

# New Constructions of Reusable Designated-Verifier NIZKs

Alex Lombardi  
MIT\*

Willy Quach  
Northeastern University†

Ron D. Rothblum  
Technion‡

Daniel Wichs  
Northeastern University§

David J. Wu  
University of Virginia¶

## Abstract

Non-interactive zero-knowledge arguments (NIZKs) for NP are an important cryptographic primitive, but we currently only have instantiations under a few specific assumptions. Notably, we are missing constructions from the learning with errors (LWE) assumption, the Diffie-Hellman (CDH/DDH) assumption, and the learning parity with noise (LPN) assumption.

In this paper, we study a relaxation of NIZKs to the *designated-verifier* setting (DV-NIZK), where a trusted setup generates a common reference string together with a secret key for the verifier. We want *reusable* schemes, which allow the verifier to reuse the secret key to verify many different proofs, and soundness should hold even if the malicious prover learns whether various proofs are accepted or rejected. Such reusable DV-NIZKs were recently constructed under the CDH assumption, but it was open whether they can also be constructed under LWE or LPN.

We also consider an extension of reusable DV-NIZKs to the *malicious designated-verifier* setting (MDV-NIZK). In this setting, the only trusted setup consists of a common random string. However, there is also an additional untrusted setup in which the verifier chooses a public/secret key needed to generate/verify proofs, respectively. We require that zero-knowledge holds even if the public key is chosen maliciously by the verifier. Such reusable MDV-NIZKs were recently constructed under the “one-more CDH” assumption, but constructions under CDH/LWE/LPN remained open.

In this work, we give new constructions of (reusable) DV-NIZKs and MDV-NIZKs using generic primitives that can be instantiated under CDH, LWE, or LPN.

## 1 Introduction

Zero-knowledge proofs [GMR89] allow a prover to convince a verifier that a statement is true without revealing anything beyond this fact. While standard zero-knowledge proof systems are interactive, Blum, Feldman, and Micali [BFM88] introduced the concept of a non-interactive zero-knowledge

---

\*Email: [alexjl@mit.edu](mailto:alexjl@mit.edu). Research supported in part by an NDSEG fellowship. Research supported in part by NSF Grants CNS-1350619 and CNS-1414119, and by the Defense Advanced Research Projects Agency (DARPA) and the U.S. Army Research Office under contracts W911NF-15-C-0226 and W911NF-15-C-0236.

†Email: [quach.w@husky.neu.edu](mailto:quach.w@husky.neu.edu).

‡Email: [rothblum@cs.technion.ac.il](mailto:rothblum@cs.technion.ac.il). Supported in part by the Israeli Science Foundation (Grant No. 1262/18) and the Technion Hiroshi Fujiwara cyber security research center and the Israel cyber directorate.

§Email: [wichs@ccs.neu.edu](mailto:wichs@ccs.neu.edu). Research supported by NSF grants CNS-1314722, CNS-1413964, CNS-1750795 and the Alfred P. Sloan Research Fellowship.

¶Email: [dwu4@virginia.edu](mailto:dwu4@virginia.edu). Part of this work was done while visiting the Technion.

(NIZK) proof, which consists of a single message sent by the prover to the verifier. Although such NIZKs cannot exist in the plain model, they are realizable in the *common reference string (CRS)* model, where a trusted third party generates and publishes a common reference string chosen either uniformly random or from some specified distribution. We currently have NIZKs for general NP languages under several specific assumptions, such as: (doubly-enhanced) trapdoor permutations, which can be instantiated from factoring [BFM88, SMP87, FLS99, SCO<sup>+</sup>01, Gol11], the Diffie-Hellman assumption over bilinear groups [CHK03, GOS06], optimal hardness of the learning with errors (LWE) assumption<sup>1</sup> [CCH<sup>+</sup>18], and circular-secure fully homomorphic encryption [CLW18]. We also have such NIZKs in the random-oracle model [FS86]. However, we are lacking constructions from several standard assumptions, most notably the computational or decisional Diffie-Hellman assumptions (CDH, DDH), the plain learning with errors (LWE) assumption, and the learning parity with noise (LPN) assumption.

**Designated-verifier NIZK.** We consider a relaxation of NIZKs to the *designated-verifier* model (DV-NIZK). In this model, a trusted third party generates a CRS together with a secret key, which is given to the verifier and is used to verify proofs. Throughout this work, we focus on the problem of achieving *reusable* (i.e., multi-theorem) security. This means that soundness should hold even if the scheme is used multiple times and a malicious prover can test whether the verifier accepts or rejects various proofs.

While reusable DV-NIZKs appear to be a non-trivial relaxation of standard NIZKs,<sup>2</sup> we did not (until recently) have any constructions of such DV-NIZKs under assumptions not known to imply standard NIZKs. Very recently, the works of [CH19, KNY19, QRW19] constructed such DV-NIZKs under the CDH assumption. However, it was left as an open problem whether such DV-NIZKs can be constructed under LWE or LPN.

We note that the work [KW18a] constructed an orthogonal notion of reusable “designated-prover” NIZKs (DP-NIZK) under LWE, where the trusted third party generates a CRS together with a secret key that is given to the prover and needed to generate proofs. In addition, the work [BCGI18] constructed “preprocessing NIZKs” (PP-NIZK), in which the trusted third party generates both a secret proving key and a secret verification key, under variants of the LPN assumption over large fields.

**Malicious-designated-verifier NIZK.** We also consider a strengthening of DV-NIZKs to the malicious-designated-verifier model (MDV-NIZKs) introduced by [QRW19]. In this model, a trusted party only generates a common *uniformly random* string. The verifier can then choose a public/secret key pair, which is used to generate/verify proofs, respectively. Soundness is required to hold when the verifier generates these keys honestly, while zero-knowledge is required to hold even when the verifier may generate the public key maliciously. MDV-NIZKs can equivalently be thought of as 2-round zero-knowledge protocols in the common random string model, in which the verifier’s first-round message is reusable; namely, the public key chosen by the verifier can be thought of as a first-round message.

Very recently, the work of [QRW19] showed how to construct such MDV-NIZKs under the “one-more CDH assumption.” This is an interactive assumption that has received much less scrutiny than standard CDH/DDH.

---

<sup>1</sup>This means that no polynomial-time attacker can break LWE with any probability better than random guessing.

<sup>2</sup>The public verifiability of traditional NIZKs immediately implies reusable soundness.

## 1.1 Our Results

In this work, we propose a framework for constructing reusable DV-NIZKs from generic assumptions. One instantiation of this framework yields reusable DV-NIZKs generically from any public-key encryption together with a secret-key encryption scheme satisfying a weak form of key-dependent message (KDM) security. Both components can be instantiated under any of the CDH/LWE/LPN assumptions, so we obtain constructions of DV-NIZKs under these assumptions. In particular, we obtain the following theorem:

**Theorem 1.1** (informal). *Assuming the existence of public-key encryption and secret-key encryption that is KDM-secure with respect to projections (see Definition 2.24), there exist reusable designated-verifier NIZK arguments for NP. In particular, there exist reusable DV-NIZKs under either the CDH assumption, the LWE assumption, or the LPN assumption with noise rate  $O(\frac{1}{\sqrt{n}})$ .*

We then show how to construct reusable *malicious* DV-NIZKs from any (receiver-extractable) 2-round oblivious transfer (OT) in the common random string model and the same form of KDM-secure SKE. This yields instantiations of MDV-NIZKs under the CDH/LWE/LPN assumptions using the OT constructions of [PVW08, DGH<sup>+</sup>19], as summarized by the following theorem.

**Theorem 1.2** (informal). *Assuming the existence of “receiver-extractable 2-message OT” (see Definition 2.14) and secret-key encryption that is KDM-secure with respect to projections, there exist reusable malicious designated-verifier NIZK arguments for NP. In particular, there exist reusable MDV-NIZKs under the CDH assumption, the LWE assumption, or the LPN assumption with noise rate  $n^{-(\frac{1}{2}+\varepsilon)}$  for any  $\varepsilon > 0$ .*

More generally, we give a compiler converting any  $\Sigma$ -protocol (or even more generally, any “zero-knowledge PCP” [KPT97, IMS12]) into a DV-NIZK using a form of *single-key attribute-based encryption* (ABE) satisfying a certain “function-hiding (under decryption queries)” property. Collusion-resistant ABE is only known from specific algebraic assumptions over bilinear maps [SW05, GPSW06] or lattices [GVW13, BGG<sup>+</sup>14], but *single-key* ABE can be constructed from any public-key encryption scheme [SS10, GVW12]. While we are unable to construct our variant of ABE (i.e., one that satisfies our function-hiding property) from an arbitrary public-key encryption (PKE) scheme, we show how to construct it by additionally relying on KDM-secure SKE, using a technique recently developed in [KW18b, KMT19]. However, in addition to this construction, we outline an alternate approach for building single-key ABE with our function-hiding property (using the standard lattice-based ABE [BGG<sup>+</sup>14]) in Appendix C. As a result, we believe that our new notion may be helpful in order to construct DV-NIZKs from other assumptions in the future. Note that if one could construct DV-NIZKs from any semantically-secure PKE scheme, it would show that semantically-secure PKE generically implies CCA-secure PKE (via the Naor-Yung paradigm [NY90]), which would resolve a major long-standing open problem. More modestly, one could hope to construct DV-NIZKs generically from any CCA-2 secure encryption. Our techniques may offer some hope towards realizing these exciting possibilities.

Our techniques depart significantly from the prior constructions of DV-NIZKs and MDV-NIZKs in [CH19, KNY19, QRW19]. In particular, those works relied on the hidden-bits model from [FLS99] and used a variant of the Cramer-Shoup hash-proof system under CDH [CS98, CS02] to instantiate the hidden bits for a designated verifier. Unfortunately, we do not have good hash-proof systems under LWE/LPN and so it does not appear that these techniques can be used when starting from “noisy assumptions” (among other concerns). As we describe below, we take a vastly different

approach and do not rely on the hidden bits model. One disadvantage of our results is that, while [CH19, KNY19, QRW19] achieve *statistically* sound (M)DV-NIZK proofs, we only get argument systems with *computational* soundness.<sup>3</sup>

**Application to reusable non-interactive secure computation.** We note that MDV-NIZKs can be used to obtain new solutions to the problem of reusable non-interactive secure computation (rNISC) [CDI<sup>+</sup>19]. In this setting, there is a public function  $f$  and a receiver (Rachel) publishes a “query” using her secret input  $x$ . Later a sender (Sam) can send a “response” using his secret input  $y$  and ensure that Rachel only learns  $f(x, y)$ . We further want Rachel’s query to be reusable so that Sam can send many different responses with various values  $y_i$  and have Rachel learn  $f(x, y_i)$  without compromising security. The main difficulty is that a malicious Sam can send malformed responses and, by observing whether Rachel aborts or not, can potentially learn information about her input  $x$ . Previously, we had instantiations of rNISC (in the CRS model) using 2-round (malicious) oblivious transfer (OT) *and* NIZKs, or more recently, via a black-box use of oblivious linear-function evaluation (OLE) [CDI<sup>+</sup>19]. However, we had no constructions under many standard assumptions, including any of CDH/DDH, LPN or LWE. It turns out that we can easily use MDV-NIZKs instead of standard NIZKs (along with 2-round malicious OT) to solve this problem. In particular, Rachel sends OT queries corresponding to her input  $x$  as well as the public-key of an MDV-NIZK. Sam then creates a garbled circuit for  $f(\cdot, y)$  with his input  $y$  hard-coded, and sends the labels via the OT responses; in addition he encrypts  $y$  (under a public key in the CRS) and proves that he computed the garbled circuit and the OT responses correctly and consistently with the encrypted  $y$ . We can simulate Sam’s view (including Rachel’s output) by checking the MDV-NIZK to decide if Rachel aborts or not; if the MDV-NIZK verifies then we can extract  $y$  from the encryption and be sure that Rachel correctly outputs  $f(x, y)$ . Using our instantiations of MDV-NIZKs along with known constructions of 2-round OT from [PVW08, DGH<sup>+</sup>19], we get instantiations of rNISC under CDH, LPN or LWE.

## 1.2 Recent Related Work

**NIZKs from LWE.** In a concurrent and independent work, Peikert and Shiehian [PS19] construct NIZKs from the plain LWE assumption, which in particular yields reusable (M)DV-NIZKs from LWE. While the [PS19] NIZK has the major advantage of being publicly verifiable, we note that our usage of LWE only relies on plain Regev (public-key) encryption [Reg05] rather than more complex lattice-based primitives.

**KDM-Secure SKE and hinting PRGs.** A preliminary version of this work [LQR<sup>+</sup>19] relied on *hinting PRGs* [KW18b] rather than KDM-secure SKE in the statement of our main results (Theorem 1.1 and Theorem 1.2). In a concurrent work, Kitagawa, Matsuda and Tanaka [KMT19] modify the “signaling technique” of [KW18b] with the goal of constructing CCA-secure encryption (similarly to [KW18b]). The [KMT19] modification of [KW18b] can be plugged into our construction of (M)DV-NIZKs to obtain our LPN-based instantiation of DV-NIZKs (in addition to our original CDH-based and LWE-based instantiations). We have updated our paper to reflect this improved result.

In addition, shortly after our update, Kitagawa and Matsuda [KM19] independently noted the same combination of [LQR<sup>+</sup>19] and [KMT19] within a follow-up work.

<sup>3</sup>Our construction is also computational zero-knowledge. None of the recent constructions of DV-NIZKs satisfy statistical zero knowledge.

**2-Message OT from CDH/LPN.** In another concurrent work, Döttling, Garg, Hajiabadi, Masny, and Wichs [DGH<sup>+</sup>19] construct 2-round OT from the CDH/LPN assumptions. Their construction can be directly combined with our generic transformations, yielding the CDH/LPN-based instantiations of MDV-NIZKs in this updated version of our paper (in addition to our original DDH-based and LWE-based instantiations).

### 1.3 Our Techniques

Our approach starts with the construction of *non-reusable* DV-NIZKs from any public-key encryption, due to Pass, shelat, and Vaikuntanathan [PsV06]. The [PsV06] construction relies on a  $\Sigma$ -protocol [CDS94] with 1-bit challenges for an NP-complete language, such as Blum’s protocol for graph Hamiltonicity [Blu86]. Recall that a  $\Sigma$ -protocol is a 3-round protocol, where the prover sends a value  $a$ , the challenger chooses a bit  $b \in \{0, 1\}$ , and the prover replies with a response  $z$ ; the verifier checks the validity of the transcript  $(a, b, z)$  at the end. The protocol should have special soundness (if there are two accepting transcripts  $(a, 0, z_0), (a, 1, z_1)$  with the same  $a$  then the statement must be true) and special honest-verifier zero-knowledge (given  $b$  ahead of time, we can simulate the transcript  $(a, b, z)$  without knowing a witness). The scheme also relies on a public-key encryption scheme PKE. The non-reusable DV-NIZK of [PsV06] is defined by invoking  $\lambda$  (security parameter) independent copies of the following base scheme in parallel:

- **Setup:** The common reference string consists of PKE public keys,  $(\text{pk}_0, \text{pk}_1)$ . The verifier’s secret verification key  $(b, \text{sk}_b)$  consists of a random bit  $b$  along with the secret key  $\text{sk}_b$  for the corresponding public key  $\text{pk}_b$ .
- **Proof generation:** On input a statement  $x$  and a witness  $w$ , the prover  $P$  first computes the first message  $a$  of the  $\Sigma$ -protocol. Then, the prover computes responses  $(z_0, z_1)$  for both possible challenge bits  $b \in \{0, 1\}$ , respectively, and outputs  $(a, \text{ct}_0 = \text{Encrypt}(\text{pk}_0, z_0), \text{ct}_1 = \text{Encrypt}(\text{pk}_1, z_1))$ .
- **Proof verification:** Given a proof  $(a, \text{ct}_0, \text{ct}_1)$ , and verification key  $(b, \text{sk}_b)$ , the verifier computes  $z = \text{Decrypt}(\text{sk}_b, \text{ct}_b)$  and accepts if and only if  $(a, b, z)$  is a valid transcript.

Zero-knowledge of the DV-NIZK holds because the simulator knows the bits  $b$  of the verifier in each invocation and can therefore simulate the  $\Sigma$ -protocol transcripts  $(a, b, z_b)$  without knowing a witness. It can create the ciphertext  $\text{ct}_b$  by encrypting  $z_b$  and can put an arbitrary dummy value in the “other” ciphertext  $\text{ct}_{1-b}$ ; this is indistinguishable by the security of the encryption.

Non-reusable soundness of the DV-NIZK follows from the special soundness of the  $\Sigma$ -protocol. If the statement is false then, for each  $a$ , there is only one challenge bit  $b$  that has a valid response  $z$ , and therefore the prover would have to correctly guess the bit  $b$  in each of the  $\lambda$  invocations of the above base protocol. This can only happen with negligible probability.

Unfortunately, as noted in [PsV06], the soundness of this scheme is completely broken if the prover is allowed to query a verification oracle to test whether arbitrary proofs accept or reject—by creating a proof of a true statement and putting an incorrect value in (say) the ciphertext  $\text{ct}_0$  of the  $i^{\text{th}}$  copy of the protocol, the adversary learns the verifier’s bit  $b$  in the  $i^{\text{th}}$  copy after learning whether the proof accepts or rejects. The adversary can eventually recover all of the verifier’s bits  $b$  this way and, once it does so, it is easy to construct a valid proof of a false statement by using the  $\Sigma$ -protocol simulator to generate valid transcripts  $(a, b, z_b)$ .

To overcome this problem, we replace the use of public-key encryption with a form of *attribute-based encryption*, so that every instance  $x$  yields a *different* sequence of challenge bits  $b$  associated to it.

**Function-hiding ABE.** The main tool that we use in this work is a variant of single-key ABE satisfying a certain *function-hiding* property. Recall that an ABE scheme ( $\text{Setup}$ ,  $\text{KeyGen}$ ,  $\text{Encrypt}$ ,  $\text{Decrypt}$ ) allows for the encryption of a message  $m$  under public parameters  $\text{pp}$  with respect to an *attribute*  $x$  resulting in a ciphertext  $\text{ct}$ . The ciphertext  $\text{ct}$  can be decrypted using a secret key  $\text{sk}_f$  associated with a function  $f$  and the message  $m$  is recovered if  $f(x) = 1$ . On the other hand, if  $f(x) = 0$ , then semantic security holds and nothing about the message is revealed even given  $\text{sk}_f$ . In this work, we focus on schemes satisfying semantic security in the presence of a *single* secret key  $\text{sk}_f$ ; ABE schemes satisfying single-key security can be constructed from any public-key encryption scheme [SS10, GVW12].

The function-hiding property we consider in this work requires that for any function  $f$ , oracle access to the decryption oracle  $\text{Decrypt}(\text{sk}_f, \cdot)$  does not reveal anything about the function  $f$  beyond whether  $\text{sk}_f$  was qualified to decrypt the ciphertexts in question. More formally, we consider schemes where the attribute  $x$  is given in the clear as part of the ciphertext  $\text{ct}$ , and require that an oracle call of the form  $\text{Decrypt}(\text{sk}_f, \text{ct})$  can be simulated using the master secret key  $\text{msk}$  along with the value  $f(x)$ , but without any additional knowledge of  $f$ .

At first glance, this property seems closely related to the standard notion of *CCA-security*, in which access to a decryption oracle does not compromise semantic security. However, these two notions appear to be incomparable. In particular, function-hiding can hold even if access to the decryption oracle completely breaks semantic security while CCA-2 security can hold even if access to the decryption oracle completely reveals the function  $f$ . Nonetheless, we observe that some of the techniques previously developed for obtaining CCA-security are also useful for obtaining our form of function-hiding.

Given this notion, our main contributions can be broken down into two steps: (1) showing that function-hiding ABE yields DV-NIZKs, and (2) giving constructions of function-hiding ABE. With respect to (1), we note that assuming the existence of public-key encryption, our notion of function-hiding ABE is actually *equivalent* to DV-NIZKs for NP; we show the converse to (1) in Appendix D.

**The compiler.** Here, we describe a simplified version of our DV-NIZK protocol using three main ingredients:

- A  $\Sigma$ -protocol [CDS94] with 1-bit challenges for an NP-complete language  $\mathcal{L}$ , such as Blum’s protocol for graph Hamiltonicity [Blu86]. (In Section 4, we describe our compiler more generally in the language of zero-knowledge PCPs, which can be instantiated via  $\Sigma$ -protocols as a special case).
- An ABE scheme  $\text{ABE} = (\text{Setup}, \text{KeyGen}, \text{Encrypt}, \text{Decrypt})$  satisfying single-key security and function-hiding as described above. (In Section 4, we describe our compiler more generally using a new primitive called attribute-based secure function evaluation (AB-SFE), for which ABE is a special case).
- A pseudorandom function PRF that can be evaluated by ABE. In this simplified scheme, it suffices for PRF to output a single bit. (In Section 4, we describe our compiler by reusing the

same PRF and ABE parameters across invocations, while here we apply parallel repetition of completely independent schemes).

Our DV-NIZK protocol is defined by invoking  $\lambda$  (security parameter) independent copies of the following base scheme in parallel.

