mathsf{vk}), \pi_{\mathsf{NIZK}})$ to $\mathcal{A}$, and waits for the response $(\mathsf{witness}, (w, R, \pi_{\mathsf{HEC}}))$ from $\mathcal{A}$. Then $\mathcal{S}_{\mathsf{Ext}}$ returns $(\mathsf{witness}, w)$ to $\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}$.

Lemma 5.7. *If Π_{OTS} is strongly unforgeable and Π_{HEC} satisfies the computational binding for evaluated commitment, then we have* $\texttt{IDEAL}_{\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}},\mathcal{S}_{\mathsf{Real}},\mathcal{Z}} \approx \texttt{IDEAL}_{\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}},\mathcal{S}_{\mathsf{Ext}},\mathcal{Z}}$.

Proof. The differences between the executions of $\mathcal{S}_{\mathsf{Real}}$ and $\mathcal{S}_{\mathsf{Ext}}$ occurs only when an honest party sends $(\mathsf{verify}, \mathsf{sid}, x, \pi)$ such that $(\mathsf{proof}, \mathsf{sid}, x, \pi)$ has not been generated through $\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}$, π is a valid proof for the statement x, and $(x, w) \notin \mathcal{R}$ where w is the first component of the witness extracted from π_{NIZK} by $\mathcal{A}$. We prove that this happens with negligible probability. If $(\mathsf{proof}, \mathsf{sid}, x, \pi = (\mathsf{com}, \pi_{\mathsf{NIZK}}, \mathsf{vk}, \sigma))$ has not been generated through $\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}$, there are the following two possible cases:

1. No proof of the form $(\mathsf{proof}, \mathsf{sid}, \star, (\mathsf{com}, \pi_{\mathsf{NIZK}}, \mathsf{vk}, \star))$ has been generated through $\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}$.
2. A proof $(\mathsf{proof}, \mathsf{sid}, x', (\mathsf{com}, \pi_{\mathsf{NIZK}}, \mathsf{vk}, \sigma'))$ such that $(x', \sigma') \neq (x, \sigma)$ has been generated through $\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}$.

In the second case, σ is a valid signature on the message $(x, \mathsf{com}, \pi_{\mathsf{NIZK}}, \mathsf{vk})$ with negligible probability due to the strong one-time security of Π_{OTS}, and thus π is not a valid proof for the statement x. Since we are interested in the case that π is a valid proof for the statement x, we consider the first case in the following. If $\pi = (\mathsf{com}, \pi_{\mathsf{NIZK}}, \mathsf{vk}, \sigma)$ is a valid proof for the statement x, then π_{NIZK} is a valid proof for the statement $(\mathsf{pp}, \mathsf{com}, \mathsf{com}_{\mathsf{eval}}, \mathsf{vk})$ by the construction of Π_{UCNIZK} where $\mathsf{com}_{\mathsf{eval}} = \mathsf{HEC.Eval}^{out}(\mathsf{ek}, C_x, \mathsf{com})$. On the other hand, a proof for the statement $(\mathsf{pp}, \mathsf{com}, \mathsf{com}_{\mathsf{eval}}, \mathsf{vk})$ with session id sid has never been generated through $\mathcal{F}^{\widetilde{\mathcal{R}}}_{\mathsf{NIZK}}$ since this happens only when a proof of the form $(\mathsf{proof}, \mathsf{sid}, \star, (\mathsf{com}, \pi_{\mathsf{NIZK}}, \mathsf{vk}, \star))$ is generated through $\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}$. This means that $\mathcal{A}$ must succeeds in extracting $(w, R, \pi_{\mathsf{HEC}})$ such that $((\mathsf{pp}, \mathsf{com}, \mathsf{vk}), (w, R, \pi_{\mathsf{HEC}})) \in \tilde{\mathcal{R}}$ by the definition of ideal functionality $\mathcal{F}^{\widetilde{\mathcal{R}}}_{\mathsf{NIZK}}$. Especially, we have $\mathsf{com} = \mathsf{HEC.Commit}(\mathsf{pp}, w; R)$. Then we have $(x, w) \in \mathcal{R}$ except a negligible probability since otherwise we succeed in breaking the computational binding of Π_{HEC} by outputting $(w, R, C_x, 1, \pi_{\mathsf{HEC}})$. (Remark that we have $C_x(w) = 0$ if $(x, w) \in \mathcal{R}$.) In summary, the difference between executions of these two simulators occur with negligible probability. □

$\mathcal{S}_{\mathsf{Sim}}$**:** This simulator works similarly to $\mathcal{S}_{\mathsf{Ext}}$ except the way of simulating a proof when it receives $(\mathsf{prove}, \mathsf{sid}, x)$ from the ideal functionality $\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}$ and the way of simulating internal coins of corrupted parties.

- When it receives $(\mathsf{prove}, P_i, \mathsf{sid}, x)$, it picks $\overline{R} \xleftarrow{\$} \mathcal{D}_{\mathcal{R}}$, computes $\mathsf{com} := \mathsf{HEC.Commit}(\mathsf{pp}, 0^m; \overline{R})$ and $\mathsf{com}_{\mathsf{eval}} := \mathsf{HEC.Eval}^{out}(\mathsf{ek}, C_x, \mathsf{com})$, generates $(\mathsf{vk}, \mathsf{sigk}) \xleftarrow{\$} \mathsf{OTS.KeyGen}(1^\kappa)$, and sends $(\mathsf{prove}, \mathsf{sid}, (\mathsf{pp}, \mathsf{com}, \mathsf{com}_{\mathsf{eval}}, \mathsf{vk}))$ to $\mathcal{A}$, which returns $(\mathsf{proof}, \mathsf{sid}, (\mathsf{pp}, \mathsf{com}, \mathsf{com}_{\mathsf{eval}}, \mathsf{vk}), \pi_{\mathsf{NIZK}})$. Then it computes $\sigma \xleftarrow{\$} \mathsf{OTS.Sign}(\mathsf{sigk}, (x, \mathsf{com}, \pi_{\mathsf{NIZK}}, \mathsf{vk}))$ and returns $(\mathsf{proof}, \mathsf{sid}, x, (\mathsf{com}, \pi_{\mathsf{NIZK}}, \mathsf{vk}, \sigma))$ to $\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}$. $\mathcal{S}_{\mathsf{Sim}}$ stores all randomness used in the above simulation (i.e., $\overline{R}$ and other randomness R' used in $\mathsf{OTS.KeyGen}$ and $\mathsf{OTS.Sign}$) along with the corresponding party P_i, session ID sid, the statement x, and the proof $\pi = (\mathsf{com}, \pi_{\mathsf{NIZK}}, \mathsf{vk}, \sigma)$.

- When the adversary $\mathcal{A}$ corrupts a party P_i, it corrupts $\tilde{P}_i$ who has generated k proofs $\pi_1, ..., \pi_k$. Then it knows from the internal state of $\tilde{P}_j$ that π_j was generated on an input (x_j, w_j) for each $j \in [k]$. For each $j \in [k]$, $\mathcal{S}_{\mathsf{Sim}}$ does the following: It first looks for the corresponding randomness $(\overline{R}_j, R'_j)$ used for simulating π_j where $\overline{R}_j$ stands for the randomness used in $\mathsf{HEC.Commit}$ and R'_j stands for all the randomness used in the other parts (i.e., $\mathsf{OTS.KeyGen}$ and $\mathsf{OTS.Sign}$) of the simulation of π_j. It parses $\pi_j = (\mathsf{com}_j, \pi_{\mathsf{NIZK},j}, \mathsf{vk}_j, \sigma_j)$, computes $R_j := \mathsf{HEC.Open}(\mathsf{msk}, (0^m, \overline{R}_j), w_j)$, and uses (R_j, R'_j) as a randomness used for generating π_j on the input $(\mathsf{prove}, \mathsf{sid}, x_j, w_j)$ in the simulation of the interaction between the corrupted party P_j and the adversary $\mathcal{A}$. (Since all internal information owned by P_j is randomness used in the generations of proofs, this suffices for the simulation.)

Lemma 5.8. *If Π_{HEC} satisfies the distributional equivalence of open, then we have $\mathtt{IDEAL}_{\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}, \mathcal{S}_{\mathsf{Ext}}, \mathcal{Z}} \approx \mathtt{IDEAL}_{\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}, \mathcal{S}_{\mathsf{Sim}}, \mathcal{Z}}$.*

Proof. The only differences between simulations by $\mathcal{S}_{\mathsf{Ext}}$ and $\mathcal{S}_{\mathsf{Sim}}$ are the way of simulating a commitment com and its underlying randomness R in each simulated proof. Namely, in the former, we generates $\mathsf{com} = \mathsf{HEC.Commit}(\mathsf{pp}, w; R)$ with a randomly chosen $R \xleftarrow{\$} \mathcal{D}_{\mathcal{R}}$ whereas in the latter, we first generates $\mathsf{com} = \mathsf{HEC.Commit}(\mathsf{pp}, 0^m; \overline{R})$ with a randomly chosen $\overline{R} \xleftarrow{\$} \mathcal{D}_{\mathcal{R}}$, and then later computes $R := \mathsf{HEC.Open}(\mathsf{msk}, (0, \overline{R}), w)$ when the corresponding w is revealed. The distributional equivalence of open property of HEC exactly says that the joint distribution of $(\mathsf{pp}, \mathsf{msk}, w, R, \mathsf{com})$ is statistically close in these two situations even when w may arbitrarily depend on $(\mathsf{pp}, \mathsf{msk})$. Therefore $\mathcal{Z}$ has negligible advantage to distinguish these two simulations. $\square$

By combining the above lemmas, we have $\mathtt{REAL}_{\Pi_{\mathsf{UCNIZK}}, \mathcal{A}, \mathcal{Z}} \approx \mathtt{IDEAL}_{\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}, \mathcal{S}_{\mathsf{Sim}}, \mathcal{Z}}$. Here, we notice that $\mathcal{S}_{\mathsf{Sim}}$ no longer uses any extended capability, and it is a legitimate simulator in the ideal world. This concludes the proof of Theorem 5.5 $\square$

5.4 Instantiations

Here, we discuss a possible instantiation for Π_{UCNIZK}. First, we recall the result by Groth, Ostrovsky, and Sahai (called GOS proof) [GOS12]. In the following, $\mathcal{U}$ denotes a uniform distribution over the set of all bit strings of a certain length.

Lemma 5.9 ([GOS12]). *If the DLIN assumption holds in a bilinear group, then for any efficiently verifiable relation $\widetilde{\mathcal{R}}$, there exists a NIZK scheme that UC-realizes $\mathcal{F}^{\widetilde{\mathcal{R}}}_{\mathsf{NIZK}}$ in the $\mathcal{F}^{\mathcal{U}}_{\mathsf{crs}}$-hybrid model. The proof size of the scheme is linear in the size of circuit that verifies the relation $\widetilde{\mathcal{R}}$.*

Then by combining Theorem 5.5 and Lemma 5.9, we obtain the following corollary.

Corollary 5.10. *If the DLIN assumption holds in a bilinear group and HEC scheme that supports a function class containing verification of a relation $\mathcal{R}$ exists, then there exists a NIZK scheme that UC-realizes $\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}$ in the $(\mathcal{F}^{\mathcal{D}_{\mathsf{HEC}}}_{\mathsf{crs}}, \mathcal{F}^{\mathcal{U}}_{\mathsf{crs}})$-hybrid model tolerating an adaptive, malicious adversary.*

We instantiate the scheme with the HEC for $\mathbf{NC}^1$ with efficient verification based on the CDHER assumption given in Section 3.2. In this case, since the running time of $\mathsf{HEC.Commit}$ and $\mathsf{HEC.Verify}$ are $\mathsf{poly}(\kappa)$, which are independent of the size of a circuit to be evaluated, the relation $\widetilde{\mathcal{R}}$, which is supported by the underlying non-compact UC-NIZK, can be verified by a circuit of size $|w| \cdot \mathsf{poly}(\kappa)$, which is independent of the complexity of $\mathcal{R}$. Since the proof size of GOS proof is linear in the circuit size to verify the relation, the proof size of the resulting UC-NIZK scheme is $|w| \cdot \mathsf{poly}(\kappa)$. On the other hand, since the HEC scheme only supports $\mathbf{NC}^1$, we have to restrict a class of relations to be verifiable in $\mathbf{NC}^1$. In summary we obtain the following corollary.

Corollary 5.11. *If the DLIN assumption and the CDHER assumption hold in a bilinear group, then for any relation $\mathcal{R}$ that is computable in $\mathbf{NC}^1$, there exists a NIZK scheme with proof size $|w| \cdot \mathsf{poly}(\kappa)$ that UC-realizes $\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}$ in the $(\mathcal{F}^{\mathcal{D}_{\mathsf{HEC}}}_{\mathsf{crs}}, \mathcal{F}^{\mathcal{U}}_{\mathsf{crs}})$-hybrid model tolerating an adaptive, malicious adversary.*

We note that though it is generally possible to modify any relation to one verifiable in $\mathbf{NC}^1$ (e.g., see [GGH+16]), this is done at the cost of making the witness size as large as the size of a circuit that verifies the relation, which is not useful in our setting. Namely, if we apply this technique in our setting, then the proof size of the resulting UC-NIZK is $|C| \cdot \mathsf{poly}(\kappa)$, which is asymptotically no better than the GOS proof.

6 Compact DV-NIZK

6.1 Preliminaries

Here, we prepare some definitions and lemmas that are needed to present our DV-NIZK.
Functional Encryption for Inner-Product. Here, we define functional encryption for inner-product (IPFE). An IPFE scheme over $\mathbb{Z}_p$ consists of PPT algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$.

$\mathsf{Setup}(1^\kappa, 1^d) \to (\mathsf{pp}, \mathsf{msk})$**:** The setup algorithm takes as inputs the security parameter 1^κ and the dimension 1^d, and outputs a public parameter pp and a master secret key msk.

$\mathsf{KeyGen}(\mathsf{msk}, \mathbf{y}) \to \mathsf{sk}$**:** The key generation algorithm takes as inputs a master secret key msk and a vector $\mathbf{y} = (y_1, ..., y_d) \in \mathbb{Z}_p^d$, and outputs a secret key sk. Without loss of generality, we assume that sk always contains $\mathbf{y}$.

$\mathsf{Enc}(\mathsf{pp}, \mathbf{x}) \to \mathsf{ct}$**:** The encryption algorithm takes as inputs a public parameter pp and a vector $\mathbf{x} = (x_1, ..., x_d) \in \mathbb{Z}_p^d$, and outputs a ciphertext ct.

$\mathsf{Dec}(\mathsf{pp}, \mathsf{sk}, \mathsf{ct}) \to z$**:** The decryption algorithm takes as inputs a public parameter pp, a secret key sk, and a ciphertext ct, and outputs $z \in \mathbb{Z}_p$.

Correctness. For all $\kappa, d \in \mathbb{N}$, $(\mathsf{pp}, \mathsf{msk}) \in \mathsf{Setup}(1^\kappa, 1^d)$, vectors $\mathbf{x} \in \mathbb{Z}_p^d$ and $\mathbf{y} \in \mathbb{Z}_p^d$, secret keys $\mathsf{sk} \in \mathsf{KeyGen}(\mathsf{msk}, \mathbf{y})$, and ciphertexts $\mathsf{ct} \in \mathsf{Enc}(\mathsf{pp}, \mathbf{x})$, we have

$$\mathsf{Dec}(\mathsf{pp}, \mathsf{ct}, \mathsf{sk}) = \langle \mathbf{x}, \mathbf{y} \rangle \mod p.$$

Security. For an adversary $\mathcal{A}$, we consider the following experiment between a challenger and an adversary $\mathcal{A}$.

1. $\mathcal{A}$ is given 1^κ and outputs 1^d.
2. The challenger generates $(\mathsf{pp}, \mathsf{msk}) \xleftarrow{\$} \mathsf{Setup}(1^\kappa, 1^d)$.
3. $\mathcal{A}$ is given pp. It is allowed to make arbitrary number of key generation queries. When it makes a key query $\mathbf{y}$, the challenger generates $\mathsf{sk} \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{msk}, \mathbf{y})$ and returns sk to $\mathcal{A}$. At some point, $\mathcal{A}$ output vectors $\mathbf{x}^{(0)}$ and $\mathbf{x}^{(1)}$.
4. The challenger randomly picks $\mathsf{coin} \xleftarrow{\$} \{0, 1\}$, generates $\mathsf{ct} \xleftarrow{\$} \mathsf{Enc}(\mathsf{pp}, \mathbf{x}^{(\mathsf{coin})})$.
5. $\mathcal{A}$ is given ct, and allowed to make arbitrary number of key generation queries again. Finally, $\mathcal{A}$ outputs coin'. We say that $\mathcal{A}$ wins if $\mathsf{coin}' = \mathsf{coin}$.

We say that $\mathcal{A}$ is adaptively admissible if for all key queries $\mathbf{y}$ made by $\mathcal{A}$ and vectors $\mathbf{x}^{(0)}$ and $\mathbf{x}^{(1)}$ output by $\mathcal{A}$, we have $\langle \mathbf{x}^{(0)}, \mathbf{y} \rangle = \langle \mathbf{x}^{(1)}, \mathbf{y} \rangle \mod p$. We say that Π_{IPFE} is adaptively secure if for all adaptively admissible adversaries $\mathcal{A}$, the probability $|\Pr[\mathcal{A} \text{ wins}] - 1/2|$ is negligible. We say that Π_{IPFE} is adaptively *single-key* secure if for all adaptively admissible adversaries $\mathcal{A}$ that makes at most one key query, the probability $|\Pr[\mathcal{A} \text{ wins}] - 1/2|$ is negligible.

Remark 6.1. (On Multi-Challenge Security.) We also consider the *multi-challenge security* where an adversary can query multiple challenge vectors $\mathbf{x}_i^{(0)}$ and $\mathbf{x}_i^{(1)}$ for each $i = 1, ..., Q$, and the challenger returns $\mathsf{Enc}(\mathsf{pp}, \mathbf{x}_i^{(\mathsf{coin})})$ for each i. We can reduce the multi-challenge security to the single-challenge security defined above by a standard hybrid argument.

Master Decryption of IPFE. Here, we define an additional algorithm for IPFE called the *master decryption* algorithm denoted by $\mathsf{MasterDec}$. This algorithm is given a master secret key and a ciphertext as input, and outputs the corresponding message to the ciphertext. We note that we only require the algorithm to correctly work if the given ciphertext is *valid* (i.e., the ciphertext is in the range of the encryption algorithm). This is a very weak requirement, and indeed we can implement $\mathsf{MasterDec}$ by just combining KeyGen and Dec as follows:

$\mathsf{MasterDec}(\mathsf{msk}, \mathsf{ct})$**:** For all $i \in [d]$, it generates $\mathsf{sk}_i \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{msk}, \mathbf{e}_i)$ where $\mathbf{e}_i$ is the vector whose i-th entry is 1 and all other entries are 0, and computes $x_i \xleftarrow{\$} \mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct})$. Then it outputs $\mathbf{x} := (x_1, ..., x_d)$.

Then we have the following: For all $\kappa, d \in \mathbb{N}$, $(\mathsf{pp}, \mathsf{msk}) \in \mathsf{Setup}(1^\kappa, 1^d)$, vectors $\mathbf{x} \in \mathbb{Z}_p^d$, we have

$$\mathsf{MasterDec}(\mathsf{msk}, \mathsf{Enc}(\mathsf{pp}, \mathbf{x})) = \mathbf{x}.$$

Remark 6.2. One may think that the concept of master decryption is similar to the extractability defined for IPFE on exponent in [KNYY19]. However, the important difference is that the master decryption algorithm is only required to work for valid ciphertexts whereas the extraction in [KNYY19] is required to work even for malformed ciphertexts. Therefore master decryption is much weaker property than the extraction, and indeed this is possible for any IPFE scheme as shown above.

Useful lemmas. Here, we recall some lemmas which are implicit or explicit in [KNYY19]. The proof of Lemma 6.3 is given in Appendix B for completeness.

Lemma 6.3. *(Implicit in [KNYY19]) Let C be a boolean circuit that computes a relation $\mathcal{R}$ on $\{0,1\}^n \times \{0,1\}^m$, i.e., for $(x, w) \in \{0,1\}^n \times \{0,1\}^m$, we have $C(x, w) = 1$ if and only if $(x, w) \in \mathcal{R}$, and p be an integer larger than $|C|$. Then there exists a deterministic algorithm $\mathsf{Exp}_{C,x}$ and an arithmetic circuit $\tilde{C}$ on $\mathbb{Z}_p$ with degree at most 3 such that we have*

- $|\mathsf{Exp}_{C,x}(w)| = |C(x, \cdot)|$ *for all* $w \in \{0,1\}^m$.
- *If* $C(x, w) = 1$, *then we have* $\tilde{C}(x, \mathsf{Exp}_{C,x}(w)) = 1 \mod p$.
- *For any* $x \in \{0,1\}^n$, *if there does not exist* $w \in \{0,1\}^m$ *such that* $C(x, w) = 1$, *then there does not exist* $w' \in \{0,1\}^{|C(x,\cdot)|}$ *such that* $\tilde{C}(x, w') = 1 \mod p$

Lemma 6.4. *([KNYY19]) There exists a deterministic polynomial-time algorithm* $\mathsf{Coefficient}$ *that satisfies the following: for any $p \in \mathbb{N}$, arithmetic circuit f over $\mathbb{Z}_p$ of degree D, $\mathbf{x} = (x_1, ..., x_\ell) \in \mathbb{Z}_p^\ell$ and $\boldsymbol{\sigma} = (\sigma_1, ..., \sigma_\ell) \in \mathbb{Z}_p^\ell$,* $\mathsf{Coefficient}(1^D, p, f, \mathbf{x}, \boldsymbol{\sigma})$ *outputs $(c_1, ..., c_D) \in \mathbb{Z}_p^D$ such that*

$$f(\sigma_1 Z + x_1, ..., \sigma_\ell Z + x_\ell) = f(x_1, ..., x_\ell) + \sum_{j=1}^{D} c_j Z^j \mod p. \tag{11}$$

where Z is an indeterminate.

6.2 Construction

Here, we give a generic construction of compact DV-NIZK. Namely, we construct DV-NIZK with the proof size $|C| + \mathsf{poly}(\kappa)$ from any (non-compact) DV-NIZK, SKE scheme whose decryption circuit is in $\mathbf{NC}^1$ with additive ciphertext overhead, and PKE scheme. First, we prepare notations and the building blocks.

- Let $\mathcal{L}$ be an **NP** language defined by a relation $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$. Let $n(\kappa)$ and $m(\kappa)$ be any fixed polynomials. Let C be a circuit that computes the relation $\mathcal{R}$ on $\{0,1\}^n \times \{0,1\}^m$, i.e., for $(x, w) \in \{0,1\}^n \times \{0,1\}^m$, we have $C(x, w) = 1$ if and only if $(x, w) \in \mathcal{R}$. Let $\mathsf{Exp}_{C,x}$ and $\tilde{C}$ be as defined in Lemma 6.3.
- Let $\Pi_{\mathsf{IPFE}} = (\mathsf{IPFE.Setup}, \mathsf{IPFE.KeyGen}, \mathsf{IPFE.Enc}, \mathsf{IPFE.Dec})$ be an adaptively single-key secure IPFE scheme with a prime modulus $p > |C|$. Such an IPFE scheme can be constructed from any PKE scheme [GVW12].
- Let $\Pi_{\mathsf{SKE}} = (\mathsf{SKE.KeyGen}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ be a CPA-secure symmetric key encryption scheme over a ciphertext space $\mathcal{CT}$ and a key space $\{0,1\}^\ell$ with additive ciphertext overhead (i.e., the ciphertext size is the message size plus $\mathsf{poly}(\kappa)$) whose decryption algorithm is computed in $\mathbf{NC}^1$. Especially, the decryption circuit can be expressed by an arithmetic circuit over $\mathbb{Z}_p$ of degree $\mathsf{poly}(\kappa)$. We note that such an SKE scheme exists under the CDH assumption [KNYY19].
- For $x \in \{0,1\}^n$ and $\mathsf{ct} \in \mathcal{CT}$, we define the function $f_{x,\mathsf{ct}}(K) := \tilde{C}(x, \mathsf{SKE.Dec}(K, \mathsf{ct}))$. Let D be the maximal degree of $f_{x,\mathsf{ct}}$ (as a multivariate polynomial). Since $\tilde{C}$'s degree is at most 3 and $\mathsf{SKE.Dec}(\cdot, \mathsf{ct})$'s degree is $\mathsf{poly}(\kappa)$, we have $D = \mathsf{poly}(\kappa)$ (which especially does not depend on $|C|$).

- Let $\Pi_{\mathsf{DVNIZK}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ be DV-NIZK for the language corresponding to the relation $\tilde{\mathcal{R}}$ defined below:

$$\left((\mathsf{pp}_{\mathsf{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \mathsf{pp}'_{\mathsf{IPFE}}, \mathsf{ct}'_{\mathsf{IPFE}}),\ \ (\{(K_i, \sigma_i, R_i)\}_{i\in[\ell]}, (c_1, ..., c_D, R')) \right) \in \tilde{\mathcal{R}}$$

  if and only if the following conditions are satisfied:

  1. For all $i \in [\ell]$, $K_i \in \{0,1\}$,
  2. For all $i \in [\ell]$, $\mathsf{IPFE.Enc}(\mathsf{pp}_{\mathsf{IPFE}}, (K_i, \sigma_i); R_i) = \mathsf{ct}^i_{\mathsf{IPFE}}$,
  3. $\mathsf{IPFE.Enc}(\mathsf{pp}'_{\mathsf{IPFE}}, (c_1, ..., c_D); R') = \mathsf{ct}'_{\mathsf{IPFE}}$.

The DV-NIZK $\Pi'_{\mathsf{DVNIZK}} = (\mathsf{Setup}', \mathsf{Prove}', \mathsf{Verify}')$ for $\mathcal{L}$ is described as follows.

$\mathsf{Setup}'(1^\kappa)$**:** This algorithm picks $s \xleftarrow{\$} \mathbb{Z}_p^*$ and generates $(\mathsf{crs}, k_{\mathsf{V}}) \xleftarrow{\$} \mathsf{Setup}(1^\kappa)$, $(\mathsf{pp}_{\mathsf{IPFE}}, \mathsf{msk}_{\mathsf{IPFE}}) \xleftarrow{\$} \mathsf{IPFE.Setup}(1^\kappa, 1^2)$, $(\mathsf{pp}'_{\mathsf{IPFE}}, \mathsf{msk}'_{\mathsf{IPFE}}) \xleftarrow{\$} \mathsf{IPFE.Setup}(1^\kappa, 1^D)$, $\mathsf{sk}_{\mathsf{IPFE}} \xleftarrow{\$} \mathsf{IPFE.KeyGen}(\mathsf{msk}_{\mathsf{IPFE}}, (1, s))$, and $\mathsf{sk}'_{\mathsf{IPFE}} \xleftarrow{\$} \mathsf{IPFE.KeyGen}$ $(\mathsf{msk}'_{\mathsf{IPFE}}, (s, ..., s^D))$. It outputs a common reference string $\mathsf{crs}' := (\mathsf{crs}, \mathsf{pp}_{\mathsf{IPFE}}, \mathsf{pp}'_{\mathsf{IPFE}})$ and a verifier key $k'_{\mathsf{V}} := (k_{\mathsf{V}}, s, \mathsf{sk}_{\mathsf{IPFE}}, \mathsf{sk}'_{\mathsf{IPFE}})$.

$\mathsf{Prove}'(\mathsf{crs}', x, w)$**:** This algorithm aborts if $(x, w) \notin \mathcal{R}$. Otherwise it parses $(\mathsf{crs}, \mathsf{pp}_{\mathsf{IPFE}}, \mathsf{pp}'_{\mathsf{IPFE}}) \leftarrow \mathsf{crs}'$, picks $K \xleftarrow{\$} \mathsf{SKE.KeyGen}(1^\kappa)$ and $\sigma_i \xleftarrow{\$} \mathbb{Z}_p$ for $i \in [\ell]$, and generates $\mathsf{ct}_{\mathsf{SKE}} \xleftarrow{\$} \mathsf{SKE.Enc}(K, \mathsf{Exp}_{C,x}(w))$ and $(c_1, ..., c_D) \leftarrow \mathsf{Coefficient}(1^D, p, f_{x,\mathsf{ct}_{\mathsf{SKE}}}, K = (K_1, ..., K_\ell), (\sigma_1, ..., \sigma_\ell))$. Then it generates $\mathsf{ct}^i_{\mathsf{IPFE}} := \mathsf{IPFE.Enc}(\mathsf{pp}_{\mathsf{IPFE}}, (K_i, \sigma_i); R_i)$ for $i \in [\ell]$ (where R_i is the randomness used by the encryption algorithm), $\mathsf{ct}'_{\mathsf{IPFE}} := \mathsf{IPFE.Enc}(\mathsf{pp}'_{\mathsf{IPFE}}, (c_1, ..., c_D); R')$ (where R' is the randomness used by the encryption algorithm), and $\pi \xleftarrow{\$} \mathsf{Prove}(\mathsf{crs}, (\mathsf{pp}_{\mathsf{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \mathsf{pp}'_{\mathsf{IPFE}}, \mathsf{ct}'_{\mathsf{IPFE}}),$ $(\{(K_i, \sigma_i, R_i)\}_{i\in[\ell]}, (c_1, ..., c_D, R')))$ and outputs a proof $\pi' := (\pi, \mathsf{ct}_{\mathsf{SKE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \mathsf{ct}'_{\mathsf{IPFE}})$.

$\mathsf{Verify}'(\mathsf{crs}', k'_{\mathsf{V}}, x, \pi')$**:** This algorithm parses $(\mathsf{crs}, \mathsf{pp}_{\mathsf{IPFE}}, \mathsf{pp}'_{\mathsf{IPFE}}) \leftarrow \mathsf{crs}'$, $(k_{\mathsf{V}}, s, \mathsf{sk}_{\mathsf{IPFE}}, \mathsf{sk}'_{\mathsf{IPFE}}) \leftarrow k'_{\mathsf{V}}$, and $(\pi, \mathsf{ct}_{\mathsf{SKE}},$ $\{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \mathsf{ct}'_{\mathsf{IPFE}}) \leftarrow \pi'$, computes $r_i \xleftarrow{\$} \mathsf{IPFE.Dec}(\mathsf{pp}_{\mathsf{IPFE}}, \mathsf{ct}^i_{\mathsf{IPFE}}, \mathsf{sk}_{\mathsf{IPFE}})$ for $i \in [\ell]$ and $t \xleftarrow{\$} \mathsf{IPFE.Dec}(\mathsf{pp}'_{\mathsf{IPFE}},$ $\mathsf{ct}'_{\mathsf{IPFE}}, \mathsf{sk}'_{\mathsf{IPFE}})$, and outputs $\top$ if we have $\mathsf{Verify}(\mathsf{crs}, (\mathsf{pp}_{\mathsf{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \mathsf{pp}'_{\mathsf{IPFE}}, \mathsf{ct}'_{\mathsf{IPFE}}), \pi) = \top$ and

$$f_{x,\mathsf{ct}_{\mathsf{SKE}}}(r_1, ..., r_\ell) = 1 + t \mod p,$$

and outputs $\perp$ otherwise.

Correctness. Suppose that $(\pi, \mathsf{ct}_{\mathsf{SKE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \mathsf{ct}'_{\mathsf{IPFE}})$ is an honestly generated proof on $(x, w) \in \mathcal{R}$. Then it is clear that we have $\mathsf{Verify}(\mathsf{crs}, (\mathsf{pp}_{\mathsf{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \mathsf{pp}'_{\mathsf{IPFE}}, \mathsf{ct}'_{\mathsf{IPFE}}), \pi) = \top$ by the way of generating the proof and the correctness of Π_{DVNIZK}. By the way of generating $(\{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \mathsf{ct}'_{\mathsf{IPFE}})$ and correctness of Π_{IPFE}, we have $r_i = K_i + \sigma_i s \mod p$ for $i \in [\ell]$ and $t = \sum_{j\in[D]} c_j s^j$ where r_i and t are generated as in the verification. Since we have $f_{x,\mathsf{ct}_{\mathsf{SKE}}}(K_1 + \sigma_1 Z, ..., K_\ell + \sigma_\ell Z) = 1 + \sum_{j\in[D]} c_j Z^j$ for an indeterminate Z by the correctness of Π_{SKE} and Lemma 6.4, we have $f_{x,\mathsf{ct}_{\mathsf{SKE}}}(r_1, ..., r_\ell) = 1 + t$ by substituting s for Z.
Proof Size. First, we remark that the relation $\tilde{\mathcal{R}}$ can be verified by a circuit whose size is a fixed polynomial in $(\kappa, \ell, \log p, D)$ that does not depend on $|C|$. Moreover, we have $|\mathsf{Exp}_{C,x}(w)| = |C(x,\cdot)| \le |C|$ for all $w \in \{0,1\}^m$ by Lemma 6.3. Then we have $|\pi| = \mathsf{poly}(\kappa, \ell, \log p, D)$, $|\mathsf{ct}_{\mathsf{SKE}}| = |C(x,\cdot)| + \mathsf{poly}(\kappa)$, $|\mathsf{ct}^i_{\mathsf{IPFE}}| = \mathsf{poly}(\kappa, \log p)$, and $|\mathsf{ct}'_{\mathsf{IPFE}}| = \mathsf{poly}(\kappa, \log p, D)$. By setting $\ell = \kappa$ and $p = 2^{O(\kappa)}$ and remarking that $D = \mathsf{poly}(\kappa)$, we have $|\pi'| = |C(x,\cdot)| + \mathsf{poly}(\kappa) \le |C| + \mathsf{poly}(\kappa)$.
Security. The security of our scheme Π'_{DVNIZK} is stated as follows. The proofs are similar to the security proof for PP-NIZK by Katsumata et al. [KNYY19].

Theorem 6.5 (Soundness.). *If* Π_{DVNIZK} *satisfies statistical (resp. computational) soundness and* $p = \kappa^{\omega(1)}$*, then* Π'_{DVNIZK} *satisfies statistical (resp. computational) soundness.*

Proof. We prove the case of the statistical soundness. The case of the computational soundness can be proven similarly. Suppose that there is an unbounded-time adversary $\mathcal{A}$ that breaks soundness. We consider the following sequence of games.

Game_0**:** This game is the original experiment that defines the soundness. Specifically, the game is described as follows.

1. The challenger picks $s \xleftarrow{\$} \mathbb{Z}_p^*$, generates $(\mathsf{crs}, k_{\mathsf{V}}) \xleftarrow{\$} \mathsf{Setup}(1^\kappa)$, $(\mathsf{pp}_{\mathsf{IPFE}}, \mathsf{msk}_{\mathsf{IPFE}}) \xleftarrow{\$} \mathsf{IPFE.Setup}(1^\kappa, 1^2)$, $(\mathsf{pp}'_{\mathsf{IPFE}}, \mathsf{msk}'_{\mathsf{IPFE}}) \xleftarrow{\$} \mathsf{IPFE.Setup}(1^\kappa, 1^D)$, $\mathsf{sk}_{\mathsf{IPFE}} \xleftarrow{\$} \mathsf{IPFE.KeyGen}(\mathsf{msk}_{\mathsf{IPFE}}, (1, s))$, and $\mathsf{sk}'_{\mathsf{IPFE}} \xleftarrow{\$} \mathsf{IPFE.KeyGen}(\mathsf{msk}'_{\mathsf{IPFE}}, (s, ..., s^D))$, sets $\mathsf{crs}' := (\mathsf{crs}, \mathsf{pp}_{\mathsf{IPFE}}, \mathsf{pp}'_{\mathsf{IPFE}})$ and $k'_{\mathsf{V}} := (k_{\mathsf{V}}, s, \mathsf{sk}_{\mathsf{IPFE}}, \mathsf{sk}'_{\mathsf{IPFE}})$, and gives crs' to $\mathcal{A}$.
2. $\mathcal{A}$ can make arbitrary number of verification queries. When $\mathcal{A}$ queries $(x, \pi' = (\pi, \mathsf{ct}_{\mathsf{SKE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \mathsf{ct}'_{\mathsf{IPFE}}))$, the challenger returns $\mathsf{Verify}'(\mathsf{crs}', k'_{\mathsf{V}}, x, \pi')$. More specifically, the challenger computes $r_i \xleftarrow{\$} \mathsf{IPFE.Dec}(\mathsf{pp}_{\mathsf{IPFE}}, \mathsf{ct}^i_{\mathsf{IPFE}}, \mathsf{sk}_{\mathsf{IPFE}})$ for $i \in [\ell]$ and $t \xleftarrow{\$} \mathsf{IPFE.Dec}(\mathsf{pp}'_{\mathsf{IPFE}}, \mathsf{ct}'_{\mathsf{IPFE}}, \mathsf{sk}'_{\mathsf{IPFE}})$, and outputs $\top$ if we have $\mathsf{Verify}(\mathsf{crs}, (\mathsf{pp}_{\mathsf{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \mathsf{pp}'_{\mathsf{IPFE}}, \mathsf{ct}'_{\mathsf{IPFE}}), \pi) = \top$ and $f_{x,\mathsf{ct}_{\mathsf{SKE}}}(r_1, ..., r_\ell) = 1 + t \mod p$, and outputs $\bot$ otherwise.
3. Finally, $\mathcal{A}$ outputs (x^*, π^*). We say that $\mathcal{A}$ wins if $x^* \notin \mathcal{L}$ and $\mathsf{Verify}(\mathsf{crs}', k'_{\mathsf{V}}, x^*, \pi^*) = \top$.

Game_1**:** This game is the same as the previous game except that the verification oracle $\mathsf{Verify}'(\mathsf{crs}', k'_{\mathsf{V}}, \cdot, \cdot)$ is modified to the following alternative oracle denoted by $\mathsf{Verify}'_1(\cdot, \cdot)$. The oracle $\mathsf{Verify}'_1(\cdot, \cdot)$ does the following:

- Given a query $(x, \pi' = (\pi, \mathsf{ct}_{\mathsf{SKE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \mathsf{ct}'_{\mathsf{IPFE}}))$.
- If $\mathsf{Verify}(\mathsf{crs}, \mathsf{pp}_{\mathsf{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \mathsf{pp}'_{\mathsf{IPFE}}, \mathsf{ct}'_{\mathsf{IPFE}}), \pi) = \bot$, returns $\bot$. Otherwise, proceeds to the next step.
- It computes $(K_i, \sigma_i) := \mathsf{MasterDec}(\mathsf{msk}_{\mathsf{IPFE}}, \mathsf{ct}^i_{\mathsf{IPFE}})$ for $i \in [\ell]$ and $(c_1, ..., c_D) := \mathsf{MasterDec}(\mathsf{msk}'_{\mathsf{IPFE}}, \mathsf{ct}'_{\mathsf{IPFE}})$, where $\mathsf{MasterDec}$ is the master decryption algorithm for Π_{IPFE} as defined in Section 6.1. If $K = (K_1, ..., K_\ell) \notin \{0,1\}^\ell$, returns $\bot$. Otherwise, proceeds to the next step.
- If we have
$$f_{x,\mathsf{ct}_{\mathsf{SKE}}}(K_1 + \sigma_1 s, ..., K_\ell + \sigma_\ell s) = 1 + \sum_{j\in[D]} c_j s^j \mod p,$$
outputs $\top$ and returns $\bot$ otherwise.

We note that the winning condition of $\mathcal{A}$ is also modified to use $\mathsf{Verify}'_1(\cdot, \cdot)$ instead of $\mathsf{Verify}'(\mathsf{crs}, k_{\mathsf{V}}, \cdot, \cdot)$.

Game_2**:** This game is the same as the previous game except that the oracle Verify'_1 is replaced with Verify'_2 described in the following. The main differences from Verify'_1 are highlighted by red underlines. The oracle $\mathsf{Verify}'_2(\cdot, \cdot)$ does the following:

- Given a query $(x, \pi' = (\pi, \mathsf{ct}_{\mathsf{SKE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \mathsf{ct}'_{\mathsf{IPFE}}))$.
- If $\mathsf{Verify}(\mathsf{crs}, \mathsf{pp}_{\mathsf{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \mathsf{pp}'_{\mathsf{IPFE}}, \mathsf{ct}'_{\mathsf{IPFE}}), \pi) = \bot$, returns $\bot$. Otherwise, proceeds to the next step.
- It computes $(K_i, \sigma_i) := \mathsf{MasterDec}(\mathsf{msk}_{\mathsf{IPFE}}, \mathsf{ct}^i_{\mathsf{IPFE}})$ for $i \in [\ell]$ and $(c_1, ..., c_D) := \mathsf{MasterDec}(\mathsf{msk}'_{\mathsf{IPFE}}, \mathsf{ct}'_{\mathsf{IPFE}})$. If $K = (K_1, ..., K_\ell) \notin \{0,1\}^\ell$, returns $\bot$. Otherwise, proceeds to the next step.
- $\underline{\text{It computes } (\hat{c}_1, ..., \hat{c}_D) \leftarrow \mathsf{Coefficient}(1^D, p, f_{x,\mathsf{ct}_{\mathsf{SKE}}}, (K_1, ..., K_\ell), (\sigma_1, ..., \sigma_\ell))}$.
- If we have
$$\underline{\sum_{j=1}^{D} (c_j - \hat{c}_j) s^j + 1 - f_{x,\mathsf{ct}_{\mathsf{SKE}}}(K_1, ..., K_\ell) = 0 \mod p},$$
it outputs $\top$ and returns $\bot$ otherwise.

We note that the winning condition of $\mathcal{A}$ is also modified to use Verify'_2 instead of Verify'_1.

Game_3**:** This game is the same as the previous game except that the oracle Verify'_2 is replaced with an alternative oracle Verify'_3 described below. The main differences from Verify'_2 are highlighted by red underlines. The oracle $\mathsf{Verify}'_3(\cdot, \cdot)$ does the following:

- Given a query $(x, \pi' = (\pi, \mathsf{ct}_{\mathsf{SKE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \mathsf{ct}'_{\mathsf{IPFE}}))$.
- If $\mathsf{Verify}(\mathsf{crs}, \mathsf{pp}_{\mathsf{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \mathsf{pp}'_{\mathsf{IPFE}}, \mathsf{ct}'_{\mathsf{IPFE}}), \pi) = \bot$, then returns $\bot$. Otherwise, proceeds to the next step.
- It computes $(K_i, \sigma_i) := \mathsf{MasterDec}(\mathsf{msk}_{\mathsf{IPFE}}, \mathsf{ct}^i_{\mathsf{IPFE}})$ for $i \in [\ell]$ and $(c_1, ..., c_D) := \mathsf{MasterDec}(\mathsf{msk}'_{\mathsf{IPFE}}, \mathsf{ct}'_{\mathsf{IPFE}})$. If $K = (K_1, ..., K_\ell) \notin \{0,1\}^\ell$, returns $\bot$. Otherwise, proceeds to the next step.
- It computes $(\hat{c}_1, ..., \hat{c}_D) \leftarrow \mathsf{Coefficient}(1^D, p, f_{x,\mathsf{ct}_{\mathsf{SKE}}}, (K_1, ..., K_\ell), (\sigma_1, ..., \sigma_\ell))$.
- If we have $f_{x,\mathsf{ct}_{\mathsf{SKE}}}(K_1, ..., K_\ell) = 1$ and $(c_1, ..., c_D) = (\hat{c}_1, ..., \hat{c}_D)$, it outputs $\top$, and returns $\bot$ otherwise.

We note that the winning condition of $\mathcal{A}$ is also modified to use Verify'_3 instead of Verify'_2.

This completes the description of games. We denote the event that $\mathcal{A}$ wins in Game_k by T_k for $k = 0, ..., 3$. We prove the following lemmas.

Lemma 6.6. *If* Π_{DVNIZK} *satisfies statistical soundness, then we have* $|\Pr[\mathsf{T}_0] - \Pr[\mathsf{T}_1]| \le \mathsf{negl}(\kappa)$.

Proof. Let F be the event that $\mathcal{A}$ ever makes a query $(x, \pi' = (\pi, \mathsf{ct}_{\mathsf{SKE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \mathsf{ct}'_{\mathsf{IPFE}}))$ to the verification oracle such that $\mathsf{Verify}(\mathsf{crs}, \mathsf{pp}_{\mathsf{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \mathsf{pp}'_{\mathsf{IPFE}}, \mathsf{ct}'_{\mathsf{IPFE}}), \pi) = \top$ and $((\mathsf{pp}_{\mathsf{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \mathsf{pp}'_{\mathsf{IPFE}}, \mathsf{ct}'_{\mathsf{IPFE}}), (\{(K_i, \sigma_i, R_i)\}_{i\in[\ell]}, (c_1, ..., c_D, R'))) \notin \tilde{\mathcal{R}}$. It is easy to see that $\Pr[\mathsf{F}]$ is negligible if Π_{DVNIZK} satisfies statistical soundness. If we have $\mathsf{Verify}(\mathsf{crs}, \mathsf{pp}_{\mathsf{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \mathsf{pp}'_{\mathsf{IPFE}}, \mathsf{ct}'_{\mathsf{IPFE}}), \pi) = \top$ and F does not happen, then each ciphertext $\mathsf{ct}^i_{\mathsf{IPFE}}$ is a valid encryption of some $(K_i, \sigma_i) \in \{0,1\} \times \mathbb{Z}_p$ and $\mathsf{ct}'_{\mathsf{IPFE}}$ is a valid encryption of some $(c_1, ..., c_D) \in \mathbb{Z}_p^D$. Then if we let $(K_i, \sigma_i) := \mathsf{MasterDec}(\mathsf{msk}_{\mathsf{IPFE}}, \mathsf{ct}^i_{\mathsf{IPFE}})$ for $i \in [\ell]$, $(c_1, ..., c_D) := \mathsf{MasterDec}(\mathsf{msk}'_{\mathsf{IPFE}}, \mathsf{ct}'_{\mathsf{IPFE}})$, $r_i \xleftarrow{\$} \mathsf{IPFE.Dec}(\mathsf{pp}_{\mathsf{IPFE}}, \mathsf{ct}^i_{\mathsf{IPFE}}, \mathsf{sk}_{\mathsf{IPFE}})$ for $i \in [\ell]$, and $t \xleftarrow{\$} \mathsf{IPFE.Dec}(\mathsf{pp}'_{\mathsf{IPFE}}, \mathsf{ct}'_{\mathsf{IPFE}}, \mathsf{sk}'_{\mathsf{IPFE}})$, then we have $K = (K_1, ..., K_\ell) \in \{0,1\}^\ell$, $r_i = K_i + \sigma_i s \mod p$ for $i \in [\ell]$ and $t = \sum_{j\in[D]} c_j s^j \mod p$, and the conditions

$$f_{x,\mathsf{ct}_{\mathsf{SKE}}}(r_1, ..., r_\ell) = 1 + t \mod p,$$

and

$$f_{x,\mathsf{ct}_{\mathsf{SKE}}}(K_1 + \sigma_1 s, ..., K_\ell + \sigma_\ell s) = 1 + \sum_{j\in[D]} c_j s^j \mod p$$

are equivalent. Therefore we have $|\Pr[\mathsf{T}_0] - \Pr[\mathsf{T}_1]| \le \Pr[\mathsf{F}]$, and the lemma is proven. □

Lemma 6.7. $\Pr[\mathsf{T}_1] = \Pr[\mathsf{T}_2]$.

Proof. Game_1 and Game_2 are identical from $\mathcal{A}$'s view since the responses from the verification oracle never differ as seen below: By the definition of $\mathsf{Coefficient}$, we have

$$f_{x,\mathsf{ct}_{\mathsf{SKE}}}(K_1 + \sigma_1 s, ..., K_\ell + \sigma_\ell s) = f_{x,\mathsf{ct}_{\mathsf{SKE}}}(K_1, ..., K_\ell) + \sum_{j\in[D]} \hat{c}_j s^j \mod p.$$

Therefore the equation

$$f_{x,\mathsf{ct}_{\mathsf{SKE}}}(K_1 + \sigma_1 s, ..., K_\ell + \sigma_\ell s) = 1 + \sum_{j\in[D]} c_j s^j \mod p$$

is equivalent to

$$\sum_{j=1}^{D} (c_j - \hat{c}_j) s^j + 1 - f_{x,\mathsf{ct}_{\mathsf{SKE}}}(K_1, ..., K_\ell) = 0 \mod p.$$

□

Lemma 6.8. $|\Pr[\mathsf{T}_2] - \Pr[\mathsf{T}_3]| \le \frac{(Q+1)D}{p-1}$ *where* Q *denotes the number of verification queries.*

Proof. Here, we regard the final output (x^*, π'^*) of $\mathcal{A}$ as the $(Q+1)$-th query for notational convenience. We consider hybrids H_k for $k = 0, 1, ..., Q+1$, which is the same as Game_2 except that Verify'_3 is used until $\mathcal{A}$'s k-th query and Verify'_2 is used for the rest of the queries. Let T'_k be the event that $\mathcal{A}$ wins in H_k. It is clear that H_0 is Game_2 and H_{Q+1} is Game_3. Thus what we have to prove is that we have $|\Pr[\mathsf{T}'_k] - \Pr[\mathsf{T}'_{k+1}]| \leq \frac{D}{p-1}$ for $k = 0, ..., Q$. Let Bad_k be the event that $\mathcal{A}$'s k-th query causes the difference between H_k and H_{k+1}, i.e., $f_{x,\mathsf{ct_{SKE}}}(K_1, ..., K_\ell) \neq 1$ or $(c_1, ..., c_D) \neq (\hat{c}_1, ..., \hat{c}_D)$ and $\sum_{j=1}^{D}(c_j - \hat{c}_j)s^j + 1 - f_{x,\mathsf{ct_{SKE}}}(K_1, ..., K_\ell) \mod p = 0$. Since H_k and H_{k+1} are completely the same games as long as Bad_{k+1} does not occur, we have $|\Pr[\mathsf{T}'_k] - \Pr[\mathsf{T}'_{k+1}]| \leq \Pr[\mathsf{Bad}_{k+1}]$. If $f_{x,\mathsf{ct_{SKE}}}(K_1, ..., K_\ell) \neq 1$ or $(c_1, ..., c_D) \neq (\hat{c}_1, ..., \hat{c}_D)$, then $\sum_{j=1}^{D}(c_j - \hat{c}_j)s^j + 1 - f_{x,\mathsf{ct_{SKE}}}(K_1, ..., K_\ell)$ is a non-zero polynomial in s of degree at most D, which has at most D roots. Moreover, in H_k and H_{k+1}, no information of s is used before $\mathcal{A}$ makes its $k+1$-th query since s is not used at all until this point. This means that s is uniformly distributed on $\mathbb{Z}_p^*$ from the view of $\mathcal{A}$. Therefore regardless of $\mathcal{A}$'s $k+1$-th query, the probability that $\sum_{j=1}^{D}(c_j - \hat{c}_j)s^j + 1 - f_{x,\mathsf{ct_{SKE}}}(K_1, ..., K_\ell) = 0 \mod p$ is at most $\frac{D}{p-1}$. □

Lemma 6.9. $\Pr[\mathsf{T}_3] = 0$.

Proof. $\mathcal{A}$ has no chance to win Game_3 since if $x^* \notin \mathcal{L}$, $f_{x,\mathsf{ct_{SKE}}}$ never outputs 1 on any input $K \in \{0,1\}^\ell$ for any $\mathsf{ct_{SKE}} \in \mathcal{CT}$. □

This completes the proof of Theorem 6.5. □

Theorem 6.10 (Zero-knowledge). *If* SKE *is CPA secure,* Π_{IPFE} *is adaptively single-key secure, and* Π_{DVNIZK} *satisfies zero-knowledge, then* Π'_{DVNIZK} *satisfies zero-knowledge.*

Proof. Let $(\mathcal{S}_1, \mathcal{S}_2)$ be the simulator for Π_{DVNIZK}. We describe the simulator $(\mathcal{S}_1, \mathcal{S}_2)$ for Π'_{DVNIZK} below.

$\mathcal{S}'_1(1^\kappa)$**:** It picks $s \xleftarrow{\$} \mathbb{Z}_p^*$ and generates $(\mathsf{pp_{IPFE}}, \mathsf{msk_{IPFE}}) \xleftarrow{\$} \mathsf{IPFE.Setup}(1^\kappa, 1^2)$, $(\mathsf{pp'_{IPFE}}, \mathsf{msk'_{IPFE}}) \xleftarrow{\$} \mathsf{IPFE.Setup}(1^\kappa, 1^D)$, $\mathsf{sk_{IPFE}} \xleftarrow{\$} \mathsf{IPFE.KeyGen}(\mathsf{msk_{IPFE}}, (1, s))$, $\mathsf{sk'_{IPFE}} \xleftarrow{\$} \mathsf{IPFE.KeyGen}(\mathsf{msk'_{IPFE}}, (s, ..., s^D))$, $(\overline{\mathsf{crs}}, \bar{k}_{\mathsf{V}}, \bar{\tau}) \xleftarrow{\$} \mathcal{S}_1(1^\kappa)$, and $K \xleftarrow{\$} \mathsf{SKE.KeyGen}(1^\kappa)$, and outputs $(\overline{\mathsf{crs}}' := (\overline{\mathsf{crs}}, \mathsf{pp_{IPFE}}, \mathsf{pp'_{IPFE}}), \bar{k}'_{\mathsf{V}} := (\bar{k}_{\mathsf{V}}, s, \mathsf{sk_{IPFE}}, \mathsf{sk'_{IPFE}}), \bar{\tau}' = (\bar{\tau}, K))$.