- **Setup:** The common reference string consists of the public parameters  $\text{pp}$  for ABE. The verifier’s secret verification key  $(k, \text{sk}_f)$  consists of a PRF key  $k$  along with an ABE secret key  $\text{sk}_f$  for evaluating the function

$$f(x, b) = 1 \iff \text{PRF}(k, x) = b.$$

- **Proof generation:** On input a statement  $x$  and a witness  $w$ , the prover  $P$  computes the first message  $a$  in the  $\Sigma$ -protocol. Then, the prover computes responses  $(z_0, z_1)$  for both possible challenge bits  $b \in \{0, 1\}$ , respectively, and computes an ABE encryption of  $z_b$  with respect to attribute  $(x, b)$ . This yields ciphertexts  $(\text{ct}_0, \text{ct}_1)$ ; the prover sends  $(a, \text{ct}_0, \text{ct}_1)$  to the verifier.
- **Proof verification:** The verifier first computes  $y = \text{PRF}(k, x)$ . Then, the verifier decrypts  $\text{ct}_y$  using its secret verification key  $\text{sk}_f$  to obtain the prover’s response  $z_y$ . Finally, the verifier checks that the proof  $(a, y, z_y)$  is valid and accepts if this is the case.

We claim that the DV-NIZK is reusably sound. Consider any fixed statement<sup>4</sup>  $x \notin \mathcal{L}$  and an adversary that makes arbitrary verification queries and eventually produces an accepting proof for  $x$ . First, without loss of generality, we claim that we can consider an adversary that never makes a verification query on  $x$  itself; if an adversary had a non-negligible probability of making such a query and getting an accepting response then it would be able to win the game without making the query! Second, we claim that the challenges  $y = \text{PRF}(k, x)$  for each invocation, which are used when verifying the adversary’s final proof for  $x$ , are pseudorandom from the prover’s perspective. This holds *even* if the prover is given oracle access to the verifier on all statements  $x' \neq x$  since, by the function-hiding of ABE, these queries can be simulated given only the values  $\text{PRF}(k, x')$  without revealing any additional info about  $k$ . But, by the special soundness of the  $\Sigma$ -protocol, the only way that the adversary can produce an accepting proof would be to guess all of the values  $y$  used in each of the  $\lambda$  invocations, which only happens with negligible probability.

Moreover, we claim that the DV-NIZK is zero-knowledge. In an honestly-generated proof  $\pi = (a, \text{ct}_0, \text{ct}_1)$ , on instance  $x$ , the verifier can compute the response  $z_y$  for  $y = \text{PRF}(k, x)$ , but the response  $z_{1-y}$  is computationally hidden by semantic security of ABE. This means that  $\pi$  can be simulated given only  $(k, (a, z_y))$ , which is in turn simulatable given only  $x$  by the special honest-verifier zero-knowledge of the  $\Sigma$ -protocol.

We provide the formal description of our compiler in Section 4.

**Constructing function-hiding ABE.** We now describe two ways<sup>5</sup> to construct a (single-key) ABE scheme that satisfies our function-hiding property. Combined with our compiler above, this suffices to construct DV-NIZKs (i.e., the results from Theorem 1.1). In the body of the paper, we

<sup>4</sup>This suffices for *non-adaptive* soundness. Adaptive soundness (in which the cheating prover is allowed to adaptively select a false statement  $x \notin \mathcal{L}$  after seeing the common reference string) can be achieved either by complexity leveraging [BB04] (see Remark 2.6) or by relying on a *trapdoor*  $\Sigma$ -protocol [CLW18] (see Remark 4.4).

<sup>5</sup>We refer to a previous version of this work [LQR<sup>+</sup>19] for an additional approach based on lossy trapdoor functions.

will focus on the second candidate based on KDM-secure SKE for two main reasons: (1) it enables instantiations from CDH/LWE/LPN (and correspondingly, DV-NIZKs from these assumptions); and (2) it readily generalizes to notions beyond ABE, which as we discuss in greater detail below, enables constructions of *MDV-NIZKs* from CDH/LWE/LPN.

- **Lattice-based ABE:** First, we observe that a simple variant of the lattice-based ABE construction from [BGG<sup>+</sup>14] satisfies our notion of function-hiding. Namely, we can modify the construction [BGG<sup>+</sup>14] so that the decryption algorithm (with either the master secret key or a function key) can *fully recover* the encryption randomness used to construct a particular ciphertext, and in doing so, verify that a ciphertext is well-formed (i.e., could be output by the honest encryption algorithm). If the scheme supports this randomness recovery property, function-privacy essentially follows from (perfect) correctness of the underlying scheme. This high-level idea of leveraging *randomness recovery* is a common theme in our constructions. We provide additional details in Appendix C.
- **PKE and KDM-secure SKE:** Following the approach of [KW18b, KMT19], we show that any single-key ABE scheme can be used to construct an ABE scheme satisfying function-hiding with respect to decryption queries. The amplification procedure additionally requires the existence of a secret-key encryption scheme SKE that is KDM-secure for a simple class of functions. As shown in [BHHO08, ACPS09, DG17, BLSV18, DGHM18], such secret-key encryption schemes can be constructed from the CDH/LWE/LPN assumptions, and hence, give instantiations of function-hiding ABE from CDH/LWE/LPN.

In fact, the exact construction of *CCA-secure* ABE in [KW18b] (and the modification introduced in [KMT19]) can also be shown to satisfy function-hiding. However, as noted above, CCA-security does not generically imply our notion of function-hiding or vice versa. In this work, we describe a simplified variant of the [KMT19] compiler that suffices to construct function-hiding ABE and then analyze its security.

We now provide a description of (our simplification of) the [KW18b, KMT19] construction. Take any (single-key) ABE scheme ABE, a secret-key encryption scheme SKE, a public-key encryption scheme PKE, an equivocable commitment scheme Com, and consider the following modified ABE scheme:

- **Public parameters:** ABE public parameters  $\text{pp}$ , PKE public key  $\text{pk}$ , and commitment common reference string  $\text{crs}$ .
- **Key generation:** This is unmodified from ABE: an ABE master secret key is used to generate keys  $\text{sk}_f$  associated to functions  $f$ .
- **Encryption algorithm:** On input the public parameters  $(\text{pp}, \text{pk}, \text{crs})$ , an attribute  $x$  and a message  $m$ :
  1. Sample a SKE secret key  $s \leftarrow \{0, 1\}^\lambda$ .
  2. Sample random strings  $\rho_i$  (for  $i \in [\lambda]$ ) and  $R_{i,b}$  (for  $i \in [\lambda], b \in \{0, 1\}$ ).
  3. Output commitments  $\text{com}_i = \text{Com}(\text{crs}, s_i; \rho_i)$  to the bits of the secret key  $s$ , a “joint encryption matrix”  $M = \{\text{ct}_{i,b}\}_{i \in [\lambda], b \in \{0,1\}}$  consisting of  $\lambda$  ABE ciphertexts and  $\lambda$  PKE ciphertexts using the strings  $R_{i,b}$  as encryption randomness. Lastly, also output a symmetric encryption  $\text{ct}_0 \leftarrow \text{SKE.Encrypt}(s, m \| \{R_{i,s_i}\}_{i \in [\lambda]})$  of the message



$m$  concatenated with a subset of  $\{R_{i,b}\}$  corresponding to the bits of  $s$  (using fresh encryption randomness).

We now elaborate on the ciphertexts  $\text{ct}_{i,b}$ :

- \* For every index  $i \in [\lambda]$ ,  $\text{ct}_{i,0}$  is an ABE ciphertext computed using  $(\text{pp}, x)$  and randomness  $R_{i,0}$ , while  $\text{ct}_{i,1}$  is a PKE ciphertext computed using  $\text{pk}$  and randomness  $R_{i,1}$ .
- \* As for the underlying messages: for every index  $i \in [\lambda]$ ,  $\text{ct}_{i,s_i}$  is an encryption of  $\rho_i$ , while  $\text{ct}_{i,1-s_i}$  is an encryption of  $\perp$  (a dummy message).

Following [KW18b, KMT19], we provide some high-level intuition for this encryption algorithm. For a fixed pair  $(i, b)$ , call a ciphertext  $\text{ct}_{i,b}$  “good” with respect to commitment  $\text{com}_i$  if there exists commitment randomness  $\rho_i$  such that  $\text{com}_i = \text{Com}(\text{crs}, b; \rho_i)$  and  $\text{ct}_{i,b}$  is a well-formed encryption of  $\rho_i$ . Then, given a qualified secret key  $\text{sk}_f$ , an honestly-generated matrix  $M = \{\text{ct}_{i,b}\}$  encodes the SKE-secret key  $s$ : we have  $s_i = b$  if and only if  $\text{ct}_{i,b}$  is “good,” so  $s_i$  can be identified by decrypting  $\text{ct}_{i,0}$  (using  $\text{sk}_f$ ) and checking whether the underlying message is  $\perp$ .<sup>6</sup>

Moreover, the binding property of the commitment scheme  $\text{Com}$  guarantees that for every  $(i, \text{com}_i, \text{ct}_{i,0}, \text{ct}_{i,1})$ , there is *at most one bit*  $b$  such that  $\text{ct}_{i,b}$  is “good”; in other words, even malformed ciphertexts encode at most one secret key  $s$ . This introduces enough redundancy in the scheme so that CCA-like security properties can be guaranteed *without* the decryption procedure fully recovering the encryption randomness. In particular, the randomness  $\{R_{i,1-s_i}\}$  can be left unrecoverable (even given a qualified key  $\text{sk}_f$ ), which is what allows for a proof of semantic security.

We leave a detailed discussion of the decryption algorithm to Section 5, but decryption roughly proceeds by recovering some of the overall encryption randomness (using  $\text{sk}_f$ )—namely,  $(s, \rho, \{R_{i,s_i}\}_{i \in [\lambda]})$ —and then checking that each ciphertext of the form  $\text{ct}_{i,s_i}$  is “good” (which can be done *without* using  $\text{sk}_f$ ). To argue semantic security, we proceed in three steps:

- Switch the commitment  $\text{crs}$  to an “equivocal mode” so that  $\text{com} = (\text{com}_i)_{i \in [\lambda]}$  can be explained as a commitment to *any string* (with an appropriate choice of randomness).
- Show that (in equivocal mode)  $M = \{\text{ct}_{i,b}\}$  can be simulated (using  $\rho$  and  $\{R_{i,b}\}$ ) without knowing  $s$ .
- At this point,  $\text{ct}_0$  is guaranteed to hide  $m$  by invoking KDM-security of SKE.

To argue function-hiding, we show that the ciphertext  $\text{ct}$  can be decrypted in two equivalent ways: (1) by using the “honest” decryption algorithm with the ABE secret key  $\text{sk}_f$ ; and (2) using *the PKE secret key associated with*  $\text{pk}$  (and outputting a message only when  $f(x) = 1$ ). Semantic security of the scheme is guaranteed to hold in the presence of  $\text{sk}_f$  (the secret key of the honest decryptor), but an adversary with oracle access to one of these two decryption functions cannot distinguish them. Since the second procedure hides  $f(x')$  for any attribute  $x'$  not queried by the adversary, function-hiding follows.

<sup>6</sup>In the actual decryption procedure, a more sophisticated mechanism is employed to identify  $s$  in order to handle malformed ciphertexts.

Combined with the instantiations of KDM-secure SKE from various assumptions [BH08, ACPS09, DG17, BLSV18, DGHM18] and the fact that single-key ABE follows from PKE [SS10, GVW12], this approach gives a single-key function-hiding ABE scheme from any of the CDH/LWE/LPN assumptions. For our LPN instantiation, we require noise rate  $1/\sqrt{n}$  to instantiate the public-key encryption scheme [Ale03]. We describe this construction and its analysis in Section 5.

**Obtaining malicious security.** So far, we have shown how to construct DV-NIZKs from function-hiding single-key ABE and provided several instantiations of the latter object from concrete assumptions. However, the DV-NIZKs obtained in this fashion necessarily requires that the verifier’s secret key be generated by a trusted party; indeed, if the verifier is malicious and allowed to set up this DV-NIZK, it can simply sample an ABE key-pair  $(\text{pp}, \text{msk})$  and remember the entire master secret key. This clearly breaks zero-knowledge.

To construct a malicious DV-NIZK scheme, we intuitively have to replace the trusted setup of an ABE scheme with a form of reusable *non-interactive* two-party computation that implements a similar functionality. Specifically, we introduce a more general primitive called *attribute-based secure function evaluation* (AB-SFE, see Definition 3.1). At a high-level, an AB-SFE scheme is a two-party protocol between a sender and a receiver and parameterized by a public function  $F: \mathcal{X} \times \mathcal{Y} \rightarrow \{0, 1\}$ . The sender holds a public attribute  $x \in \mathcal{X}$  and a secret message  $m$  while the receiver holds a secret input  $y \in \mathcal{Y}$ . At the end of the protocol, the receiver should learn  $m$  only if  $F(x, y) = 1$  (otherwise, the receiver should learn nothing). The protocol should be non-interactive in the following sense: at the beginning of the protocol, the receiver publishes a public key  $\text{pk}_y$  based on its secret input  $y$ ; thereafter, the sender with its attribute-message pair  $(x, m)$  can send a single message to the receiver that allows the receiver to learn  $m$  whenever  $F(x, y) = 1$ . The receiver’s initial message  $\text{pk}_y$  should both hide  $y$  and be *reusable* for arbitrarily many protocol executions. We say AB-SFE schemes satisfying this property are “key-hiding”. In addition, we are interested in security even against malicious receivers that choose  $\text{pk}_y$  maliciously. We note that a single-key ABE scheme can be used to construct a secure AB-SFE scheme satisfying a much weaker security notion where a trusted party generates the receiver’s message  $\text{pk}_y$ .

Similarly to our use of ABE in the generic compiler above, an AB-SFE scheme can be used to compile a  $\Sigma$ -protocol (or more generally, zero-knowledge PCPs) to obtain a reusable DV-NIZK; moreover, this compiled DV-NIZK is secure even against *malicious* verifiers and is therefore an MDV-NIZK. Specifically, in our construction, we replace the ABE scheme with an AB-SFE scheme with respect to the function  $F$  where  $F((x, b), k) = 1$  if and only if  $\text{PRF}(k, x) = b$ . If we use a maliciously-secure AB-SFE scheme, we only rely on a trusted setup to generate a *uniformly random* common reference string. We then allow the verifier to (1) sample a PRF key  $k$  and (2) compute the receiver message  $\text{pk}_k$  for the AB-SFE protocol (with private input  $k$ ) itself. Malicious security of the AB-SFE protocol exactly allows us to prove malicious zero-knowledge of the compiled protocol. As in the case of ABE, *soundness* of the compiled protocol relies on a form of AB-SFE security where the receiver’s input  $y$  is hidden from the sender even given access to an appropriately-defined decryption oracle.

We obtain AB-SFE schemes that can be plugged into our compiler in two steps:

- **Constructing weak key-hiding AB-SFE.** First, we combine a form of malicious-secure 2-message OT [PVW08, DGH<sup>+</sup>19] with garbled circuits to obtain an AB-SFE scheme that

satisfies *weak key-hiding*. Namely, the receiver’s input  $y$  is hidden to an adversary that does *not* have access to the decryption oracle. We describe this construction in Section 5.2.

- **Amplifying weak key-hiding to strong key-hiding.** Then, we apply the [KW18b, KMT19] transformation to the weak key-hiding AB-SFE scheme from above to obtain an AB-SFE scheme that satisfies strong key-hiding where the receiver’s input  $y$  is hidden even in the presence of the decryption oracle. This allows for new instantiations of MDV-NIZK from any of the CDH/LWE/LPN assumptions (Theorem 1.2). Our LPN-based instantiation requires noise rate  $n^{-(\frac{1}{2}+\epsilon)}$  for some  $\epsilon > 0$  in order to implement the [DGH<sup>+</sup>19] OT protocol. We describe this construction in Section 5.3.

We provide a formal definition of AB-SFE in Section 3, and the full construction and analysis in Section 5.

## 2 Preliminaries

We write  $\lambda$  to denote a security parameter. We say that a function  $f$  is negligible in  $\lambda$ , denoted  $\text{negl}(\lambda)$ , if  $f(\lambda) = o(1/\lambda^c)$  for all  $c \in \mathbb{N}$ . We say an event happens with negligible probability if the probability of the event happening is negligible, and that it happens with overwhelming probability if its complement occurs with negligible probability. We say that an algorithm is efficient if it runs in probabilistic polynomial-time (PPT) in the length of its inputs. We write  $\text{poly}(\lambda)$  to denote a function bounded by a (fixed) polynomial in  $\lambda$ . We say that two families of distributions  $\mathcal{D}_1 = \{\mathcal{D}_{1,\lambda}\}_{\lambda \in \mathbb{N}}$  and  $\mathcal{D}_2 = \{\mathcal{D}_{2,\lambda}\}_{\lambda \in \mathbb{N}}$  are computationally indistinguishable if no PPT adversary can distinguish samples from  $\mathcal{D}_1$  and  $\mathcal{D}_2$  except with negligible probability, and we denote this by writing  $\mathcal{D}_1 \stackrel{c}{\approx} \mathcal{D}_2$ . We write  $\mathcal{D}_1 \stackrel{s}{\approx} \mathcal{D}_2$  to denote that  $\mathcal{D}_1$  and  $\mathcal{D}_2$  are statistically indistinguishable (i.e., the statistical distance between  $\mathcal{D}_1$  and  $\mathcal{D}_2$  is bounded by a negligible function).

For an integer  $n \geq 1$ , we write  $[n]$  to denote the set of integers  $\{1, \dots, n\}$ . For a finite set  $S$ , we write  $x \stackrel{R}{\leftarrow} S$  to denote that  $x$  is sampled uniformly at random from  $S$ . For a distribution  $\mathcal{D}$ , we write  $x \leftarrow \mathcal{D}$  to denote that  $x$  is sampled from  $\mathcal{D}$ . For two finite sets  $\mathcal{X}$  and  $\mathcal{Y}$ , we write  $\text{Funs}[\mathcal{X}, \mathcal{Y}]$  to denote the set of functions from  $\mathcal{X}$  to  $\mathcal{Y}$ . We now review the main cryptographic building blocks we use in this work.

**Definition 2.1** (Pseudorandom Function [GGM84]). A pseudorandom function (PRF) with key-space  $\mathcal{K}$ , domain  $\mathcal{X}$ , and range  $\mathcal{Y}$  is an efficiently-computable function  $\text{PRF}: \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$  such that for all PPT adversaries  $\mathcal{A}$ ,

$$\Pr[k \stackrel{R}{\leftarrow} \mathcal{K} : \mathcal{A}^{\text{PRF}(k,\cdot)}(1^\lambda) = 1] - \Pr[f \stackrel{R}{\leftarrow} \text{Funs}[\mathcal{X}, \mathcal{Y}] : \mathcal{A}^{f(\cdot)}(1^\lambda) = 1] = \text{negl}(\lambda).$$

**Definition 2.2** (Public-Key Encryption). A public-key encryption (PKE) scheme with message space  $\mathcal{M}$  is a tuple  $\text{PKE} = (\text{PKE.KeyGen}, \text{PKE.Encrypt}, \text{PKE.Decrypt})$  with the following properties:

- $\text{PKE.KeyGen}(1^\lambda) \rightarrow (\text{pk}, \text{sk})$ : On input the security parameter  $\lambda$ , the key-generation algorithm outputs a public key  $\text{pk}$  and a secret key  $\text{sk}$ .
- $\text{PKE.Encrypt}(\text{pk}, m) \rightarrow \text{ct}$ : On input a public key  $\text{pk}$  and a message  $m \in \mathcal{M}$ , the encryption algorithm outputs a ciphertext  $\text{ct}$ .

- $\text{PKE.Decrypt}(\text{sk}, \text{ct}) \rightarrow m/\perp$ : On input a secret key  $\text{sk}$  and a ciphertext  $\text{ct}$ , the decryption algorithm either outputs a message  $m \in \mathcal{M}$  or a special symbol  $\perp$ .

Moreover, PKE should satisfy the following properties:

- **Correctness:** PKE is perfectly correct if for all message  $m \in \mathcal{M}$ ,

$$\Pr[(\text{pk}, \text{sk}) \leftarrow \text{PKE.KeyGen}(1^\lambda) : \text{PKE.Decrypt}(\text{sk}, \text{PKE.Encrypt}(\text{pk}, m)) = m] = 1.$$

- **Semantic security:** For all PPT adversaries  $\mathcal{A}$ ,

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

where  $(\text{pk}, \text{sk}) \leftarrow \text{PKE.KeyGen}(1^\lambda)$  and the encryption oracle  $\mathcal{O}_b(\text{pk}, m_0, m_1)$  outputs the ciphertext  $\text{ct}_b \leftarrow \text{PKE.Encrypt}(\text{pk}, m_b)$ .

## 2.1 Designated-Verifier NIZKs

We now introduce the notion of a designated-verifier non-interactive zero-knowledge (DV-NIZK) argument. We use a refined notion where there are separate setup and key-generation algorithms. The setup algorithm outputs a common reference string (possibly a common *random* string) for the scheme. The CRS can be reused by different verifiers, who would generate their own public and private keys. In the traditional notion of designated-verifier NIZKs, the setup and key-generation algorithms are combined, and the public key  $\text{pk}$  is simply included as part of the CRS.