$\mathcal{S}'_2(\overline{\mathsf{crs}}' = (\overline{\mathsf{crs}}, \mathsf{pp_{IPFE}}, \mathsf{pp'_{IPFE}}), \bar{k}'_{\mathsf{V}} = (\bar{k}_{\mathsf{V}}, s, \mathsf{sk_{IPFE}}, \mathsf{sk'_{IPFE}}), \bar{\tau}' = (\bar{\tau}, K), x)$**:** It picks $r_i \xleftarrow{\$} \mathbb{Z}_p$ for $i \in [\ell]$ computes $\mathsf{ct_{SKE}} \xleftarrow{\$} \mathsf{SKE.Enc}(K, 0^{|C(x,\cdot)|})$, $t := f_{x,\mathsf{ct_{SKE}}}(r_1, ..., r_\ell) - 1 \mod p$, $\mathsf{ct}^i_{\mathsf{IPFE}} \xleftarrow{\$} \mathsf{IPFE.Enc}(\mathsf{pp_{IPFE}}, (r_i, 0))$ for $i \in [\ell]$, $\mathsf{ct'_{IPFE}} \xleftarrow{\$} \mathsf{IPFE.Enc}(\mathsf{pp'_{IPFE}}, (t \cdot s^{-1}, 0, ..., 0))$, and $\pi \xleftarrow{\$} \mathcal{S}_2(\overline{\mathsf{crs}}, \bar{k}_{\mathsf{V}}, \bar{\tau}, (\mathsf{pp_{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \mathsf{pp'_{IPFE}}, \mathsf{ct'_{IPFE}}))$ and outputs a proof $\pi' := (\pi, \mathsf{ct_{SKE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \mathsf{ct'_{IPFE}})$.

This completes the description of the simulator. We prove that proofs simulated by the above simulator are computationally indistinguishable from the honestly generated proofs. To prove this, we consider the following sequence of games between a PPT adversary $\mathcal{A}$ and a challenger.

Game_0**:** In this game, proofs are generated honestly. Namely,

1. The challenger picks $s \xleftarrow{\$} \mathbb{Z}_p^*$ and generates $(\mathsf{crs}, k_{\mathsf{V}}) \xleftarrow{\$} \mathsf{Setup}(1^\kappa)$, $(\mathsf{pp_{IPFE}}, \mathsf{msk_{IPFE}}) \xleftarrow{\$} \mathsf{IPFE.Setup}(1^\kappa, 1^2)$, $(\mathsf{pp'_{IPFE}}, \mathsf{msk'_{IPFE}}) \xleftarrow{\$} \mathsf{IPFE.Setup}(1^\kappa, 1^D)$, $\mathsf{sk_{IPFE}} \xleftarrow{\$} \mathsf{IPFE.KeyGen}(\mathsf{msk_{IPFE}}, (1, s))$, and $\mathsf{sk'_{IPFE}} \xleftarrow{\$} \mathsf{IPFE.KeyGen}(\mathsf{msk'_{IPFE}}, (s, ..., s^D))$. It defines a common reference string $\mathsf{crs}' := (\mathsf{crs}, \mathsf{pp_{IPFE}}, \mathsf{pp'_{IPFE}})$ and a verifier key $k'_{\mathsf{V}} := (k_{\mathsf{V}}, s, \mathsf{sk_{IPFE}}, \mathsf{sk'_{IPFE}})$.
2. $\mathcal{A}$ is given $(1^\kappa, \mathsf{crs}', k'_{\mathsf{V}})$, and allowed to query $\mathcal{O}(\mathsf{crs}, \cdot, \cdot)$, which works as follows. When $\mathcal{A}$ queries (x, w), if $(x, w) \notin \mathcal{R}$, then the oracle returns $\perp$. Otherwise, it picks $K \xleftarrow{\$} \mathsf{SKE.KeyGen}(1^\kappa)$ and $\sigma_i \xleftarrow{\$} \mathbb{Z}_p$ for $i \in [\ell]$, and generates $\mathsf{ct_{SKE}} \xleftarrow{\$} \mathsf{SKE.Enc}(K, \mathsf{Exp}_{C,x}(w))$, $(c_1, ..., c_D) \leftarrow \mathsf{Coefficient}(1^D, p, f_{x,\mathsf{ct_{SKE}}}, K = (K_1, ..., K_\ell), (\sigma_1, ..., \sigma_\ell))$, $\mathsf{ct}^i_{\mathsf{IPFE}} \xleftarrow{\$} \mathsf{IPFE.Enc}(\mathsf{pp_{IPFE}}, (K_i, \sigma_i); R_i)$ for $i \in [\ell]$ (where R_i is the randomness used by the encryption algorithm), $\mathsf{ct'_{IPFE}} \xleftarrow{\$} \mathsf{IPFE.Enc}(\mathsf{pp'_{IPFE}}, (c_1, ..., c_D); R')$ (where R' is the randomness used by the encryption algorithm), and $\pi \xleftarrow{\$} \mathsf{Prove}(\mathsf{crs}, \mathsf{pp_{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \mathsf{pp'_{IPFE}}, \mathsf{ct'_{IPFE}}), (\{(K_i, \sigma_i, R_i)\}_{i\in[\ell]}, (c_1, ..., c_D, R')))$ and returns a proof $\pi' := (\pi, \mathsf{ct_{SKE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \mathsf{ct'_{IPFE}})$.

3. Finally, $\mathcal{A}$ returns a bit β.

Game_1**:** This game is the same as the previous game except that crs, k_{V}, and π are generated differently. Namely, the challenger generates $(\mathsf{crs}, k_{\mathsf{V}}, \tau) \xleftarrow{\$} \mathcal{S}_1(1^\kappa)$ at the beginning of the game, and π is generated as $\pi \xleftarrow{\$} \mathcal{S}_2(\mathsf{crs}, k_{\mathsf{V}}, \tau, \mathsf{pp}_{\mathsf{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \mathsf{pp}'_{\mathsf{IPFE}}, \mathsf{ct}'_{\mathsf{IPFE}}))$ for each oracle query.

Game_2**:** This game is the same as the previous game except that $\{\mathsf{ct}_{\mathsf{IPFE}}\}_{i\in[\ell]}$ is generated differently when responding to each query. Namely, the oracle computes $r_i := K_i + \sigma_i s \mod p$ and $\mathsf{ct}^i_{\mathsf{IPFE}} \xleftarrow{\$} \mathsf{IPFE.Enc}(\mathsf{pp}_{\mathsf{IPFE}}, (r_i, 0))$ for $i \in [\ell]$.

Game_3**:** This game is the same as the previous game except that $\mathsf{ct}'_{\mathsf{IPFE}}$ is generated differently when responding to each query. Namely, the oracle computes $t := f_{x,\mathsf{ct}_{\mathsf{SKE}}}(r_1, ..., r_\ell) - 1 \mod p$ and $\mathsf{ct}'_{\mathsf{IPFE}} \xleftarrow{\$} \mathsf{IPFE.Enc}(\mathsf{pp}'_{\mathsf{IPFE}}, (t \cdot s^{-1}, 0, ..., 0))$. We note that in this game, $(c_1, ..., c_D)$ is need not be computed since it is not used for generating $\mathsf{ct}'_{\mathsf{IPFE}}$.

Game_4**:** This game is the same as the previous game except that r_i is randomly chosen from $\mathbb{Z}_p$ for $i \in [\ell]$ in each query.

Game_5**:** This game is the same as the previous game except that $\mathsf{ct}_{\mathsf{SKE}}$ is generated as $\mathsf{ct}_{\mathsf{SKE}} \xleftarrow{\$} \mathsf{SKE.Enc}(K, 0^{|C(x,\cdot)|})$.

Let T_i be the event that $\mathcal{A}$ returns 1 in Game_i for $i \in \{0, 1, 2, 3, 4, 5\}$. It is easy to see that proofs are generated by $\mathcal{S}' = (\mathcal{S}'_1, \mathcal{S}'_2)$ in Game_5. Thus we have to prove that $|\Pr[\mathsf{T}_0] - \Pr[\mathsf{T}_5]|$ is negligible. We prove this by the following lemmas.

Lemma 6.11. *If* Π_{DVNIZK} *satisfies zero-knowledge w.r.t. the simulator* $\mathcal{S}$*, then* $|\Pr[\mathsf{T}_0] - \Pr[\mathsf{T}_1]| = \mathsf{negl}(\kappa)$.

Proof. We assume that $|\Pr[\mathsf{T}_0] - \Pr[\mathsf{T}_1]|$ is non-negligible, and construct a PPT adversary $\mathcal{B}$ that breaks the zero-knowledge of Π_{DVNIZK}. The description of $\mathcal{B}$ is given below.

$\mathcal{B}^{\mathcal{O}(\cdot,\cdot)}(\mathsf{crs}, k_{\mathsf{V}})$**:** It picks $s \xleftarrow{\$} \mathbb{Z}_p^*$ and generates $(\mathsf{pp}_{\mathsf{IPFE}}, \mathsf{msk}_{\mathsf{IPFE}}) \xleftarrow{\$} \mathsf{IPFE.Setup}(1^\kappa, 1^2)$, $(\mathsf{pp}'_{\mathsf{IPFE}}, \mathsf{msk}'_{\mathsf{IPFE}}) \xleftarrow{\$} \mathsf{IPFE.Setup}(1^\kappa, 1^D)$, $\mathsf{sk}_{\mathsf{IPFE}} \xleftarrow{\$} \mathsf{IPFE.KeyGen}(\mathsf{msk}_{\mathsf{IPFE}}, (1, s))$, and $\mathsf{sk}'_{\mathsf{IPFE}} \xleftarrow{\$} \mathsf{IPFE.KeyGen}(\mathsf{msk}'_{\mathsf{IPFE}}, (s, ..., s^D))$. It defines a common reference string $\mathsf{crs}' := (\mathsf{crs}, \mathsf{pp}_{\mathsf{IPFE}}, \mathsf{pp}'_{\mathsf{IPFE}})$ and a verifier key $k'_{\mathsf{V}} := (k_{\mathsf{V}}, s, \mathsf{sk}_{\mathsf{IPFE}}, \mathsf{sk}'_{\mathsf{IPFE}})$, and runs $\mathcal{A}^{\mathcal{O}'(\cdot,\cdot)}(\mathsf{crs}', k'_{\mathsf{V}})$ where $\mathcal{B}$ simulates $\mathcal{O}'(\cdot,\cdot)$ as follows. When $\mathcal{A}$ makes a query (x, w) to $\mathcal{O}'(\cdot)$, $\mathcal{B}$ picks $K \xleftarrow{\$} \mathsf{SKE.KeyGen}(1^\kappa)$ and $\sigma_i \xleftarrow{\$} \mathbb{Z}_p$ for $i \in [\ell]$, generates $\mathsf{ct}_{\mathsf{SKE}} \xleftarrow{\$} \mathsf{SKE.Enc}(K, \mathsf{Exp}_{C,x}(w))$, $(c_1, ..., c_D) \leftarrow \mathsf{Coefficient}(1^D, p, f_{x,\mathsf{ct}_{\mathsf{SKE}}}, K = (K_1, ..., K_\ell), (\sigma_1, ..., \sigma_\ell))$, $\mathsf{ct}^i_{\mathsf{IPFE}} \xleftarrow{\$} \mathsf{IPFE.Enc}(\mathsf{pp}_{\mathsf{IPFE}}, (K_i, \sigma_i); R_i)$ for $i \in [\ell]$ (where R_i is the randomness used by the encryption algorithm), and $\mathsf{ct}'_{\mathsf{IPFE}} \xleftarrow{\$} \mathsf{IPFE.Enc}(\mathsf{pp}'_{\mathsf{IPFE}}, (c_1, ..., c_D); R')$ (where R' is the randomness used by the encryption algorithm), and queries $(\mathsf{pp}_{\mathsf{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \mathsf{pp}'_{\mathsf{IPFE}}, \mathsf{ct}'_{\mathsf{IPFE}})$, $(\{(K_i, \sigma_i, R_i)\}_{i\in[\ell]}, (c_1, ..., c_D, R')))$ to $\mathcal{O}(\cdot,\cdot)$ to obtain π (if $\perp$ is returned, then $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$ as a response by $\mathcal{O}'(\cdot,\cdot)$). Then it returns a proof $\pi' := (\pi, \mathsf{ct}_{\mathsf{SKE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \mathsf{ct}'_{\mathsf{IPFE}})$.

This completes the description of $\mathcal{B}$. It is easy to see that if $\mathcal{O}$ generates proofs honestly, then $\mathcal{B}$ perfectly simulates Game_0, and if $\mathcal{O}$ generates proofs by using the simulator, then $\mathcal{B}$ perfectly simulates Game_1. Therefore if $\mathcal{A}$ distinguishes these two games with non-negligible probability, then $\mathcal{B}$ succeeds in distinguishing these two cases, which means it breaks the zero-knowledge. □

Lemma 6.12. *If* IPFE *is adaptively single-key secure, then* $|\Pr[\mathsf{T}_1] - \Pr[\mathsf{T}_2]| = \mathsf{negl}(\kappa)$.

Proof. Suppose that $|\Pr[\mathsf{T}_1] - \Pr[\mathsf{T}_2]|$ is non-negligible. We construct an adversary $\mathcal{B}$ that breaks the multi-challenge adaptive single-key security of Π_{IPFE} as follows.

$\mathcal{B}(1^\kappa)$**:** It declares the dimension 1^2 and obtains a public parameter $\mathsf{pp}_{\mathsf{IPFE}}$ for Π_{IPFE} with dimension 2. It picks $s \xleftarrow{\$} \mathbb{Z}_p^*$, queries a vector $(1, s)$ to its key generation oracle to obtain $\mathsf{sk}_{\mathsf{IPFE}}$. Then it generates $(\mathsf{crs}, k_{\mathsf{V}}, \tau) \xleftarrow{\$} \mathcal{S}_1(1^\kappa)$, $(\mathsf{pp}'_{\mathsf{IPFE}}, \mathsf{msk}'_{\mathsf{IPFE}}) \xleftarrow{\$} \mathsf{IPFE.Setup}(1^\kappa, 1^D)$, and $\mathsf{sk}'_{\mathsf{IPFE}} \xleftarrow{\$} \mathsf{IPFE.KeyGen}(\mathsf{msk}'_{\mathsf{IPFE}}, (s, ..., s^D))$. It sets a common reference string $\mathsf{crs}' := (\mathsf{crs}, \mathsf{pp}_{\mathsf{IPFE}}, \mathsf{pp}'_{\mathsf{IPFE}})$ and a verifier key $k'_{\mathsf{V}} := (k_{\mathsf{V}}, s, \mathsf{sk}_{\mathsf{IPFE}}, \mathsf{sk}'_{\mathsf{IPFE}})$, and gives $(1^\kappa, \mathsf{crs}', k'_{\mathsf{V}})$ to $\mathcal{A}$ as input. When $\mathcal{A}$ makes a query (x, w), $\mathcal{B}$ returns $\perp$ if $(x, w) \notin \mathcal{R}$. Otherwise it picks $K \xleftarrow{\$} \mathsf{SKE.KeyGen}(1^\kappa)$ and $\sigma_i \xleftarrow{\$} \mathbb{Z}_p$ for $i \in [\ell]$, computes $r_i := K_i + \sigma_i s \mod p$ for $i \in [\ell]$, and generates $\mathsf{ct}_{\mathsf{SKE}} \xleftarrow{\$} \mathsf{SKE.Enc}(K, \mathsf{Exp}_{C,x}(w))$, $(c_1, ..., c_D) \leftarrow \mathsf{Coefficient}(1^D, p, f_{x,\mathsf{ct}_{\mathsf{SKE}}}, K = (K_1, ..., K_\ell), (\sigma_1, ..., \sigma_\ell))$, and $\mathsf{ct}'_{\mathsf{IPFE}} \xleftarrow{\$} \mathsf{IPFE.Enc}(\mathsf{pp}'_{\mathsf{IPFE}}, (c_1, ..., c_D))$. Then $\mathcal{B}$ queries pairs of vectors $\mathbf{x}_i^{(0)} := (K_i, \sigma_i)$ and $\mathbf{x}_i^{(1)} := (r_i, 0)$ for $i \in [\ell]$ to its challenge oracle to obtain ciphertexts $\{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i \in [\ell]}$. Then it generates $\pi \xleftarrow{\$} \mathcal{S}_2(\mathsf{crs}, k_{\mathsf{V}}, \tau, (\mathsf{pp}_{\mathsf{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i \in [\ell]}, \mathsf{pp}'_{\mathsf{IPFE}}, \mathsf{ct}'_{\mathsf{IPFE}}))$, and returns a proof $\pi' := (\pi, \mathsf{ct}_{\mathsf{SKE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i \in [\ell]}, \mathsf{ct}'_{\mathsf{IPFE}})$ to $\mathcal{A}$. Finally, when $\mathcal{A}$ outputs a bit coin', $\mathcal{B}'$ also outputs coin'.

This completes the construction of $\mathcal{B}$. First, we remark that $\mathcal{B}$ is a valid adversary against the multi-challenge adaptive single-key security of Π_{IPFE} since we have $(K_i, \sigma_i) \cdot (1, s)^T = (r_i, 0) \cdot (1, s)^T \mod p$ for $i \in [\ell]$. It is easy to see that $\mathcal{B}$ perfectly simulates Game_1 to $\mathcal{A}$ if the coin picked by the IPFE challenger is 0 and it perfectly simulates Game_2 otherwise. Therefore we have $|\Pr[\mathsf{T}_2] - \Pr[\mathsf{T}_1]| \leq \mathsf{negl}(\kappa)$ if Π_{IPFE} satisfies the multi-challenge adaptive single-key security, which follows from the single-challenge version of it as remarked in Remark 6.1.

□

Lemma 6.13. *If* IPFE *is adaptively single-key secure, then* $|\Pr[\mathsf{T}_2] - \Pr[\mathsf{T}_3]| = \mathsf{negl}(\kappa)$.

Proof. Suppose that $|\Pr[\mathsf{T}_2] - \Pr[\mathsf{T}_3]|$ is non-negligible. We construct an adversary $\mathcal{B}$ that breaks the multi-challenge adaptive single-key security of Π_{IPFE} as follows.

$\mathcal{B}(1^\kappa)$**:** First, it declares the dimension D to obtain a public parameter $\mathsf{pp}'_{\mathsf{IPFE}}$ for Π_{IPFE} with dimension D. Then it picks $s \xleftarrow{\$} \mathbb{Z}_p^*$ and queries $(s, ..., s^D)$ to its key generation oracle to obtain a key $\mathsf{sk}'_{\mathsf{IPFE}}$. Then it generates $(\mathsf{crs}, k_{\mathsf{V}}, \tau) \xleftarrow{\$} \mathcal{S}_1(1^\kappa)$, $(\mathsf{pp}_{\mathsf{IPFE}}, \mathsf{msk}_{\mathsf{IPFE}}) \xleftarrow{\$} \mathsf{IPFE.Setup}(1^\kappa, 1^2)$, and $\mathsf{sk}_{\mathsf{IPFE}} \xleftarrow{\$} \mathsf{IPFE.KeyGen}(\mathsf{msk}_{\mathsf{IPFE}}, (1, s))$. It sets a common reference string $\mathsf{crs}' := (\mathsf{crs}, \mathsf{pp}_{\mathsf{IPFE}}, \mathsf{pp}'_{\mathsf{IPFE}})$ and a verifier key $k'_{\mathsf{V}} := (k_{\mathsf{V}}, s, \mathsf{sk}_{\mathsf{IPFE}}, \mathsf{sk}'_{\mathsf{IPFE}})$, and gives $(1^\kappa, \mathsf{crs}', k'_{\mathsf{V}})$ to $\mathcal{A}$ as input. When $\mathcal{A}$ makes a query (x, w), $\mathcal{B}$ returns $\perp$ if $(x, w) \notin \mathcal{R}$. Otherwise it picks $K \xleftarrow{\$} \mathsf{SKE.KeyGen}(1^\kappa)$ and $\sigma_i \xleftarrow{\$} \mathbb{Z}_p$ for $i \in [\ell]$, computes $r_i := K_i + \sigma_i s \mod p$ for $i \in [\ell]$ and $t := f_{x,\mathsf{ct}_{\mathsf{SKE}}}(r_1, ..., r_\ell) - 1 \mod p$, and generates $(c_1, ..., c_D) \leftarrow \mathsf{Coefficient}(1^D, p, f_{x,\mathsf{ct}_{\mathsf{SKE}}}, K = (K_1, ..., K_\ell), (\sigma_1, ..., \sigma_\ell))$, $\mathsf{ct}_{\mathsf{SKE}} \xleftarrow{\$} \mathsf{SKE.Enc}(K, \mathsf{Exp}_{C,x}(w))$, and $\mathsf{ct}^i_{\mathsf{IPFE}} \xleftarrow{\$} \mathsf{IPFE.Enc}(\mathsf{pp}_{\mathsf{IPFE}}, (r_i, 0))$ for $i \in [\ell]$. Then it queries vectors $\mathbf{x}^{(0)} := (c_1, ..., c_D)$ and $\mathbf{x}^{(1)} := (t \cdot s^{-1}, 0, ..., 0)$ to its challenge oracle to obtain a ciphertext $\mathsf{ct}'_{\mathsf{IPFE}}$. Then it generates $\pi \xleftarrow{\$} \mathcal{S}_2(\mathsf{crs}, k_{\mathsf{V}}, \tau, (\mathsf{pp}_{\mathsf{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i \in [\ell]}, \mathsf{pp}'_{\mathsf{IPFE}}, \mathsf{ct}'_{\mathsf{IPFE}}))$ and returns a proof $\pi' := (\pi, \mathsf{ct}_{\mathsf{SKE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i \in [\ell]}, \mathsf{ct}'_{\mathsf{IPFE}})$ to $\mathcal{A}$. Finally, when $\mathcal{A}$ outputs a bit coin', $\mathcal{B}'$ also outputs coin'.

This completes the description of $\mathcal{B}$. First, we show that $\mathcal{B}$ is a valid adversary against the multi-challenge adaptive single-key security of Π_{IPFE}. In the simulation of each query, by the way of generating $(c_1, ..., c_D)$, we have

$$f_{x,\mathsf{ct}_{\mathsf{SKE}}}(K_1 + \sigma_1 s, ..., K_\ell + \sigma_\ell s) = f_{x,\mathsf{ct}_{\mathsf{SKE}}}(K_1, ..., K_\ell) + \sum_{j=1}^{D} c_j s^j \mod p$$

by Lemma 6.4. Moreover, when the query is not returned by $\perp$, we have $(x, w) \in \mathcal{R}$ and thus $f_{x,\mathsf{ct}_{\mathsf{SKE}}}(K_1, ..., K_\ell) = 1 \mod p$ since we have $\mathsf{ct}_{\mathsf{SKE}} = \mathsf{SKE.Enc}(K, \mathsf{Exp}_{x,C}(w))$. Since we have $t = f_{x,\mathsf{ct}_{\mathsf{SKE}}}(r_1, ..., r_\ell) - 1 = f(K_1 + \sigma_1 s, ..., K_\ell + \sigma_\ell s) - 1$ by the definition, we have $(c_1, ..., c_D) \cdot (s, ..., s^D)^T = (t \cdot s^{-1}, 0, ..., 0) \cdot (s, ..., s^D)^T \mod p$. Therefore $\mathcal{B}$ is a valid adversary against the multi-challenge adaptive single-key security of Π_{IPFE}.

It is easy to see that $\mathcal{B}$ perfectly simulates Game_2 if the coin picked by the IPFE challenger is 0 and it perfectly simulates Game_2 otherwise. Therefore we have $|\Pr[\mathsf{T}_2] - \Pr[\mathsf{T}_3]]| \leq \mathsf{negl}(\kappa)$ if Π_{IPFE} satisfies the multi-challenge adaptive single-key security, which follows from the single-challenge version of it as remarked in Remark 6.1.

□

Lemma 6.14. $\Pr[\mathsf{T}_3] = \Pr[\mathsf{T}_4]$.

Proof. First, we observe that in the simulation of each proof, σ_i is used only for generating r_i as $r_i := K_i + \sigma_i s \mod p$ in Game_3. (Remark that the computation of $(c_1, ..., c_D) \leftarrow \mathsf{Coefficient}(1^D, p, f_{x,\mathsf{ct}_{\mathsf{SKE}}}, K = (K_1, ..., K_\ell), (\sigma_1, ..., \sigma_\ell))$

is no longer needed due to the modification made in Game_3.) Since we have $s \in \mathbb{Z}_p^*$, $\sigma_i s$ is uniformly distributed on $\mathbb{Z}_p$, and thus r_i is distributed uniformly on $\mathbb{Z}_p$ independently of K_i or any other values whose partial information may be given to $\mathcal{A}$. Thus these two games are completely identical from the view of $\mathcal{A}$. □

Lemma 6.15. *If* SKE *is CPA-secure, then* $|\Pr[\mathsf{T}_4] - \Pr[\mathsf{T}_5]| = \mathsf{negl}(\kappa)$.

Proof. Since the only part in Game_4 where K is used is the generation of $\mathsf{ct}_{\mathsf{SKE}}$. Therefore it is straightforward to reduce the indistinguishability between these two games to the CPA-security of SKE. □

This completes the proof of Theorem 6.10.

□

Remark 6.16. (On variants in other models) By the essentially the same construction as the case of DV-NIZK, we can generically construct PP-NIZK with compact proof size (i.e., $|C| + \mathsf{poly}(\kappa)$) from any PP-NIZK additionally assuming an SKE scheme whose decryption circuit is in $\mathbf{NC}^1$ with additive ciphertext overhead. We note that in this case, we need not assume PKE since we can use adaptively-secure single-key *secret-key* IPFE, which can be constructed from any one-way function [GVW12]. On the other hand, the construction does not work in the CRS and DP settings since it is essential that the randomly chosen $s \in \mathbb{Z}_p^*$ is hidden from the adversary in the proof of the soundness.

Instantiation. The above construction can be instantiated based on the CDH assumption on a pairing-free group since

- An adaptively single-key secure IPFE scheme exists under any PKE scheme [GVW12], and there exists a PKE scheme based on the CDH assumption.
- An SKE scheme whose decryption circuit is in $\mathbf{NC}^1$ with additive ciphertext overhead exists under the CDH assumption [KNYY19].
- DV-NIZK for all of $\mathbf{NP}$ exists under the CDH assumption [CH19, KNYY19, QRW19]

Therefore we obtain the following corollary.

Corollary 6.17. *If the CDH assumption holds on a pairing-free group, then there exists DV-NIZK for all of* $\mathbf{NP}$ *with proof size* $|C| + \mathsf{poly}(\kappa)$.

Variant with Sublinear Proof Size. Similarly to the case of CRS-NIZK as discussed in Section 4.3, we can make the proof size of the above DV-NIZK sublinear in $|C|$ if C is a leveled circuit. More detailed explanation can be found in Appendix C. Namely, we obtain the following corollary:

Corollary 6.18. *If the CDH assumption holds on a pairing-free group, then there exists DV-NIZK for all* $\mathbf{NP}$ *languages whose corresponding relation is computable by a leveled circuit with proof size* $|w| + |C|/\log\kappa + \mathsf{poly}(\kappa)$.

6.3 DV- NIZK for $\mathbf{NC}^1$ with Shorter Proof from CDHI Assumption

Here, we give a variant of the DV-NIZK in Section 6.2 with shorter proof size $|w| + \mathsf{poly}(\kappa)$ that supports $\mathbf{NC}^1$ relations based on the ℓ-computational Diffie-Hellman inversion (ℓ-CDHI) assumption on pairing-free group. The differences from the construction in the previous section are that we directly encrypt w by SKE instead of "expanding" it and we use the group-based trick inspired by [CF18] instead of the D-dimensional IPFE to reduce the proof size.

ℓ-CDHI assumption. Here, we recall the ℓ-CDHI assumption.

Definition 6.19 (**ℓ-Computational Diffie-Hellman Inversion Assumption [MSK02, CF18]**)**.** *We say that the ℓ-Computational Diffie-Hellman inversion (ℓ-CDHI) assumption holds relative to* GGen *if for all PPT adversaries* $\mathcal{A}$,

$$\Pr\left[\, \mathcal{G} = (\mathbb{G}, p, g) \xleftarrow{\$} \mathsf{GGen}(1^\kappa),\ s \xleftarrow{\$} \mathbb{Z}_p :\ g^{s^{-1}} \leftarrow \mathcal{A}(1^\kappa, \mathcal{G}, g^s, \ldots, g^{s^\ell}) \,\right] \le \mathsf{negl}(\kappa).$$

Construction. We prepare notations for the building blocks.

- Let $\mathcal{L}$ be an **NP** language defined by a relation $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$. Let $n(\kappa)$ and $m(\kappa)$ be any fixed polynomials. Let C be a circuit that computes the relation $\mathcal{R}$ on $\{0,1\}^n \times \{0,1\}^m$, i.e., for $(x,w) \in \{0,1\}^n \times \{0,1\}^m$, we have $C(x,w) = 1$ if and only if $(x,w) \in \mathcal{R}$. We assume that $C(x,\cdot)$ can be computed in $\mathbf{NC}^1$.
- Let $\Pi_{\mathsf{IPFE}} = (\mathsf{IPFE.Setup}, \mathsf{IPFE.KeyGen}, \mathsf{IPFE.Enc}, \mathsf{IPFE.Dec})$ be an adaptively single-key secure IPFE scheme with a modulus p. Such an IPFE scheme can be constructed from any PKE scheme [GVW12].
- Let $\Pi_{\mathsf{SKE}} = (\mathsf{SKE.KeyGen}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ be a CPA secure symmetric key encryption scheme over a ciphertext space $\mathcal{CT}$ and a key space $\{0,1\}^\ell$ with additive ciphertext overhead (i.e., the ciphertext size is the message size plus $\mathsf{poly}(\kappa)$) whose decryption algorithm is computed in $\mathbf{NC}^1$. We note that such an SKE scheme exists under the CDH assumption [KNYY19].
- For $x \in \{0,1\}^n$ and $\mathsf{ct} \in \mathcal{CT}$, we define the function $f_{x,\mathsf{ct}}(K) := C(x, \mathsf{SKE.Dec}(K,\mathsf{ct}))$. Let D be the maximal degree of $f_{x,\mathsf{ct}}$ (as a multivariate polynomial). Since both $C(x,\cdot)$ and $\mathsf{SKE.Dec}(\cdot,\mathsf{ct})$ can be computable in $\mathbf{NC}^1$, we can assume $D = \mathsf{poly}(\kappa)$ (which especially does not depend on $|C|$).
- Let $\mathbb{G}$ be a cyclic group of a prime order p on which $(D-1)$-CDHI assumption holds.[14]
- Let $\Pi_{\mathsf{DVNIZK}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ be DV-NIZK for the language corresponding to the relation $\tilde{\mathcal{R}}$ defined below:
$((\mathsf{pp}_{\mathsf{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}), \{(K_i,\sigma_i,R_i)\}_{i\in[\ell]}) \in \tilde{\mathcal{R}}$ if and only if the following conditions are satisfied:
    1. For all $i \in [\ell]$, $K_i \in \{0,1\}$,
    2. For all $i \in [\ell]$, $\mathsf{IPFE.Enc}(\mathsf{pp}_{\mathsf{IPFE}}, (K_i,\sigma_i); R_i) = \mathsf{ct}^i_{\mathsf{IPFE}}$.

The DV-NIZK $\Pi'_{\mathsf{DVNIZK}} = (\mathsf{Setup}', \mathsf{Prove}', \mathsf{Verify}')$ for $\mathcal{L}$ is described as follows.

$\mathsf{Setup}'(1^\kappa)$**:** This algorithm picks $s \xleftarrow{\$} \mathbb{Z}_p^*$ and $g \xleftarrow{\$} \mathbb{G}$, computes $h_j := g^{s^j}$ for $j \in [D]$ and generates $(\mathsf{crs}, k_{\mathsf{V}}) \xleftarrow{\$} \mathsf{Setup}(1^\kappa)$, $(\mathsf{pp}_{\mathsf{IPFE}}, \mathsf{msk}_{\mathsf{IPFE}}) \xleftarrow{\$} \mathsf{IPFE.Setup}(1^\kappa, 1^2)$ and $\mathsf{sk}_{\mathsf{IPFE}} \xleftarrow{\$} \mathsf{IPFE.KeyGen}(\mathsf{msk}_{\mathsf{IPFE}}, (1,s))$ for $i \in [\ell]$. It outputs a common reference string $\mathsf{crs}' := (\mathsf{crs}, \mathsf{pp}_{\mathsf{IPFE}}, \{h_j\}_{j\in[D]})$ and a verifier key $k'_{\mathsf{V}} := (k_{\mathsf{V}}, s, \{\mathsf{sk}_{\mathsf{IPFE}}\}_{i\in[\ell]}, g)$.

$\mathsf{Prove}'(\mathsf{crs}', x, w)$**:** This algorithm aborts if $(x,w) \notin \mathcal{R}$. Otherwise it parses $(\mathsf{crs}, \mathsf{pp}_{\mathsf{IPFE}}, \{h_j\}_{j\in[D]}) \leftarrow \mathsf{crs}'$, picks $K \xleftarrow{\$} \mathsf{SKE.KeyGen}(1^\kappa)$ and $\sigma_i \xleftarrow{\$} \mathbb{Z}_p$ for $i \in [\ell]$, and generates $\mathsf{ct}_{\mathsf{SKE}} \xleftarrow{\$} \mathsf{SKE.Enc}(K,w)$ and $(c_1,...,c_D) \leftarrow \mathsf{Coefficient}(1^D, p, f_{x,\mathsf{ct}_{\mathsf{SKE}}}, K = (K_1,...,K_\ell), (\sigma_1,...,\sigma_\ell))$. Then it generates $\mathsf{ct}^i_{\mathsf{IPFE}} := \mathsf{IPFE.Enc}(\mathsf{pp}_{\mathsf{IPFE}}, (K_i,\sigma_i); R_i)$ for $i \in [\ell]$ (where R_i is the randomness used by the encryption algorithm), $\pi \xleftarrow{\$} \mathsf{Prove}(\mathsf{crs}, (\mathsf{pp}_{\mathsf{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}), \{(K_i,\sigma_i,R_i)\}_{i\in[\ell]})$, and $\Lambda := \prod_{j=1}^D h_j^{c_j}$ and outputs a proof $\pi' := (\pi, \mathsf{ct}_{\mathsf{SKE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \Lambda)$.

$\mathsf{Verify}'(\mathsf{crs}', k'_{\mathsf{V}}, x, \pi')$**:** This algorithm parses $(\mathsf{crs}, \mathsf{pp}_{\mathsf{IPFE}}, \{h_j\}_{j\in[D]}) \leftarrow \mathsf{crs}'$, $(k_{\mathsf{V}}, s, \{\mathsf{sk}_{\mathsf{IPFE}}\}_{i\in[\ell]}, g) \leftarrow k'_{\mathsf{V}}$, and $(\pi, \mathsf{ct}_{\mathsf{SKE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \Lambda) \leftarrow \pi'$, computes $r_i \xleftarrow{\$} \mathsf{IPFE.Dec}(\mathsf{pp}_{\mathsf{IPFE}}, \mathsf{ct}^i_{\mathsf{IPFE}}, \mathsf{sk}_{\mathsf{IPFE}})$ for $i \in [\ell]$ and $t := f_{x,\mathsf{ct}_{\mathsf{SKE}}}(r_1,...,r_\ell) - 1 \mod p$, and outputs $\top$ if we have $\mathsf{Verify}(\mathsf{crs}, (\mathsf{pp}_{\mathsf{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}), \pi) = \top$ and $g^t = \Lambda$, and outputs $\bot$ otherwise.

Correctness. Suppose that $(\pi, \mathsf{ct}_{\mathsf{SKE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \Lambda)$ is an honestly generated proof on $(x,w) \in \mathcal{R}$. Then it is clear that we have $\mathsf{Verify}(\mathsf{crs}, (\mathsf{pp}_{\mathsf{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}), \pi) = \top$ by the way of generating the proof and the correctness of Π_{DVNIZK}. By the way of generating $(\{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \Lambda)$ and correctness of Π_{IPFE}, we have $r_i = K_i + \sigma_i s \mod p$ for $i \in [\ell]$ and $\Lambda = g^{\sum_{j\in[D]} c_j s^j}$. Since we have $f_{x,\mathsf{ct}_{\mathsf{SKE}}}(K_1 + \sigma_1 Z, ..., K_\ell + \sigma_\ell Z) = 1 + \sum_{j\in[D]} c_j Z^j$ for an indeterminate Z by the correctness of Π_{SKE} and Lemma 6.4, we have $g^{f_{x,\mathsf{ct}_{\mathsf{SKE}}}(r_1,...,r_\ell)-1} = \Lambda$ by substituting s for Z.
Proof Size. First, we remark that the relation $\tilde{\mathcal{R}}$ can be verified by a circuit whose size is a fixed polynomial in $(\kappa, \ell, \log p, D)$ that does not depend on $|C|$. Then we have $|\pi| = \mathsf{poly}(\kappa, \ell, \log p, D)$, $|\mathsf{ct}_{\mathsf{SKE}}| = |w| + \mathsf{poly}(\kappa)$, $|\mathsf{ct}^i_{\mathsf{IPFE}}| = \mathsf{poly}(\kappa, \log p)$. By setting $\ell = \kappa$, $p = 2^{O(\kappa)}$, we have $|\pi'| = |w| + \mathsf{poly}(\kappa)$.
Security. The security of our scheme Π'_{DVNIZK} is stated as follows. The proofs are similar to the security proof for DV-NIZK given in Section 6.2.

[14]Precisely speaking, the $(D-1)$-CDHI assumption was defined for a group generator. We describe our construction as if it was defined for a fixed group $\mathbb{G}$ for notational simplicity.

Theorem 6.20 (Soundness.). *If Π_{DVNIZK} satisfies computational soundness and $(D-1)$-DDHI assumption holds on $\mathbb{G}$, then Π'_{DVNIZK} satisfies computational soundness.*

Proof. Suppose that there is a PPT adversary $\mathcal{A}$ that breaks soundness. We consider the following sequence of games.

Game_0**:** This game is the original experiment that defines the soundness. Specifically, the game is described as follows.

1. The challenger picks $s \xleftarrow{\$} \mathbb{Z}_p^*$ and $g \xleftarrow{\$} \mathbb{G}$, computes $h_j := g^{s^j}$ for $j \in [D]$, generates $(\mathsf{crs}, k_{\mathsf{V}}) \xleftarrow{\$} \mathsf{Setup}(1^\kappa)$, $(\mathsf{pp}_{\mathsf{IPFE}}, \mathsf{msk}_{\mathsf{IPFE}}) \xleftarrow{\$} \mathsf{IPFE.Setup}(1^\kappa, 1^2)$, and $\mathsf{sk}_{\mathsf{IPFE}} \xleftarrow{\$} \mathsf{IPFE.KeyGen}(\mathsf{msk}_{\mathsf{IPFE}}, (1, s))$, sets $\mathsf{crs}' := (\mathsf{crs}, \mathsf{pp}_{\mathsf{IPFE}}, \{h_j\}_{j\in[D]})$ and $k'_{\mathsf{V}} := (k_{\mathsf{V}}, s, \{\mathsf{sk}_{\mathsf{IPFE}}\}_{i\in[\ell]}, g)$, and gives crs' to $\mathcal{A}$.
2. $\mathcal{A}$ can make arbitrary number of verification queries. When $\mathcal{A}$ queries $(x, \pi' = (\pi, \mathsf{ct}_{\mathsf{SKE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \Lambda))$, the challenger returns $\mathsf{Verify}'(\mathsf{crs}', k'_{\mathsf{V}}, x, \pi')$. More specifically, the challenger computes $r_i \xleftarrow{\$} \mathsf{IPFE.Dec}(\mathsf{pp}_{\mathsf{IPFE}}, \mathsf{ct}^i_{\mathsf{IPFE}}, \mathsf{sk}_{\mathsf{IPFE}})$ for $i \in [\ell]$ and $u := f_{x,\mathsf{ct}_{\mathsf{SKE}}}(r_1, ..., r_\ell) - 1 \mod p$, and outputs $\top$ if we have $\mathsf{Verify}(\mathsf{crs}, (\mathsf{pp}_{\mathsf{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}), \pi) = \top$ and $g^t = \Lambda$, and outputs $\bot$ otherwise.
3. Finally, $\mathcal{A}$ outputs (x^*, π'^*). We say that $\mathcal{A}$ wins if $x^* \notin \mathcal{L}$ and $\mathsf{Verify}(\mathsf{crs}', k'_{\mathsf{V}}, x^*, \pi'^*) = \top$.

Game_1**:** This game is the same as the previous game except that the verification oracle $\mathsf{Verify}'(\mathsf{crs}', k'_{\mathsf{V}}, \cdot, \cdot)$ is modified to the following alternative oracle denoted by $\mathsf{Verify}'_1(\cdot, \cdot)$. The oracle $\mathsf{Verify}'_1(\cdot, \cdot)$ does the following:

- Given a query $(x, \pi' = (\pi, \mathsf{ct}_{\mathsf{SKE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \Lambda))$.
- If $\mathsf{Verify}(\mathsf{crs}, (\mathsf{pp}_{\mathsf{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}), \pi) = \bot$, returns $\bot$. Otherwise, proceeds to the next step.
- It computes $(K_i, \sigma_i) := \mathsf{MasterDec}(\mathsf{msk}_{\mathsf{IPFE}}, \mathsf{ct}^i_{\mathsf{IPFE}})$ for $i \in [\ell]$, where $\mathsf{MasterDec}$ is the master decryption algorithm for Π_{IPFE} as defined in Section 6.1. If $K = (K_1, ..., K_\ell) \notin \{0,1\}^\ell$, returns $\bot$. Otherwise, proceeds to the next step.
- It computes $t := f_{x,\mathsf{ct}_{\mathsf{SKE}}}(K_1 + \sigma_1 s, ..., K_\ell + \sigma_\ell s) - 1 \mod p$. If we have $g^t = \Lambda$, outputs $\top$ and returns $\bot$ otherwise.

We note that the winning condition of $\mathcal{A}$ is also modified to use $\mathsf{Verify}'_1(\cdot, \cdot)$ instead of $\mathsf{Verify}'(\mathsf{crs}', k'_{\mathsf{V}}, \cdot, \cdot)$.

Game_2**:** This game is the same as the previous game except that the oracle Verify'_1 is replaced with Verify'_2 described in the following. The main differences from Verify'_1 are highlighted by red underlines. The oracle $\mathsf{Verify}'_2(\cdot, \cdot)$ does the following:

- Given a query $(x, \pi' = (\pi, \mathsf{ct}_{\mathsf{SKE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \Lambda))$.
- If $\mathsf{Verify}(\mathsf{crs}, (\mathsf{pp}_{\mathsf{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}), \pi) = \bot$, returns $\bot$. Otherwise, proceeds to the next step.
- It computes $(K_i, \sigma_i) := \mathsf{MasterDec}(\mathsf{msk}_{\mathsf{IPFE}}, \mathsf{ct}^i_{\mathsf{IPFE}})$ for $i \in [\ell]$. If $K = (K_1, ..., K_\ell) \notin \{0,1\}^\ell$, returns $\bot$. Otherwise, proceeds to the next step.
- $\underline{\text{It computes } (\hat{c}_1, ..., \hat{c}_D) \leftarrow \mathsf{Coefficient}(1^D, p, f_{x,\mathsf{ct}_{\mathsf{SKE}}}, (K_1, ..., K_\ell), (\sigma_1, ..., \sigma_\ell)).}$
  If we have $\underline{\prod_{j=1}^{D} h_j^{\hat{c}_j} \cdot g^{f_{x,\mathsf{ct}_{\mathsf{SKE}}}(K_1,...,K_\ell)-1}} = \Lambda$, outputs $\top$ and returns $\bot$ otherwise.

We note that the winning condition of $\mathcal{A}$ is also modified to use Verify'_2 instead of Verify'_1.

Game_3**:** This game is the same as the previous game except that the oracle Verify'_2 is replaced with an alternative oracle Verify'_3 described below. The main differences from Verify'_2 are highlighted by red underlines. The oracle $\mathsf{Verify}'_3(\cdot, \cdot)$ does the following:

- Given a query $(x, \pi' = (\pi, \mathsf{ct}_{\mathsf{SKE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \Lambda))$.
- If $\mathsf{Verify}(\mathsf{crs}, (\mathsf{pp}_{\mathsf{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}), \pi) = \bot$, returns $\bot$. Otherwise, proceeds to the next step.
- It computes $(K_i, \sigma_i) := \mathsf{MasterDec}(\mathsf{msk}_{\mathsf{IPFE}}, \mathsf{ct}^i_{\mathsf{IPFE}})$ for $i \in [\ell]$. If $K = (K_1, ..., K_\ell) \notin \{0,1\}^\ell$, returns $\bot$. Otherwise, proceeds to the next step.

- It computes $(\hat{c}_1, ..., \hat{c}_D) \leftarrow \mathsf{Coefficient}(1^D, p, f_{x,\mathsf{ct}_{\mathsf{SKE}}}, (K_1, ..., K_\ell), (\sigma_1, ..., \sigma_\ell))$.
  If we have $\underline{f_{x,\mathsf{ct}_{\mathsf{SKE}}}(K_1, ..., K_\ell) = 1 \mod p \text{ and } \prod_{j=1}^{D} h_j^{\hat{c}_j} = \Lambda}$, it outputs $\top$ and returns $\bot$ otherwise.

We note that the winning condition of $\mathcal{A}$ is also modified to use Verify'_3 instead of Verify'_2.

This completes the description of games. We denote the event that $\mathcal{A}$ wins in Game_k by T_k for $k = 0, ..., 3$. We prove the following lemmas.

Lemma 6.21. *If Π_{DVNIZK} satisfies computational soundness, then we have* $|\Pr[\mathsf{T}_0] - \Pr[\mathsf{T}_1]| \le \mathsf{negl}(\kappa)$.

Proof. Let F be the event that $\mathcal{A}$ ever makes a query $(x, \pi' = (\pi, \mathsf{ct}_{\mathsf{SKE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \Lambda))$ to the verification oracle such that $\mathsf{Verify}(\mathsf{crs}, (\mathsf{pp}_{\mathsf{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}), \pi) = \top$ and $\big((\mathsf{pp}_{\mathsf{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}), \{(K_i, \sigma_i, R_i)\}_{i\in[\ell]}\big) \notin \tilde{\mathcal{R}}$. It is easy to see that $\Pr[\mathsf{F}]$ is negligible if Π_{DVNIZK} satisfies computational soundness. If we have $\mathsf{Verify}(\mathsf{crs}, (\mathsf{pp}_{\mathsf{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}), \pi) = \top$ and F does not happen, each ciphertext $\mathsf{ct}^i_{\mathsf{IPFE}}$ is a valid encryption of some $(K_i, \sigma_i) \in \{0,1\} \times \mathbb{Z}_p$. Then if we let $(K_i, \sigma_i) := \mathsf{MasterDec}(\mathsf{msk}_{\mathsf{IPFE}}, \mathsf{ct}^i_{\mathsf{IPFE}})$ and $r_i \xleftarrow{\$} \mathsf{IPFE.Dec}(\mathsf{pp}_{\mathsf{IPFE}}, \mathsf{ct}^i_{\mathsf{IPFE}}, \mathsf{sk}_{\mathsf{IPFE}})$ for $i \in [\ell]$, then we have $K = (K_1, ..., K_\ell) \in \{0,1\}^\ell$ and $r_i = K_i + \sigma_i s \mod p$ for $i \in [\ell]$, and thus we have $f_{x,\mathsf{ct}_{\mathsf{SKE}}}(r_1, ..., r_\ell) = f_{x,\mathsf{ct}_{\mathsf{SKE}}}(K_1 + \sigma_\ell s, ..., K_\ell + \sigma_\ell s)$. Therefore we have $|\Pr[\mathsf{T}_0] - \Pr[\mathsf{T}_1]| \le \Pr[\mathsf{F}]$, and the lemma is proven. □

Lemma 6.22. $\Pr[\mathsf{T}_1] = \Pr[\mathsf{T}_2]$.

Proof. Game_1 and Game_2 are identical from $\mathcal{A}$'s view since the responses from the verification oracle never differ as seen below: By the definition of $\mathsf{Coefficient}$, we have

$$f_{x,\mathsf{ct}_{\mathsf{SKE}}}(K_1 + \sigma_1 s, ..., K_\ell + \sigma_\ell s) = f_{x,\mathsf{ct}_{\mathsf{SKE}}}(K_1, ..., K_\ell) + \sum_{j\in[D]} \hat{c}_j s^j \mod p.$$

Therefore the equation

$$g^{f_{x,\mathsf{ct}_{\mathsf{SKE}}}(K_1+\sigma_1 s,...,K_\ell+\sigma_\ell s)-1} = \Lambda$$

is equivalent to

$$\prod_{j=1}^{D} h_j^{\hat{c}_j} \cdot g^{f_{x,\mathsf{ct}_{\mathsf{SKE}}}(K_1,...,K_\ell)-1} = \Lambda$$

□

Lemma 6.23. *If the $(D-1)$-CDHI assumption holds, then* $|\Pr[\mathsf{T}_2] - \Pr[\mathsf{T}_3]| \le \mathsf{negl}(\kappa)$.

Proof. Here, we regard the final output (x^*, π'^*) of $\mathcal{A}$ as the $(Q+1)$-th query for notational convenience. We consider hybrids H_k for $k = 0, 1, ..., Q+1$, which is the same as Game_2 except that Verify'_3 is used until $\mathcal{A}$'s k-th query and Verify'_2 is used for the rest of the queries. Let T'_k be the event that $\mathcal{A}$ wins in H_k. It is clear that H_0 is Game_2 and H_{Q+1} is Game_3. Thus what we have to prove is that we have $|\Pr[\mathsf{T}'_k] - \Pr[\mathsf{T}'_{k+1}]| \le \mathsf{negl}(\kappa)$ for $k = 0, ..., Q$. It is easy to see that H_k and H_{k+1} differ only when we have $f_{x,\mathsf{ct}_{\mathsf{SKE}}}(K_1, ..., K_\ell) = 0$ and $\prod_{j=1}^{D} h_j^{\hat{c}_j} \cdot g^{-1} = \Lambda$ in the simulation of the verification oracle for $\mathcal{A}$'s $(k+1)$-th query. If we denote this event by Bad_{k+1}, then we have $|\Pr[\mathsf{T}'_k] - \Pr[\mathsf{T}'_{k+1}]| \le \Pr[\mathsf{Bad}_{k+1}]$. We prove that $\Pr[\mathsf{Bad}_{k+1}]$ is negligible assuming $(D-1)$-CDHI assumption. Suppose that $\Pr[\mathsf{Bad}_{k+1}]$ is non-negligible. Then we construct an adversary $\mathcal{B}$ that breaks the $(D-1)$-CDHI assumption as follows.

$\mathcal{B}(\tilde{g}, \tilde{g}^s, ..., \tilde{g}^{s^{D-1}})$**:** Given a problem instance $(\tilde{g}, \tilde{g}^s, ..., \tilde{g}^{s^{D-1}})$, it sets $h_j := \tilde{g}^{s^{j-1}}$ for $j \in [D]$ (which implicitly defines $g := \tilde{g}^{s^{-1}}$), generates $(\mathsf{crs}, k_{\mathsf{V}}) \xleftarrow{\$} \mathsf{Setup}(1^\kappa)$, $(\mathsf{pp}_{\mathsf{IPFE}}, \mathsf{msk}_{\mathsf{IPFE}}) \xleftarrow{\$} \mathsf{IPFE.Setup}(1^\kappa, 1^2)$, and $\mathsf{sk}_{\mathsf{IPFE}} \xleftarrow{\$} \mathsf{IPFE.KeyGen}(\mathsf{msk}_{\mathsf{IPFE}}, (1, s))$, sets $\mathsf{crs}' := (\mathsf{crs}, \mathsf{pp}_{\mathsf{IPFE}}, \{h_j\}_{j\in[D]})$, and gives crs' to $\mathcal{A}$. $\mathcal{B}$ simulates the verification oracle $\mathsf{Verify}_3(\cdot,\cdot)$ until $\mathcal{A}$ makes its $(k+1)$-th query. (We note that this can be done without knowing $g = \tilde{g}^{s^{-1}}$ since Verify'_3 does not use g.) Let $(x, \pi = (\pi, \mathsf{ct}_{\mathsf{SKE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \Lambda))$ be the $\mathcal{A}$'s $(k+1)$-th query. Then $\mathcal{B}$ computes $(K_i, \sigma_i) := \mathsf{MasterDec}(\mathsf{msk}_{\mathsf{IPFE}}, \mathsf{ct}^i_{\mathsf{IPFE}})$ for $i \in [\ell]$ and $(\hat{c}_1, ..., \hat{c}_D) \leftarrow \mathsf{Coefficient}(1^D, p, f_{x,\mathsf{ct}_{\mathsf{SKE}}}, (K_1, ..., K_\ell), (\sigma_1, ..., \sigma_\ell))$, and outputs $\prod_{j=1}^{D} h_j^{\hat{c}_j} \cdot \Lambda^{-1}$.