**Definition 2.3** (Designated-Verifier NIZK Argument). Let  $\mathcal{L}$  be an NP language associated with an NP relation  $\mathcal{R}$ . A *designated-verifier non-interactive zero-knowledge (DV-NIZK) argument for  $\mathcal{L}$*  consists of a tuple of three efficient algorithms  $\text{dvNIZK} = (\text{dvNIZK.Setup}, \text{dvNIZK.KeyGen}, \text{dvNIZK.Prove}, \text{dvNIZK.Verify})$  with the following properties:

- $\text{dvNIZK.Setup}(1^\lambda) \rightarrow \text{crs}$ : On input the security parameter  $\lambda$ , the setup algorithm outputs a common reference string  $\text{crs}$ . If  $\text{dvNIZK.Setup}$  outputs a *uniformly random string*, then we say that the DV-NIZK scheme is in the *common random string* model.
- $\text{dvNIZK.KeyGen}(\text{crs}) \rightarrow (\text{pk}, \text{sk})$ : On input a common reference string  $\text{crs}$ , the key-generation algorithm outputs a public key  $\text{pk}$  and a secret verification key  $\text{sk}$ .
- $\text{dvNIZK.Prove}(\text{crs}, \text{pk}, x, w) \rightarrow \pi$ : On input the common reference string  $\text{crs}$ , a public key  $\text{pk}$ , a statement  $x$ , and a witness  $w$ , the prove algorithm outputs a proof  $\pi$ .
- $\text{dvNIZK.Verify}(\text{crs}, \text{sk}, x, \pi) \rightarrow \{0, 1\}$ : On input the common reference string  $\text{crs}$ , a secret key  $\text{sk}$ , a statement  $x$ , and a proof  $\pi$ , the verification algorithm outputs a bit  $b \in \{0, 1\}$ .

Moreover,  $\text{dvNIZK}$  should satisfy the following properties:

- **Completeness:** For all  $(x, w) \in \mathcal{R}$ , and taking  $\text{crs} \leftarrow \text{dvNIZK.Setup}(1^\lambda)$  and  $(\text{pk}, \text{sk}) \leftarrow \text{dvNIZK.KeyGen}(\text{crs})$ , we have that

$$\Pr[\pi \leftarrow \text{dvNIZK.Prove}(\text{crs}, \text{pk}, x, w) : \text{dvNIZK.Verify}(\text{crs}, \text{sk}, x, \pi) = 1] = 1.$$

- **Soundness:** We consider two variants of soundness:

- **Non-adaptive soundness:** For all  $x \notin \mathcal{L}$ , all PPT adversaries  $\mathcal{A}$ ,

$$\Pr \left[ \pi \leftarrow \mathcal{A}^{\text{dvNIZK.Verify}(\text{crs}, \text{sk}, \cdot, \cdot)}(1^\lambda, \text{crs}, \text{pk}, x) : \text{dvNIZK.Verify}(\text{crs}, \text{sk}, x, \pi) = 1 \right] = \text{negl}(\lambda),$$

where  $\text{crs} \leftarrow \text{dvNIZK.Setup}(1^\lambda)$  and  $(\text{pk}, \text{sk}) \leftarrow \text{dvNIZK.KeyGen}(\text{crs})$ .

- **Adaptive soundness:** For all PPT adversaries  $\mathcal{A}$ ,

$$\Pr \left[ (x, \pi) \leftarrow \mathcal{A}^{\text{dvNIZK.Verify}(\text{crs}, \text{sk}, \cdot, \cdot)}(1^\lambda, \text{crs}) : \right. \\ \left. x \notin \mathcal{L} \wedge \text{dvNIZK.Verify}(\text{crs}, \text{sk}, x, \pi) = 1 \right] = \text{negl}(\lambda),$$

where  $(\text{crs}, \text{pk}, \text{sk}) \leftarrow \text{dvNIZK.Setup}(1^\lambda)$  and  $(\text{pk}, \text{sk}) \leftarrow \text{dvNIZK.KeyGen}(\text{crs})$ .

- **Zero-knowledge:** For all PPT adversaries  $\mathcal{A}$ , there exists a PPT simulator  $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$  such that

$$\left| \Pr[\mathcal{A}^{\mathcal{O}_0(\text{crs}, \text{pk}, \cdot, \cdot)}(\text{crs}, \text{pk}, \text{sk}) = 1] - \Pr[\mathcal{A}^{\mathcal{O}_1(\text{st}_{\mathcal{S}}, \cdot, \cdot)}(\overline{\text{crs}}, \overline{\text{pk}}, \overline{\text{sk}}) = 1] \right| = \text{negl}(\lambda),$$

where  $\text{crs} \leftarrow \text{dvNIZK.Setup}(1^\lambda)$ ,  $(\text{pk}, \text{sk}) \leftarrow \text{dvNIZK.KeyGen}(\text{crs})$ , and  $(\text{st}_{\mathcal{S}}, \overline{\text{crs}}, \overline{\text{pk}}, \overline{\text{sk}}) \leftarrow \mathcal{S}_1(1^\lambda)$ , the oracle  $\mathcal{O}_0(\text{crs}, \text{pk}, x, w)$  outputs  $\text{dvNIZK.Prove}(\text{crs}, \text{pk}, x, w)$  if  $\mathcal{R}(x, w) = 1$  and  $\perp$  otherwise, and the oracle  $\mathcal{O}_1(\text{st}_{\mathcal{S}}, x, w)$  outputs  $\mathcal{S}_2(\text{st}_{\mathcal{S}}, x)$  if  $\mathcal{R}(x, w) = 1$  and  $\perp$  otherwise.

**Definition 2.4** (Malicious Designated-Verifier NIZKs [QRW19]). Let  $\text{dvNIZK}$  be a DV-NIZK for a language  $\mathcal{L}$  (with associated NP relation  $\mathcal{R}$ ). For an adversary  $\mathcal{A}$ , and a simulator  $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ , we define two experiments  $\text{ExptReal}_{\mathcal{A}}(\lambda)$  and  $\text{ExptSim}_{\mathcal{A}, \mathcal{S}}(\lambda)$  as follows:

- **Setup:** In  $\text{ExptReal}_{\mathcal{A}}(\lambda)$ , the challenger samples  $\text{crs} \leftarrow \text{dvNIZK.Setup}(1^\lambda)$  and in  $\text{ExptSim}_{\mathcal{A}, \mathcal{S}}(\lambda)$ , the challenger samples  $(\text{st}_{\mathcal{S}}, \overline{\text{crs}}) \leftarrow \mathcal{S}_1(1^\lambda)$ . In  $\text{ExptReal}_{\mathcal{A}}(\lambda)$ , the challenger gives  $\text{crs}$  to  $\mathcal{A}$ , while in  $\text{ExptSim}_{\mathcal{A}, \mathcal{S}}(\lambda)$ , the challenger gives  $\overline{\text{crs}}$  to  $\mathcal{A}$ . Then,  $\mathcal{A}$  outputs a public key  $\text{pk}$ .
- **Verification queries:** Algorithm  $\mathcal{A}$  is then given access to a verification oracle. In both experiments, if  $\mathcal{R}(x, w) \neq 1$ , then the challenger replies with  $\perp$ . Otherwise, in  $\text{ExptReal}_{\mathcal{A}}(\lambda)$ , the challenger replies with  $\pi \leftarrow \text{dvNIZK.Prove}(\text{crs}, \text{pk}, x, w)$ , and in  $\text{ExptSim}_{\mathcal{A}, \mathcal{S}}(\lambda)$ , the challenger replies with  $\overline{\pi} \leftarrow \mathcal{S}_2(\text{st}_{\mathcal{S}}, \text{pk}, x)$ .
- **Output:** At the end of the experiment, the adversary outputs a bit  $b' \in \{0, 1\}$ , which is the output of the experiment.

We say that  $\text{dvNIZK}$  provides *zero-knowledge against malicious verifiers* if for all PPT adversaries  $\mathcal{A}$ , there exists an efficient simulator  $\mathcal{S}$  such that  $\text{ExptReal}_{\mathcal{A}}(\lambda) \stackrel{c}{\approx} \text{ExptSim}_{\mathcal{A}, \mathcal{S}}(\lambda)$ . If  $\text{dvNIZK}$  satisfies this property (in addition to completeness and soundness), then we say that  $\text{dvNIZK}$  is a *malicious-designated-verifier NIZK* (MDV-NIZK).

**Remark 2.5** (Reusability of the CRS with Many Public Keys). The zero-knowledge property of Definition 2.4 only provides (multi-theorem) zero-knowledge with respect to a *single* maliciously-generated public key  $\text{pk}$ . Using the “OR trick” transformation from [FLS99], any MDV-NIZK can be generically compiled into one where a single CRS can be reused with an arbitrary polynomial

number of (potentially maliciously-generated) public keys, while preserving zero-knowledge. Note that the original transformation compiled any NIZK in the CRS model with single-theorem zero-knowledge into a multi-theorem version; we note that it also directly applies to the (malicious) designated-verifier setting (essentially because proofs can still be generated publicly). Additionally, if the original MDV-NIZK is in the common random string model, then the resulting protocol is also in the common random string model.

**Remark 2.6** (Adaptive Soundness via Complexity Leveraging). Using the standard technique of complexity leveraging [BB04], a DV-NIZK satisfying non-adaptive soundness also satisfies adaptive soundness at the expense of a super-polynomial loss in the security reduction.

## 2.2 Zero-Knowledge PCPs

**Definition 2.7** (Zero-Knowledge PCP [KPT97, IMS12]). Let  $\mathcal{R}: \{0, 1\}^n \times \{0, 1\}^h \rightarrow \{0, 1\}$  be an NP relation and  $\mathcal{L} \subseteq \{0, 1\}^n$  be the associated language. A *non-adaptive,  $\ell$ -query zero-knowledge PCP* (with alphabet  $\Sigma$ ) for  $\mathcal{L}$  is a tuple of algorithms  $\text{zkPCP} = (\text{zkPCP.Prove}, \text{zkPCP.Query}, \text{zkPCP.Verify})$  with the following properties:

- $\text{zkPCP.Prove}(x, w) \rightarrow \pi$ : On input a statement  $x \in \{0, 1\}^n$  and a witness  $w \in \{0, 1\}^h$ , the prove algorithm outputs a proof  $\pi \in \Sigma^m$ .
- $\text{zkPCP.Query}(x) \rightarrow (\text{st}_x, q_1, \dots, q_\ell)$ : On input a statement  $x \in \{0, 1\}^n$ , the query-generation algorithm outputs a verification state  $\text{st}_x$  and  $\ell$  query indices  $q_1, \dots, q_\ell \in [m]$ .
- $\text{zkPCP.Verify}(\text{st}_x, s_1, \dots, s_\ell) \rightarrow \{0, 1\}$ : On input the verification state  $\text{st}$  and a set of responses  $s_1, \dots, s_\ell \in \Sigma$ , the verify algorithm outputs a bit  $b \in \{0, 1\}$ .

Moreover,  $\text{zkPCP}$  should satisfy the following properties:

- **Efficiency:** The running time of  $\text{zkPCP.Prove}$ ,  $\text{zkPCP.Query}$ , and  $\text{zkPCP.Verify}$  should be bounded by  $\text{poly}(n)$ . In particular, this means that  $m = \text{poly}(n)$ .
- **Completeness:** For all  $x \in \{0, 1\}^n$  and  $w \in \{0, 1\}^h$  where  $\mathcal{R}(x, w) = 1$ ,

$$\Pr[\text{zkPCP.Verify}(\text{st}_x, \pi_{q_1}, \dots, \pi_{q_\ell}) = 1] = 1,$$

where  $\pi \leftarrow \text{zkPCP.Prove}(x, w)$  and  $(\text{st}_x, q_1, \dots, q_\ell) \leftarrow \text{zkPCP.Query}(x)$ .

- **Soundness:** For all  $x \notin \mathcal{L}$ , all proof strings  $\pi \in \Sigma^m$ ,

$$\Pr[\text{zkPCP.Verify}(\text{st}_x, \pi_{q_1}, \dots, \pi_{q_\ell}) = 1] = \text{negl}(n),$$

where  $(\text{st}_x, q_1, \dots, q_\ell) \leftarrow \text{zkPCP.Query}(x)$ .

- **Zero-knowledge:** For all PPT adversaries  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ , there exists an efficient simulator  $\mathcal{S}$  such that

$$\left| \Pr[b = 1 \mid \mathcal{R}(x, w) = 1] - \Pr[\tilde{b} = 1 \mid \mathcal{R}(x, w) = 1] \right| = \text{negl}(n),$$

where  $(\text{st}_\mathcal{A}, x, w, q_1, \dots, q_\ell) \leftarrow \mathcal{A}_1(1^n)$ ,  $\pi \leftarrow \text{zkPCP.Prove}(x, w)$ ,  $(\tilde{\pi}_1, \dots, \tilde{\pi}_\ell) \leftarrow \mathcal{S}(x, q_1, \dots, q_\ell)$ ,  $b \leftarrow \mathcal{A}_2(\text{st}_\mathcal{A}, \pi_{q_1}, \dots, \pi_{q_\ell})$ , and  $\tilde{b} \leftarrow \mathcal{A}_2(\text{st}_\mathcal{A}, \tilde{\pi}_1, \dots, \tilde{\pi}_\ell)$ ,

**Semi-malicious zero-knowledge.** The zero-knowledge requirement in Definition 2.7 requires that there exists a PPT simulator for an adversary that reads any set of  $\ell$  bits of the PCP, including subsets that would never be output by `zkPCP.Query`. In our constructions, we can rely on the relaxed notion of *semi-malicious* zero-knowledge which only requires simulation for subsets of bits that are output by an invocation of `zkPCP.Query` (for some setting of the randomness). Specifically, we define the following:

**Definition 2.8** (Semi-Malicious Zero-Knowledge). A zero-knowledge PCP `zkPCP` for a language  $\mathcal{L}$  with associated NP relation  $\mathcal{R}$  satisfies semi-malicious zero-knowledge if for all PPT adversaries  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ , there exists a PPT simulator  $\mathcal{S}$  such that

$$|\Pr[\mathcal{A}_2(\text{st}_{\mathcal{A}}, \pi_{q_1}, \dots, \pi_{q_\ell}) = 1 \mid \mathcal{R}(x, w) = 1] - \Pr[\mathcal{A}_2(\text{st}_{\mathcal{A}}, \tilde{\pi}_1, \dots, \tilde{\pi}_\ell) = 1 \mid \mathcal{R}(x, w) = 1]| = \text{negl}(n),$$

where  $(\text{st}_{\mathcal{A}}, x, w, r) \leftarrow \mathcal{A}_1(1^n)$ ,  $(q_1, \dots, q_\ell) \leftarrow \text{zkPCP.Query}(x; r)$ ,  $\pi \leftarrow \text{zkPCP.Prove}(x, w)$ , and  $(\tilde{\pi}_1, \dots, \tilde{\pi}_\ell) \leftarrow \mathcal{S}(x, q_1, \dots, q_\ell)$ .

**Instantiating zero-knowledge PCPs.** As noted by Ishai et al. [IMS12], the original zero-knowledge protocol by Goldreich et al. [GMW86] makes implicit use of an honest-verifier zero-knowledge PCP for graph 3-coloring. To briefly recall, the prover takes a 3-coloring of the graph, randomly permutes the colors, and writes down the colors for each vertex as the PCP. To check the PCP, the (honest) verifier samples a random edge in the graph and reads the colors for the two nodes associated with the edge. It is straightforward to see that if `zkPCP.Query` always outputs a pair of nodes corresponding to some edge in the graph, then this PCP satisfies semi-malicious zero-knowledge. To achieve negligible soundness, we rely on parallel amplification (e.g., by concatenating many independent copies of the PCP) and note that semi-malicious zero-knowledge is indeed preserved under parallel repetition. We state this instantiation below:

**Theorem 2.9** (Semi-Malicious Zero-Knowledge PCP [GMW86]). *Let  $\mathcal{L} \subseteq \{0, 1\}^n$  be an NP language. Then, there exists an  $\ell$ -query zero-knowledge PCP for  $\mathcal{L}$  with alphabet  $\Sigma = \{0, 1, 2\}$  and  $\ell = \text{poly}(n)$ .*

We note that there are many other ways to instantiate the zero-knowledge PCP with the desired properties. For instance, Blum’s protocol for graph Hamiltonicity [Blu86] also implicitly uses a (semi-malicious) zero-knowledge PCP. We can also construct zero-knowledge PCPs (with fully malicious zero knowledge) using multiparty computation (MPC) protocols by using the MPC-in-the-head technique of Ishai et al. [IKOS07]. More broadly,  $\Sigma$ -protocols with a polynomial-size challenge space can generally be viewed as implicitly implementing a (semi-malicious) zero-knowledge PCP.

### 2.3 Attribute-Based Encryption

**Definition 2.10** (Attribute-Based Encryption). An attribute-based encryption (ABE) scheme over a message space  $\mathcal{M}$ , an attribute space  $\mathcal{X}$ , and a function family  $\mathcal{F} = \{f: \mathcal{X} \rightarrow \{0, 1\}\}$  is a tuple of algorithms  $\text{ABE} = (\text{ABE.Setup}, \text{ABE.KeyGen}, \text{ABE.Encrypt}, \text{ABE.Decrypt})$  with the following properties:

- $\text{ABE.Setup}(1^\lambda) \rightarrow (\text{pp}, \text{msk})$ : On input the security parameter  $\lambda$ , the setup algorithm outputs the public parameters  $\text{pp}$  and the master secret key  $\text{msk}$ .
- $\text{ABE.KeyGen}(\text{pp}, \text{msk}, f) \rightarrow \text{sk}_f$ : On input the public parameters  $\text{pp}$ , the master secret key  $\text{msk}$  and a function  $f \in \mathcal{F}$ , the key-generation algorithm outputs a decryption key  $\text{sk}_f$ .

- $\text{ABE.Encrypt}(\text{pp}, x, m) \rightarrow \text{ct}_{x,m}$ : On input the public parameters  $\text{pp}$ , an attribute  $x \in \mathcal{X}$ , and a message  $m \in \mathcal{M}$ , the encryption algorithm outputs a ciphertext  $\text{ct}_{x,m}$ .
- $\text{ABE.Decrypt}(\text{pp}, \text{sk}, \text{ct}) \rightarrow (x, m)$ : On input the public parameters  $\text{pp}$ , a secret key  $\text{sk}$  (which could be the master secret key), and a ciphertext  $\text{ct}$ , the decryption algorithm either outputs an attribute-message pair  $(x, m) \in \mathcal{X} \times \mathcal{M}$  or a special symbol  $\perp$ .

**Definition 2.11** (Correctness). An ABE scheme  $\text{ABE}$  is (perfectly) *correct* if for all messages  $m \in \mathcal{M}$ , all attributes  $x \in \mathcal{X}$ , and all predicates  $f \in \mathcal{F}$ , and setting  $(\text{pp}, \text{msk}) \leftarrow \text{ABE.Setup}(1^\lambda)$ ,

- $\Pr[\text{ABE.Decrypt}(\text{pp}, \text{msk}, \text{ABE.Encrypt}(\text{pp}, x, m)) = (x, m)] = 1$ .
- If  $f(x) = 1$ , then

$$\Pr[\text{ABE.Decrypt}(\text{pp}, \text{ABE.KeyGen}(\text{pp}, \text{msk}, f), \text{ABE.Encrypt}(\text{pp}, x, m)) = (x, m)] = 1.$$

**Definition 2.12** (Security). Let  $\text{ABE}$  be an ABE scheme over an attribute space  $\mathcal{X}$ , message space  $\mathcal{M}$ , and function family  $\mathcal{F}$ . For a security parameter  $\lambda$  and an adversary  $\mathcal{A}$ , we define the ABE security experiment  $\text{Expt}_{\mathcal{A}}^{\text{ABE}}(\lambda, b)$  as follows. The challenger begins by sampling  $(\text{pp}, \text{msk}) \leftarrow \text{ABE.Setup}(1^\lambda)$  and gives  $\text{pp}$  to the adversary  $\mathcal{A}$ . Then  $\mathcal{A}$  is given access to the following oracles:

- **Key-generation oracle:** On input a function  $f \in \mathcal{F}$ , the challenger responds with a key  $\text{sk}_f \leftarrow \text{ABE.KeyGen}(\text{pp}, \text{msk}, f)$ .
- **Challenge oracle:** On input an attribute  $x \in \mathcal{X}$  and a pair of messages  $m_0, m_1 \in \mathcal{M}$ , the challenger responds with a ciphertext  $\text{ct} \leftarrow \text{ABE.Encrypt}(\text{pp}, x, m_b)$ .

At the end of the game, the adversary outputs a bit  $b' \in \{0, 1\}$ , which is also the output of the experiment. An adversary  $\mathcal{A}$  is admissible for the attribute-based encryption security game if it makes one challenge query  $(x, m_0, m_1)$ , and for all key-generation queries  $f$  the adversary makes,  $f(x) = 0$ . We say that  $\text{ABE}$  is *secure* if for all efficient and admissible adversaries  $\mathcal{A}$ ,

$$\left| \Pr[\text{Expt}_{\mathcal{A}}^{\text{ABE}}(\lambda, 0) = 1] - \Pr[\text{Expt}_{\mathcal{A}}^{\text{ABE}}(\lambda, 1) = 1] \right| = \text{negl}(\lambda).$$

Moreover, we say that  $\text{ABE}$  is *single-key secure* if the above property holds for all efficient and admissible adversaries  $\mathcal{A}$  that make at most one key-generation query.