This completes the description of $\mathcal{B}$. It is easy to see that $\mathcal{B}$ perfectly simulates H_k and H_{k+1} until $\mathcal{A}$ makes its $(k+1)$-th query. If Bad_{k+1} happens, then we have $\prod_{j=1}^{D} h_j^{\hat{c}_j} \cdot g^{-1} = \Lambda$, and thus we have $\prod_{j=1}^{D} h_j^{\hat{c}_j} \cdot \Lambda^{-1} = g = \hat{g}^{s^{-1}}$. This means that $\mathcal{B}$ succeeds in breaking the $(D-1)$-CDHI assumption. Therefore $\Pr[\mathsf{Bad}_{k+1}] \leq \mathsf{negl}(\kappa)$ under the $(D-1)$-CDHI assumption. □

Lemma 6.24. $\Pr[\mathsf{T}_3] = 0$.

Proof. $\mathcal{A}$ has no chance to win Game_3 since if $x^* \notin \mathcal{L}$, $f_{x,\mathsf{ct}_{\mathsf{SKE}}}$ never outputs 1 on any input $K \in \{0,1\}^\ell$ for any $\mathsf{ct}_{\mathsf{SKE}} \in \mathcal{CT}$. □

This completes the proof of Theorem 6.20. □

Theorem 6.25 (Zero-knowledge). *If* SKE *is CPA secure,* Π_{IPFE} *is adaptively single-key secure, and* Π_{DVNIZK} *satisfies zero-knowledge, then* Π'_{DVNIZK} *satisfies zero-knowledge.*

Proof. Let $(\mathcal{S}_1, \mathcal{S}_2)$ be the simulator for Π_{DVNIZK}. We describe the simulator $(\mathcal{S}_1, \mathcal{S}_2)$ for Π'_{DVNIZK} below.

$\mathcal{S}'_1(1^\kappa)$**:** It picks $s \xleftarrow{\$} \mathbb{Z}_p^*$ and $g \xleftarrow{\$} \mathbb{G}$, computes $h_j := g^{s^j}$ for $j \in [D]$, and generates $(\mathsf{pp}_{\mathsf{IPFE}}, \mathsf{msk}_{\mathsf{IPFE}}) \xleftarrow{\$} \mathsf{IPFE.Setup}(1^\kappa, 1^2)$ and $\mathsf{sk}_{\mathsf{IPFE}} \xleftarrow{\$} \mathsf{IPFE.KeyGen}(\mathsf{msk}_{\mathsf{IPFE}}, (1,s))$, $(\overline{\mathsf{crs}}, \bar{k}_{\mathsf{V}}, \bar{\tau}) \xleftarrow{\$} \mathcal{S}_1(1^\kappa)$, and $K \xleftarrow{\$} \mathsf{SKE.KeyGen}(1^\kappa)$, and outputs $(\overline{\mathsf{crs}}' := (\overline{\mathsf{crs}}, \mathsf{pp}_{\mathsf{IPFE}}, \{h_j\}_{j\in[D]}), \bar{k}'_{\mathsf{V}} := (\bar{k}_{\mathsf{V}}, s, \{\mathsf{sk}_{\mathsf{IPFE}}\}_{i\in[\ell]}), \bar{\tau}' = (\bar{\tau}, K))$.

$\mathcal{S}'_2(\overline{\mathsf{crs}}' = (\overline{\mathsf{crs}}, \mathsf{pp}_{\mathsf{IPFE}}, \{h_j\}_{j\in[D]}), \bar{k}'_{\mathsf{V}} = (\bar{k}_{\mathsf{V}}, s, \{\mathsf{sk}_{\mathsf{IPFE}}\}_{i\in[\ell]}), \bar{\tau}' = (\bar{\tau}, K), x)$**:** It picks $r_i \xleftarrow{\$} \mathbb{Z}_p$ for $i \in [\ell]$ computes $\mathsf{ct}_{\mathsf{SKE}} \xleftarrow{\$} \mathsf{SKE.Enc}(K, 0^m)$, $t := f_{x,\mathsf{ct}_{\mathsf{SKE}}}(r_1, ..., r_\ell) - 1 \mod p$, $\mathsf{ct}^i_{\mathsf{IPFE}} \xleftarrow{\$} \mathsf{IPFE.Enc}(\mathsf{pp}_{\mathsf{IPFE}}, (r_i, 0))$ for $i \in [\ell]$, $\Lambda := g^t$, and $\pi \xleftarrow{\$} \mathcal{S}_2(\overline{\mathsf{crs}}, \bar{k}_{\mathsf{V}}, \bar{\tau}, (\mathsf{pp}_{\mathsf{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}))$ and outputs a proof $\pi' := (\pi, \mathsf{ct}_{\mathsf{SKE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \Lambda)$.

This completes the description of the simulator. We prove that proofs simulated by the above simulator are computationally indistinguishable from the honestly generated proofs. To prove this, we consider the following sequence of games between a PPT adversary $\mathcal{A}$ and a challenger.

Game_0**:** In this game, proofs are generated honestly. Namely,

1. The challenger picks $s \xleftarrow{\$} \mathbb{Z}_p^*$ and $g \xleftarrow{\$} \mathbb{G}$, computes $h_j := g^{s^j}$ for $j \in [D]$, and generates $(\mathsf{crs}, k_{\mathsf{V}}) \xleftarrow{\$} \mathsf{Setup}(1^\kappa)$, and $(\mathsf{pp}_{\mathsf{IPFE}}, \mathsf{msk}_{\mathsf{IPFE}}) \xleftarrow{\$} \mathsf{IPFE.Setup}(1^\kappa, 1^2)$ and $\mathsf{sk}_{\mathsf{IPFE}} \xleftarrow{\$} \mathsf{IPFE.KeyGen}(\mathsf{msk}_{\mathsf{IPFE}}, (1,s))$. It defines a common reference string $\mathsf{crs}' := (\mathsf{crs}, \mathsf{pp}_{\mathsf{IPFE}}, \{h_j\}_{j\in[D]})$ and a verifier key $k'_{\mathsf{V}} := (k_{\mathsf{V}}, s, \{\mathsf{sk}_{\mathsf{IPFE}}\}_{i\in[\ell]})$.
2. $\mathcal{A}$ is given $(1^\kappa, \mathsf{crs}', k'_{\mathsf{V}})$, and allowed to query $\mathcal{O}(\mathsf{crs}, \cdot, \cdot)$, which works as follows. When $\mathcal{A}$ queries (x, w), if $(x,w) \notin \mathcal{R}$, then the oracle returns $\perp$. Otherwise, it picks $K \xleftarrow{\$} \mathsf{SKE.KeyGen}(1^\kappa)$ and $\sigma_i \xleftarrow{\$} \mathbb{Z}_p$ for $i \in [\ell]$, and generates $\mathsf{ct}_{\mathsf{SKE}} \xleftarrow{\$} \mathsf{SKE.Enc}(K, w)$, $(c_1, ..., c_D) \leftarrow \mathsf{Coefficient}(1^D, p, f_{x,\mathsf{ct}_{\mathsf{SKE}}}, K = (K_1, ..., K_\ell), (\sigma_1, ..., \sigma_\ell))$, $\mathsf{ct}^i_{\mathsf{IPFE}} \xleftarrow{\$} \mathsf{IPFE.Enc}(\mathsf{pp}_{\mathsf{IPFE}}, (K_i, \sigma_i); R_i)$ for $i \in [\ell]$ (where R_i is the randomness used by the encryption algorithm), $\pi \xleftarrow{\$} \mathsf{Prove}(\mathsf{crs}, (\mathsf{pp}_{\mathsf{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}), \{(K_i, \sigma_i, R_i)\}_{i\in[\ell]})$, computes $\Lambda := \prod_{j=1}^{D} h_j^{c_j}$, and returns a proof $\pi' := (\pi, \mathsf{ct}_{\mathsf{SKE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \Lambda)$.
3. Finally, $\mathcal{A}$ returns a bit β.

Game_1**:** This game is the same as the previous game except that crs, k_{V}, and π are generated differently. Namely, the challenger generates $(\mathsf{crs}, k_{\mathsf{V}}, \tau) \xleftarrow{\$} \mathcal{S}_1(1^\kappa)$ at the beginning of the game, and π is generated as $\pi \xleftarrow{\$} \mathcal{S}_2(\mathsf{crs}, k_{\mathsf{V}}, \tau, (\mathsf{pp}_{\mathsf{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}))$ for each oracle query.

Game_2**:** This game is the same as the previous game except that $\{\mathsf{ct}_{\mathsf{IPFE}}\}_{i\in[\ell]}$ is generated differently when responding to each query. Namely, the oracle computes $r_i := K_i + \sigma_i s \mod p$ and $\mathsf{ct}^i_{\mathsf{IPFE}} \xleftarrow{\$} \mathsf{IPFE.Enc}(\mathsf{pp}_{\mathsf{IPFE}}, (r_i, 0))$ for $i \in [\ell]$.

Game_3**:** This game is the same as the previous game except that Λ is generated differently when responding to each query. Namely, the oracle computes $t := f_{x,\mathsf{ct}_{\mathsf{SKE}}}(r_1, ..., r_\ell) - 1 \mod p$, and sets $\Lambda := g^t$. We note that in this game, $(c_1, ..., c_D)$ is need not be computed since it is not used for generating Λ.

Game_4**:** This game is the same as the previous game except that r_i is randomly chosen from $\mathbb{Z}_p$ for $i \in [\ell]$ in each query.

Game_5**:** This game is the same as the previous game except that $\mathsf{ct_{SKE}}$ is generated as $\mathsf{ct_{SKE}} \xleftarrow{\$} \mathsf{SKE.Enc}(K, 0^m)$.

Let T_i be the event that $\mathcal{A}$ returns 1 in Game_i for $i \in \{0,1,2,3,4,5\}$. It is easy to see the the way of generating proofs in Game_5 is identical to the one by $\mathcal{S}' = (\mathcal{S}'_1, \mathcal{S}'_2)$. Thus we have to prove that $|\Pr[\mathsf{T}_0] - \Pr[\mathsf{T}_3]|$ is negligible. We prove this by the following lemmas.

Lemma 6.26. *If* Π_{DVNIZK} *satisfies zero-knowledge w.r.t. the simulator* $\mathcal{S}$*, then* $|\Pr[\mathsf{T}_0] - \Pr[\mathsf{T}_1]| = \mathsf{negl}(\kappa)$.

Proof. We assume that $|\Pr[\mathsf{T}_0] - \Pr[\mathsf{T}_1]|$ is non-negligible, and construct a PPT adversary $\mathcal{B}$ that breaks the zero-knowledge of Π_{DVNIZK}. The description of $\mathcal{B}$ is given below.

$\mathcal{B}^{\mathcal{O}(\cdot,\cdot)}(\mathsf{crs}, k_{\mathsf{V}})$**:** It picks $s \xleftarrow{\$} \mathbb{Z}_p^*$ and $g \xleftarrow{\$} \mathbb{G}$, computes $h_j := g^{s^j}$ for $j \in [D]$, and generates $(\mathsf{pp_{IPFE}}, \mathsf{msk_{IPFE}}) \xleftarrow{\$} \mathsf{IPFE.Setup}(1^\kappa, 1^2)$ and $\mathsf{sk_{IPFE}} \xleftarrow{\$} \mathsf{IPFE.KeyGen}(\mathsf{msk_{IPFE}}, (1,s))$ for $i \in [\ell]$. It defines a common reference string $\mathsf{crs}' := (\mathsf{crs}, \mathsf{pp_{IPFE}}, \{h_j\}_{j\in[D]})$ and a verifier key $k'_{\mathsf{V}} := (k_{\mathsf{V}}, s, \{\mathsf{sk_{IPFE}}\}_{i\in[\ell]})$. and runs $\mathcal{A}^{\mathcal{O}'(\cdot,\cdot)}(\mathsf{crs}', k'_{\mathsf{V}})$ where $\mathcal{B}$ simulates $\mathcal{O}'(\cdot,\cdot)$ as follows. When $\mathcal{A}$ makes a query (x, w) to $\mathcal{O}'(\cdot)$, $\mathcal{B}$ picks $K \xleftarrow{\$} \mathsf{SKE.KeyGen}(1^\kappa)$ and $\sigma_i \xleftarrow{\$} \mathbb{Z}_p$ for $i \in [\ell]$, generates $\mathsf{ct_{SKE}} \xleftarrow{\$} \mathsf{SKE.Enc}(K, w)$, $(c_1, ..., c_D) \leftarrow \mathsf{Coefficient}(1^D, p, f_{x,\mathsf{ct_{SKE}}}, K = (K_1, ..., K_\ell), (\sigma_1, ..., \sigma_\ell))$, and $\mathsf{ct}^i_{\mathsf{IPFE}} \xleftarrow{\$} \mathsf{IPFE.Enc}(\mathsf{pp_{IPFE}}, (K_i, \sigma_i); R_i)$ for $i \in [\ell]$ (where R_i is the randomness used by the encryption algorithm), computes $\Lambda := \prod_{j=1}^{D} h_j^{c_j}$, and queries $((\mathsf{pp_{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}), \{(K_i, \sigma_i, R_i)\}_{i\in[\ell]})$ to $\mathcal{O}(\cdot,\cdot)$ to obtain π (if $\perp$ is returned, then $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$ as a response by $\mathcal{O}'(\cdot,\cdot)$). Then it returns a proof $\pi' := (\pi, \mathsf{ct_{SKE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \Lambda)$.

This completes the description of $\mathcal{B}$. It is easy to see that if $\mathcal{O}$ generates proofs honestly, then $\mathcal{B}$ perfectly simulates Game_0, and if $\mathcal{O}$ generates proofs by using the simulator, then $\mathcal{B}$ perfectly simulates Game_1. Therefore if $\mathcal{A}$ distinguishes these two games with non-negligible probability, then $\mathcal{B}$ succeeds in distinguishing these two cases, which means it breaks the zero-knowledge. $\square$

Lemma 6.27. *If* IPFE *is adaptively single-key secure, then* $|\Pr[\mathsf{T}_1] - \Pr[\mathsf{T}_2]| = \mathsf{negl}(\kappa)$.

Proof. Suppose that $|\Pr[\mathsf{T}_1] - \Pr[\mathsf{T}_2]|$ is non-negligible. Then we construct an adversary $\mathcal{B}$ that breaks the multi-challenge adaptive single-key security of Π_{IPFE} as follows.

$\mathcal{B}(1^\kappa)$**:** It declares the dimension 1^2 and obtains a public parameter $\mathsf{pp_{IPFE}}$ for Π_{IPFE} with dimension 2. It picks $s \xleftarrow{\$} \mathbb{Z}_p^*$, queries a vector $(1, s)$ to its key generation oracle to obtain $\mathsf{sk_{IPFE}}$. Then it picks $g \xleftarrow{\$} \mathbb{G}$ and $s \xleftarrow{\$} \mathbb{Z}_p^*$, sets $h_j := g^{s^j}$ for $j \in [D]$, and generates $(\mathsf{crs}, k_{\mathsf{V}}, \tau) \xleftarrow{\$} \mathcal{S}_1(1^\kappa)$. It sets a common reference string $\mathsf{crs}' := (\mathsf{crs}, \mathsf{pp_{IPFE}}, \{h_j\}_{j\in[D]})$ and a verifier key $k'_{\mathsf{V}} := (k_{\mathsf{V}}, s, \{\mathsf{sk_{IPFE}}\}_{i\in[\ell]}, g)$, and gives $(1^\kappa, \mathsf{crs}', k'_{\mathsf{V}})$ to $\mathcal{A}$ as input. When $\mathcal{A}$ makes its k-th query (x, w), $\mathcal{B}$ returns $\perp$ if $(x, w) \notin \mathcal{R}$. Otherwise it picks $K \xleftarrow{\$} \mathsf{SKE.KeyGen}(1^\kappa)$ and $\sigma_i \xleftarrow{\$} \mathbb{Z}_p$ for $i \in [\ell]$, computes $r_i := K_i + \sigma_i s \mod p$ for $i \in [\ell]$, and generates $\mathsf{ct_{SKE}} \xleftarrow{\$} \mathsf{SKE.Enc}(K, w)$ and $(c_1, ..., c_D) \leftarrow \mathsf{Coefficient}(1^D, p, f_{x,\mathsf{ct_{SKE}}}, K = (K_1, ..., K_\ell), (\sigma_1, ..., \sigma_\ell))$. Then $\mathcal{B}$ queries pairs vectors $\mathbf{x}_i^{(0)} := (K_i, \sigma_i)$ and $\mathbf{x}_i^{(1)} := (r_i, 0)$ to its challenge oracle to obtain ciphertexts $\{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}$. Then it generates $\pi \xleftarrow{\$} \mathcal{S}_2(\mathsf{crs}, k_{\mathsf{V}}, \tau, (\mathsf{pp_{IPFE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}))$, computes $\Lambda := \prod_{j=1}^{D} h_j^{c_j}$, and returns a proof $\pi' := (\pi, \mathsf{ct_{SKE}}, \{\mathsf{ct}^i_{\mathsf{IPFE}}\}_{i\in[\ell]}, \Lambda)$ to $\mathcal{A}$. Finally, when $\mathcal{A}$ outputs a bit coin', $\mathcal{B}$ also outputs coin'.

This completes the construction of $\mathcal{B}$. First, we remark that $\mathcal{B}$ is a valid adversary against the multi-challenge adaptive single-key security of Π_{IPFE} since we have $(K_i, \sigma_i) \cdot (1, s)^T = (r_i, 0) \cdot (1, s)^T \mod p$. It is easy to see that $\mathcal{B}$ perfectly simulates Game_1 to $\mathcal{A}$ if the coin picked by the IPFE challenger is 0 and it perfectly simulates Game_2 otherwise. Therefore we have $|\Pr[\mathsf{T}_1] - \Pr[\mathsf{T}_2]| \leq \mathsf{negl}(\kappa)$ if Π_{IPFE} satisfies the multi-challenge adaptive single-key security, which follows from the single-challenge version of it as remarked in Remark 6.1.

$\square$

Lemma 6.28. $\Pr[\mathsf{T}_2] = \Pr[\mathsf{T}_3]$.

Proof. In a simulation for each query, by the way of generating $(c_1, ..., c_D)$, we have

$$f_{x,\mathsf{ct}_{\mathsf{SKE}}}(K_1 + \sigma_1 s, ..., K_\ell + \sigma_\ell s) = f_{x,\mathsf{ct}_{\mathsf{SKE}}}(K_1, ..., K_\ell) + \sum_{j=1}^{D} c_j s^j \mod p$$

by Lemma 6.4. Moreover, when the query is not returned by $\perp$, we have $(x, w) \in \mathcal{R}$ and thus $f_{x,\mathsf{ct}_{\mathsf{SKE}}}(K_1, ..., K_\ell) = 1 \mod p$ since we have $\mathsf{ct}_{\mathsf{SKE}} = \mathsf{SKE.Enc}(K, \mathsf{Exp}(w))$. Since we have $t = f_{x,\mathsf{ct}_{\mathsf{SKE}}}(r_1, ..., r_\ell) - 1 = f(K_1 + \sigma_1 s, ..., K_\ell + \sigma_\ell s) - 1$ by the definition, we have $t = \sum_{j=1}^{D} c_j s^j$. Therefore we have $\prod_{j=1}^{D} h_j^{c_j} = g^t$, and thus Λ is defined to be exactly the same value in Game_2 and Game_3. □

Lemma 6.29. $\Pr[\mathsf{T}_3] = \Pr[\mathsf{T}_4]$.

Proof. First, we observe that in a simulation of each proof, σ_i is used only for generating r_i as $r_i := K_i + \sigma_i s \mod p$ in Game_3. (Remark that the computation of $(c_1, ..., c_D) \leftarrow \mathsf{Coefficient}(1^D, p, f_{x,\mathsf{ct}_{\mathsf{SKE}}}, K = (K_1, ..., K_\ell), (\sigma_1, ..., \sigma_\ell))$ is no longer needed due to the modification made in Game_3.) Since we have $s \in \mathbb{Z}_p^*$, $\sigma_i s$ is uniformly distributed on $\mathbb{Z}_p$, and thus r_i is distributed uniformly on $\mathbb{Z}_p$ independently of K_i or any other values whose partial information may be given to $\mathcal{A}$. Thus these two games are completely identical from the view of $\mathcal{A}$. □

Lemma 6.30. *If* SKE *is CPA-secure, then* $|\Pr[\mathsf{T}_4] - \Pr[\mathsf{T}_5]| = \mathsf{negl}(\kappa)$.

Proof. Since the only part in Game_4 where K is used is the generation of $\mathsf{ct}_{\mathsf{SKE}}$. Therefore it is straightforward to reduce the indistinguishability between these two games to the CPA-security of SKE. □

This completes the proof of Theorem 6.25. □

Instantiation. The above construction can be instantiated based on the $\mathsf{poly}(\kappa)$-CDHI assumption on a pairing-free group since all building blocks except for the group $\mathbb{G}$ can be instantiated based on the CDH assumption as seen in Section 6.2, and the CDH assumption holds under the 1-CDHI assumption. Therefore we obtain the following corollary.

Corollary 6.31. *If the* $\mathsf{poly}(\kappa)$*-CDHI assumption holds on a pairing-free group, then there exists DV-NIZK for all* **NP** *languages whose corresponding relation is computable in* $\mathbf{NC}^1$ *with proof size* $|w| + \mathsf{poly}(\kappa)$.

7 CRS-NIZK with Efficient Prover From Laconic Function Evaluation

In this section, we present a NIZK proof system where a prover is efficient, that is, the running time of a prover is smaller than the size of circuit that computes the relation. We use laconic function evaluation to achieve our NIZK proof system.

Before describing the construction, we prepare some building blocks and notations.

- Let $\mathcal{L}$ be an **NP** language defined by a relation $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$. Let $n(\kappa)$ and $m(\kappa)$ be any fixed polynomials. Let C be a circuit that computes the relation $\mathcal{R}$ on $\{0,1\}^n \times \{0,1\}^m$, i.e., for $(x, w) \in \{0,1\}^n \times \{0,1\}^m$, we have $C(x, w) = 1$ if and only if $(x, w) \in \mathcal{R}$
- Let $\mathsf{LFE} = (\mathsf{LFE.crsGen}, \mathsf{LFE.Compress}, \mathsf{LFE.Enc}, \mathsf{LFE.Dec})$ be a LFE scheme whose function class $\mathcal{C}$ is the class of all circuits with $\mathsf{params} = (1^k, 1^d)$ consisting of the input size k and the depth d of the circuits and contains $\{C\}$ that computes the relation $\mathcal{R}$ for NP-complete language.
- Let $\Pi_{\mathsf{CRSNIZK}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ be a CRS-NIZK for the language corresponding to the relation $\tilde{\mathcal{R}}$ defined below:

$$((x, \mathsf{lfe.crs}, \mathsf{digest}_C, \mathsf{lfe.ct}), (w, r)) \in \tilde{\mathcal{R}} \iff \mathsf{LFE.Enc}(\mathsf{lfe.crs}, \mathsf{digest}_C, (x, w); r) = \mathsf{lfe.ct}\,.$$

The CRS-NIZK $\Pi'_{\mathsf{CRSNIZK}} = (\mathsf{Setup}', \mathsf{Prove}', \mathsf{Verify}')$ for $\mathcal{L}$ is described as follows.

$\mathsf{Setup}'(1^\kappa)$**:** This algorithm generates $\mathsf{crs} \xleftarrow{\$} \mathsf{Setup}(1^\kappa)$ and $\mathsf{lfe.crs} \xleftarrow{\$} \mathsf{LFE.crsGen}(1^\kappa, \mathsf{params})$. It generates $\mathsf{digest}_C := \mathsf{LFE.Compress}(\mathsf{lfe.crs}, C)$. It outputs a common reference string $\mathsf{crs}' = (\mathsf{crs}, \mathsf{lfe.crs}, \mathsf{digest}_C)$.

$\mathsf{Prove}'(\mathsf{crs}', x, w)$: This algorithm aborts if $\mathcal{R}(x, w) = 0$. Otherwise it parses $(\mathsf{crs}, \mathsf{lfe.crs}, \mathsf{digest}_C) \leftarrow \mathsf{crs}'$, generates $\mathsf{lfe.ct} := \mathsf{LFE.Enc}(\mathsf{lfe.crs}, \mathsf{digest}_C, (x, w); r)$ where r is the randomness for $\mathsf{LFE.Enc}$ and $\pi_{\mathsf{NIZK}} \overset{\$}{\leftarrow} \mathsf{Prove}(\mathsf{crs}, (x, \mathsf{lfe.crs}, \mathsf{digest}_C, \mathsf{lfe.ct}), (w, r))$. It outputs a proof $\pi' := (\mathsf{lfe.ct}, \pi_{\mathsf{NIZK}})$.

$\mathsf{Verify}'(\mathsf{crs}', x, \pi')$: This algorithm parses $(\mathsf{crs}, \mathsf{lfe.crs}, \mathsf{digest}_C) \leftarrow \mathsf{crs}'$, $(\mathsf{lfe.ct}, \pi_{\mathsf{NIZK}}) \leftarrow \pi'$, and computes $t := \mathsf{Verify}(\mathsf{crs}, (x, \mathsf{lfe.crs}, \mathsf{digest}_C, \mathsf{lfe.ct}), \pi_{\mathsf{NIZK}})$. If $t = \bot$ or $0 \overset{\$}{\leftarrow} \mathsf{LFE.Dec}(\mathsf{lfe.crs}, C, \mathsf{lfe.ct})$, then outputs $\bot$. Otherwise, outputs $\top$.

Completeness. By the completeness of Π_{CRSNIZK}, the proof π_{NIZK} in an honestly generated proof π' passes the verification of Π_{CRSNIZK}. That is, it holds that $\mathsf{Verify}(\mathsf{crs}, (x, \mathsf{lfe.crs}, \mathsf{digest}_C, \mathsf{lfe.ct}), \pi_{\mathsf{NIZK}}) = \top$. By the correctness of LFE, it holds that $1 = C(x, w) \overset{\$}{\leftarrow} \mathsf{LFE.Dec}(\mathsf{lfe.crs}, C, \mathsf{lfe.ct})$ with probability 1. Thus, the completeness follows.

Prover Efficiency. First, we remark that the relation $\tilde{\mathcal{R}}$ can be verified by a circuit whose size is $|\mathsf{LFE.Enc}|$ since the relation is about the validity of LFE ciphertexts. The running time of Prove' is the sum of those of $\mathsf{LFE.Enc}$ and Prove. We defer concrete efficiency analysis until Section 7.1 since the running time depends on instantiations of $\mathsf{LFE.Enc}$ and Prove.

Security. Finally, we prove the security of Π'_{CRSNIZK}.

Theorem 7.1 (Soundness). *Π'_{CRSNIZK} is computationally/statistically sound if Π_{CRSNIZK} is computationally/statistically sound, respectively.*

Proof of Theorem 7.1. In this proof, we focus on the computational soundness case. We can prove the statistical soundness in a similar manner.

Suppose that there is a PPT adversary $\mathcal{A}$ that breaks soundness. Then we construct a PPT adversary $\mathcal{B}$ that breaks the soundness of Π_{CRSNIZK} as follows.

$\mathcal{B}(1^\kappa, \mathsf{crs})$: It generates $\mathsf{lfe.crs} \overset{\$}{\leftarrow} \mathsf{LFE.crsGen}(1^\kappa, \mathsf{params})$ and $\mathsf{digest}_C := \mathsf{LFE.Compress}(\mathsf{lfe.crs}, C)$, sets $\mathsf{crs}' := (\mathsf{crs}, \mathsf{lfe.crs}, \mathsf{digest}_C)$, and runs $\mathcal{A}(1^\kappa, \mathsf{crs}')$ to obtain (x^*, π'^*). Then, it parses $\pi'^* = (\mathsf{lfe.ct}, \pi_{\mathsf{NIZK}})$ and computes $y \overset{\$}{\leftarrow} \mathsf{LFE.Dec}(\mathsf{lfe.crs}, \mathsf{digest}_C, \mathsf{lfe.ct})$. If $y = 1$, then it outputs $(x' := (x^*, \mathsf{lfe.crs}, \mathsf{digest}_C, \mathsf{lfe.ct}), \pi_{\mathsf{NIZK}})$. If $y = 0$, outputs $\bot$.

This completes the description of $\mathcal{B}$. In the following, we show that $\mathcal{B}$ breaks the soundness of Π_{CRSNIZK}. Since we assume $\mathcal{A}$ breaks the soundness of Π'_{CRSNIZK}, we have $x^* \notin \mathcal{L}$, $\mathsf{Verify}(\mathsf{crs}, (x^*, \mathsf{lfe.crs}, \mathsf{digest}_C, \mathsf{lfe.ct}), \pi_{\mathsf{NIZK}}) = \top$, and $\mathsf{LFE.Dec}(\mathsf{lfe.crs}, \mathsf{digest}_C, \mathsf{lfe.ct}) = 1$ with non-negligible probability. In the following, we assume that this is the case.

In this case, we prove that $x' = (x^*, \mathsf{lfe.crs}, \mathsf{digest}_C, \mathsf{lfe.ct}) \notin \tilde{\mathcal{L}}$. Assume by contradiction that $x' \in \tilde{\mathcal{L}}$, that is, $\mathsf{lfe.ct}$ is an encryption of (x^*, w^*) for some w^* under some randomness r. Then by the condition that $\mathsf{LFE.Dec}(\mathsf{lfe.crs}, \mathsf{digest}_C, \mathsf{lfe.ct}) = 1$, we have that $\mathcal{R}(x^*, w^*) = 1$, i.e., $C(x^*, w^*) = 1$, due to the correctness of LFE. However, the condition $\mathcal{R}(x^*, w^*) = 1$ contradicts $x^* \notin \mathcal{L}$, so we must have $x' \notin \tilde{\mathcal{L}}$. Thus, $\mathcal{B}$ succeeds in breaking the soundness since $x' \notin \tilde{\mathcal{L}}$ and $\mathsf{Verify}(\mathsf{crs}, (x^*, \mathsf{lfe.crs}, \mathsf{digest}_C, \mathsf{lfe.ct}), \pi_{\mathsf{NIZK}}) = \top$ holds. □

Theorem 7.2 (Zero-Knowledge). *Π'_{CRSNIZK} is computational zero-knowledge if Π_{CRSNIZK} is zero-knowledge and* LFE *is adaptively secure.*

Proof of Theorem 7.2. Let $(\mathcal{S}_1, \mathcal{S}_2)$ be the simulator for Π_{CRSNIZK}. We describe the simulator $(\mathcal{S}'_1, \mathcal{S}'_2)$ for Π'_{CRSNIZK} below.

$\mathcal{S}'_1(1^\kappa)$: It generates $(\mathsf{crs}, \tau) \overset{\$}{\leftarrow} \mathcal{S}_1(1^\kappa)$, $\mathsf{lfe.crs} \overset{\$}{\leftarrow} \mathsf{LFE.crsGen}(1^\kappa, \mathsf{params})$, and $\mathsf{digest}_C := \mathsf{LFE.Compress}(\mathsf{lfe.crs}, C)$. It outputs $\mathsf{crs}' := (\mathsf{crs}, \mathsf{lfe.crs}, \mathsf{digest}_C)$ and $\tau' := \tau$.

$\mathcal{S}'_2(\mathsf{crs}', \tau' = \tau, x)$: It parses $(\mathsf{crs}, \mathsf{lfe.crs}, \mathsf{digest}_C) \leftarrow \mathsf{crs}'$, computes $\mathsf{lfe.ct} \overset{\$}{\leftarrow} \mathsf{LFE.Sim}(\mathsf{lfe.crs}, C, \mathsf{digest}_C, 1)$ and $\pi_{\mathsf{NIZK}} \overset{\$}{\leftarrow} \mathcal{S}_2(\mathsf{crs}, \tau, (x, \mathsf{lfe.crs}, \mathsf{digest}_C, \mathsf{lfe.ct}))$, and outputs $\pi' := (\mathsf{lfe.ct}, \pi_{\mathsf{NIZK}})$.

This completes the description of the simulator. We prove that proofs simulated by the above simulator are computationally indistinguishable from the honestly generated proofs. To prove this, we consider the following sequence of games between a PPT adversary $\mathcal{A}$ and a challenger.

Game_0**:** In this game, proofs are generated honestly. Namely,

1. The challenger generates $\mathsf{crs} \xleftarrow{\$} \mathsf{Setup}(1^\kappa)$, $\mathsf{lfe.crs} \xleftarrow{\$} \mathsf{LFE.crsGen}(1^\kappa, \mathsf{params})$, and $\mathsf{digest}_C := \mathsf{LFE.Compress}(\mathsf{lfe.crs}, C)$. It gives $\mathsf{crs}' := (\mathsf{crs}, \mathsf{lfe.crs}, \mathsf{digest}_C)$ to $\mathcal{A}$.
2. $\mathcal{A}$ is given $(1^\kappa, \mathsf{crs}')$, and allowed to query $\mathcal{O}(\mathsf{crs}, \cdot, \cdot)$, which works as follows. When $\mathcal{A}$ queries (x, w), if $(x, w) \notin \mathcal{R}$, then the oracle returns $\perp$. Otherwise, it computes $\mathsf{lfe.ct} \xleftarrow{\$} \mathsf{LFE.Enc}(\mathsf{lfe.crs}, \mathsf{digest}_C, (x, w); r)$ where r is the randomness and $\pi_{\mathsf{NIZK}} \xleftarrow{\$} \mathsf{Prove}(\mathsf{crs}, (x, \mathsf{lfe.crs}, \mathsf{digest}_C, \mathsf{lfe.ct}), (w, r))$. It returns a proof $\pi' := (\mathsf{lfe.ct}, \pi_{\mathsf{NIZK}})$.
3. Finally, $\mathcal{A}$ returns a bit β.

Game_1**:** This game is identical to the previous game except that crs and π_{NIZK} are generated differently. Namely, the challenger generates $(\mathsf{crs}, \tau) \xleftarrow{\$} \mathcal{S}_1(1^\kappa)$ at the beginning of the game, and π_{NIZK} is generated as $\pi_{\mathsf{NIZK}} \xleftarrow{\$} \mathcal{S}_2(\mathsf{crs}, \tau, x)$ for each oracle query.

Game_2**:** This game is identical to the previous game except that $\mathsf{lfe.ct}$ is generated differently. Namely, the challenger generates $\mathsf{lfe.ct} \xleftarrow{\$} \mathsf{LFE.Sim}(\mathsf{lfe.crs}, C, \mathsf{digest}_C, 1)$.

Let T_i be the event that $\mathcal{A}$ returns 1 in Game_i for $i = 0, 1, 2$. It is easy to see that proofs generated by $\mathcal{S}' = (\mathcal{S}_1', \mathcal{S}_2')$ are the same as those in Game_2. Thus we have to prove that $|\Pr[\mathsf{T}_0] - \Pr[\mathsf{T}_2]|$ is negligible. The following lemmas are straightforward to prove.

Lemma 7.3. *If* Π_{CRSNIZK} *satisfies the computational zero-knowledge w.r.t. the simulator* $\mathcal{S}$*, then* $|\Pr[\mathsf{T}_0] - \Pr[\mathsf{T}_1]| = \mathsf{negl}(\kappa)$.

Proof of Lemma 7.3. Suppose that there is a PPT adversary $\mathcal{A}$ that distinguishes Game_1 from Game_0. Then we construct a PPT adversary $\mathcal{B}$ that breaks the zero-knowledge of Π_{CRSNIZK} as follows.

$\mathcal{B}(1^\kappa, \mathsf{crs})$**:** It generates $\mathsf{lfe.crs} \xleftarrow{\$} \mathsf{LFE.crsGen}(1^\kappa, \mathsf{params})$ and $\mathsf{digest}_C := \mathsf{LFE.Compress}(\mathsf{lfe.crs}, C)$, sets $\mathsf{crs}' := (\mathsf{crs}, \mathsf{lfe.crs}, \mathsf{digest}_C)$, and runs $\mathcal{A}(1^\kappa, \mathsf{crs}')$. When $\mathcal{A}$ sends (x, w) as a query, if $\mathcal{R}(x, w) = 0$, then outputs $\perp$. Otherwise, it computes $\mathsf{lfe.ct} \xleftarrow{\$} \mathsf{LFE.Enc}(\mathsf{lfe.crs}, \mathsf{digest}_C, (x, w); r)$ where r is the randomness, sends $((x, \mathsf{lfe.crs}, \mathsf{digest}_C, \mathsf{lfe.ct}), (w, r))$ to its oracle $\mathcal{O}^{\mathsf{NIZK}}$, and receivers π_{NIZK}. Finally, it returns $(\mathsf{lfe.ct}, \pi_{\mathsf{NIZK}})$ to $\mathcal{A}$ and outputs what $\mathcal{A}$ outputs.

This completes the description of $\mathcal{B}$. In the following, we show that $\mathcal{B}$ breaks the zero-knowledge of Π_{CRSNIZK}. If $\mathcal{B}$ has oracle access to $\mathcal{O}_0^{\mathsf{NIZK}}$ (real proof), $\mathcal{B}$ perfectly simulates the oracle $\mathcal{O}_0^{\mathsf{NIZK}'}$ (proofs of Π_{CRSNIZK}' in Game_0) since $\mathcal{B}$ honestly generates $\mathsf{lfe.ct}$ by using w and r. If $\mathcal{B}$ has an oracle access to $\mathcal{O}_1^{\mathsf{NIZK}}$ (simulated proof), $\mathcal{B}$ perfectly simulates the oracle $\mathcal{O}_1^{\mathsf{NIZK}'}$ (proofs of Π_{CRSNIZK}' in Game_1) since $\mathcal{B}$ receives $\pi_{\mathsf{NIZK}} \xleftarrow{\$} \mathcal{S}_2(\mathsf{crs}, \tau, x)$. Therefore, if $\mathcal{A}$ distinguishes two games, $\mathcal{B}$ also distinguishes real proofs from simulated proofs. ☐

Lemma 7.4. *If* LFE *is adaptively secure,* $|\Pr[\mathsf{T}_1] - \Pr[\mathsf{T}_2]| = \mathsf{negl}(\kappa)$.

Proof of Lemma 7.4. In this proof, we use the multi-challenge adaptive security of LFE, which is implied by adaptive security for a single-challenge as noted in Definition 2.7. Suppose that there is a PPT adversary $\mathcal{A}$ that distinguishes Game_2 from Game_1. Then we construct a PPT adversary $\mathcal{B}$ that breaks the adaptive security of LFE as follows.

$\mathcal{B}(1^\kappa)$**:** It generates parameters k, d for a circuit family that contains C, sends $(1^k, 1^d)$ to the challenger, receives $\mathsf{lfe.crs}$, and computes $\mathsf{digest}_C := \mathsf{LFE.Compress}(\mathsf{crs}, C)$ where C is the circuit that computes $\mathcal{R}$. It runs $(\mathsf{crs}, \tau) \xleftarrow{\$} \mathcal{S}_1(1^\kappa)$ and sends $\mathsf{crs}' := (\mathsf{crs}, \mathsf{lfe.crs}, \mathsf{digest}_C)$ to $\mathcal{A}$. When $\mathcal{B}$ sends (x, w) as a query, if $\mathcal{R}(x, w) = 0$, outputs $\perp$. Otherwise, it sends $((x, w), C)$ to the challenger of LFE, receives $\mathsf{lfe.ct}$, and runs $\pi_{\mathsf{NIZK}} \xleftarrow{\$} \mathcal{S}_2(\mathsf{crs}, \tau, (x, \mathsf{lfe.crs}, \mathsf{digest}_C, \mathsf{lfe.ct}))$. It returns $(\mathsf{lfe.ct}, \pi_{\mathsf{NIZK}})$ to $\mathcal{A}$ as an answer. Finally, it outputs what $\mathcal{A}$ outputs.

This completes the description of $\mathcal{B}$. In the following, we show that $\mathcal{B}$ breaks the adaptive security of LFE. If $\mathcal{B}$ is given $\mathsf{lfe.ct} \xleftarrow{\$} \mathsf{LFE.Enc}(\mathsf{lfe.crs}, \mathsf{digest}_C, x)$ (real LFE ciphertext), it is easy to see that $\mathcal{B}$ perfectly simulates the oracle $\mathcal{O}_1^{\mathsf{NIZK}'}$ (proofs of Π'_{CRSNIZK} in Game_1). If $\mathcal{B}$ is given $\mathsf{lfe.ct} \xleftarrow{\$} \mathsf{LFE.Sim}(\mathsf{lfe.crs}, C, \mathsf{digest}_C, C(x,w))$ (simulated LFE ciphertext), $\mathcal{B}$ perfectly simulates the oracle $\mathcal{O}_2^{\mathsf{NIZK}'}$ (proofs of Π'_{CRSNIZK} in Game_2, that is, simulated proof of Π'_{CRSNIZK}) since $\mathcal{B}$ uses (x,w) such that $\mathcal{R}(x,w) = 1$, that is, $C(x,w) = 1$. Therefore, if $\mathcal{A}$ distinguishes two games, $\mathcal{B}$ also distinguishes the real experiment from the simulated experiment of LFE. □

By Lemmata 7.3 and 7.4, we complete the proof of Theorem 7.2. □

By the analysis of completeness and Theorems 7.1 and 7.2, Π'_{CRSNIZK} is a secure NIZK proof system.

7.1 Instantiations

We can consider two cases since there are two instantiations of adaptively secure LFE.

1. (Under sub-exponential security of the LWE assumption with sub-exponential modulus-to-noise ratio): By Lemma 2.8, it holds that $|\mathsf{lfe.crs}| = \mathsf{poly}(\kappa, |x|, |w|, d)$, $|\mathsf{digest}_C| = \mathsf{poly}(\kappa)$, $|\mathsf{lfe.ct}| = \mathsf{poly}(\kappa, |x|, |w|, d)$, and the running time of $\mathsf{LFE.Enc}$ is $\mathsf{poly}(\kappa, |x|, |w|, d)$ where d is the depth of C since the input length of C is $|x| + |w|$. In this case, we use a NIZK whose prover running time is $\mathsf{poly}(\tilde{C}, \kappa)$ where $\tilde{C}$ is a circuit that computes the relation $\tilde{\mathcal{R}}$, which holds for any NIZK. In this case, $\tilde{C}$ just runs $\mathsf{LFE.Enc}$, so it takes $|\mathsf{LFE.Enc}| + \mathsf{poly}(|\mathsf{LFE.Enc}|, \kappa)$ time to generate π_{NIZK}. Thus, the running time of the prover is $\mathsf{poly}(\kappa, |x|, |w|, d)$.

2. (Under the *adaptive* LWE assumption with sub-exponential modulus-to-noise ratio): By Lemma 2.8, it holds that $|\mathsf{lfe.crs}| = (|x| + |w|) \cdot \mathsf{poly}(\kappa, d)$, $|\mathsf{digest}_C| = \mathsf{poly}(\kappa)$, $|\mathsf{lfe.ct}| = \tilde{O}(|x| + |w|) \cdot \mathsf{poly}(\kappa, d)$, and the running time of $\mathsf{LFE.Enc}$ is $\tilde{O}(|x| + |w|) \cdot \mathsf{poly}(\kappa, d)$ where d is the depth of C since the input length of C is $|x| + |w|$. In this case, we use a NIZK whose prover running time is $|\tilde{C}| \cdot \mathsf{poly}(\kappa)$. An example of such a NIZK is the NIZK by Groth et al. [GOS12]. By using the efficiency of Groth et al. NIZK, it takes $|\mathsf{LFE.Enc}| + |\mathsf{LFE.Enc}| \cdot \mathsf{poly}(\kappa)$ time to generate π_{NIZK}. Thus, the running time of the prover is $\tilde{O}(|x|+|w|) \cdot \mathsf{poly}(\kappa, d) \cdot \mathsf{poly}(\kappa) = \tilde{O}(|x|+|w|) \cdot \mathsf{poly}(\kappa, d)$.

Therefore, we obtain the following two corollaries by using Lemmata 2.8 and 5.9.

Corollary 7.5. *If a CRS-NIZK scheme for all of* ***NP*** *exists and the sub-exponentially secure LWE assumption with sub-exponential modulus-to-noise ratio holds, then there exists a CRS-NIZK scheme for all of* ***NP*** *whose prover running time is* $\mathsf{poly}(\kappa, |x|, |w|, d)$.

Corollary 7.6. *If the DLIN assumption in a bilinear group and the adaptive LWE assumption with sub-exponential modulus-to-noise ratio hold, then there exists a CRS-NIZK scheme for all of* ***NP*** *whose prover running time is* $\tilde{O}(|x| + |w|)\mathsf{poly}(\kappa, d)$.

Acknowledgement. We thank anonymous reviewers of Crypto 2019 for their helpful comments. The first and the third authors were supported by JST CREST Grant Number JPMJCR19F6. The third author was supported by JSPS KAKENHI Grant Number 16K16068.

References

[ABDP15] Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. Simple functional encryption schemes for inner products. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 733–751. Springer, Heidelberg, March / April 2015. (Cited on page 8.)

[Abu13] Hamza Abusalah. Generic instantiations of the hidden bits model for non-interactive zero-knowledge proofs for NP, 2013. Master's thesis, RWTH-Aachen University. (Cited on page 5, 31.)

[AFG$^+$16] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-preserving signatures and commitments to group elements. *Journal of Cryptology*, 29(2):363–421, April 2016. (Cited on page 4.)

[ALS16] Shweta Agrawal, Benoît Libert, and Damien Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 333–362. Springer, Heidelberg, August 2016. (Cited on page 8.)

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In Shafi Goldwasser, editor, *ITCS 2012*, pages 326–349. ACM, January 2012. (Cited on page 3.)

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *20th ACM STOC*, pages 103–112. ACM Press, May 1988. (Cited on page 2.)

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 509–539. Springer, Heidelberg, August 2016. (Cited on page 31, 62.)

[BMW03] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 614–629. Springer, Heidelberg, May 2003. (Cited on page 2, 3.)

[BP15] Nir Bitansky and Omer Paneth. ZAPs and non-interactive witness indistinguishability from indistinguishability obfuscation. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 401–427. Springer, Heidelberg, March 2015. (Cited on page 2.)

[BPW16] Nir Bitansky, Omer Paneth, and Daniel Wichs. Perfect structure on the edge of chaos - trapdoor permutations from indistinguishability obfuscation. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 474–502. Springer, Heidelberg, January 2016. (Cited on page 2.)

[BY96] Mihir Bellare and Moti Yung. Certifying permutations: Noninteractive zero-knowledge based on any trapdoor permutation. *Journal of Cryptology*, 9(3):149–166, June 1996. (Cited on page 2.)

[Can00] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. `http://eprint.iacr.org/2000/067`. (Cited on page 32.)

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001. (Cited on page 32, 33.)

[CC18] Pyrros Chaidos and Geoffroy Couteau. Efficient designated-verifier non-interactive zero-knowledge proofs of knowledge. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 193–221. Springer, Heidelberg, April / May 2018. (Cited on page 2.)

[CCH$^+$19] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-Shamir: from practice to theory. 2019. (Cited on page 2.)

[CCRR18] Ran Canetti, Yilei Chen, Leonid Reyzin, and Ron D. Rothblum. Fiat-Shamir and correlation intractability from strong KDM-secure encryption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 91–122. Springer, Heidelberg, April / May 2018. (Cited on page 2.)

[CD04] Ronald Cramer and Ivan Damgård. Secret-key zero-knowlegde and non-interactive verifiable exponentiation. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 223–237. Springer, Heidelberg, February 2004. (Cited on page 10.)

[CF18] Dario Catalano and Dario Fiore. Practical homomorphic message authenticators for arithmetic circuits. *Journal of Cryptology*, 31(1):23–59, January 2018. (Cited on page 4, 8, 9, 46.)

[CG15] Pyrros Chaidos and Jens Groth. Making sigma-protocols non-interactive without random oracles. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 650–670. Springer, Heidelberg, March / April 2015. (Cited on page 2.)

[CH19] Geoffroy Couteau and Dennis Hofheinz. Designated-verifier pseudorandom generators, and their applications. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 562–592. Springer, Heidelberg, May 2019. (Cited on page 2, 5, 9, 46.)

[CHK07] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. *Journal of Cryptology*, 20(3):265–294, July 2007. (Cited on page 5, 31.)

[CL18] Ran Canetti and Amit Lichtenberg. Certifying trapdoor permutations, revisited. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part I*, volume 11239 of *LNCS*, pages 476–506. Springer, Heidelberg, November 2018. (Cited on page 2.)

[CsW19] Ran Cohen, abhi shelat, and Daniel Wichs. Adaptively secure mpc with sublinear communication complexity. In *CRYPTO*, 2019. (Cited on page 2, 4, 5, 7, 32, 33, 35.)

[Cv91] David Chaum and Eugène van Heyst. Group signatures. In Donald W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 257–265. Springer, Heidelberg, April 1991. (Cited on page 2.)

[Dam93] Ivan Damgård. Non-interactive circuit based proofs and non-interactive perfect zero-knowledge with proprocessing. In Rainer A. Rueppel, editor, *EUROCRYPT'92*, volume 658 of *LNCS*, pages 341–355. Springer, Heidelberg, May 1993. (Cited on page 10.)

[DDN00] Danny Dolev, Cynthia Dwork, and Moni Naor. Nonmalleable cryptography. *SIAM J. Comput.*, 30(2):391–437, 2000. (Cited on page 2.)

[DFGK14] George Danezis, Cédric Fournet, Jens Groth, and Markulf Kohlweiss. Square span programs with applications to succinct NIZK arguments. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 532–550. Springer, Heidelberg, December 2014. (Cited on page 3.)

[DFN06] Ivan Damgård, Nelly Fazio, and Antonio Nicolosi. Non-interactive zero-knowledge from homomorphic encryption. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 41–59. Springer, Heidelberg, March 2006. (Cited on page 2, 12.)

[DMP90] Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Non-interactive zero-knowledge with preprocessing. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 269–282. Springer, Heidelberg, August 1990. (Cited on page 10.)

[FLS99] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. *SIAM J. Comput.*, 29(1):1–28, 1999. (Cited on page 2, 5.)

[Gen09] Craig Gentry. A fully homomorphic encryption scheme, 2009. Ph.D. thesis, Stanford University. (Cited on page 2, 16.)

[GGH$^+$16] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *SIAM J. Comput.*, 45(3):882–929, 2016. (Cited on page 31, 37.)

[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. *Journal of Cryptology*, 28(4):820–843, October 2015. (Cited on page 2, 3, 5.)

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Heidelberg, May 2013. (Cited on page 2, 3.)

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989. (Cited on page 2.)

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987. (Cited on page 2.)

[GO94] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7(1):1–32, December 1994. (Cited on page 2.)

[Gol04] Oded Goldreich. Foundations of cryptography: Volume 2, basic applications. 2004. (Cited on page 2, 5.)

[GOS12] Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *J. ACM*, 59(3):11:1–11:35, 2012. (Cited on page 2, 3, 4, 5, 9, 37, 55.)

[GPSW06a] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. *IACR Cryptology ePrint Archive*, 2006:309, 2006. Version 20061007:061901. (Cited on page 17.)

[GPSW06b] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *CCS*, pages 89–98. ACM, 2006. (Cited on page 24.)

[Gro10a] Jens Groth. Short non-interactive zero-knowledge proofs. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 341–358. Springer, Heidelberg, December 2010. (Cited on page 2, 3, 5.)

[Gro10b] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Heidelberg, December 2010. (Cited on page 2.)

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016. (Cited on page 3.)

[GS12] Jens Groth and Amit Sahai. Efficient noninteractive proof systems for bilinear groups. *SIAM J. Comput.*, 41(5):1193–1232, 2012. (Cited on page 2.)

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 162–179. Springer, Heidelberg, August 2012. (Cited on page 9, 39, 46, 47.)

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 469–477. ACM Press, June 2015. (Cited on page 4, 10, 14, 64, 66, 68.)

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 99–108. ACM Press, June 2011. (Cited on page 2.)

[IKOS09] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge proofs from secure multiparty computation. *SIAM J. Comput.*, 39(3):1121–1152, 2009. (Cited on page 10.)

[Kat17] Shuichi Katsumata. On the untapped potential of encoding predicates by arithmetic circuits and their applications. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 95–125. Springer, Heidelberg, December 2017. (Cited on page 61.)

[Kil94] Joe Kilian. On the complexity of bounded-interaction and noninteractive zero-knowledge proofs. In *35th FOCS*, pages 466–477. IEEE Computer Society Press, November 1994. (Cited on page 3.)

[KMO90] Joe Kilian, Silvio Micali, and Rafail Ostrovsky. Minimum resource zero-knowledge proofs (extended abstract). In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 545–546. Springer, Heidelberg, August 1990. (Cited on page 10.)