**Function hiding.** Our generic constructions of designated-verifier NIZKs from ABE (and generalizations thereof) relies on an additional (weak) notion of function hiding. While the traditional notion of function hiding asks that the secret decryption key hides the function, our construction relies on a *weaker* notion where we require that *oracle access* to the decryption function does not reveal information about the underlying function (other than what can be directly inferred by the input-output behavior of the function). We give the formal definition below:

**Definition 2.13** (Weak Function Hiding). Let  $\text{ABE}$  be an ABE scheme, and let  $t = t(\lambda)$  be a bound on the length of ciphertext in ABE. We say that  $\text{ABE}$  satisfies *weak function hiding* if there exists an efficient simulator  $\mathcal{S}$  such that for all functions  $f \in \mathcal{F}$ ,  $(\text{pp}, \text{msk}) \leftarrow \text{ABE.Setup}(1^\lambda)$ , and  $\text{sk}_f \leftarrow \text{ABE.KeyGen}(\text{pp}, \text{msk}, f)$

$$\left| \Pr[\mathcal{A}^{\mathcal{O}_1(\text{pp}, \text{sk}_f, \cdot)}(1^\lambda, \text{pp}) = 1] - \Pr[\mathcal{A}^{\mathcal{O}_2(\text{pp}, \text{msk}, \cdot)}(1^\lambda, \text{pp}) = 1] \right| = \text{negl}(\lambda),$$

where the oracles  $\mathcal{O}_1, \mathcal{O}_2$  are defined as follows:



- **Real decryption oracle:** On input  $\text{pp}, \text{sk}_f, \text{ct} \in \{0, 1\}^t$ , the real decryption oracle  $\mathcal{O}_1(\text{pp}, \text{sk}_f, \text{ct})$  outputs  $\text{ABE.Decrypt}(\text{pp}, \text{sk}_f, \text{ct})$ .
- **Ideal decryption oracle:** On input  $\text{pp}, \text{msk}$ , and a string  $\text{ct} \in \{0, 1\}^t$ , the ideal decryption oracle  $\mathcal{O}_2(\text{pp}, \text{msk}, \text{ct})$  outputs  $\mathcal{S}^{f(\cdot)}(\text{pp}, \text{msk}, \text{ct})$ . Moreover, we restrict the simulator  $\mathcal{S}$  to make at most one oracle query to  $f$  per invocation.

## 2.4 Receiver-Extractable 2-Message OT

**Definition 2.14** (Receiver-Extractable 2-Message OT). A receiver-extractable 2-message 1-out-of-2 OT scheme on  $k$ -bit messages is a tuple of PPT algorithms  $\text{OT} = (\text{OT.Setup}, \text{OT}_1, \text{OT}_2, \text{OT.Receive})$  satisfying the following properties:

- $\text{OT.Setup}(1^\lambda) \rightarrow \text{crs}$ : On input the security parameter  $\lambda$ , the setup algorithm outputs a *common reference string*  $\text{crs}$ . If  $\text{OT.Setup}$  outputs a uniformly random string, we say that the OT scheme is in the *common random string* model.
- $\text{OT}_1(\text{crs}, b; r) \rightarrow M^{(1)}$ : On input the common reference string  $\text{crs}$  and a choice bit  $b \in \{0, 1\}$ ,  $\text{OT}_1$  outputs the first message of the OT (from receiver to sender). Here,  $r \in \{0, 1\}^\tau$  denotes the random coins used in the process.
- $\text{OT}_2(\text{crs}, M^{(1)}, m_0, m_1) \rightarrow M^{(2)}$ : On input the common reference string  $\text{crs}$ , the OT first message  $M^{(1)}$  and the sender's messages  $m_0, m_1 \in \{0, 1\}^k$ ,  $\text{OT}_2$  outputs the the second OT message (from sender to receiver).
- $\text{OT.Receive}(\text{crs}, M^{(2)}, b, r) \rightarrow m$ : On input the common reference string  $\text{crs}$ , the OT second message  $M^{(2)}$ , the receiver's choice bit  $b \in \{0, 1\}$  and random coins  $r \in \{0, 1\}^\tau$ , the receiver algorithm outputs a message  $m \in \{0, 1\}^k$ .

**Definition 2.15** (Correctness). An OT scheme  $\text{OT}$  is (perfectly) correct if for all security parameters  $\lambda \in \mathbb{N}$ , choice bits  $b \in \{0, 1\}$  and messages  $m_0, m_1 \in \{0, 1\}^k$ , we have:

$$\Pr \left[ \text{OT.Receive}(\text{crs}, M^{(2)}, b, r) = m_b \right] = 1,$$

where  $\text{crs} \leftarrow \text{OT.Setup}$ ,  $r \xleftarrow{\text{R}} \{0, 1\}^\tau$ ,  $M^{(1)} = \text{OT}_1(\text{crs}, b; r)$ , and  $M^{(2)} \leftarrow \text{OT}_2(\text{crs}, M^{(1)}, m_0, m_1)$ .

**Definition 2.16** (Computational Receiver Security). An OT scheme  $\text{OT}$  satisfies (computational) receiver privacy if

$$\{r \xleftarrow{\text{R}} \{0, 1\}^\tau : (\text{crs}, \text{OT}_1(\text{crs}, 0; r))\} \stackrel{c}{\approx} \{r \xleftarrow{\text{R}} \{0, 1\}^\tau : (\text{crs}, \text{OT}_1(\text{crs}, 1; r))\},$$

where  $\text{crs} \leftarrow \text{OT.Setup}(1^\lambda)$ .

**Definition 2.17** (Extractability against Malicious Receivers). An OT scheme  $\text{OT}$  is *extractable against malicious receivers* if there exist PPT algorithms  $(\text{crs}, \text{td}) \leftarrow \text{OT.SetupExt}(1^\lambda)$  and  $y \leftarrow \text{OT.Ext}(\text{td}, M^{(1)})$  such that the following two properties hold:

- **CRS indistinguishability:** The common reference strings generated by  $\text{OT.Setup}$  and  $\text{OT.SetupExt}$  are computationally indistinguishable. Namely, we require

$$\{\text{crs} \leftarrow \text{OT.Setup}(1^\lambda) : \text{crs}\} \stackrel{c}{\approx} \{(\text{crs}, \text{td}) \leftarrow \text{OT.SetupExt}(1^\lambda) : \text{crs}\}.$$

- **Sender privacy in extraction mode:** For a security parameter  $\lambda$  and an adversary  $\mathcal{A}$ , let  $\text{Expt}_{\text{OT}, \mathcal{A}}^{\text{OT-Ext}}(\lambda, b)$  be the following experiment:
  - **Setup:** The challenger samples  $(\text{crs}, \text{td}) \leftarrow \text{OT.SetupExt}(1^\lambda)$  and gives  $\text{crs}$  to the adversary.
  - **Initial message:** The adversary picks  $M^{(1)}$  and sends it to the challenger. The challenger computes  $\beta \leftarrow \text{OT.Ext}(\text{td}, M^{(1)})$  and sends it to the adversary.
  - **Challenge query:** The adversary picks two messages  $m_0, m_1 \in \{0, 1\}^k$  and sends them to the challenger. The challenger sets  $\tilde{m}_\beta = m_\beta$  and  $\tilde{m}_{1-\beta} = m_{1-\beta}$  if  $b = 0$  and  $\tilde{m}_{1-\beta} = 0^k$  if  $b = 1$ , and sends  $M^{(2)} \leftarrow \text{OT}_2(\text{crs}, M^{(1)}, \tilde{m}_0, \tilde{m}_1)$  to the adversary.
  - **Output:** The adversary outputs a bit  $b' \in \{0, 1\}$ , which is the output of the experiment.

We say that OT satisfies *sender privacy in extraction mode* if for all PPT adversaries  $\mathcal{A}$ :

$$\left| \Pr[\text{Expt}_{\text{OT}, \mathcal{A}}^{\text{OT-Ext}}(\lambda, 0) = 1] - \Pr[\text{Expt}_{\text{OT}, \mathcal{A}}^{\text{OT-Ext}}(\lambda, 1) = 1] \right| = \text{negl}(\lambda).$$

**Remark 2.18** (Batch OT). In the following we will assume that  $\text{OT}_1$  takes as input an  $\ell$ -bit choice string  $y \in \{0, 1\}^\ell$  for some  $\ell = \text{poly}(\lambda)$  (rather than just a single choice bit  $b \in \{0, 1\}$ ), and that the  $\text{OT}_2$  algorithm takes as input  $2\ell$  messages  $\{(m_0^{(i)}, m_1^{(i)})\}_{i \in [\ell]}$ . Correctness now says that the receiver receives messages  $m_{y_i}^{(i)}$  for  $i \in [\ell]$ . Receiver security says that  $\text{OT}_1(\text{crs}, y)$  computationally hides  $y$ . Extractability against malicious receivers still requires that the extraction CRS is computationally indistinguishable from the real CRS, and moreover, that the extraction algorithm  $\text{OT.Ext}$  now extracts  $y \in \{0, 1\}^\ell$  from  $M^{(1)}$ , and that the OT second message  $M^{(2)}$  hides all of the messages  $m_{(1-y_i)}^{(i)}$  for  $i \in [\ell]$ . This is without loss of generality, since we can simply run  $\ell$  independent instances of the basic 1-out-of-2 OT scheme in parallel (where the new CRS is the concatenation of  $\ell$  independent common reference strings for the base OT). Those properties then follow directly via a standard hybrid argument.

**Constructing receiver-extractable OT.** Finally, we note that the OT constructions of [PVW08, DGH<sup>+</sup>19] imply receiver-extractable 2-message (batch) OTs from the CDH/LWE/LPN assumptions. We state this in the following theorem:

**Theorem 2.19** (Receiver-Extractable 2-Message OT from CDH/LWE/LPN [PVW08, DGH<sup>+</sup>19]). *Assuming either CDH, LWE with a polynomial modulus-to-noise ratio (and polynomial modulus), or LPN (with noise rate  $n^{-(\frac{1}{2}+\varepsilon)}$  for any  $\varepsilon > 0$ ), there exists a receiver-extractable 2-message 1-out-of-2 OT for  $k$ -bit messages (for any fixed  $k = \text{poly}(\lambda)$ ) in the common random string model.*

We show in Appendix B that the [PVW08] OT scheme satisfies our notion of receiver extractability. As for [DGH<sup>+</sup>19], we use their intermediate “half-UC OT scheme;” it follows directly from their proof of security that our notion of sender privacy holds (in fact, this is true even if we give the extraction trapdoor itself to the adversary) holds. Moreover, the CRS of this intermediate scheme consists of a uniformly random string along with the public key of a public-key encryption scheme; since CDH and LPN both imply public-key encryption schemes with pseudorandom public keys, this allows the overall OT protocol to be instantiated with a uniformly random CRS.

## 2.5 Garbling Scheme

**Definition 2.20** (Garbling Scheme [Yao86, LP09, BHR12]). A garbling scheme for Boolean circuits consists of two algorithms  $\text{Yao} = (\text{Yao.Garble}, \text{Yao.Eval})$  with the following properties:

- $\text{Yao.Garble}(1^\lambda, C) \rightarrow (\tilde{C}, \overrightarrow{\text{lab}})$ : On input a security parameter  $\lambda$ , a Boolean circuit  $C$  on  $n$ -bit inputs, the garbling algorithm outputs a garbled circuit  $\tilde{C}$  and a collection of labels  $\overrightarrow{\text{lab}} = \{\text{lab}_{i,b}\}_{i \in [n], b \in \{0,1\}}$  where  $\text{lab}_{i,b} \in \{0,1\}^\lambda$  for all  $i \in [n]$  and  $b \in \{0,1\}$ .
- $\text{Yao.Eval}(\tilde{C}, \overrightarrow{\text{lab}}) \rightarrow y$ : On input a garbled circuit  $\tilde{C}$  and a collection of labels  $\overrightarrow{\text{lab}} = \{\text{lab}_{i,x_i}\}_{i \in [n]}$ , the evaluation algorithm outputs a string  $y$ .

A garbling scheme should satisfy the following two properties:

- **Correctness:** For all circuits  $C: \{0,1\}^n \rightarrow \{0,1\}^m$  and inputs  $x \in \{0,1\}^n$  we have that

$$\Pr [C(x) = \text{Yao.Eval}(\tilde{C}, \{\text{lab}_{i,x_i}\}_{i \in [n]})] = 1,$$

where  $(\tilde{C}, \{\text{lab}_{i,b}\}_{i \in [n], b \in \{0,1\}}) \leftarrow \text{Yao.Garble}(1^\lambda, C)$ .

- **Security:** There exists a PPT simulator  $\mathcal{S}$  such that for all circuits  $C: \{0,1\}^n \rightarrow \{0,1\}^m$  and inputs  $x \in \{0,1\}^n$ , we have that

$$(\tilde{C}, \{\text{lab}_{i,x_i}\}_{i \in [n]}) \stackrel{c}{\approx} \mathcal{S}(1^{|C|}, 1^n, C(x)),$$

where  $(\tilde{C}, \{\text{lab}_{i,b}\}_{i \in [n], b \in \{0,1\}}) \leftarrow \text{Yao.Garble}(1^\lambda, C)$ .

**Theorem 2.21** (Yao's Garbling Scheme [Yao86, LP09, BHR12]). *Assuming one-way functions exist, there exists a secure garbling scheme for Boolean circuits.*

## 2.6 Non-Interactive Equivocable Commitments

**Definition 2.22** (Non-Interactive Equivocable Commitment [CIO98]). A non-interactive equivocable commitment scheme  $\text{Com} = (\text{Com.Setup}, \text{Com.Commit})$  with message space  $\mathcal{M}$  is defined as follows:

- $\text{Com.Setup}(1^\lambda) \rightarrow \text{crs}$ : On input the security parameter  $\lambda$ , the setup algorithm outputs a common reference string  $\text{crs}$ . If  $\text{Com.Setup}$  outputs a uniformly random string, we say that the commitment scheme is in the *common random string* model.
- $\text{Com.Commit}(\text{crs}, m; r) \rightarrow c$ : On input a reference string  $\text{crs}$ , a message  $m \in \mathcal{M}$ , and randomness  $r \in \{0,1\}^\lambda$ , the commit algorithm outputs a commitment  $c$ .

Moreover,  $\text{Com}$  should satisfy the following properties:

- **Statistical binding:** For all values  $c$  and distinct messages  $m_0, m_1 \in \mathcal{M}$

$$\Pr[\exists r_0, r_1 \in \{0,1\}^\lambda : \text{Com.Commit}(\text{crs}, m_0; r_0) = c = \text{Com.Commit}(\text{crs}, m_1; r_1)] = \text{negl}(\lambda),$$

where  $\text{crs} \leftarrow \text{Com.Setup}(1^\lambda)$ .

- **Equivocation:** There exists PPT algorithms ( $\text{Com.SetupEquiv}, \text{Com.Equivocate}$ ) with the following properties:
  - $\text{Com.SetupEquiv}(1^\lambda) \rightarrow (\overline{\text{crs}}, \bar{c}, \text{td})$ : On input the security parameter  $\lambda$ , the setup algorithm outputs a common reference string  $\overline{\text{crs}}$ , a commitment  $\bar{c}$ , and a trapdoor  $\text{td}$ .
  - $\text{Com.Equivocate}(\text{td}, m, c) \rightarrow r$ : On input a trapdoor  $\text{td}$ , a message  $m \in \mathcal{M}$ , and a commitment  $c$ , the equivocate algorithm outputs  $r \in \{0, 1\}^\lambda$ .

Moreover, for all messages  $m \in \mathcal{M}$ , we require that

$$(\text{crs}, c, r) \stackrel{c}{\approx} (\overline{\text{crs}}, \bar{c}, \bar{r}),$$

where  $\text{crs} \leftarrow \text{Com.Setup}(1^\lambda)$ ,  $r \xleftarrow{\text{R}} \{0, 1\}^\lambda$ , and  $c \leftarrow \text{Com.Commit}(\text{crs}, m; r)$  are sampled honestly, while  $(\overline{\text{crs}}, \bar{c}, \text{td}) \leftarrow \text{Com.SetupEquiv}(1^\lambda)$  and  $\bar{r} \leftarrow \text{Com.Equivocate}(\text{td}, m, \bar{c})$  are sampled in equivocation mode. In particular, this implies that for all messages  $m \in \mathcal{M}$ ,

$$\Pr[r \leftarrow \text{Com.Equivocate}(\text{td}, m, \bar{c}) : \bar{c} \neq \text{Commit}(\overline{\text{crs}}, m; r)] = \text{negl}(\lambda),$$

where  $(\overline{\text{crs}}, \bar{c}, \text{td}) \leftarrow \text{Com.SetupEquiv}(1^\lambda)$ .

**Theorem 2.23** (Equivocable Bit-Commitments from One-Way Functions [Nao91, CIO98]). *Assuming one-way functions exist, there exists a non-interactive equivocable commitment scheme with message space  $\{0, 1\}$  in the common random string model.*

## 2.7 KDM-Secure Secret-Key Encryption

**Definition 2.24** (One-Time KDM-Secure SKE). A secret-key encryption (SKE) scheme  $\text{SKE} = (\text{SKE.Encrypt}, \text{SKE.Decrypt})$  is said to be *one-time KDM secure* for a function class  $\mathcal{F}$  (with many-bit outputs) if for every function  $f \in \mathcal{F}$ , the following two distributions are computationally indistinguishable:

$$\{s \xleftarrow{\text{R}} \{0, 1\}^\lambda : \text{SKE.Encrypt}(s, f(s))\} \stackrel{c}{\approx} \{s \xleftarrow{\text{R}} \{0, 1\}^\lambda : \text{SKE.Encrypt}(s, 0^{|f(s)|})\}$$

**Remark 2.25** (KDM-Secure SKE Constructions for Projection Functions). We say a function  $f: \{0, 1\}^\lambda \rightarrow \{0, 1\}^m$  is a *projection function* if each bit of  $f(s)$  depends on at most one bit of  $s$ . As in [KMT19], we consider the class  $\mathcal{F} = \mathcal{F}_{\text{proj}}$  of projection functions. Secret-key encryption schemes that are KDM-secure for the class of projection functions can be constructed from the CDH [BH08, BLSV18], LWE (with polynomial modulus) [ACPS09, BLSV18], and constant-noise LPN [ACPS09] assumptions.

## 3 Attribute-Based Secure Function Evaluation

In this section, we formally introduce our notion of an attribute-based secure function evaluation scheme (AB-SFE), which can be viewed as a generalization of a single-key ABE scheme. We then define two main security requirements on AB-SFE schemes: message-hiding and key-hiding. For each notion, we introduce a “weak” variant and a “strong” variant of the notion.

**Definition 3.1** (Attribute-Based Secure Function Evaluation). An attribute-based secure function evaluation (AB-SFE) scheme for a function  $F: \mathcal{X} \times \mathcal{Y} \rightarrow \{0, 1\}$  with message space  $\mathcal{M}$  consists of a tuple  $\text{ABSFE} = (\text{ABSFE.Setup}, \text{ABSFE.KeyGen}, \text{ABSFE.Encrypt}, \text{ABSFE.Decrypt})$  of PPT algorithms with the following properties:

- $\text{ABSFE.Setup}(1^\lambda) \rightarrow \text{crs}$ : On input the security parameter  $\lambda$ , the setup algorithm outputs a *common reference string*  $\text{crs}$ . We say that the AB-SFE scheme is in the *common random string model* if  $\text{Setup}$  simply outputs a uniformly random string.
- $\text{ABSFE.KeyGen}(\text{crs}, y) \rightarrow (\text{pk}, \text{sk})$ : On input the common reference string  $\text{crs}$  and a value  $y \in \mathcal{Y}$ , the key-generation algorithm outputs a public key  $\text{pk}$  and a secret key  $\text{sk}$ .
- $\text{ABSFE.Encrypt}(\text{crs}, \text{pk}, x, m) \rightarrow \text{ct}$ : On input the common reference string  $\text{crs}$ , a public key  $\text{pk}$ , a value  $x \in \mathcal{X}$ , and a message  $m \in \mathcal{M}$ , the encryption algorithm outputs a ciphertext  $\text{ct}$ .
- $\text{ABSFE.Decrypt}(\text{crs}, \text{sk}, x, \text{ct}) \rightarrow m$ : On input the common reference string  $\text{crs}$ , a secret key  $\text{sk}$ , an attribute  $x \in \mathcal{X}$ , and a ciphertext  $\text{ct}$ , the decryption algorithm outputs a message  $m \in \mathcal{M} \cup \{\perp\}$ .

**Definition 3.2** (Correctness). An AB-SFE scheme  $\text{ABSFE}$  is (perfectly) *correct* if for all messages  $m \in \mathcal{M}$ , all  $x \in \mathcal{X}, y \in \mathcal{Y}$  where  $F(x, y) = 1$ ,

$$\Pr [\text{ABSFE.Decrypt}(\text{crs}, \text{sk}, x, \text{ABSFE.Encrypt}(\text{crs}, \text{pk}, x, m)) = m] = 1,$$

where  $\text{crs} \leftarrow \text{ABSFE.Setup}(1^\lambda)$  and  $(\text{pk}, \text{sk}) \leftarrow \text{ABSFE.KeyGen}(\text{crs}, y)$ .