[KNYY19] Shuichi Katsumata, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Designated verifier/prover and preprocessing NIZKs from diffie-hellman assumptions. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 622–651. Springer, Heidelberg, May 2019. (Cited on page 2, 3, 4, 5, 6, 7, 8, 9, 10, 16, 19, 22, 31, 39, 40, 46, 47, 62, 66, 68.)

[KP98] Joe Kilian and Erez Petrank. An efficient noninteractive zero-knowledge proof system for NP with general assumptions. *Journal of Cryptology*, 11(1):1–27, January 1998. (Cited on page 3.)

[KRR17] Yael Tauman Kalai, Guy N. Rothblum, and Ron D. Rothblum. From obfuscation to the security of Fiat-Shamir for proofs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 224–251. Springer, Heidelberg, August 2017. (Cited on page 2.)

[KW18a] Sam Kim and David J. Wu. Multi-theorem preprocessing NIZKs from lattices. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 733–765. Springer, Heidelberg, August 2018. (Cited on page 6, 8, 10.)

[KW18b] Sam Kim and David J. Wu. Multi-theorem preprocessing NIZKs from lattices. Cryptology ePrint Archive, Report 2018/272, 2018. https://eprint.iacr.org/2018/272. (Cited on page 32.)

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 169–189. Springer, Heidelberg, March 2012. (Cited on page 2, 3.)

[Lip13] Helger Lipmaa. Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part I*, volume 8269 of *LNCS*, pages 41–60. Springer, Heidelberg, December 2013. (Cited on page 3.)

[Lip17] Helger Lipmaa. Optimally sound sigma protocols under DCRA. In Aggelos Kiayias, editor, *FC 2017*, volume 10322 of *LNCS*, pages 182–203. Springer, Heidelberg, April 2017. (Cited on page 2.)

[LS91] Dror Lapidot and Adi Shamir. Publicly verifiable non-interactive zero-knowledge proofs. In Alfred J. Menezes and Scott A. Vanstone, editors, *CRYPTO'90*, volume 537 of *LNCS*, pages 353–365. Springer, Heidelberg, August 1991. (Cited on page 10.)

[LW11] Allison B. Lewko and Brent Waters. Decentralizing attribute-based encryption. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 568–588. Springer, Heidelberg, May 2011. (Cited on page 17.)

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718. Springer, Heidelberg, April 2012. (Cited on page 61.)

[MSK02] Shigeo Mitsunari, Ryuichi Sakai, and Masao Kasahara. A new traitor tracing. *IEICE Transactions*, E85-A(2):481–484, February 2002. (Cited on page 4, 46.)

[Nao03] Moni Naor. On cryptographic assumptions and challenges (invited talk). In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 96–109. Springer, Heidelberg, August 2003. (Cited on page 2.)

[NS98] David Naccache and Jacques Stern. A new public key cryptosystem based on higher residues. In Li Gong and Michael K. Reiter, editors, *ACM CCS 98*, pages 59–66. ACM Press, November 1998. (Cited on page 2.)

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd ACM STOC*, pages 427–437. ACM Press, May 1990. (Cited on page 2.)

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Heidelberg, August 1992. (Cited on page 4.)

[PHGR16] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: nearly practical verifiable computation. *Commun. ACM*, 59(2):103–112, 2016. (Cited on page 3.)

[PsV06] Rafael Pass, abhi shelat, and Vinod Vaikuntanathan. Construction of a non-malleable encryption scheme from any semantically secure one. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 271–289. Springer, Heidelberg, August 2006. (Cited on page 2, 12.)

[QRW19] Willy Quach, Ron D. Rothblum, and Daniel Wichs. Reusable designated-verifier NIZKs for all NP from CDH. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 593–621. Springer, Heidelberg, May 2019. (Cited on page 2, 5, 9, 46.)

[QWW18] Willy Quach, Hoeteck Wee, and Daniel Wichs. Laconic function evaluation and applications. In Mikkel Thorup, editor, *59th FOCS*, pages 859–870. IEEE Computer Society Press, October 2018. (Cited on page 3, 4, 5, 9, 13, 14, 61.)

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):34:1–34:40, 2009. (Cited on page 60.)

[RST01] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 552–565. Springer, Heidelberg, December 2001. (Cited on page 2.)

[Sah99] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th FOCS*, pages 543–553. IEEE Computer Society Press, October 1999. (Cited on page 2.)

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Heidelberg, May 1997. (Cited on page 16.)

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014. (Cited on page 2.)

[VV09] Carmine Ventre and Ivan Visconti. Co-sound zero-knowledge with public keys. In Bart Preneel, editor, *AFRICACRYPT 09*, volume 5580 of *LNCS*, pages 287–304. Springer, Heidelberg, June 2009. (Cited on page 2.)

A Learning with Errors

. In this section, we review the definitions about learning with errors (LWE) assumptions.

Definition A.1 (B-bounded distribution). *We say that a distribution χ over $\mathbb{Z}$ is B-bounded if* $\Pr[\chi \in [-B, B]] = 1$.

Definition A.2 (LWE assumption [Reg09]). *Let $n = n(\kappa)$ and $q = q(\kappa)$ be integer parameters and $\chi = \chi(\kappa)$ a distribution over $\mathbb{Z}$. We say the $\mathsf{LWE}(n, q, \chi)$ assumption holds if for any PPT adversary $\mathcal{A}$, its advantage*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{LWE}}(\kappa) := \left| \Pr\left[\mathcal{A}(1^\kappa, \boldsymbol{A}, \boldsymbol{s}^\top \boldsymbol{A} + \boldsymbol{e}) = 1 \;\middle|\; \begin{array}{l} \boldsymbol{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}, \\ \boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^n, \\ \boldsymbol{e} \xleftarrow{\$} \chi^m \end{array} \right] - \Pr\left[\mathcal{A}(1^\kappa, \boldsymbol{A}, \boldsymbol{u}) = 1 \;\middle|\; \begin{array}{l} \boldsymbol{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}, \\ \boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^m \end{array} \right] \right|$$

is negligible in κ.

We assume that for any polynomial $p = p(\kappa)$, there exists some polynomial $n = n(\kappa)$, some $q = q(\kappa) = 2^{\mathsf{poly}(\kappa)}$, and some $B = B(\kappa)$-bounded distribution $\chi = \chi(\kappa)$ such that $q/B \geq 2^p$ (sub-exponential modulus-to-noise ratio) and $\mathsf{LWE}_{n,q,\chi}$ assumption holds.

Definition A.3 (Gadget Matrix [MP12]). *For any integer $q \geq 2$, let $\boldsymbol{g} := (1, 2, \ldots, 2^{\lceil \log q \rceil - 1}) \in \mathbb{Z}_q^{1 \times \lceil \log q \rceil}$. The gadget matrix is defined as $\boldsymbol{G} := \boldsymbol{g} \otimes \boldsymbol{I}_n \in \mathbb{Z}_q^{n \times m}$ where $n \in \mathbb{N}$, $m := n \lceil \log q \rceil$, and $\boldsymbol{I}_n$ is $n \times n$ identity matrix.*

Definition A.4 (Adaptive LWE assumption [QWW18]). *Let $n = n(\kappa)$, $q = q(\kappa)$, and $k = k(\kappa)$ be integer parameters, $\chi = \chi(\kappa)$ a distribution over $\mathbb{Z}$, and $\boldsymbol{G} \in \mathbb{Z}_q^{n \times m}$ the gadget matrix in Definition A.3. We define $m := n \lceil \log q \rceil$. For any polynomial $m' = m'(\kappa)$, we define experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{ALWE}}(1^\kappa, b)$ between an adversary $\mathcal{A}$ and challenger as follows.*

1. *The challenger chooses k random matrices $\boldsymbol{A}_i \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$ for $i \leq k$, and sends them to $\mathcal{A}$.*
2. *$\mathcal{A}$ chooses $x \in \{0,1\}^k$ and sends it to the challenger.*
3. *The challenger chooses $\boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^n$ and $\boldsymbol{A}_{k+1} \xleftarrow{\$} \mathbb{Z}_q^{n \times m'}$ and do the following:*
   - *If $b = 0$, then computes $\mathbf{b}_i := \boldsymbol{s}^\top(\boldsymbol{A}_i - x_i \boldsymbol{G}) + \boldsymbol{e}_i^\top$ where $\boldsymbol{e}_i \xleftarrow{\$} \chi^m$ for $i \leq k$ and $\mathbf{b}_{k+1} := \boldsymbol{s}^\top \boldsymbol{A}_{k+1} + \boldsymbol{e}_{k+1}^\top$ where $\boldsymbol{e}_{k+1} \xleftarrow{\$} \chi^{m'}$.*
   - *If $b = 1$, then chooses $\mathbf{b}_i \xleftarrow{\$} \mathbb{Z}_q^m$ for $i \leq k$ and $\mathbf{b}_{k+1} \xleftarrow{\$} \mathbb{Z}_q^{m'}$.*
4. *Finally, the challenger sends $\boldsymbol{A}_{k+1}$ and $\{\mathbf{b}_i\}_{i \leq k+1}$ to $\mathcal{A}$ and the experiment outputs whatever $\mathcal{A}$ outputs.*

We say the $\mathsf{ALWE}(n, q, \chi)$ assumption holds if for any polynomial $m = m(\kappa)$ and any PPT adversary $\mathcal{A}$, its advantage

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{ALWE}}(\kappa) := \left| \Pr\left[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{ALWE}}(1^\kappa, 0) = 1\right] - \Pr\left[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{ALWE}}(1^\kappa, 1) = 1\right] \right|$$

is negligible in κ.

Quach et al. [QWW18] observed that the sub-exponentially secure LWE assumption implies the adaptive LWE assumption (n, q, χ can depend on k). Although they did not give any concrete reduction, they conjectured that the adaptive LWE assumption plausibly holds on its own (n, q, χ do not depend on k).

B Proof of Lemma 6.3

Proof. For simplicity, we assume that $C(x, \cdot)$ consists of only NAND gates. (This is just for notational simplicity, and a similar result can be obtained as long as each gate of C can be expressed as a degree 2 polynomial over $\mathbb{Z}_p$.) Now, we fix $(x, w) \in \mathcal{R} \cap \{0,1\}^n \times \{0,1\}^m$ and a prime $p > |C|$. Let Gates denote the set of all gates of $C(x, \cdot)$, and for each gate $g \in \mathsf{Gates}$ of the circuit $C(x, \cdot)$ let s_g denote the output value of g when we evaluate $C(x, \cdot)$ on the input w. We let $\mathsf{Exp}_{C,x}(w) := \{s_g\}_{g \in \mathsf{Gates}}$. Clearly, the size of $\mathsf{Exp}_{C,x}(w)$ is $|\mathsf{Gates}|$. For each gate $g \in \mathsf{Gates}$, let A_g and B_g denote the gates whose output wire is connected to the input wire of g. Then we have $s_g = 1 - s_{A_g} \cdot s_{B_g} \mod p$ for every gate g and $s_{\mathsf{out}} = 1 \mod p$ where out is the output gate of $C(x, \cdot)$. Conversely, it is easy to see that there exists $w \in \{0,1\}^m$ such that $C(x, w) = 1$ if and only if there exists $\{s_g\}_{g \in \mathsf{Gates}} \in \{0,1\}^{|\mathsf{Gates}|}$ such that $s_g = 1 - s_{A_g} \cdot s_{B_g} \mod p$ for every gate g and $s_{\mathsf{out}} = 1 \mod p$, which is equivalent to

$$\prod_{g \in \mathsf{Gates}} (1 - (1 - s_{A_g} \cdot s_{B_g} - s_g)^2) \cdot (1 - (1 - s_{\mathsf{out}})^2) = 1 \mod p. \tag{12}$$

Here, we apply a similar trick to the one used by Katsumata [Kat17] to reduce the degree of the above equation. Namely, we first remark that the above equation is satisfied if and only if all terms $(1 - s_{A_g} \cdot s_{B_g} - s_g)$ and $(1 - s_{\mathsf{out}})$ are 0. Also, remark that $|\mathsf{Gates}| \leq |C| < p$ and all terms $(1 - s_{A_g} \cdot s_{B_g} - s_g)$ and $(1 - s_{\mathsf{out}})$ are in $\{-1, 0, 1\}$, we have

$\sum_{g \in \mathsf{Gates}}(1 - s_{A_g} \cdot s_{B_g} - s_g)^2 + (1 - s_{\mathsf{out}})^2 < p$, and thus this sum is equal to 0 modulo p if and only if all terms $(1 - s_{A_g} \cdot s_{B_g} - s_g)$ and $(1 - s_{\mathsf{out}})$ are 0. Therefore Equation (12) is equivalent to the condition that

$$\sum_{g \in \mathsf{Gates}} (1 - s_{A_g} \cdot s_{B_g} - s_g)^2 + (1 - s_{\mathsf{out}})^2 = 0 \mod p.$$

Then if we define

$$\tilde{C}(x, \{s_g\}_{g \in \mathsf{Gates}}) := 1 + \sum_{g \in \mathsf{Gates}} (1 - s_{A_g} \cdot s_{B_g} - s_g)^2 + (1 - s_{\mathsf{out}})^2,$$

then there exists $\{s_g\}_{g \in \mathsf{Gates}} \in \{0,1\}^{|\mathsf{Gates}|}$ such that $\tilde{C}(x, \{s_g\}_{g \in \mathsf{Gates}}) = 1$ if and only if there exists $w \in \{0,1\}^m$ such that $C(x, w) = 1$. Finally, we remark that we have $s^2 = s$ for any $s \in \{0,1\}$, and thus the degree of $\tilde{C}(x, \cdot)$ is at most 3. This completes the proof. □

C (CRS,DV)-NIZK for Leveled Relations with Sublinear Proof Size

Here, we give variants of our compact (CRS,DV)-NIZK whose proof size is sublinear in the size of the circuit that computes the **NP** relation to prove. This construction only works for **NP** languages with "leveled" relation, which is a relation that can be expressed by a leveled circuit, i.e., a circuit whose gates are divided into L levels, and all incoming wires to a gate of level $i+1$ come from gates of level i. For this case, the proof size of the scheme becomes $|w| + |C|/\log\kappa + \mathsf{poly}(\kappa)$. We note that the construction is basically same as the similar result for (DP,PP)-NIZK in [KNYY19], and many parts of this section is taken verbatim from their paper. We include this section in this paper for completeness.

C.1 Leveled Circuits and Relations.

First, we define leveled circuits and its "special" levels following [BGI16]. We say that a circuit is a leveled circuit of depth D if its gates are partitioned into $D+1$ levels, all input gates are of level 0, all output gates are of level $D+1$, and all incoming wires to a gate of level $i+1$ come from gates of level i for each $i \in [D]$. The width at level i is defined to be the number of gates of level i. For a leveled circuit C of depth D, we define a set $\mathcal{S}_C \subset \{0, ..., D+1\}$ of "special" levels in the following manner. For each $j \in \{0, ..., \lfloor D/\log\kappa \rfloor - 1\}$, $\mathcal{S}_C$ contains one level i in the interval $[j\log\kappa + 1, ..., (j+1)\log\kappa]$ such that the width at level i is the minimum among the width at levels in this interval. (If there exist multiple levels whose width are minimum, we choose the smallest level.) We say that i is a special level if $i \in \mathcal{S}_C$. Let $\mathsf{pre}(i)$ denote the precedent special level of i, i.e., the maximal $i' < i$ such that $i' \in \mathcal{S}_C$ (if such i' does not exist, then we define $\mathsf{pre}(i) := 0$) and L_C denote the largest special level of C, i.e., the largest i such that $i \in \mathcal{S}_C$. It is easy to see that the number of gates of a special level is at most $|C|/\log\kappa$ since $\mathcal{S}_C$ contains levels whose width are the smallest in the corresponding interval of length $\log\kappa$. For any gate g of a special level $i \in \mathcal{S}_C$, we can compute the output value of g as a function of output values of gates of level $\mathsf{pre}(i)$. We denote this function by $\mathsf{EvalfromPre}_g$. Since each special level is at most $2\log\kappa$ far apart from its precedent special level, $\mathsf{EvalfromPre}_g$ can be expressed as a circuit of depth at most $2\log\kappa$. Similarly, we define a function $\mathsf{EvalfromPre}_{\mathsf{out}}$ to be a function that computes the output value of C given output values gates of level L_C as input. Similarly, $\mathsf{EvalfromPre}_{\mathsf{out}}$ can be expressed as a circuit of depth at most $2\log\kappa$.

An **NP** relation $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ is said to be a *leveled relation* if there exists a family $\{C_{n,m} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}\}$ of leveled circuits such that for $x \in \{0,1\}^n$ and $w \in \{0,1\}^m$, we have $C_{n,m}(x,w) = 1$ if and only if $(x,w) \in \mathcal{R}$. In the following, we fix n and m, and omit the subscripts n and m from C for notational simplicity. For $x \in \{0,1\}^n$, we let $\mathsf{SGates}[C(x,\cdot)]$ be the set of all gates of $C(x,\cdot)$ whose level is a special level. For a gate g of $C(x,\cdot)$, we let s_g be the output value of the gate g when $C(x,\cdot)$ is evaluated on input w. We call $w' := (w, \{s_g\}_{g \in \mathsf{SGates}[C(x,\cdot)]})$ an expanded witness of w w.r.t. x and C. It is easy to see that we have $|w'| \le |w| + |C|/\log\kappa$ since $|\mathsf{SGates}[C(x,\cdot)]|$ is at most $|C|/\log\kappa$. Then we define an expanded circuit $\mathsf{ExpCir}_{C(x,\cdot)}$ for the expanded witness as follows.

$\mathsf{ExpCir}_{C(x,\cdot)}(w')$**:** It parses $(w, \{s_g\}_{g \in \mathsf{SGates}[C(x,\cdot)]}) \leftarrow w'$. For all $i \in \mathcal{S}_C$, we denote the output values of gates of level i (in a canonical order) by S_i and we let $S_0 := w$. For all gates g of a special level $i \in \mathcal{S}_C$, it verifies if $s_g = \mathsf{EvalfromPre}_g(S_{\mathsf{pre}(i)})$ holds, and returns 0 if this does not hold. If all check pass, it outputs $\mathsf{EvalfromPre}_{\mathsf{out}}(S_{L_{C(x,\cdot)}})$.

It is easy to see that for any $x \in \{0,1\}^n$, there exists an expanded witness w' such that $\mathsf{ExpCir}_{C(x,\cdot)}(w') = 1$ if and only if there exists a witness $w \in \{0,1\}^m$ such that $C(x,w) = 1$. We can implement $\mathsf{ExpCir}_{C(x,\cdot)}$ by a circuit of depth at most $2\log\kappa + \log(|C|/\log\kappa + 1)$. This can be seen by observing that $\mathsf{ExpCir}_{C(x,\cdot)}$ first computes $\mathsf{EvalfromPre}_g$ for at most $|C|/\log\kappa$ different g and $\mathsf{EvalfromPre}_{\mathsf{out}}$, each of which can be computed by a circuit of depth at most $2\log\kappa$, followed by taking the AND of them. Since the last AND is fan-in at most $|C|/\log\kappa + 1$, this can be implemented by a circuit of depth $\log(|C|/\log\kappa + 1)$ and fan-in 2. Especially, if $|C| = \mathsf{poly}(\kappa)$, then there exists a constant c such that $\mathsf{ExpCir}_{C(x,\cdot)}$ can be computed by a circuit of depth at most $c\log\kappa$.

C.2 CRS-NIZK with Sublinear Proof Size.

For constructing CRS-NIZK with sublinear proof size, we instantiate the construction of CRS-NIZK given in Section 4.2 with the following setting.

- Let $n(\kappa)$ and $m(\kappa)$ be any fixed polynomials. Let $\mathcal{L}$ be an **NP** language defined by a leveled relation $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$. Namely, there exists a leveled circuit C such that for all $x \in \{0,1\}^n$ and $w \in \{0,1\}^m$, we have $C(x,w) = 1$ if and only if $(x,w) \in \mathcal{R}$. We assume that $\mathsf{ExpCir}_{C(x,\cdot)}$ as defined in the previous paragraph can be computed by a circuit of depth at most $c\log\kappa$ for a constant c for all $x \in \{0,1\}^n$.
- Let $\mathsf{SKE} = (\mathsf{SKE.KeyGen}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ be a symmetric key encryption scheme with ciphertext space $\mathcal{CT}$ and key space $\{0,1\}^\ell$ whose ciphertext overhead (i.e., ciphertext length minus message length) is $\mathsf{poly}(\kappa)$ where poly is a polynomial independent of the message length, and whose decryption algorithm is computed by a circuit of depth at most $D_{\mathsf{Dec}}(\kappa)$
- For $x \in \{0,1\}^n$ and $\mathsf{ct} \in \mathcal{CT}$, we define a function $f_{x,\mathsf{ct}}(\mathsf{K}_{\mathsf{SKE}}) := \mathsf{ExpCir}_{C(x,\cdot)}(\mathsf{SKE.Dec}(\mathsf{K}_{\mathsf{SKE}}, \mathsf{ct}))$. We note that the depth of $f_{x,\mathsf{ct}}$ is at most $c\log\kappa + D_{\mathsf{Dec}}(\kappa)$ for every $x \in \{0,1\}^n$ and $\mathsf{ct} \in \mathcal{CT}$.
- Let $\Pi_{\mathsf{HEC}} = (\mathsf{HEC.Setup}, \mathsf{HEC.Commit}, \mathsf{HEC.Open}, \mathsf{HEC.Eval}^{in}, \mathsf{HEC.Eval}^{out}, \mathsf{HEC.Verify})$ be a HEC scheme that supports all circuits of depth at most $c\log\kappa + D_{\mathsf{Dec}}(\kappa)$. Recall that "compact" means that the size of an evaluated signature is $\mathsf{poly}(\kappa)$ that does not depend on the message length or the function to evaluate.
- Let $\Pi_{\mathsf{CRSNIZK}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ be an extractable CRS-NIZK for the language corresponding to the relation $\widetilde{\mathcal{R}}$ defined below:
  $((\mathsf{pp}, \mathsf{com}, \mathsf{com}_{\mathsf{eval}}), (K, R, \pi_{\mathsf{HEC}})) \in \widetilde{\mathcal{R}}$ if and only if the followings are satisfied:
  1. $K \in \{0,1\}^\ell$,
  2. $\mathsf{HEC.Commit}(\mathsf{pp}, K; R) = \mathsf{com}$,
  3. $\mathsf{HEC.Verify}(\mathsf{pp}, \mathsf{com}_{\mathsf{eval}}, 1, \pi_{\mathsf{HEC}}) = \top$.
- In the proving algorithm, we set $\mathsf{ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{K}_{\mathsf{SKE}}, w')$ instead of $\mathsf{ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{K}_{\mathsf{SKE}}, w)$ where w' is the expanded witness of w w.r.t. x and C.

Since we simply changed the language to prove in the construction given in Section 4.2, the security can be proven similarly based on the same assumption.

Proof Size. In the above construction, a proof consists of an encryption ct of the expanded witness w', a commitment com of an SKE key $K \in \{0,1\}^\ell$, and an NIZK proof π_{NIZK} of Π_{CRSNIZK}. The size of expanded witness is at most $|w| + |C|/\log\kappa$, and therefore the size of ct is at most $|w| + |C|/\log\kappa + \mathsf{poly}(\kappa)$. The sizes of com and π_{NIZK} are $\mathsf{poly}(\kappa)$ as analyzed in Section 4.3. In summary, the proof size of the above scheme is at most $|w| + |C|/\log\kappa + \mathsf{poly}(\kappa)$.

C.3 DV-NIZK with Sublinear Proof Size.

For applying the similar idea in the DV setting, we first show a variant of Lemma 6.3.

Lemma C.1. *Let C be a leveled circuit that computes a relation $\mathcal{R}$ on $\{0,1\}^n \times \{0,1\}^m$, i.e., for $(x,w) \in \{0,1\}^n \times \{0,1\}^m$, we have $C(x,w) = 1$ if and only if $(x,w) \in \mathcal{R}$, and p be an integer larger than $|C|$. Then there exists a deterministic algorithm $\mathsf{Exp}'_{C,x}$ and an arithmetic circuit $\tilde{C}'$ on $\mathbb{Z}_p$ with degree at most κ^4 such that we have*

- $|\mathsf{Exp}'_{C,x}(w)| = |w| + |C|/\log\kappa$ *for all $w \in \{0,1\}^m$.*
- *If $C(x,w) = 1$, then we have $\tilde{C}'(x, \mathsf{Exp}'_{C,x}(w)) = 1 \mod p$.*
- *For any $x \in \{0,1\}^n$, if there does not exist $w \in \{0,1\}^m$ such that $C(x,w) = 1$, then there does not exist w' such that $\tilde{C}'(x,w') = 1 \mod p$*

Proof. We let $\mathsf{Exp}'_{C,x}(w)$ be the expanded witness defined in the previous paragraph. As already shown, we have $|\mathsf{Exp}'_{C,x}(w)| = |w| + |C|/\log\kappa$. By the definition, if we let $\mathsf{Exp}'_{C,x}(w) = (w, \{s_g\}_{g\in\mathsf{SGates}})$, then $C(x,w)$ can be computed as

$$\prod_{g\in\mathsf{SGates}} (1-(s_g - \mathsf{EvalfromPre}_g(S_{\mathsf{pre}(i_g)}))^2)\cdot(1-(1-\mathsf{EvalfromPre}_{\mathsf{out}}(S_{L_{C(x,\cdot)}}))^2)$$

where i_g denotes g's level. By using a similar trick to the one used in the proof of Lemma 6.3, the condition that $C(x,w)=1$ is equivalent to the condition that

$$\sum_{g\in\mathsf{SGates}} (s_g - \mathsf{EvalfromPre}_g(S_{\mathsf{pre}(i_g)}))^2 + (1-\mathsf{EvalfromPre}_{\mathsf{out}}(S_{L_{C(x,\cdot)}}))^2 = 0 \mod p.$$

Therefore if we define

$$\tilde{C}'(x, w' = (w, \{s_g\}_{g\in\mathsf{SGates}})) := 1 + \sum_{g\in\mathsf{SGates}} (s_g - \mathsf{EvalfromPre}_g(S_{\mathsf{pre}(i_g)}))^2 + (1-\mathsf{EvalfromPre}_{\mathsf{out}}(S_{L_{C(x,\cdot)}}))^2,$$

then it satisfies the condition required in the lemma. Since the degrees of $\mathsf{EvalfromPre}_g$ $\mathsf{EvalfromPre}_{\mathsf{out}}$ are at most κ^2 as they are implemented by a circuit of depth at most $2\log\kappa$, the degree of $\tilde{C}'(x,\cdot)$ is at most κ^4 as required. □

Then we instantiate the construction of DV-NIZK given in Section 6.2 with replacing $\mathsf{Exp}_{C,x}$ and $\tilde{C}$ with $\mathsf{Exp}'_{C,x}$ and $\tilde{C}'$, respectively. Security can be proven similarly. The size of $\mathsf{ct}_{\mathsf{SKE}} = \mathsf{SKE.Enc}(K, \mathsf{Exp}'_{C,x}(w))$ is $|w| + |C|/\log\kappa + \mathsf{poly}(\kappa)$. Moreover, we note that we still have $D = \mathsf{poly}(\kappa)$ since the degree of $f_{x,\mathsf{ct}}(\cdot) := \tilde{C}'(x, \mathsf{SKE.Dec}(\cdot, \mathsf{ct}))$ is $\mathsf{poly}(\kappa)$ since the degree of $\tilde{C}'$ is at most κ^4 as shown above. Therefore the sizes of all other components of a proof still remain $\mathsf{poly}(\kappa)$. In summary, the total proof size is $|w| + |C|/\log\kappa + \mathsf{poly}(\kappa)$.