**Message-hiding.** The first security requirement on an AB-SFE scheme is message-hiding. The basic notion (or “weak” notion) is essentially semantic security: namely, a ciphertext with attribute  $x \in \mathcal{X}$  encrypted under a public key for  $y \in \mathcal{Y}$  where  $F(x, y) = 0$  should hide the underlying message. Next, we define a “strong” notion of message-hiding, which says semantic security holds even in the setting where the public-key is *maliciously* chosen. In this case, we require that there exists an efficient algorithm that can extract an attribute  $y$  from any (possibly malformed) public key  $\text{pk}$ , and ciphertexts encrypted to any attribute  $x$  where  $F(x, y) = 0$  still hide the underlying message.

**Definition 3.3** (Weak Message-Hiding). Let  $\text{ABSFE}$  be an AB-SFE scheme. For a bit  $b \in \{0, 1\}$ , we define the following game between an adversary  $\mathcal{A}$  and a challenger:

- **Setup:** The adversary  $\mathcal{A}$  begins by sending an input  $y \in \mathcal{Y}$  to the challenger. The challenger samples  $\text{crs} \leftarrow \text{ABSFE.Setup}(1^\lambda)$ ,  $(\text{pk}, \text{sk}) \leftarrow \text{ABSFE.KeyGen}(\text{crs}, y)$  and gives  $\text{crs}, \text{pk}, \text{sk}$  to  $\mathcal{A}$ .
- **Challenge query:** The adversary  $\mathcal{A}$  then makes a challenge query  $(x, m_0, m_1)$  to the challenger where  $x \in \mathcal{X}$ ,  $m_0, m_1 \in \mathcal{M}$ , and  $F(x, y) = 0$ . The challenger replies with  $\text{ct} \leftarrow \text{ABSFE.Encrypt}(\text{crs}, \text{pk}, x, m_b)$  and gives  $\text{ct}$  to  $\mathcal{A}$ .
- **Output:** The adversary  $\mathcal{A}$  outputs a bit  $b' \in \{0, 1\}$ .

We say that  $\text{ABSFE}$  provides *weak message-hiding* if for all PPT adversaries  $\mathcal{A}$ ,

$$|\Pr[b' = 1|b = 0] - \Pr[b' = 1|b = 1]| = \text{negl}(\lambda).$$

**Definition 3.4** (Strong Message-Hiding). An AB-SFE scheme ABSFE provides *strong message-hiding* if there exists a PPT “extractable-setup” algorithm  $(\text{crs}, \text{td}) \leftarrow \text{ABSFE.SetupExt}(1^\lambda)$  and a PPT extractor  $y \leftarrow \text{ABSFE.Ext}(\text{td}, \text{pk})$  with the following properties:

- **CRS indistinguishability:** The common reference strings output by  $\text{ABSFE.Setup}$  and  $\text{ABSFE.SetupExt}$  are computationally indistinguishable:

$$\{\text{crs} \leftarrow \text{ABSFE.Setup}(1^\lambda) : \text{crs}\} \stackrel{c}{\approx} \{(\text{crs}, \text{td}) \leftarrow \text{ABSFE.SetupExt}(1^\lambda) : \text{crs}\}.$$

- **Ciphertext indistinguishability in extraction mode:** For a bit  $b \in \{0, 1\}$ , we define the following game between an adversary  $\mathcal{A}$  and a challenger:
  - **Setup:** The challenger samples  $(\text{crs}, \text{td}) \leftarrow \text{ABSFE.SetupExt}(1^\lambda)$  and gives  $\text{crs}$  to  $\mathcal{A}$ .
  - **Public key selection:** The adversary chooses a public key  $\text{pk}$ . The challenger computes  $y \leftarrow \text{ABSFE.Ext}(\text{td}, \text{pk})$  and gives  $y \in \mathcal{Y}$  to  $\mathcal{A}$ .
  - **Challenge query:** The adversary  $\mathcal{A}$  makes a challenge query  $(x, m_0, m_1)$  where  $x \in \mathcal{X}$ ,  $m_0, m_1 \in \mathcal{M}$ , and  $F(x, y) = 0$ . The challenger computes  $\text{ct} \leftarrow \text{ABSFE.Encrypt}(\text{crs}, \text{pk}, x, m_b)$  and gives  $\text{ct}$  to  $\mathcal{A}$ .
  - **Output:** The adversary  $\mathcal{A}$  outputs a bit  $b' \in \{0, 1\}$ .

We require that for all PPT adversaries  $\mathcal{A}$ ,  $|\Pr[b' = 1|b = 0] - \Pr[b' = 1|b = 1]| = \text{negl}(\lambda)$ .

**Remark 3.5** (Multiple Challenge Queries). By a standard hybrid argument, any AB-SFE scheme that satisfies weak message-hiding (resp., strong message-hiding) against an adversary that makes a single challenge query  $(x, m_0, m_1)$  is also secure against an adversary that makes polynomially-many challenge queries. Note that in the strong message-hiding setting, the challenger encrypts each challenge message with respect to the *same* public key chosen by the adversary (and correspondingly, the *same* value of  $y$  is used to check admissibility of each of the adversary’s challenge queries). It is essential for the hybrid argument to use the same public key together with the same extracted attribute  $y$ , which is known to the adversary (otherwise, the reduction algorithm is unable to simulate the other ciphertexts in the hybrid argument, and correspondingly, single-challenge security does not necessarily imply multiple-challenge security).

**Key-hiding.** The second security requirement on an AB-SFE scheme is key-hiding. Similar to the case of message-hiding security, we consider a “weak” notion and a “strong” notion. The weak notion requires that a public key  $\text{pk}$  associated with an attribute  $y$  hides  $y$ , while the strong notion requires that  $y$  remains hidden even if the adversary has access to a decryption oracle (with the associated secret key  $\text{sk}$ ). Strong key-hiding is reminiscent of the weak function-hiding property we defined for ABE (Definition 2.13), and indeed, we show in Section 5.1 that ABE schemes satisfying weak function-hiding imply AB-SFE schemes that satisfy strong key-hiding.

**Definition 3.6** (Weak Key-Hiding). An AB-SFE scheme ABSFE satisfies *weak key-hiding* if there exists a PPT simulator  $\mathcal{S}$  such that for all  $y \in \mathcal{Y}$  and all PPT adversaries  $\mathcal{A}$ ,

$$\left| \Pr[\mathcal{A}(1^\lambda, \text{crs}, \text{pk}) = 1] - \Pr[\mathcal{A}(1^\lambda, \overline{\text{crs}}, \overline{\text{pk}}) = 1] \right| = \text{negl}(\lambda),$$

where  $\text{crs} \leftarrow \text{ABSFE.Setup}(1^\lambda)$ ,  $(\text{pk}, \text{sk}) \leftarrow \text{ABSFE.KeyGen}(\text{crs}, y)$ , and  $(\overline{\text{crs}}, \overline{\text{pk}}) \leftarrow \mathcal{S}(1^\lambda)$ .

**Definition 3.7** (Strong Key-Hiding). An AB-SFE scheme ABSFE satisfies *strong key-hiding* if there exists a PPT simulator  $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$  such that for all  $y \in \mathcal{Y}$  and all PPT adversaries  $\mathcal{A}$  we have:

$$\left| \Pr[\mathcal{A}^{\mathcal{O}_1(\text{crs}, \text{sk}, \cdot)}(1^\lambda, \text{crs}, \text{pk}) = 1] - \Pr[\mathcal{A}^{\mathcal{O}_2(\text{st}_{\mathcal{S}}, \cdot)}(1^\lambda, \overline{\text{crs}}, \overline{\text{pk}}) = 1] \right| = \text{negl}(\lambda),$$

where  $\text{crs} \leftarrow \text{ABSFE.Setup}(1^\lambda)$ ,  $(\text{pk}, \text{sk}) \leftarrow \text{ABSFE.KeyGen}(\text{crs}, y)$ ,  $(\text{st}_{\mathcal{S}}, \overline{\text{crs}}, \overline{\text{pk}}) \leftarrow \mathcal{S}_1(1^\lambda)$  and the oracles  $\mathcal{O}_1, \mathcal{O}_2$  are defined as follows:

- **Real decryption oracle  $\mathcal{O}_1$ :** On input a string  $\text{crs}$ , a secret key  $\text{sk}$ , a value  $x \in \mathcal{X}$ , and a ciphertext  $\text{ct}$ , output  $\text{ABSFE.Decrypt}(\text{crs}, \text{sk}, x, \text{ct})$ .
- **Ideal decryption oracle  $\mathcal{O}_2$ :** On input a state  $\text{st}_{\mathcal{S}}$ ,  $x \in \mathcal{X}$  and a ciphertext  $\text{ct}$ , output  $\mathcal{S}_2(\text{st}_{\mathcal{S}}, x, \text{ct}, F(x, y))$ .

## 4 Designated-Verifier NIZKs from AB-SFE

In this section, we show how to construct a DV-NIZK from any AB-SFE scheme that provides weak message-hiding and strong key-hiding. In Appendix D, we show that a converse of this statement also holds: given any public-key encryption scheme and a DV-NIZK, we can obtain an AB-SFE scheme that provides weak message-hiding and strong key-hiding. This means that assuming public-key encryption exists, our notion of AB-SFE is *equivalent* to DV-NIZK. Next, we strengthen our construction and show that if the underlying AB-SFE scheme satisfies strong message-hiding (and strong key-hiding), then we obtain a DV-NIZK with security against malicious verifiers. We give our main construction below:

**Construction 4.1** (Designated-Verifier NIZKs from AB-SFE). Let  $\lambda$  be a security parameter. Let  $\mathcal{L} \subseteq \{0, 1\}^n$  be an NP language associated with an NP relation  $\mathcal{R} \subseteq \{0, 1\}^n \times \{0, 1\}^h$ , where  $n = n(\lambda)$ ,  $h = h(\lambda)$ . Our construction relies on the following building blocks:

- Let  $\text{zkPCP} = (\text{zkPCP.Prove}, \text{zkPCP.Query}, \text{zkPCP.Verify})$  be an efficient  $\ell$ -query, non-adaptive, zero-knowledge PCP (with alphabet  $\Sigma$ ) for  $\mathcal{L}$  (Definition 2.7). Let  $m = m(\lambda)$  be the length of the PCP and  $\rho = \rho(\lambda)$  be a bound on the number of random bits needed for  $\text{zkPCP.Query}$ .
- Let  $\text{PRF}: \mathcal{K} \times \{0, 1\}^n \rightarrow \{0, 1\}^\rho$  be a pseudorandom function (Definition 2.1).
- Let  $F: (\{0, 1\}^n \times [m]) \times \mathcal{K} \rightarrow \{0, 1\}$  be the function

$$F((x, i), k) := \begin{cases} 1 & \exists j \in [\ell] \text{ where } i = q_j \\ 0 & \text{otherwise,} \end{cases} \quad (4.1)$$

where  $(\text{st}_x, q_1, \dots, q_\ell) \leftarrow \text{zkPCP.Query}(x; \text{PRF}(k, x))$ .

- Let  $\text{ABSFE} = (\text{ABSFE.Setup}, \text{ABSFE.KeyGen}, \text{ABSFE.Encrypt}, \text{ABSFE.Decrypt})$  be an AB-SFE scheme (Definition 3.1) for  $F$  with message space  $\mathcal{M} = \Sigma$  and attribute spaces  $\mathcal{X} = \{0, 1\}^n \times [m]$  and  $\mathcal{Y} = \mathcal{K}$ .

We construct a designated-verifier NIZK  $\text{dvNIZK} = (\text{dvNIZK.Setup}, \text{dvNIZK.KeyGen}, \text{dvNIZK.Prove}, \text{dvNIZK.Verify})$  for  $\mathcal{L}$  as follows:

- $\text{dvNIZK.Setup}(1^\lambda)$ : Output  $\text{crs} \leftarrow \text{ABSFE.Setup}(1^\lambda)$ .
- $\text{dvNIZK.KeyGen}(\text{crs})$ : Sample  $k \xleftarrow{\mathcal{R}} \mathcal{K}$ , and  $(\text{pk}', \text{sk}') \leftarrow \text{ABSFE.KeyGen}(\text{crs}, k)$ . Output the public key  $\text{pk} = \text{pk}'$ , and the secret verification key  $\text{sk} = (k, \text{sk}')$ .
- $\text{dvNIZK.Prove}(\text{crs}, \text{pk}, x, w)$ : Construct a PCP  $\pi^{(\text{PCP})} \leftarrow \text{zkPCP.Prove}(x, w)$ . Then, for each  $i \in [m]$ , compute ciphertexts  $\text{ct}_i \leftarrow \text{ABSFE.Encrypt}(\text{crs}, \text{pk}, (x, i), \pi_i^{(\text{PCP})})$ , and finally, output the proof  $\pi = (\text{ct}_1, \dots, \text{ct}_m)$ .
- $\text{dvNIZK.Verify}(\text{crs}, \text{sk}, x, \pi)$ : On input the verification key  $\text{sk} = (k, \text{sk}')$ , a statement  $x \in \{0, 1\}^n$  and a proof  $\pi = (\text{ct}_1, \dots, \text{ct}_m)$ , compute  $(\text{st}_x, q_1, \dots, q_\ell) \leftarrow \text{zkPCP.Query}(x; \text{PRF}(k, x))$ . For each  $j \in [\ell]$ , compute  $s_j \leftarrow \text{ABSFE.Decrypt}(\text{crs}, \text{sk}, (x, q_j), \text{ct}_{q_j})$ , and finally, output  $\text{zkPCP.Verify}(\text{st}_x, s_1, \dots, s_\ell)$ .

**Security analysis.** We now state and prove the completeness, soundness, and zero-knowledge theorems for Construction 4.1.

**Theorem 4.2** (Completeness). *If  $\text{zkPCP}$  is complete and  $\text{ABSFE}$  is correct, then  $\text{dvNIZK}$  from Construction 4.1 is complete.*

*Proof.* Follows immediately by completeness of  $\text{zkPCP}$  and correctness of  $\text{ABSFE}$ .  $\square$

**Theorem 4.3** (Soundness). *If  $\text{PRF}$  is a secure PRF,  $\text{ABSFE}$  satisfies strong key-hiding, and  $\text{zkPCP}$  is sound, then  $\text{dvNIZK}$  from Construction 4.1 satisfies non-adaptive computational soundness.*

*Proof.* Take any statement  $x^* \notin \mathcal{L}$ . We proceed with a hybrid argument:

- $\text{Hyb}_0$ : This is the real soundness experiment. At the beginning of the game, the challenger samples  $k \xleftarrow{\mathcal{R}} \mathcal{K}$ ,  $\text{crs} \leftarrow \text{ABSFE.Setup}(1^\lambda)$ , and  $(\text{pk}', \text{sk}') \leftarrow \text{ABSFE.KeyGen}(\text{crs}, k)$ . It gives  $\text{crs}$  and  $\text{pk} = \text{pk}'$  to the adversary and sets  $\text{sk} = (k, \text{sk}')$ . When the adversary makes a verification query  $(x, \pi)$ , the challenger replies with  $\text{dvNIZK.Verify}(\text{crs}, \text{sk}, x, \pi)$ . At the end of the experiment, the adversary outputs a proof  $\pi^*$ . The output of the experiment is 1 if  $\text{dvNIZK.Verify}(\text{crs}, \text{sk}, x^*, \pi^*) = 1$  and 0 otherwise.
- $\text{Hyb}_1$ : Same as  $\text{Hyb}_0$ , except the challenger uses the simulator  $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$  for the strong key-hiding game for  $\text{ABSFE}$  to construct the CRS, the public key, and to implement the decryption algorithm when responding to the verification queries. Namely, the challenger begins by computing  $(\text{st}_{\mathcal{S}}, \overline{\text{crs}}, \overline{\text{pk}}) \leftarrow \mathcal{S}_1(1^\lambda)$ , and gives  $\overline{\text{crs}}, \overline{\text{pk}}$  to the adversary. When simulating verification queries  $\text{dvNIZK.Verify}(\text{crs}, \text{sk}, x, \pi)$ , the challenger proceeds as follows:
  - Compute  $(\text{st}_x, q_1, \dots, q_\ell) \leftarrow \text{zkPCP.Query}(x; \text{PRF}(k, x))$ .
  - Write  $\pi = (\text{ct}_1, \dots, \text{ct}_m)$ . For each  $j \in [\ell]$ , it computes  $s_j \leftarrow \mathcal{S}_2(\text{st}_{\mathcal{S}}, (x, q_j), \text{ct}_{q_j}, 1)$ .
  - The challenge replies with  $\text{zkPCP.Verify}(\text{st}_x, s_1, \dots, s_\ell)$ .

Notably, everything in this experiment can be simulated just given *oracle* access to  $\text{PRF}(k, \cdot)$ .

This is computationally indistinguishable from  $\text{Hyb}_0$  by strong key-hiding security of  $\text{ABSFE}$ . In particular, we use the fact that by construction,  $F((x, q_j), k) = 1$  for each verification query.



- **Hyb<sub>2</sub>**: Same as Hyb<sub>1</sub>, except the challenger samples a random function  $f \xleftarrow{R} \text{Funs}[\{0, 1\}^n, \{0, 1\}^\rho]$  at the beginning of the experiment. Then, whenever it needs to compute  $\text{PRF}(k, x)$ , it instead computes  $f(x)$ .

This is computationally indistinguishable from Hyb<sub>1</sub> by security of PRF.

- **Hyb<sub>3</sub>**: Same as Hyb<sub>2</sub>, except whenever the challenger needs to compute  $\text{dvNIZK.Verify}(\text{crs}, \text{sk}, x^*, \pi)$  for any proof string  $\pi$ , it instead sets the output to 0.

This is statistically indistinguishable from Hyb<sub>2</sub> by soundness of zkPCP. To see this, let  $Q = \text{poly}(\lambda)$  be the number of verification queries the adversary makes. By construction, the challenger in Hyb<sub>2</sub> and Hyb<sub>3</sub> will compute  $\text{dvNIZK.Verify}(\text{crs}, \text{sk}, \cdot, \cdot)$  at most  $Q + 1$  times (when responding to the  $Q$  verification queries and for computing the output of the experiment). For  $t \in \{0, \dots, Q + 1\}$ , let Hyb<sub>2,t</sub> be the experiment where the first  $t$  calls to  $\text{dvNIZK.Verify}(\text{crs}, \text{sk}, \cdot, \cdot)$  are answered according to the specification in Hyb<sub>3</sub> while the remaining queries are answered according to the specification in Hyb<sub>2</sub>. By construction,  $\text{Hyb}_{2,0} \equiv \text{Hyb}_2$  and  $\text{Hyb}_{2,Q+1} \equiv \text{Hyb}_3$ . We now show that for  $t \in [Q + 1]$ , the outputs of Hyb<sub>2,t-1</sub> and Hyb<sub>2,t</sub> are statistically indistinguishable. By construction, the challenger’s behavior in Hyb<sub>2,t-1</sub> and Hyb<sub>2,t</sub> is identical except in how it implements the  $t^{\text{th}}$  call to  $\text{dvNIZK.Verify}(\text{crs}, \text{sk}, \cdot, \cdot)$ .

Let  $(x_t, \pi_t)$  be the inputs to the  $t^{\text{th}}$  invocation of  $\text{dvNIZK.Verify}(\text{crs}, \text{sk}, \cdot, \cdot)$ . If  $x_t \neq x^*$ , then the verifier’s response is identically distributed in the two experiments. If  $x_t = x^*$ , then in Hyb<sub>2,t-1</sub>, the challenger replies with  $\text{dvNIZK.Verify}(\text{crs}, \text{sk}, x^*, \pi)$  while in Hyb<sub>2,t</sub>, the challenger replies with 0. We argue that  $\text{dvNIZK.Verify}(\text{crs}, \text{sk}, x, \pi)$  outputs 0 in Hyb<sub>2,t-1</sub> with overwhelming probability. Suppose otherwise. Then, the challenger in Hyb<sub>2,t-1</sub> must have first invoked  $(\text{st}_{x^*}, q_1, \dots, q_\ell) \xleftarrow{R} \text{zkPCP.Query}(x^*; r)$  where  $r = f(x^*)$  is uniformly random and independent of all previous queries (since the challenger’s response in all of the previous verification queries do not depend on  $f(x^*)$ <sup>7</sup>). Moreover, the challenger must have computed  $s_1, \dots, s_\ell \in \{0, 1\}$  such that  $\text{zkPCP.Verify}(\text{st}_{x^*}, s_1, \dots, s_\ell) = 1$ . But since  $x^* \notin \mathcal{L}$ , by soundness of zkPCP,  $\Pr[\text{zkPCP.Verify}(\text{st}_{x^*}, s_1, \dots, s_\ell) = 1] = \text{negl}(\lambda)$ , where the probability is taken over the randomness in  $\text{zkPCP.Query}$ . Thus, with overwhelming probability,  $\text{dvNIZK.Verify}(\text{sk}, x^*, \pi) = 0$  in Hyb<sub>2,t-1</sub>.

By construction, for any proof  $\pi$  for challenge statement  $x^*$ , the challenger in Hyb<sub>3</sub> rejects with probability 1, and soundness follows.  $\square$

**Remark 4.4** (Adaptive Soundness without Complexity Leveraging). Theorem 4.3 shows that Construction 4.1 gives a non-adaptively sound DV-NIZK. As noted in Remark 2.6, we can always use complexity leveraging to obtain adaptive soundness. Here, we note that we can avoid complexity leveraging and sub-exponential hardness assumptions if we instead apply our general compiler to zero-knowledge PCPs based on “trapdoor  $\Sigma$ -protocols” [CLW18]. Informally, these are 3-message proof systems satisfying two properties.

- **Special soundness:** For every false statement, and every first message  $a$  in the  $\Sigma$ -protocol, there is exactly a single challenge that would cause the verifier to accept.

<sup>7</sup>By the specification of Hyb<sub>2,t-1</sub>, if the adversary made a verification query on  $(x^*, \pi)$  for any  $\pi$  as one of its first  $t - 1$  queries, the challenger simply replies with 0 (without computing  $f(x^*)$ ).

- **Trapdoor computation of bad challenges:** For every false statement  $x$  and first message  $a$ , the single “bad challenge” above can be efficiently computed given special trapdoor information.

For any  $\Sigma$ -protocol with special soundness, adaptive soundness of the AB-SFE-compiled protocol is broken only if the verifier’s randomness (derived from the PRF) happens to coincide with the “bad” challenge associated with the prover’s statement. For trapdoor  $\Sigma$ -protocols, this condition is efficiently checkable (given the trapdoor), so the security of the PRF then says that this happens with negligible probability, and adaptive soundness follows (via a similar sequence of hybrid arguments as in the proof of Theorem 4.3). As shown in [CLW18], we can construct a trapdoor  $\Sigma$ -protocol from Blum’s protocol for graph Hamiltonicity [Blu86]. We describe and analyze this formally in Appendix A.

**Theorem 4.5 (Zero-Knowledge).** *If ABSFE satisfies weak message-hiding (resp., strong message-hiding) and zkPCP satisfies semi-malicious zero-knowledge, then the designated-verifier NIZK  $\text{dvNIZK}$  from Construction 4.1 satisfies computational zero-knowledge (resp., computational zero-knowledge against malicious verifiers).*

*Proof.* We show that if ABSFE satisfies strong message-hiding and zkPCP satisfies semi-malicious zero-knowledge, then  $\text{dvNIZK}$  satisfies zero-knowledge against malicious verifiers. The simpler case of weak message-hiding implying vanilla zero-knowledge follows by an analogous hybrid argument (in this case, the simulator can just sample the verifier’s public and secret keys as in the real scheme instead of extracting it from the adversary’s public key). Thus, consider the case where ABSFE is strong message-hiding. Then, there exists additional algorithms  $\text{ABSFE.SetupExt}$  and  $\text{ABSFE.Ext}$  with the properties defined in Definition 3.4. We construct the zero-knowledge simulator  $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$  below:

- $\mathcal{S}_1(1^\lambda)$ : On input the security parameter  $\lambda$ , the simulator runs  $(\overline{\text{crs}}, \text{td}) \leftarrow \text{ABSFE.SetupExt}(1^\lambda)$ . It also samples randomness  $r \xleftarrow{\mathcal{R}} \{0, 1\}^\tau$  that it will use for the  $\text{ABSFE.Ext}$  algorithm (where  $\tau$  is a bound on the number of bits of randomness  $\text{ABSFE.Ext}$  takes). Finally, it outputs  $\overline{\text{crs}}$  and a state  $\text{st}_{\mathcal{S}} = (\text{td}, r)$ . Note that the randomness  $r$  is used to ensure that the simulator extracts the *same* key  $k$  from the adversary’s public key when simulating the proofs.
- $\mathcal{S}_2(\text{st}_{\mathcal{S}}, \text{pk}, x)$ : On input the simulation state  $\text{st}_{\mathcal{S}} = (\text{td}, r)$ , a public key  $\text{pk}$ , and a statement  $x$ , the simulator begins by extracting a key  $k \leftarrow \text{ABSFE.Ext}(\text{td}, \text{pk}; r)$ . Then, it computes  $(\text{st}_x, q_1, \dots, q_\ell) \leftarrow \text{zkPCP.Prove}(x; \text{PRF}(k, x))$  and  $(\tilde{\pi}_1^{(\text{PCP})}, \dots, \tilde{\pi}_\ell^{(\text{PCP})}) \leftarrow \mathcal{S}^{(\text{PCP})}(x, q_1, \dots, q_\ell)$ . For  $i \in [m]$ , the simulator takes  $z_i \leftarrow \tilde{\pi}_j^{(\text{PCP})}$  if  $i = q_j$  for some  $j \in [\ell]$ , and otherwise, it sets  $z_i \leftarrow 0$ . Finally, for each  $i \in [m]$ , it computes  $\text{ct}_i \leftarrow \text{ABSFE.Encrypt}(\text{crs}, \text{pk}, (x, i), z_i)$  and outputs the simulated proof  $\bar{\pi} = (\text{ct}_1, \dots, \text{ct}_m)$ .

To complete the proof, we define a sequence of hybrid experiments:

- $\text{Hyb}_0$ : This is  $\text{ExptReal}_{\mathcal{A}}(\lambda)$ . Namely, at the beginning of the game, the challenger samples  $\text{crs} \leftarrow \text{ABSFE.Setup}(1^\lambda)$  and gives  $\text{crs}$  to the adversary. The adversary  $\mathcal{A}$  then outputs a public key  $\text{pk}$ . When  $\mathcal{A}$  makes a verification query on  $(x, w)$  where  $\mathcal{R}(x, w) = 1$ , the challenger replies with  $\text{dvNIZK.Prove}(\text{crs}, \text{pk}, x, w)$ . At the end of the experiment, algorithm  $\mathcal{A}$  outputs a bit  $b \in \{0, 1\}$ , which is also the output of the experiment.
- $\text{Hyb}_1$ : Same as  $\text{Hyb}_0$ , except we replace the real common reference string  $\text{crs} \leftarrow \text{ABSFE.Setup}(1^\lambda)$  with a simulated one  $\overline{\text{crs}}$  where  $(\overline{\text{crs}}, \text{td}) \leftarrow \text{ABSFE.SetupExt}(1^\lambda)$ .

This is computationally indistinguishable from  $\text{Hyb}_0$  by the CRS indistinguishability of ABSFE (from Definition 3.4).

- $\text{Hyb}_2$ : Same as  $\text{Hyb}_1$ , except the challenger implements the adversary’s verification queries using the following modified procedure. First, after  $\mathcal{A}$  outputs a public key  $\text{pk}$ , the challenger computes  $k \leftarrow \text{ABSFE.Ext}(\text{td}, \text{pk})$ . The same key  $k$  is used when responding to the subsequent verification queries. On a verification query  $(x, w)$  where  $\mathcal{R}(x, w) = 1$ , the challenger constructs a PCP  $\pi^{(\text{PCP})} \leftarrow \text{zkPCP.Prove}(x, w)$ . It additionally computes  $(\text{st}_x, q_1, \dots, q_\ell) \leftarrow \text{zkPCP.Query}(x; \text{PRF}(k, x))$  and for each  $i \in [m]$ , it sets  $z_i \leftarrow \pi_i^{(\text{PCP})}$  if  $i = q_j$  for some  $j \in [\ell]$  and  $z_i \leftarrow 0$  otherwise. It then computes  $\text{ct}_i \leftarrow \text{ABSFE.Encrypt}(\text{crs}, \text{pk}, (x, i), z_i)$  for each  $i \in [m]$  and outputs the proof  $\pi = (\text{ct}_1, \dots, \text{ct}_m)$ .

This is computationally indistinguishable from  $\text{Hyb}_1$  by ciphertext indistinguishability of ABSFE (from Definition 3.4). Namely, the only difference between  $\text{Hyb}_1$  and  $\text{Hyb}_2$  is that in  $\text{Hyb}_2$ , instead of computing the ciphertexts as  $\text{ct}_i \leftarrow \text{ABSFE.Encrypt}(\text{crs}, \text{pk}, (x, i), \pi_i^{(\text{PCP})})$  when  $F((x, i), k) = 0$ , the challenger instead computes  $\text{ct}_i \leftarrow \text{ABSFE.Encrypt}(\text{crs}, \text{pk}, (x, i), 0)$ . In the formal reduction, after the zero-knowledge adversary outputs its (possibly maliciously-generated) public key, the ciphertext-indistinguishability adversary forwards  $\text{pk}$  to the challenger to obtain the PRF key  $k$ . Afterwards, the ciphertext indistinguishability adversary uses his knowledge of  $k$  to issue challenge queries on all  $i \in [m]$  such that  $F((x, i), k) = 0$ . In particular, we assume that the adversary is able to make multiple challenge queries, all under the same  $\text{pk}$  (see Remark 3.5).

- $\text{Hyb}_3$ : Same as  $\text{Hyb}_2$ , except the challenger simulates the bits of the PCP using  $\mathcal{S}^{(\text{PCP})}$  instead of computing them via  $\text{zkPCP.Prove}$ . Specifically, on a verification query  $(x, w)$  where  $\mathcal{R}(x, w) = 1$ , after computing  $(\text{st}_x, q_1, \dots, q_\ell) \leftarrow \text{zkPCP.Prove}(x; \text{PRF}(k, x))$ , the challenger constructs a simulated PCP  $(\tilde{\pi}_1^{(\text{PCP})}, \dots, \tilde{\pi}_\ell^{(\text{PCP})}) \leftarrow \mathcal{S}^{(\text{PCP})}(x, q_1, \dots, q_\ell)$ . For each  $i \in [m]$ , it sets  $z_i = \tilde{\pi}_i^{(\text{PCP})}$  if  $i = q_j$  for some  $j \in [\ell]$  and  $z_i = 0$  otherwise. The output is then computed as in  $\text{Hyb}_2$ . Notably, this is the  $\text{ExptSim}_{\mathcal{A}, \mathcal{S}}(\lambda)$  experiment.

This is computationally indistinguishable from  $\text{Hyb}_2$  by zero-knowledge of  $\text{zkPCP}$ .

We conclude that the adversary’s output in the real distribution is computationally indistinguishable from its output in the simulated distribution, and computational zero-knowledge against a malicious verifier follows.  $\square$

**Remark 4.6** (DV-NIZKs in the Common Random String Model). If the public parameters of ABSFE (i.e., the output of  $\text{ABSFE.Setup}$ ) in Construction 4.1 are uniformly random strings, then the resulting DV-NIZK is also in the common random string model. More generally, because we are working with computational notions of soundness *and* zero-knowledge, this is true even if the public parameters are only *pseudorandom*. In this case, computational soundness and zero-knowledge would still follow by a standard hybrid argument, but completeness may be downgraded from perfect to statistical.

## 5 Constructing AB-SFE Schemes

In this section, we describe several approaches to construct AB-SFE schemes satisfying different flavors of message-hiding and key-hiding. First, in Section 5.1, we show how to build weak message-hiding AB-SFE from any single-key ABE scheme. In Section 5.2, we show how to construct AB-SFE schemes with strong message-hiding (and weak key-hiding) from receiver-extractable OT. Then, in Section 5.3, we show how to generically boost an AB-SFE scheme satisfying weak key-hiding into one that satisfies strong key-hiding (Definition 3.7) via a KDM-secure secret-key encryption scheme (while preserving weak/strong message-hiding). Combining the constructions in Section 5.2 and 5.3, we obtain AB-SFE schemes that provide both strong message-hiding and strong key-hiding (which suffice to realize our strongest notion of MDV-NIZK via Construction 4.1). Finally, in Section 5.4, we describe how to instantiate the different building blocks from the CDH, DDH, or LWE assumptions.

### 5.1 Weak Message-Hiding AB-SFE from Single-Key ABE

As noted in Section 1.3, an AB-SFE scheme can be viewed as a generalization of a single-key ABE scheme. In this section, we describe two simple constructions of AB-SFE schemes from single-key ABE schemes. Both of these schemes provide weak message-hiding.

**Construction 5.1** (AB-SFE from Single-Key ABE). Take a function  $F: \mathcal{X} \times \mathcal{Y} \rightarrow \{0, 1\}$  and a message space  $\mathcal{M}$ . Let ABE be a single-key ABE scheme that supports evaluation of functions of the form  $F_y(x) = F(x, y)$  (for hard-coded  $y$ ) and with message space  $\mathcal{M}$  (Definition 2.10). We construct an AB-SFE scheme ABSFE as follows:

- ABSFE.Setup( $1^\lambda$ ): Output  $\text{crs} = 1^\lambda$ .
- ABSFE.KeyGen( $\text{crs}, y$ ): On input  $\text{crs} = 1^\lambda$ , sample  $(\text{pp}, \text{msk}) \leftarrow \text{ABE.Setup}(1^\lambda)$ , compute  $\text{sk}_y \leftarrow \text{ABE.KeyGen}(\text{msk}, F_y)$  for the function  $F_y(x) := F(x, y)$ , and output  $(\text{pp}, \text{sk}_y)$ .
- ABSFE.Encrypt( $\text{pk}, x, m$ ): Output  $\text{ct} \leftarrow \text{ABE.Encrypt}(\text{pk}, x, m)$ .
- ABSFE.Decrypt( $\text{sk}, x, \text{ct}$ ): Output  $m' \leftarrow \text{ABE.Decrypt}(\text{sk}, x, \text{ct})$ .

**Lemma 5.2** (AB-SFE from Single-Key ABE). *If ABE is secure, then the AB-SFE scheme from Construction 5.1 satisfies weak message-hiding and weak key-hiding.*

*Proof.* Correctness follows immediately from correctness of ABE. Moreover, weak key-hiding is immediate because an honestly generated  $(\text{crs}, \text{pk} = \text{pp})$  is generated independently of  $y$ . Finally, weak message-hiding follows immediately via semantic security of ABE.  $\square$

**Corollary 5.3** (AB-SFE from Public-Key Encryption). *Assuming public-key encryption exists, then for any function ensemble  $F = \{F_\lambda\}$  computable by a family of polynomial-size circuits, there is an AB-SFE scheme for  $F$  that satisfies weak message-hiding and weak key-hiding.*

*Proof.* We can instantiate a single-key ABE scheme (for general predicates) using any public-key encryption scheme [SS10, GVW12]. The claim then follows by Lemma 5.2.  $\square$

**Remark 5.4** (Strong Key-Hiding AB-SFE from Weakly Function-Private ABE). Similarly, any single-key ABE scheme satisfying weak function hiding (Definition 2.13) directly implies the existence of an AB-SFE scheme satisfying weak message-hiding and *strong* key-hiding (Definition 3.7).

## 5.2 Strong Message-Hiding AB-SFE from Receiver-Extractable OT

Towards our goal of obtaining a malicious-designated-verifier NIZK, we show in this section how to construct an AB-SFE scheme that provides *strong message-hiding* from any receiver-extractable 2-message OT scheme (Definition 2.14). The resulting scheme satisfies weak key-hiding, and we show how to amplify key-hiding security in Section 5.3.

**Construction 5.5** (Strong Message-Hiding AB-SFE from OT). Take a function  $F: \mathcal{X} \times \mathcal{Y} \rightarrow \{0, 1\}$  and a message space  $\mathcal{M}$ . Our construction relies on the following ingredients:

- For an attribute  $x \in \mathcal{X}$  and a message  $m \in \mathcal{M}$ , let  $C_{x,m}: \mathcal{Y} \rightarrow \mathcal{M} \cup \{\perp\}$  be a circuit that on input  $y'$  outputs  $m$  if  $F(x, y') = 1$  and  $\perp$  otherwise. Let  $\ell = \text{poly}(\lambda)$  be a bound on the bit-length of elements in  $\mathcal{Y}$ .
- Let Yao = (Yao.Garble, Yao.Eval) be a garbling scheme (Definition 2.20) that supports the circuit class  $\mathcal{C} = \{x \in \mathcal{X}, m \in \mathcal{M} : C_{x,m}\}$ .
- Let OT = (OT.Setup, OT<sub>1</sub>, OT<sub>2</sub>, OT.Receive) be a receiver-extractable 2-message batch OT scheme on  $k$ -bit messages with batch size  $\ell$  (Definition 2.14, Remark 2.18), where  $k = \text{poly}(\lambda)$  is a bound on the length of the labels output by Yao. Let  $\{0, 1\}^\tau$  be the randomness space for the first OT message.

We construct an AB-SFE scheme as follows:

- ABSFE.Setup( $1^\lambda$ ): Output  $\text{crs} \leftarrow \text{OT.Setup}(1^\lambda)$ .
- ABSFE.KeyGen( $\text{crs}, y$ ): Sample  $\text{sk} = r \xleftarrow{\mathcal{R}} \{0, 1\}^\tau$ , and set  $\text{pk} \leftarrow \text{OT}_1(\text{crs}, y; r)$ . Output  $(\text{pk}, \text{sk})$ .
- ABSFE.Encrypt( $\text{crs}, \text{pk}, x, m$ ): Compute  $(\tilde{C}_{x,m}, \overrightarrow{\text{lab}}) \leftarrow \text{Yao.Garble}(1^\lambda, C_{x,m})$ , where  $\overrightarrow{\text{lab}} = \{\text{lab}_{i,b}\}_{i \in [\ell], b \in \{0,1\}}$  and  $\text{lab}_{i,b} \in \{0, 1\}^t$  for all  $i \in [\ell]$  and  $b \in \{0, 1\}$ . Output the ciphertext  $\text{ct} = (\tilde{C}_{x,m}, \text{OT}_2(\text{crs}, \text{pk}, \overrightarrow{\text{lab}}))$ .
- ABSFE.Decrypt( $\text{crs}, \text{sk}, x, \text{ct}$ ): On input the common reference string  $\text{crs}$ , a secret key  $\text{sk} = r$ , an attribute  $x \in \mathcal{X}$ , and a ciphertext  $\text{ct} = (\tilde{C}, \overrightarrow{\text{ct}'})$ , the decryption algorithm computes  $\overrightarrow{\text{lab}} \leftarrow \text{OT.Receive}(\text{crs}, r, \overrightarrow{\text{ct}'})$  and outputs Yao.Eval( $\tilde{C}, \overrightarrow{\text{lab}}$ ).

**Theorem 5.6** (Strong Message-Hiding AB-SFE from OT). *If Yao is a secure garbling scheme and OT is a receiver-extractable 2-message batch OT scheme on  $k$ -bit messages, then the AB-SFE scheme ABSFE from Construction 5.5 satisfies strong message-hiding and weak key-hiding.*

*Proof.* We show each property individually:

**Correctness.** Correctness of ABSFE follows immediately from the correctness properties of OT and Yao, respectively.

**Weak key-hiding.** Follows immediately from receiver privacy of OT.

**Strong message-hiding.** We first define the extraction algorithms (ABSFE.SetupExt, ABSFE.Ext):

- ABSFE.SetupExt( $1^\lambda$ ): Output  $(\text{crs}, \text{td}) \leftarrow \text{OT.SetupExt}(1^\lambda)$
- ABSFE.Ext( $\text{td}, \text{pk}$ ): Output OT.Ext( $\text{td}, \text{pk}$ )

We now show that strong message-hiding follows from the extractable sender privacy of OT and security of Yao. First, CRS indistinguishability for ABSFE follows from CRS indistinguishability of OT. To argue ciphertext indistinguishability, we use the following hybrid argument:

- **Hyb<sub>0</sub>**: This is the ciphertext indistinguishability game for ABSFE. Namely, the challenger begins by sampling  $(\text{crs}, \text{td}) \leftarrow \text{OT.SetupExt}(1^\lambda)$  and gives  $\text{crs}$  to  $\mathcal{A}$ . The adversary chooses a public key  $\text{pk}$  and the challenger replies with  $y \leftarrow \text{OT.Ext}(\text{td}, \text{pk})$ . Finally, when the adversary makes a challenge query  $(x, m_0, m_1)$ , the challenger garbles  $(\tilde{C}, \{\text{lab}_{i,b}\}_{i \in [n], b \in \{0,1\}}) \leftarrow \text{Yao.Garble}(1^\lambda, C_{x, m_b})$  and replies with the ciphertext  $\text{ct}'_b = (\tilde{C}, \text{OT}_2(\text{crs}, \text{pk}, \{\text{lab}_{i,b}\}_{i \in [n], b \in \{0,1\}}))$ .
- **Hyb<sub>1</sub>**: Same as **Hyb<sub>0</sub>** except the challenger instead constructs the challenge ciphertext as  $\text{ct}'_b \leftarrow (\tilde{C}, \text{OT}_2(\text{crs}, \text{pk}, \{(\text{lab}_{i,y_i}, \text{lab}_{i,y_i})\}_{i \in [n]}))$ .

Hybrids **Hyb<sub>0</sub>** and **Hyb<sub>1</sub>** are computationally indistinguishable by sender privacy of OT in extraction mode.

- **Hyb<sub>2</sub>**: Same as **Hyb<sub>1</sub>**, except instead of sampling  $(\tilde{C}, \{\text{lab}_{i,b}\}_{i \in [n], b \in \{0,1\}}) \leftarrow \text{Yao.Garble}(1^\lambda, C_{x, m_b})$ , the challenger uses the simulator  $\mathcal{S}$  for Yao to sample the garbled circuit. Namely, the challenger samples  $(\tilde{C}, \{\text{lab}_i\}_{i \in [n]}) \leftarrow \text{Sim}(1^t, 1^n, 0)$ , where  $t$  is a bound on the size of the circuit in  $\mathcal{C}$ , and sets  $\text{ct}'_b \leftarrow (\tilde{C}, \text{OT}_2(\text{crs}, \text{pk}, \{(\text{lab}_i, \text{lab}_i)\}_{i \in [n]}))$ .