D HEC from Homomorphic Trapdoor Function

Here, we show that a homomorphic trapdoor function (HTDF) implies HEC. We note that this was already observed by Gorbunov et al. [GVW15] though they did not give a formal definition of HEC. We include the construction for completeness.

D.1 Definition of HTDF

A homomorphic trapdoor function (HTDF) with domain $\mathcal{U}$ on which a distribution $\mathcal{D}_{\mathcal{U}}$ is defined , range $\mathcal{V}$, index space $\mathcal{X}$ for a function class $\mathcal{C} = \{C : \mathcal{X}^\ell \to \mathcal{X}\}$ consists of five PPT algorithms $(\mathsf{HTDF.KeyGen}, f, \mathsf{Inv}, \mathsf{HTDF.Eval}^{in}, \mathsf{HTDF.Eval}^{out})$.

$\mathsf{HTDF.KeyGen}(1^\kappa)$: The key generation algorithm takes as input the security parameter 1^κ, and outputs a public key pk and a secret key sk.

$f_{\mathsf{pk},x}(u)$ This is a polynomial-time computable polynomial function from $\mathcal{U}$ to $\mathcal{V}$ indexed by pk and $x \in \mathcal{X}$.

$\mathsf{Inv}_{\mathsf{sk},x}(v)$: The inversion algorithm is indexed by sk and $x \in \mathcal{X}$, takes $v \in \mathcal{V}$ as input and outputs $u \in \mathcal{U}$.

$\mathsf{HTDF.Eval}^{in}(\mathsf{pk}, C, (x_1, u_1), ..., (x_\ell, u_\ell))$**:** The inner evaluation algorithm is a deterministic algorithm that takes a public key pk, a circuit $C \in \mathcal{C}$ and $(x_i, u_i) \in \mathcal{X} \times \mathcal{U}$ for $i \in [\ell]$ as input, and outputs $u^* \in \mathcal{U}$.

$\mathsf{HTDF.Eval}^{out}(\mathsf{pk}, C, v_1, ..., v_\ell)$**:** The outer evaluation algorithm is a deterministic algorithm that takes a public key pk, a circuit $C \in \mathcal{C}$ and $v_i \in \mathcal{X} \times \mathcal{V}$ for $i \in [\ell]$ as input, and outputs $v^* \in \mathcal{V}$.

Correctness. For all $\kappa \in \mathbb{N}$, $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{HTDF.KeyGen}(1^\kappa)$, $x_1, ..., x_\ell \in \mathcal{X}$, and $C \in \mathcal{C}$, we have

$$f_{\mathsf{pk},C(x_1,...,x_\ell)}(\mathsf{HTDF.Eval}^{in}(C, (x_1, u_1), ..., (x_\ell, u_\ell))) = \mathsf{HTDF.Eval}^{out}(C, v_1, ..., v_\ell).$$

Distributional Equivalence of Inversion. We have

$$\{\mathsf{pk}, \mathsf{sk}, x, u, v\} \overset{\text{stat}}{\approx} \{\mathsf{pk}, \mathsf{sk}, x, u', v'\}$$

where $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{HTDF.KeyGen}$, x is an arbitrary random variable that depends on $(\mathsf{pk}, \mathsf{sk})$, $u \xleftarrow{\$} \mathcal{D}_\mathcal{U}$, $v := f_{\mathsf{pk},x}(u)$, $v' \xleftarrow{\$} \mathcal{V}$, $u' \xleftarrow{\$} \mathsf{Inv}_{\mathsf{sk},x}(v')$.

Claw-free. For all PPT adversary $\mathcal{A}$, we have

$$\Pr\left[\begin{array}{r} f_{\mathsf{pk},x}(u) = f_{\mathsf{pk},x'}(u') \\ u, u' \in \mathcal{U},\ x, x' \in \mathcal{X},\ x \neq x' \end{array} \middle| \begin{array}{rl} (\mathsf{pk}, \mathsf{sk}) & \xleftarrow{\$} \mathsf{HTDF.KeyGen}(1^\kappa), \\ (u, u', x, x') & \xleftarrow{\$} \mathcal{A}(\mathsf{pk}) \end{array}\right] \leq \mathsf{negl}(\kappa).$$

D.2 HEC from HTDF

Here, we give a construction of HEC from HTDF. Let $\Pi_{\mathsf{HTDF}} = (\mathsf{HTDF.KeyGen}, f, \mathsf{Inv}, \mathsf{HTDF.Eval}^{in}, \mathsf{HTDF.Eval}^{out})$ be HTDF with domain $\mathcal{U}$ on which a distribution $\mathcal{D}_\mathcal{U}$ is defined , range $\mathcal{V}$, and index space $\mathcal{X}$ for a function class $\mathcal{C} = \{C : \mathcal{X}^\ell \to \mathcal{X}\}$. Then we construct a HEC scheme $\Pi_{\mathsf{HEC}} = (\mathsf{HEC.Setup}, \mathsf{HEC.Commit}, \mathsf{HEC.Open}, \mathsf{HEC.Eval}^{in}, \mathsf{HEC.Eval}^{out}, \mathsf{HEC.Verify})$ with the message space $\mathcal{X}^\ell$, randomness space $\mathcal{U}^\ell$, and a randomness distribution $\mathcal{D}_\mathcal{U}^\ell$ over $\mathcal{U}^\ell$ for a circuit class $\mathcal{C} = \{C : \mathcal{X}^\ell \to \mathcal{X}\}$ as follows.

$\mathsf{HEC.Setup}(1^\kappa)$: Generate $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{HTDF.KeyGen}(1^\kappa)$ and outputs $\mathsf{pp} := \mathsf{pk}$, $\mathsf{ek} := \mathsf{pk}$, and $\mathsf{msk} := \mathsf{sk}$.

$\mathsf{HEC.Commit}(\mathsf{pp}, \mathbf{x}; \mathbf{u})$**:** Parse $\mathsf{pp} = \mathsf{pk}$, $\mathbf{x} = (x_1, ..., x_\ell)$ and $\mathbf{u} = (u_1, ..., u_\ell)$, and compute $v_i := f_{\mathsf{pk},x_i}(u_i)$ for $i \in [\ell]$, and output $\mathsf{com} := (v_1, ..., v_\ell)$.

$\mathsf{HEC.Open}(\mathsf{msk}, (\mathbf{x}, \mathbf{u}), \mathbf{x}')$**:** Parse $\mathsf{msk} = \mathsf{sk}$, $\mathbf{x} = (x_1, ..., x_\ell)$, $\mathbf{u} = (u_1, ..., u_\ell)$, and $\mathbf{x}' = (x'_1, ..., x'_\ell)$, compute $v_i := f_{\mathsf{pk},x_i}(u_i)$ for $i \in [\ell]$ and $u'_i \xleftarrow{\$} \mathsf{Inv}_{\mathsf{sk},x'_i}(v_i)$, and outputs $\mathbf{u} := (u_1, ..., u_\ell)$.

$\mathsf{HEC.Eval}^{in}(\mathsf{ek}, C, \mathbf{x}, \mathbf{u})$**:** Parse $\mathsf{ek} = \mathsf{pk}$, $\mathbf{x} = (x_1, ..., x_\ell)$, and $\mathbf{u} = (u_1, ..., u_\ell)$, compute $u^* \xleftarrow{\$} \mathsf{HTDF.Eval}^{in}(C, (x_1, u_1), ..., (x_\ell, u_\ell))$, and output $\pi := u^*$.

$\mathsf{HEC.Eval}^{out}(\mathsf{ek}, C, \mathsf{com})$**:** Parse $\mathsf{ek} = \mathsf{pk}$ and $\mathsf{com} = (v_1, ..., v_\ell)$, compute $v^* \xleftarrow{\$} \mathsf{HTDF.Eval}^{out}(C, v_1, ..., v_\ell)$, and output $\mathsf{com}_{\mathsf{eval}} := v^*$

$\mathsf{HEC.Verify}(\mathsf{pp}, \mathsf{com}_{\mathsf{eval}}, z, \pi)$**:** Parse $\mathsf{pp} = \mathsf{pk}$, $\mathsf{com}_{\mathsf{eval}} = v^*$, and $\pi = u^*$, and outputs $\top$ if $v^* = f_{\mathsf{pk},z}(u^*)$ holds and outputs $\perp$ otherwise.

Theorem D.1 (Correctness). *If* Π_{HTDF} *satisfies correctness, then* Π_{HEC} *satisfies correctness.*

Theorem D.2 (Distributional Equivalence of Open). *If* Π_{HTDF} *satisfies the distributional equivalence of inversion, then* Π_{HEC} *satisfies the distributional equivalence of open.*

Proof. (sketch.) First, we remark that for any $x \in \mathcal{X}$, if we generate $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{HTDF.KeyGen}(1^\kappa)$, picks $u \xleftarrow{\$} \mathcal{D}_\mathcal{U}$, and computes $v := f_{\mathsf{pk},x}(u)$, then v is almost uniformly distributed on $\mathcal{V}$ by the distributional equivalence of inversion of Π_{HTDF}. Then it is straightforward to reduce the distributional equivalence of open of Π_{HEC} to the distributional equivalence of inversion of Π_{HTDF}. □

Theorem D.3 (Computational Binding for Evaluated Commitment). *If Π_{HTDF} satisfies the claw-freeness, then Π_{HEC} satisfies the computational binding for evaluated commitment.*

Proof. (sketch.) Suppose that there exists a PPT adversary $\mathcal{A}$ that breaks the computational binding for evaluated commitment of Π_{HEC}. Then we construct $\mathcal{B}$ that breaks the claw-freeness of Π_{HTDF} as follows.

$\mathcal{B}(\mathsf{pk})$**:** Run $\mathcal{A}(\mathsf{pk}, \mathsf{pk})$ to obtain $(\mathbf{x}, \mathbf{u}, C, z^*, u^*)$. Then it parses $\mathbf{x} = (x_1, ..., x_\ell)$ and $\mathbf{u} = (u_1, ..., u_\ell)$, computes $v_i := f_{\mathsf{pk},x_i}(u_i)$ for $i \in [\ell]$, $\tilde{u}^* := \mathsf{HTDF.Eval}^{in}(\mathsf{pk}, C, (x_1, u_1), ..., (x_n, u_n))$, and $\tilde{z}^* := C(x_1, ..., x_\ell)$, and outputs $(u^*, \tilde{u}^*, z^*, \tilde{z}^*)$.

This completes the description of $\mathcal{B}$. If $\mathcal{A}$ succeeds in breaking the computational binding for evaluated commitment, then we have $f_{\mathsf{pk},z^*}(u^*) = v^*$ where $v^* := \mathsf{HTDF.Eval}^{out}(\mathsf{pk}, C, v_1, ..., v_\ell)$ and $z^* \neq \tilde{z}^*$. On the other hand, by the correctness of Π_{HTDF}, we have $f_{\mathsf{pk},\tilde{z}^*}(\tilde{u}^*) = v^*$. This means that $\mathcal{B}$ breaks the claw-freeness of Π_{HTDF}. □

Remark D.4. (Efficiency of Committing and Verification.) The above construction satisfies the requirement of efficient committing and verification since both committing and verification just involve computations of the function f.

E Homomorphic Signature from HEC

Here, we give a construction of homomorphic signature (HomSig) scheme from HEC. The construction is similar to the scheme by Gorbunov, Vaikuntanathan, Wichs [GVW15] except that HTDF is replaced with HEC.

E.1 Definition

Here, we only define single-data HomSig for simplicity. The definition of multi-data version can be found in [GVW15, KNYY19]. The following definition is taken from [KNYY19].

Syntax. Let $\{\mathcal{X}_\kappa\}_{\kappa \in \mathbb{N}}$ be a family of message spaces. Let $\{\mathcal{C}_\kappa\}_{\kappa \in \mathbb{N}}$ be a family of circuits, where $\mathcal{C}_\kappa$ is a set of polynomial sized circuits with domain $\mathcal{X}_\kappa^{\ell(\kappa)}$ and range $\mathcal{Z}_\kappa$. Let $\{\Sigma_{\mathsf{Fresh}_\kappa}\}_{\kappa \in \mathbb{N}}$ and $\{\Sigma_{\mathsf{Evaled}_\kappa}\}_{\kappa \in \mathbb{N}}$ be families of signature spaces, where each of them corresponds to the output space of fresh signatures and evaluated signatures, respectively.

Definition E.1 (Homomorphic Signatures). *A homomorphic signature (HomSig) scheme Π_{HS} with message space $\mathcal{X}$ for the circuit class $\mathcal{C}$ is defined by the following five algorithms:*

$\mathsf{HS.KeyGen}(1^\kappa, 1^\ell) \to (\mathsf{vk}, \mathsf{sk})$*: The key generation algorithm takes as input the security parameter 1^κ and the message length 1^ℓ and outputs a verification key* vk *and a signing key* sk.

$\mathsf{HS.Sign}(\mathsf{sk}, \mathbf{x} = (x_1, \cdots, x_\ell)) \to \boldsymbol{\sigma}$*: The signing algorithm takes as input a signing key* sk *and messages $\mathbf{x} \in \mathcal{X}^\ell$, and outputs a signature $\boldsymbol{\sigma} \in \Sigma_{\mathsf{Fresh}}$.*

$\mathsf{HS.Eval}(\mathsf{vk}, C, \mathbf{x}, \boldsymbol{\sigma}) \to \sigma$*: The signature-evaluation algorithm takes as input a verification key* vk*, a circuit $C : \mathcal{X}^\ell \to \mathcal{Z}$ in $\mathcal{C}$, messages $\mathbf{x} \in \mathcal{X}^\ell$, and a signature $\boldsymbol{\sigma} \in \Sigma_{\mathsf{Fresh}}$ and outputs an evaluated signature $\sigma \in \Sigma_{\mathsf{Evaled}}$.*

$\mathsf{HS.VerifyFresh}(\mathsf{vk}, \mathbf{x}, \boldsymbol{\sigma}) \to \top$ *or* $\bot$*: The fresh verification algorithm takes as input a verification key* vk*, messages $\mathbf{x} \in \mathcal{X}^\ell$, and a signature $\boldsymbol{\sigma} \in \Sigma_{\mathsf{Fresh}}$, and outputs $\top$ if the signature is valid and outputs $\bot$ otherwise.*

$\mathsf{HS.VerifyEvaled}(\mathsf{vk}, C, z, \sigma) \to \top$ *or* $\bot$*: The evaluated verification algorithm takes as input the verification key* vk*, a circuit $C \in \mathcal{C}$, a message $z \in \mathcal{Z}$, and a signature $\sigma \in \Sigma_{\mathsf{Evaled}}$, and outputs $\top$ if the signature is valid and outputs $\bot$ otherwise.*

Correctness. There are two types of correctness which a HomSig scheme must satisfy: signing correctness and evaluation correctness. Formally, they are defined as follows:

Definition E.2 (Correctness). *We say a homomorphic authentication scheme* Π_{HS} *is correct, if for all* $\kappa \in \mathbb{N}$, $\ell \in \mathsf{poly}(\kappa)$, *messages* $\mathbf{x} = (x_1, \cdots, x_\ell) \in \mathcal{X}^\ell$, *and* $(\mathsf{vk}, \mathsf{sk}) \in \mathsf{HS.KeyGen}(1^\kappa, 1^\ell)$ *the following two conditions hold:*
(1) Signing Correctness: *For all* $\boldsymbol{\sigma} \in \mathsf{HS.Sign}(\mathsf{sk}, \mathbf{x})$ *in* Σ_{Fresh}, *we have*

$$\Pr[\mathsf{HS.VerifyFresh}(\mathsf{vk}, \mathbf{x}, \boldsymbol{\sigma}) = \top] = 1.$$

(2) Evaluation Correctness: *For all circuits* $C \in \mathcal{C}$, *signatures* $\boldsymbol{\sigma}$ *such that* $\mathsf{HS.VerifyFresh}(\mathsf{vk}, \mathbf{x}, \boldsymbol{\sigma}) = \top$, *and* $\sigma \in \mathsf{HS.Eval}(C, \mathbf{x}, \boldsymbol{\sigma})$ *in* Σ_{Evaled}, *we have*

$$\Pr[\mathsf{HS.VerifyEvaled}(\mathsf{vk}, C, C(\mathbf{x}), \sigma) = \top] = 1.$$

(Single-Shot) Unforgeability. We now define *single-shot* unforgeability for a HomSig scheme, where the adversary must declare the challenge messages all at once. Below we assume that checking membership of $\mathcal{C}, \Sigma_{\mathsf{Fresh}}, \Sigma_{\mathsf{Evaled}}$ can be done efficiently. The security notion is defined formally by the following game between a challenger and an adversary $\mathcal{A}$.

Setup: At the beginning of the game, the adversary $\mathcal{A}$ is given 1^κ as input and sends 1^ℓ to the challenger. Then the challenger generates a signing-verification key pair $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{HS.KeyGen}(1^\kappa, 1^\ell)$ and gives vk to $\mathcal{A}$.

Signing Query: The adversary $\mathcal{A}$ submits a set of messages $\mathbf{x} \in \mathcal{X}^\ell$ to be signed. The challenger responds by creating a signature $\boldsymbol{\sigma} \xleftarrow{\$} \mathsf{HS.Sign}(\mathsf{sk}, \mathbf{x})$ and sends $\boldsymbol{\sigma} \in \Sigma_{\mathsf{Fresh}}$ to $\mathcal{A}$. Here, $\mathcal{A}$ can query a set of messages only once.

Forgery: Then the adversary $\mathcal{A}$ outputs a circuit $C^\star$, a message $z^\star \in \mathcal{Z}$, and a signature $\sigma^\star$ as the forgery. We say that $\mathcal{A}$ wins the game if:

1. $C^\star \in \mathcal{C}$ and $\sigma^\star \in \Sigma_{\mathsf{Evaled}}$;
2. $C^\star(\mathbf{x}) \neq z^\star$; and
3. $\mathsf{HS.VerifyEvaled}(\mathsf{vk}, C, z^\star, \sigma^\star) = \top$.

The advantage of an adversary winning the above game is defined by $\Pr[\mathcal{A} \text{ wins}]$, where the probability is taken over the randomness used by the challenger and the adversary.

Definition E.3 ((Single-Shot) Unforgeability). *A homomorphic signature scheme* Π_{HS} *is said to satisfy (single-shot)* statistical *unforgeability if for any (possibly inefficient) adversary* $\mathcal{A}$ *the advantage* $\Pr[\mathcal{A} \textit{ wins}]$ *of the above game is negligible. In case it only holds for adversaries that are computationally bounded, we say it satisfies* computational *unforgeability.*

We say Π_{HS} satisfies a weaker notion of *selective* (single-shot) unforgeability in case no adversary $\mathcal{A}$ that commits to the challenge messages $\mathbf{x}$ before seeing the verification key vk can win the above game with more than negligible probability.

Context-Hiding. We now define context-hiding for a HomSig scheme. This security notion roughly states that an evaluated signature σ_C does not leak any information of the initial messages $\mathbf{x}$ other than the value $C(\mathbf{x})$.

Definition E.4 (Statistical Context-Hiding). *A homomorphic signature scheme* Π_{HS} *is statistically context-hiding if for all* $\kappa \in \mathbb{N}$, $\ell \in \mathsf{poly}(\kappa)$, *there exists a PPT simulator* $\mathsf{HS.Sim}$ *such that, for any* $(\mathsf{vk}, \mathsf{sk}) \in \mathsf{HS.KeyGen}(1^\kappa, 1^\ell)$, $C \in \mathcal{C}$, *any pair* $(\mathbf{x}, z) \in \{(\mathbf{x}, z) \in \mathcal{X}^\ell \times \mathcal{Z} \mid C(\mathbf{x}) = z\}$, *and* $\boldsymbol{\sigma} \in \mathsf{HS.Sign}(\mathsf{sk}, \mathbf{x})$, *we have*

$$\{\sigma \xleftarrow{\$} \mathsf{HS.Eval}(C, \mathbf{x}, \boldsymbol{\sigma})\} \overset{\text{stat}}{\approx} \{\sigma \xleftarrow{\$} \mathsf{HS.Sim}(\mathsf{vk}, \mathsf{sk}, C, z)\},$$

where the probability is only over the randomness used by the algorithms $\mathsf{HS.Eval}$ *and* $\mathsf{HS.Sim}$.

E.2 Construction of Homomorphic Signatures from HEC

Let $\Pi_{\mathsf{HEC}} = (\mathsf{HEC.Setup}, \mathsf{HEC.Commit}, \mathsf{HEC.Open}, \mathsf{HEC.Eval}^{in}, \mathsf{HEC.Eval}^{out}, \mathsf{HEC.Verify})$ be a HEC scheme with the message space $\mathcal{X}^\ell$, randomness space $\mathcal{R}$, and a randomness distribution $\mathcal{D}_\mathcal{R}$ over $\mathcal{R}$, for a circuit class $\mathcal{C} = \{C : \mathcal{X}^\ell \to \mathcal{Z}\}$. Here, we assume $0 \in \mathcal{X}$ for simplicity, but $0^\ell \in \mathcal{X}^\ell$ that appears in the following description can be replaced with an arbitrary element in $\mathcal{X}^\ell$. Then our construction of single-data HomSig is provided as follows:

$\mathsf{HS.KeyGen}(1^\kappa, 1^\ell)$: On input the security parameter 1^κ and the message length 1^ℓ, generate $(\mathsf{HEC.pp}, \mathsf{HEC.ek}, \mathsf{HEC.msk}) \xleftarrow{\$} \mathsf{HEC.Setup}(1^\kappa)$, pick $\overline{R} \xleftarrow{\$} \mathcal{D}_\mathcal{R}$, compute $\mathsf{com} := \mathsf{HEC.Commit}(\mathsf{HEC.pp}, 0^\ell; \overline{R})$, and outputs $\mathsf{pk} := (\mathsf{HEC.pp}, \mathsf{HEC.ek}, \mathsf{com})$ and $\mathsf{sk} := (\mathsf{HEC.msk}, \overline{R})$.

$\mathsf{HS.Sign}(\mathsf{sk}, \mathbf{x})$: On input $\mathsf{sk} = (\mathsf{HEC.msk}, \overline{R})$ and $\mathbf{x} \in \mathcal{X}^\ell$, compute $R \xleftarrow{\$} \mathsf{HEC.Open}(\mathsf{HEC.msk}, (0^\ell, \overline{R}), \mathbf{x})$, and output $\boldsymbol{\sigma} = R$.

$\mathsf{HS.Eval}(\mathsf{pk}, C, \mathbf{x}, \boldsymbol{\sigma})$: Parse $\mathsf{pk} = (\mathsf{HEC.pp}, \mathsf{HEC.ek}, \mathsf{com})$ and $\boldsymbol{\sigma} = R$, compute $\pi_{\mathsf{HEC}} \xleftarrow{\$} \mathsf{HEC.Eval}^{in}(\mathsf{HEC.ek}, C, \mathbf{x}, R)$, and output $\sigma = \pi_{\mathsf{HEC}}$.

$\mathsf{HS.VerifyFresh}(\mathsf{pk}, \mathbf{x}, \boldsymbol{\sigma})$: Parse $\mathsf{pk} = (\mathsf{HEC.pp}, \mathsf{HEC.ek}, \mathsf{com})$ and $\boldsymbol{\sigma} = R$, and checks if $\mathsf{com} = \mathsf{HEC.Commit}(\mathsf{HEC.pp}, \mathbf{x}; R)$ holds. If this holds output $\top$, otherwise output $\bot$.

$\mathsf{HS.VerifyEvaled}(\mathsf{pk}, C, z, \sigma)$: Parse $\mathsf{pk} = (\mathsf{HEC.pp}, \mathsf{HEC.ek}, \mathsf{com})$ and $\sigma = \pi_{\mathsf{HEC}}$, computes $\mathsf{com}_{\mathsf{eval}} := \mathsf{HEC.Eval}^{out}(\mathsf{HEC.ek}, C, \mathsf{com})$, and checks if $\mathsf{HEC.Verify}(\mathsf{HEC.pp}, \mathsf{com}_{\mathsf{eval}}, z, \pi_{\mathsf{HEC}}) = \top$ holds. If this holds output $\top$, otherwise output $\bot$.

Correctness.

Theorem E.5 (Correctness). *If Π_{HEC} satisfies correctness and the distributional equivalence for open, then Π_{HS} satisfies the evaluation correctness and signing correctness.*

Proof. The evaluation correctness immediately follows from the correctness of HEC. The signing correctness immediately follows from the equivocality of HEC, which in turn follows from the distributional equivalence for open. □

Security. The following theorems are straightforward to prove.

Theorem E.6 (Unforgeability). *If Π_{HEC} satisfies the binding for evaluated commitment, then Π_{HS} satisfies the selective single-shot unforgeability.*

Theorem E.7 (Context-hiding). *If Π_{HEC} satisfies the context-hiding, then Π_{HS} satisfies the context-hiding.*

Remark E.8. (Instantiations.) If we instantiate the scheme based on the HEC in Section 3.2, we obtain a compact context-hiding HomSig scheme based on the CDHER assumption where "compact" means that the signature size does not depend on the size of the circuit to evaluate. We note the resulting scheme is exactly the same as the HomSig scheme given in [KNYY19]. If we instantiate the scheme based on the HEC in Section 3.3, then we obtain a non-compact context-hiding HomSig scheme based on the CDH assumption.

Remark E.9. (Extension to multi-data scheme) By using the generic transformation from single-data scheme to multi-data scheme by Gorbunov et al. [GVW15], we obtain multi-data HomSig scheme based on HEC. On the other hand, unlike the HTDF-based construction, the resulting scheme *does not* satisfy the amortized verification efficiency, which means that a verifier only needs to perform a computation depending on the size of the circuit only once and then he can reuse it for verifying signatures generated by multiple signers w.r.t. multiple data sets.

Contents