Hybrids **Hyb<sub>1</sub>** and **Hyb<sub>2</sub>** are computationally indistinguishable by security of Yao.

Adversary  $\mathcal{A}$ 's view in hybrid **Hyb<sub>2</sub>** does not depend on the challenge bit  $b$ , and the claim follows.  $\square$

### 5.3 Amplifying Weak Key-Hiding AB-SFE to Strong Key-Hiding AB-SFE

We now show how to generically upgrade weak key-hiding to strong key-hiding via KDM-secure secret-key encryption (Definition 2.24) and an equivocable non-interactive commitment scheme (Definition 2.22) [CIO98]. Before presenting our main construction, we first define a useful property on PKE and AB-SFE schemes that we will use in our construction.

**Definition 5.7** (Recovery from Randomness [KW18b]). A public-key encryption scheme  $\text{PKE} = (\text{PKE.KeyGen}, \text{PKE.Encrypt}, \text{PKE.Decrypt})$  with message space  $\mathcal{M}$  satisfies the *recover from randomness property* if there exists an efficient algorithm  $\text{PKE.Recover}$  with the following property:

- $\text{PKE.Recover}(\text{pk}, \text{ct}, r) \rightarrow m/\perp$ : On input a public key  $\text{pk}$ , a ciphertext  $\text{ct}$ , and a string  $r$ , output a message  $m \in \mathcal{M}$  or  $\perp$ .

Then, for all messages  $m \in \mathcal{M}$ , if  $(\text{pk}, \text{sk}) \leftarrow \text{PKE.KeyGen}(1^\lambda)$ ,  $\text{ct} \leftarrow \text{PKE.Encrypt}(\text{pk}, m; r)$ , then  $\text{Recover}(\text{pk}, \text{ct}, r) = m$ . Alternatively, if there is no pair  $(m, r)$  where  $\text{ct} = \text{PKE.Encrypt}(\text{pk}, m; r)$ , then  $\text{Recover}(\text{pk}, \text{ct}, r) = \perp$ . We extend this definition to the AB-SFE setting accordingly: in this case,  $\text{ABSFE.Recover}(\text{crs}, \text{pk}, \text{ct}, r)$  either outputs  $(x, m)$  if  $\text{ct} = \text{ABSFE.Encrypt}(\text{crs}, \text{pk}, x, m; r)$  and  $\perp$  if there does not exist any  $(x, m)$  such that  $\text{ct} = \text{ABSFE.Encrypt}(\text{crs}, \text{pk}, x, m; r)$ .

**Remark 5.8** (Recovery from Randomness [KW18b]). It is straightforward to upgrade any PKE (resp., AB-SFE) scheme to have the recovery from randomness property. As noted in [KW18b], we simply modify the encryption algorithm to use part of the encryption randomness to construct a symmetric encryption of the underlying message (resp., underlying attribute-message pair).

**Construction 5.9** (Weak Key-Hiding to Strong Key-Hiding). Let ABSFE be an AB-SFE scheme for  $F: \mathcal{X} \times \mathcal{Y} \rightarrow \{0, 1\}$  with message space  $\mathcal{M}$  that satisfies weak key-hiding and the recovery from randomness property (Definition 5.7, Remark 5.8). To construct an AB-SFE scheme satisfying strong key-hiding, we additionally rely on the following building blocks:

- Let PKE be a public-key encryption scheme with message space  $\{0, 1\}^\lambda$  and which supports the recovery from randomness property (Definition 2.2, Remark 5.8).
- Let  $\ell = \ell(\lambda)$  be a bound on the number of bits of randomness PKE.Encrypt and ABSFE.Encrypt use.
- Let SKE denote a secret-key encryption scheme with message-space  $\mathcal{M} \times \{0, 1\}^{\ell\lambda}$  that is one-time KDM-secure for the class of projection functions (Definition 2.24, Remark 2.25).
- Let Com be a non-interactive equivocable commitment scheme with message space  $\{0, 1\}$  (Definition 2.22).

We construct an augmented AB-SFE scheme Aug as follows:

- **Aug.Setup**( $1^\lambda$ ): Sample  $\text{ABSFE.crs} \leftarrow \text{ABSFE.Setup}(1^\lambda)$ ,  $(\text{PKE.pk}, \text{PKE.sk}) \leftarrow \text{PKE.Gen}(1^\lambda)$ , and for each  $i \in [\lambda]$ ,  $\text{Com.crs}_i \leftarrow \text{Com.Setup}(1^\lambda)$ . It outputs the common reference string  $\text{crs} = (\text{ABSFE.crs}, \text{PKE.pk}, \{\text{Com.crs}_i\}_{i \in [\lambda]})$ .
- **Aug.KeyGen**( $\text{crs}, y$ ): On input  $\text{crs} = (\text{ABSFE.crs}, \text{PKE.pk}, \{\text{Com.crs}_i\}_{i \in [\lambda]})$ , sample an AB-SFE key-pair  $(\text{ABSFE.pk}, \text{ABSFE.sk}) \leftarrow \text{ABSFE.KeyGen}(\text{ABSFE.crs}, y)$  and output the public key  $\text{pk} = \text{ABSFE.pk}$  and the secret key  $\text{sk} = (y, \text{ABSFE.pk}, \text{ABSFE.sk})$ .
- **Aug.Encrypt**( $\text{crs}, \text{pk}, x, m$ ): Parse  $\text{crs} = (\text{ABSFE.crs}, \text{PKE.pk}, \{\text{Com.crs}_i\}_{i \in [\lambda]})$  and  $\text{pk} = \text{ABSFE.pk}$ , and proceed as follows:
  - Sample a secret key  $s \xleftarrow{\text{R}} \{0, 1\}^\lambda$  for SKE.
  - For every  $i \in [\lambda]$ , sample  $\rho_i \xleftarrow{\text{R}} \{0, 1\}^\lambda$  and compute  $c_i \leftarrow \text{Com.Commit}(\text{Com.crs}_i, s_i; \rho_i)$ .
  - For every  $i \in [\lambda]$ , define  $M_{i,s_i} = \rho_i$  and  $M_{i,1-s_i} = \perp$ . Then, sample  $R_{i,0}, R_{i,1} \xleftarrow{\text{R}} \{0, 1\}^\ell$  and construct the ciphertexts
$$\begin{aligned} \text{ct}_{i,0} &\leftarrow \text{ABSFE.Encrypt}(\text{ABSFE.crs}, \text{ABSFE.pk}, x, M_{i,0}; R_{i,0}) \\ \text{ct}_{i,1} &\leftarrow \text{PKE.Encrypt}(\text{PKE.pk}, M_{i,1}; R_{i,1}). \end{aligned}$$
  - Let  $\text{ct}_0 \leftarrow \text{SKE.Encrypt}(s, (m, (R_{i,s_i})_{i \in [\lambda]}))$ .
  - Output  $\text{ct} = (\text{ct}_0, (c_i, \text{ct}_{i,0}, \text{ct}_{i,1})_{i \in [\lambda]})$ .
- **Aug.Decrypt**( $\text{crs}, \text{sk}, x, \text{ct}$ ): On input  $\text{crs} = (\text{ABSFE.crs}, \text{PKE.pk}, \{\text{Com.crs}_i\}_{i \in [\lambda]})$ , a secret key  $\text{sk} = (y, \text{ABSFE.pk}, \text{ABSFE.sk})$ , and a ciphertext  $\text{ct} = (\text{ct}_0, (c_i, \text{ct}_{i,0}, \text{ct}_{i,1})_{i \in [\lambda]})$ , proceed as follows:
  1. If  $F(x, y) = 0$ , output  $\perp$ .
  2. For every  $i \in [\lambda]$ , compute  $\rho'_i \leftarrow \text{ABSFE.Decrypt}(\text{ABSFE.crs}, \text{ABSFE.sk}, x, \text{ct}_{i,0})$ . If  $\rho'_i \neq \perp$  and  $\text{Com.Commit}(\text{Com.crs}_i, 0; \rho'_i) = c_i$ , set  $s'_i = 0$ ; otherwise, set  $s'_i = 1$ .

3. Compute  $(m', (r'_i)_{i \in [\lambda]}) \leftarrow \text{SKE.Decrypt}(s', \text{ct}_0)$ .
4. For every  $i \in [\lambda]$ , perform the following checks:
  - If  $s'_i = 0$ , then check if  $\text{Recover}(\text{ABSFE.crs}, \text{ABSFE.pk}, \text{ct}_{i,0}, r'_i) = (x, \tilde{\rho}_i)$  for some  $\tilde{\rho}_i$ , and output  $\perp$  if the check fails.
  - If  $s'_i = 1$ , then compute  $\tilde{\rho}_i \leftarrow \text{PKE.Recover}(\text{PKE.pk}, \text{ct}_{i,1}, r'_i)$  and output  $\perp$  if the recovery procedure fails.
  - Finally, check if  $c_i = \text{Com.Commit}(\text{Com.crs}_i, s'_i; \tilde{\rho}_i)$ . Output  $\perp$  if this check fails.
5. If all checks pass, output  $m'$ .

**Theorem 5.10** (Correctness). *If ABSFE and PKE is (perfectly) correct, then the AB-SFE scheme Aug from Construction 5.9 is also (perfectly) correct.*

*Proof.* Follows immediately by correctness of ABSFE and PKE. □

**Theorem 5.11** (Strong Key-Hiding). *If ABSFE satisfies weak key-hiding, and Com is statistically-binding, then the AB-SFE scheme Aug from Construction 5.9 satisfies strong key-hiding.*

*Proof.* Let  $\mathcal{S}_{\text{weak}}$  be the weak key-hiding simulator for ABSFE. We construct a strong key-hiding simulator  $(\mathcal{S}_1, \mathcal{S}_2)$  as follows:

- $\mathcal{S}_1(1^\lambda)$ : Run  $(\text{ABSFE.crs}, \text{ABSFE.pk}) \leftarrow \mathcal{S}_{\text{weak}}(1^\lambda)$ . Then, sample  $\text{PKE.pk}$ ,  $\text{PKE.sk}$ , and  $\{\text{Com.crs}_i\}_{i \in [\lambda]}$  as in  $\text{Aug.Setup}(1^\lambda)$ , and set

$$\overline{\text{crs}} = (\text{ABSFE.crs}, \text{PKE.pk}, \{\text{Com.crs}_i\}_{i \in [\lambda]}).$$

Output  $\text{st}_{\mathcal{S}} = (\overline{\text{crs}}, \text{ABSFE.pk}, \text{PKE.sk})$ ,  $\overline{\text{crs}}$ , and  $\overline{\text{pk}} = \text{ABSFE.pk}$ .

- $\mathcal{S}_2(\text{st}_{\mathcal{S}}, x, \text{ct}, \beta)$ : Parse  $\text{st}_{\mathcal{S}} = (\overline{\text{crs}}, \text{ABSFE.pk}, \text{PKE.sk})$ ,  $\overline{\text{crs}} = (\text{ABSFE.crs}, \text{PKE.pk}, \{\text{Com.crs}_i\}_{i \in [\lambda]})$  and  $\text{ct} = (\text{ct}_0, (c_i, \text{ct}_{i,0}, \text{ct}_{i,1})_{i \in [\lambda]})$ , and proceed as follows:
  1. If  $\beta = 0$ , output  $\perp$ .
  2. For every  $i \in [\lambda]$ , decrypt  $\rho'_i \leftarrow \text{PKE.Decrypt}(\text{PKE.sk}, \text{ct}_{i,1})$ . If  $\rho'_i \neq \perp$  and moreover,  $\text{Com.Commit}(\text{Com.crs}_i, 1; \rho'_i) = c_i$ , set  $s'_i = 1$ ; otherwise, set  $s'_i = 0$ .
  3. Compute  $(m', (r'_i)_{i \in [\lambda]}) \leftarrow \text{SKE.Decrypt}(s', \text{ct}_0)$ .
  4. For every  $i \in [\lambda]$ , perform the following checks:
    - If  $s'_i = 0$ , then check if  $\text{Recover}(\text{ABSFE.crs}, \text{ABSFE.pk}, \text{ct}_{i,0}, r'_i) = (x, \tilde{\rho}_i)$  for some  $\tilde{\rho}_i$ , and output  $\perp$  if the check fails.
    - If  $s'_i = 1$ , then compute  $\tilde{\rho}_i \leftarrow \text{PKE.Recover}(\text{PKE.pk}, \text{ct}_{i,1}, r'_i)$  and output  $\perp$  if the recovery procedure fails.
    - Finally, check if  $c_i = \text{Com.Commit}(\text{Com.crs}_i, s'_i; \tilde{\rho}_i)$ . Output  $\perp$  if this check fails.
  5. If all checks pass, output  $m'$ .

To argue that the simulator satisfies the desired properties, we define the following sequence of hybrid experiments between a challenger and an adversary  $\mathcal{A}$ :

- $\text{Hyb}_0$ : This is the real distribution where  $\text{crs} \leftarrow \text{Aug.Setup}(1^\lambda)$ , and  $(\text{pk}, \text{sk}) \leftarrow \text{Aug.KeyGen}(\text{crs}, y)$ . The oracle queries are handled by computing  $\text{Aug.Decrypt}(\text{crs}, \text{sk}, x, \text{ct})$ . At the end of the experiment,  $\mathcal{A}$  outputs a bit, which is the output of the experiment, denoted  $\text{Hyb}_0(\mathcal{A})$ .



- $\text{Hyb}_1$ : Same as  $\text{Hyb}_0$ , except we use  $\mathcal{S}_2$  to implement the decryption queries (but leaves  $\text{crs}$ ,  $\text{pk}$  unchanged). Namely, in this experiment, the challenger still samples  $\text{crs} \leftarrow \text{Aug.Setup}(1^\lambda)$  and  $(\text{pk}, \text{sk}) \leftarrow \text{Aug.KeyGen}(\text{crs}, y)$ . It also sets  $\text{st}_{\mathcal{S}} = (\text{crs}, \text{ABSFE.pk}, \text{PKE.sk})$ . When responding to the adversary's oracle queries  $(x, \text{ct})$ , the challenger computes  $\mathcal{S}_2(\text{st}_{\mathcal{S}}, x, \text{ct}, F(x, y))$ .

We show in Lemma 5.12 that  $\text{Hyb}_0$  and  $\text{Hyb}_1$  are computationally indistinguishable.

- $\text{Hyb}_2$ : Same as  $\text{Hyb}_1$ , except the simulator uses  $\mathcal{S}_{\text{weak}}$  to sample the components  $\text{ABSFE.crs}$  and  $\text{ABSFE.pk}$  in  $\text{crs}$ ,  $\text{pk}$ , and  $\text{st}_{\mathcal{S}}$ . This is the simulated distribution.

This is computationally indistinguishable from  $\text{Hyb}_1$  since  $\text{ABSFE}$  satisfies weak key-hiding.

**Lemma 5.12.** *If  $\text{Com}$  is statistically binding, and if  $\text{ABSFE}$  and  $\text{PKE}$  have perfect decryption correctness, then for all adversaries  $\mathcal{A}$ , the outputs of  $\text{Hyb}_0(\mathcal{A})$  and  $\text{Hyb}_1(\mathcal{A})$  are statistically indistinguishable.*

*Proof.* Since  $\text{Hyb}_0$  and  $\text{Hyb}_1$  only differ in how the decryption queries are handled, it suffices to argue that the challenger's response to each of the adversary's decryption queries are identical with overwhelming probability. Consider a decryption query  $(x, \text{ct})$  where  $\text{ct} = (\text{ct}_0, (c_i, \text{ct}_{i,0}, \text{ct}_{i,1})_{i \in [\lambda]})$ . First, if  $F(x, y) = 0$ , then in both  $\text{Hyb}_0$  and  $\text{Hyb}_1$ , the challenger replies with  $\perp$ . We consider the case where  $F(x, y) = 1$ . First, we argue that with overwhelming probability, the challenger either recovers the same seed  $s' \in \{0, 1\}^n$  in both  $\text{Hyb}_0$  and  $\text{Hyb}_1$ , or outputs  $\perp$  in both experiments.

- Suppose the challenger sets  $s'_i = 0$  in  $\text{Hyb}_0$  (without aborting). This means the challenger obtained  $\rho'_i$  and  $r'_i$  where  $\text{ct}_{i,0} = \text{ABSFE.Encrypt}(\text{ABSFE.crs}, \text{ABSFE.pk}, \rho'_i; r'_i)$  and in addition,  $c_i = \text{Com.Commit}(\text{Com.crs}_i, 0; \rho'_i)$ . Since the commitment scheme is statistically binding, with overwhelming probability over the choice of  $\text{Com.crs}_i$ , there does not exist  $\rho^*$  such that  $c_i = \text{Com.Commit}(\text{Com.crs}_i, 1, \rho^*)$ . Hence, the challenger in  $\text{Hyb}_1$  will also set  $s'_i = 0$ .
- Suppose the challenger sets  $s'_i = 1$  in  $\text{Hyb}_0$  (without aborting). This means that the challenger must have recovered  $\tilde{\rho}_i$  and  $r'_i$  such that  $\text{ct}_{i,1} = \text{PKE.Encrypt}(\text{PKE.pk}, \tilde{\rho}_i; r'_i)$  and  $c_i = \text{Com.Commit}(\text{Com.crs}_i, 1; \tilde{\rho}_i)$ . By correctness of  $\text{PKE}$ , the challenger in  $\text{Hyb}_1$  also sets  $s'_i = 1$ .

Thus, we see that if the challenger recovers  $s'$  in  $\text{Hyb}_0$  in a successful decryption query, then with overwhelming probability, it recovers the same  $s'$  in  $\text{Hyb}_1$ . Because the decryption procedures in  $\text{Hyb}_0$  and  $\text{Hyb}_1$  are identical after the recovery of  $s'$ , this means that it also outputs the same message in  $\text{Hyb}_1$  as it does in  $\text{Hyb}_0$ . By an entirely analogous argument, the converse also holds, so we conclude that with overwhelming probability, the challenger responds identically to each decryption query in  $\text{Hyb}_0$  and  $\text{Hyb}_1$ . The claim then follows by a union bound over the total number of decryption queries the adversary makes.  $\square$

We conclude that the outputs of  $\text{Hyb}_0$  and  $\text{Hyb}_2$  are computationally indistinguishable.  $\square$

**Remark 5.13** (Relaxing Perfect Correctness). Although our proof of Lemma 5.12 seems to rely on the perfect decryption correctness of  $\text{ABSFE}$  and  $\text{PKE}$ , it suffices for the two encryption schemes to satisfy the following weaker notion of correctness: with probability  $1 - \text{negl}(\lambda)$  over the choice of public parameters, decryption is perfectly correct over the choice of encryption randomness  $r$ . This is the notion of *almost-all-keys* perfect correctness from [DNR04]. This will be important later for our LPN-based instantiation.

**Theorem 5.14** (Message-Hiding). *If ABSFE satisfies weak message-hiding (resp., strong message-hiding), PKE is semantically secure, SKE is one-time KDM-secure for the class of projection functions, and Com is equivocable, then the AB-SFE scheme Aug from Construction 5.9 also satisfies weak message-hiding (resp., strong message-hiding).*

*Proof.* We show the claim for the setting where ABSFE satisfies strong message-hiding. The case where ABSFE satisfies weak message-hiding follows by the same hybrid structure (and without the additional need to define an extraction algorithm). By assumption, since ABSFE satisfies strong message-hiding, there exist efficient algorithms (ABSFE.SetupExt, ABSFE.Ext) with the properties from Definition 3.4. We use these to define the extraction algorithms (Aug.SetupExt, Aug.Ext):

- Aug.SetupExt( $1^\lambda$ ): Call  $(\text{ABSFE.crs}, \text{ABSFE.td}) \leftarrow \text{ABSFE.SetupExt}(1^\lambda)$  and output  $(\text{crs}, \text{td})$  where  $\text{crs} = (\text{ABSFE.crs}, \text{PKE.pk}, \{\text{Com.crs}_i\}_{i \in [\lambda]})$ , where the other components are generated using the same procedure as in Aug.Setup and  $\text{td} = \text{ABSFE.td}$ .
- Aug.Ext( $\text{td}, \text{pk}$ ): Output  $\text{ABSFE.Ext}(\text{td}, \text{pk})$ .

CRS indistinguishability for (Aug.Setup, Aug.SetupExt) follows immediately from the same property for ABSFE. It remains to show ciphertext indistinguishability. We use the following sequence of hybrid experiments:

- Hyb<sub>0</sub>: This is the ciphertext indistinguishability game for Aug.
- Hyb<sub>1</sub>: Same as Hyb<sub>0</sub>, except the challenger substitutes equivocable commitments for real commitments. Namely, at the beginning of the game, the challenger samples for each  $i \in [\lambda]$ ,  $(\text{Com.crs}_i, \tilde{c}_i, \text{td}_i) \leftarrow \text{Com.SetupEquiv}(1^\lambda)$ , and uses  $\text{Com.crs}_i$  in place of  $\text{Com.crs}_i$  in the crs and  $\tilde{c}_i$  in place of  $c_i$  when constructing the challenge ciphertext. In addition, when constructing the challenge ciphertexts, the challenger instead computes  $\rho_i \leftarrow \text{Com.Equivocate}(\text{td}_i, s_i, \tilde{c}_i)$ .

This is computationally indistinguishable from Hyb<sub>0</sub> by the equivocation property of Com.

- Hyb<sub>2</sub>: Same as Hyb<sub>1</sub>, except that when computing the challenge ciphertext, instead of setting  $M_{i,1-s_i} = \perp$ , the challenger computes  $\rho'_i \leftarrow \text{Com.Equivocate}(\text{td}_i, 1-s_i, \tilde{c}_i)$  and sets  $M_{i,1-s_i} = \rho'_i$ .

This is computationally indistinguishable from Hyb<sub>1</sub> by semantic security of PKE and strong message-hiding security of ABSFE (where the challenge public key  $\text{pk}$ , the extracted value  $y \leftarrow \text{ABSFE.Ext}(\text{td}, \text{pk})$  and the challenge attribute  $x \in \mathcal{X}$  are the same in both the base scheme ABSFE and Aug, and satisfy  $F(x, y) = 0$ ). In particular, by construction,  $R_{i,1-s_i}$  is uniformly random and does not appear elsewhere in the experiment. Thus, each  $\text{ct}_{i,1-s_i}$  is an honestly-generated encryption of  $M_{i,1-s_i}$  under either PKE or ABSFE.

- Hyb<sub>3</sub>: Same as Hyb<sub>2</sub>, except that when computing the challenge ciphertext, the challenger simply sets  $M_{i,b} \leftarrow \text{Com.Equivocate}(\text{td}_i, b, \tilde{c}_i)$  for all  $i \in [\lambda]$  and  $b \in \{0, 1\}$ . Namely  $M_{i,b}$  no longer depends on  $s$ .

This is just a relabeling of indices and this experiment is identical to Hyb<sub>3</sub>. Namely in Hyb<sub>2</sub>,  $M_{i,s_i} = \rho_i \leftarrow \text{Com.Equivocate}(\text{td}_i, s_i, c_i)$  and  $M_{i,1-s_i} = \rho'_i \leftarrow \text{Com.Equivocate}(\text{td}_i, 1-s_i, c_i)$ .

- **Hyb<sub>4</sub>**: Same as **Hyb<sub>3</sub>**, except the challenger computes  $\text{ct}_0 \leftarrow \text{SKE.Encrypt}(s, 0^{k+\lambda\ell})$  for the challenge ciphertext, where  $k$  is the bit-length of messages in  $\mathcal{M}$ .

This is computationally indistinguishable from **Hyb<sub>3</sub>** by the KDM-security of SKE. In particular, note that the plaintext  $(m, (R_{i,s_i})_{i \in [\lambda]})$  is a projection function of  $s$ .

Finally, any adversary’s view in **Hyb<sub>4</sub>** is independent of the message  $m$ , and the claim follows.  $\square$

## 5.4 Instantiations

In this section, we describe how to instantiate each of the building blocks needed to obtain an AB-SFE scheme satisfying strong key-hiding and strong (respectively, weak) message-hiding from either the CDH assumption, the LWE assumption, or the LPN assumption with noise rate  $n^{-(\frac{1}{2}+\epsilon)}$  (respectively, the CDH assumption, the LWE assumption, or the LPN assumption with noise rate  $O(1/\sqrt{n})$ ). All of our LWE-based instantiations can use a polynomial modulus-to-noise ratio.

The resulting weak message-hiding AB-SFE instantiations correspondingly yield designated-verifier NIZKs. Moreover, the resulting strong message-hiding AB-SFE schemes have *uniformly random* public parameters, thus yielding designated-verifier NIZKs with security against malicious verifiers in the common *random* string model. We instantiate each building block as follows:

- By Theorem 2.19 and Remark 2.18, there exists a receiver-extractable 2-message batch OT scheme in the common random string model under the CDH/LWE/LPN assumptions (with the parameters specified above). By Theorem 2.21, there exists a garbling scheme from one-way functions. Thus, by Theorem 5.6, we obtain an AB-SFE scheme with strong message-hiding and weak key-hiding under the CDH/LWE/LPN assumptions in the common random string model.
- There exist public-key encryption schemes with pseudorandom (or uniformly random) public keys from the CDH assumption [Gam84], the LWE assumption [Reg05], or the LPN assumption [Ale03] with the parameters specified above. Because we only use the associated secret key in the proof of security, we can replace the public key  $\text{PKE.pk}$  from Construction 5.9 with a uniformly random string, while maintaining security (by a standard hybrid argument) and perfect correctness.
- By Lemma 5.3, there exists an AB-SFE scheme with weak message-hiding and weak key-hiding under any assumption implying PKE.
- By Remark 2.25, there exists a KDM-secure secret-key encryption scheme for projection functions under the CDH assumption, the LWE assumption, or the LPN assumption with constant noise rate.
- By Theorem 2.23, there exists a non-interactive equivocable commitment scheme from one-way functions in the common random string model.

**Remark 5.15** (Almost-All-Keys Perfect Decryption Correctness). Some of the PKE/OT schemes above (such as the PKE scheme of [Ale03]) do not actually satisfy perfect decryption correctness as required by Definition 2.2. However, the transformation of [DNR04] shows that these encryption schemes can be modified to satisfy the following “almost-all-keys perfect correctness” property: with

probability  $1 - \text{negl}(\lambda)$  over the randomness of  $\text{PKE.KeyGen}(\cdot)$ , decryption is perfectly correct with probability 1 over the choice of encryption randomness. Encryption schemes satisfying this notion of almost-all-keys perfect correctness suffice for all of the constructions in this paper (Remark 5.13).

**Instantiations.** Combining the above primitives in Construction 5.9, we now obtain the following corollaries. In all cases, we only rely on polynomial hardness of the underlying assumption.

**Corollary 5.16** (Weak Message-Hiding, Strong Key-Hiding AB-SFE from LPN). *Assuming polynomial hardness of the LPN assumption with noise rate  $O(\frac{1}{\sqrt{n}})$ , there exists an AB-SFE scheme with that satisfies strong key-hiding and weak message-hiding.*

**Corollary 5.17** (Strong Message-Hiding, Strong Key-Hiding AB-SFE from CDH/LWE/LPN). *Assuming polynomial hardness of either CDH, LWE, or LPN with noise rate  $n^{-(\frac{1}{2}+\epsilon)}$  for any  $\epsilon > 0$ , there exists an AB-SFE scheme with uniformly random public parameters that satisfies strong key-hiding and strong message-hiding security.*

Combining Theorem 2.7 now with Construction 4.1 (and Remarks 4.4 and 4.6), we obtain the following instantiations of designated-verifier NIZKs:

**Corollary 5.18** (Designated-Verifier NIZKs from LPN). *Assuming polynomial hardness of the LPN assumption with noise rate  $O(\frac{1}{\sqrt{n}})$ , there exists a designated-verifier NIZK argument for NP that is adaptively sound and provides computational zero-knowledge in the common reference string model.*

**Corollary 5.19** (Malicious-Designated-Verifier NIZKs from CDH/LWE/LPN). *Assuming polynomial hardness of either CDH, LWE, or LPN with noise rate  $n^{-(\frac{1}{2}+\epsilon)}$  for any  $\epsilon > 0$ , there exists a designated-verifier NIZK argument for NP that is adaptively sound and provides computational zero-knowledge against malicious verifiers in the common random string model.*

## Acknowledgments

We thank Yuval Ishai and Brent Waters for many helpful discussions and comments on this work.

## References

- [ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In *EUROCRYPT*, pages 553–572, 2010.
- [ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *CRYPTO*, pages 595–618, 2009.
- [Ajt99] Miklós Ajtai. Generating hard instances of the short basis problem. In *ICALP*, pages 1–9, 1999.
- [Ale03] Michael Alekhnovich. More on average case vs approximation complexity. In *FOCS*, pages 298–307, 2003.

- [AP09] Joël Alwen and Chris Peikert. Generating shorter bases for hard random lattices. In *STACS*, pages 75–86, 2009.
- [BB04] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *EUROCRYPT*, pages 223–238, 2004.
- [BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In *ACM CCS*, pages 896–912, 2018.
- [BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *STOC*, pages 103–112, 1988.
- [BGG<sup>+</sup>14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *EUROCRYPT*, pages 533–556, 2014.
- [BHHO08] Dan Boneh, Shai Halevi, Michael Hamburg, and Rafail Ostrovsky. Circular-secure encryption from decision diffie-hellman. In *CRYPTO*, pages 108–125, 2008.
- [BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *ACM CCS*, pages 784–796, 2012.
- [BLP<sup>+</sup>13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *STOC*, pages 575–584, 2013.
- [BLSV18] Zvika Brakerski, Alex Lombardi, Gil Segev, and Vinod Vaikuntanathan. Anonymous IBE, leakage resilience and circular security from new assumptions. In *EUROCRYPT*, pages 535–564, 2018.
- [Blu86] Manuel Blum. How to prove a theorem so no one else can claim it. In *Proceedings of the International Congress of Mathematicians*, volume 1, page 2, 1986.
- [BPR12] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In *EUROCRYPT*, pages 719–737, 2012.
- [CCH<sup>+</sup>18] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, and Ron D. Rothblum. Fiat-Shamir from simpler assumptions. *IACR Cryptology ePrint Archive*, 2018:1004, 2018.
- [CDI<sup>+</sup>19] Melissa Chase, Yevgeniy Dodis, Yuval Ishai, Daniel Kraschewski, Tianren Liu, Rafail Ostrovsky, and Vinod Vaikuntanathan. Reusable non-interactive secure computation. In *CRYPTO*, 2019.
- [CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, pages 174–187, 1994.
- [CH19] Geoffroy Couteau and Dennis Hofheinz. Designated-verifier pseudorandom generators, and their applications. In *EUROCRYPT*, 2019.

- [CHK03] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. *IACR Cryptology ePrint Archive*, 2003:83, 2003.
- [CHKP10] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In *EUROCRYPT*, pages 523–552, 2010.
- [CIO98] Giovanni Di Crescenzo, Yuval Ishai, and Rafail Ostrovsky. Non-interactive and non-malleable commitment. In *STOC*, pages 141–150, 1998.
- [CLW18] Ran Canetti, Alex Lombardi, and Daniel Wichs. Fiat-Shamir: From practice to theory, part ii (NIZK and correlation intractability from circular-secure FHE). *IACR Cryptology ePrint Archive*, 2018:1248, 2018.
- [CS98] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO*, pages 13–25, 1998.
- [CS02] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *EUROCRYPT*, pages 45–64, 2002.
- [DG17] Nico Döttling and Sanjam Garg. Identity-based encryption from the Diffie-Hellman assumption. In *CRYPTO*, pages 537–569, 2017.
- [DGH<sup>+</sup>19] Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, Daniel Masny, and Daniel Wichs. Two-round oblivious transfer from CDH or LPN. *IACR Cryptology ePrint Archive*, 2019, 2019.
- [DGHM18] Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, and Daniel Masny. New constructions of identity-based and key-dependent message secure encryption schemes. In *PKC*, pages 3–31, 2018.
- [DNR04] Cynthia Dwork, Moni Naor, and Omer Reingold. Immunizing encryption schemes from decryption errors. In *EUROCRYPT*, pages 342–360, 2004.
- [DRS04] Yevgeniy Dodis, Leonid Reyzin, and Adam D. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *EUROCRYPT*, pages 523–540, 2004.
- [FLS99] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs under general assumptions. *SIAM J. Comput.*, 29(1):1–28, 1999.
- [FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.
- [Gam84] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*, pages 10–18, 1984.
- [GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *FOCS*, pages 464–479, 1984.
- [GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

- [GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In *FOCS*, pages 174–187, 1986.
- [Gol11] Oded Goldreich. Basing non-interactive zero-knowledge on (enhanced) trapdoor permutations: The state of the art. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pages 406–421. Springer, 2011.
- [GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Non-interactive Zaps and new techniques for NIZK. In *CRYPTO*, pages 97–111, 2006.
- [GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM CCS*, pages 89–98, 2006.
- [GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206, 2008.
- [GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In *CRYPTO*, pages 162–179, 2012.
- [GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In *STOC*, pages 545–554, 2013.
- [HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.
- [IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *STOC*, pages 21–30, 2007.
- [IMS12] Yuval Ishai, Mohammad Mahmoody, and Amit Sahai. On efficient zero-knowledge PCPs. In *TCC*, pages 151–168, 2012.
- [KM19] Fuyuki Kitagawa and Takahiro Matsuda. Cpa-to-cca transformation for kdm security. *IACR Cryptology ePrint Archive*, 2019:609, 2019. <https://eprint.iacr.org/2019/609>.
- [KMT19] Fuyuki Kitagawa, Takahiro Matsuda, and Keisuke Tanaka. Cca security and trapdoor functions via key-dependent-message security. In *CRYPTO*, 2019. <https://eprint.iacr.org/2019/291>.
- [KNYY19] Shuichi Katsumata, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Designated verifier/prover and preprocessing NIZKs from Diffie-Hellman assumptions. In *EUROCRYPT*, 2019.
- [KPT97] Joe Kilian, Erez Petrank, and Gábor Tardos. Probabilistically checkable proofs with zero knowledge. In *STOC*, pages 496–505, 1997.
- [KW18a] Sam Kim and David J. Wu. Multi-theorem preprocessing NIZKs from lattices. In *CRYPTO*, pages 733–765, 2018.

- [KW18b] Venkata Koppula and Brent Waters. Realizing chosen ciphertext security generically in attribute-based encryption and predicate encryption. *IACR Cryptology ePrint Archive*, 2018:847, 2018.
- [LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.
- [LQR<sup>+</sup>19] Alex Lombardi, Willy Quach, Ron D. Rothblum, Daniel Wichs, and David J. Wu. New constructions of reusable designated-verifier NIZKs. *IACR Cryptology ePrint Archive*, 2019, 2019. Preliminary Version: <https://eprint.iacr.org/2019/242/20190228:191300>.
- [LW15] Vadim Lyubashevsky and Daniel Wichs. Simple lattice trapdoor sampling from a broad class of distributions. In *PKC*, pages 716–730, 2015.
- [MM11] Daniele Micciancio and Petros Mol. Pseudorandom knapsacks and the sample complexity of LWE search-to-decision reductions. In *CRYPTO*, pages 465–484, 2011.
- [MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*, pages 700–718, 2012.
- [Nao91] Moni Naor. Bit commitment using pseudorandomness. *J. Cryptology*, 4(2):151–158, 1991.
- [NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *STOC*, pages 427–437, 1990.
- [Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In *STOC*, pages 333–342, 2009.
- [PS19] Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for NP from (plain) learning with errors. *IACR Cryptology ePrint Archive*, 2019:158, 2019.
- [PsV06] Rafael Pass, abhi shelat, and Vinod Vaikuntanathan. Construction of a non-malleable encryption scheme from any semantically secure one. In *CRYPTO*, pages 271–289, 2006.
- [PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO*, pages 554–571, 2008.
- [QRW19] Willy Quach, Ron D. Rothblum, and Daniel Wichs. Reusable designated-verifier NIZKs for all NP from CDH. In *EUROCRYPT*, 2019.
- [Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93, 2005.
- [SCO<sup>+</sup>01] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In *CRYPTO*, pages 566–598, 2001.
- [SMP87] Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Non-interactive zero-knowledge proof systems. In *CRYPTO*, pages 52–72, 1987.



- [SS10] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In *ACM CCS*, pages 463–472, 2010.
- [SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.
- [Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.

## A Adaptive Soundness via Trapdoor Zero-Knowledge PCPs

In this section, we describe how to achieve adaptive soundness in our protocols by relying on the trapdoor  $\Sigma$ -protocols from [CLW18, §6]. We present our construction more generally in the language of zero-knowledge PCPs by first defining the notion of a “trapdoor zero-knowledge PCP.” At a high-level, a trapdoor zero-knowledge PCP for an NP language  $\mathcal{L}$  has the following properties:

- Without loss of generality, we allow the prover algorithm to output some auxiliary data  $\mathbf{aux}$  together with the PCP  $\pi$ . The verifier is given the auxiliary data  $\mathbf{aux}$  in addition to its chosen bits of  $\pi$ . This is just for notational convenience since we can always include  $\mathbf{aux}$  as part of the PCP, and have the verifier always read the bits corresponding to  $\mathbf{aux}$ .
- We allow the prover and verifier algorithms to take a common reference (or random) string  $\mathbf{crs}$  as additional input.
- For every choice of common reference string  $\mathbf{crs}$ , every false statement  $x \notin \mathcal{L}$ , every choice of auxiliary data  $\mathbf{aux}$ , and every proof string  $\pi \in \Sigma^m$ , there exists exactly one setting of the randomness  $r = f(\mathbf{crs}, x, \mathbf{aux})$  such that  $\text{zkPCP.Verify}(\mathbf{crs}, \mathbf{aux}, \mathbf{st}_x, \pi_{q_1}, \dots, \pi_{q_\ell}) = 1$ , where  $(\mathbf{st}_x, q_1, \dots, q_\ell) \leftarrow \text{zkPCP.Query}(\mathbf{crs}, x)$ . Note that  $f$  does not have to be efficiently-computable, but becomes efficiently computable with knowledge of a trapdoor (associated with the  $\mathbf{crs}$ ).

We define the notion more formally below:

**Definition A.1** (Trapdoor Zero-Knowledge PCP). Let  $\mathcal{R}: \{0, 1\}^n \times \{0, 1\}^h \rightarrow \{0, 1\}$  be an NP relation and  $\mathcal{L} \subseteq \{0, 1\}^n$  be the associated language. A *non-adaptive,  $\ell$ -query trapdoor zero-knowledge PCP* (with alphabet  $\Sigma$ ) for  $\mathcal{L}$  is a tuple of algorithms  $\text{zkPCP} = (\text{zkPCP.Setup}, \text{zkPCP.Prove}, \text{zkPCP.Query}, \text{zkPCP.Verify})$  with the following properties:

- $\text{zkPCP.Setup}(1^\lambda) \rightarrow \mathbf{crs}$ : On input a security parameter  $\lambda$ , the setup algorithm outputs a common reference string  $\mathbf{crs}$ .
- $\text{zkPCP.Prove}(\mathbf{crs}, x, w) \rightarrow (\mathbf{aux}, \pi)$ : On input the common reference string  $\mathbf{crs}$ , a statement  $x \in \{0, 1\}^n$ , and a witness  $w \in \{0, 1\}^h$ , the prove algorithm outputs some auxiliary data  $\mathbf{aux}$  and a proof  $\pi \in \Sigma^m$ .
- $\text{zkPCP.Query}(\mathbf{crs}, x) \rightarrow (\mathbf{st}_x, q_1, \dots, q_\ell)$ : On input the common reference string  $\mathbf{crs}$ , a statement  $x \in \{0, 1\}^n$ , the query-generation algorithm outputs a verification state  $\mathbf{st}_x$  and  $\ell$  query indices  $q_1, \dots, q_\ell \in [m]$ .

- $\text{zkPCP.Verify}(\text{crs}, \text{st}_x, \text{aux}, s_1, \dots, s_\ell) \rightarrow \{0, 1\}$ : On input the common reference string  $\text{crs}$ , a verification state  $\text{st}_x$ , auxiliary proof data  $\text{aux}$ , and a set of responses  $s_1, \dots, s_\ell \in \Sigma$ , the verify algorithm outputs a bit  $b \in \{0, 1\}$ .

Moreover,  $\text{zkPCP}$  should satisfy the following properties:

- **Efficiency:** The running times of  $\text{zkPCP.Setup}$ ,  $\text{zkPCP.Prove}$ ,  $\text{zkPCP.Query}$ , and  $\text{zkPCP.Verify}$  are  $\text{poly}(\lambda)$ .
- **Completeness:** For all  $x \in \{0, 1\}^n$  and  $w \in \{0, 1\}^h$  where  $\mathcal{R}(x, w) = 1$ ,

$$\Pr[\text{zkPCP.Verify}(\text{crs}, \text{aux}, \text{st}_x, \pi_{q_1}, \dots, \pi_{q_\ell}) = 1] = 1,$$

where  $\text{crs} \leftarrow \text{zkPCP.Setup}(1^\lambda)$ ,  $(\text{aux}, \pi) \leftarrow \text{zkPCP.Prove}(\text{crs}, x, w)$  and  $(\text{st}_x, q_1, \dots, q_\ell) \leftarrow \text{zkPCP.Query}(\text{crs}, x)$ .

- **Soundness:** There exist a function  $f$  (not necessarily efficiently-computable), such that for all common reference strings  $\text{crs}$ , all  $x \notin \mathcal{L}$ , all auxiliary data  $\text{aux}$ , and all proof strings  $\pi \in \Sigma^m$ , for all  $r \neq f(\text{crs}, x, \text{aux})$ ,

$$\text{zkPCP.Verify}(\text{crs}, \text{aux}, \text{st}_x, \pi_{q_1}, \dots, \pi_{q_\ell}) = 0,$$

where  $(\text{st}_x, q_1, \dots, q_\ell) \leftarrow \text{zkPCP.Query}(\text{crs}, x; r)$ .

- **Zero-knowledge:** For a security parameter  $\lambda$  an adversary  $\mathcal{A}$ , and a simulator  $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ , let  $\text{Expt}_{\text{zk}, \mathcal{A}, \mathcal{S}}(\lambda, b)$  as follows:

- **Setup:** If  $b = 0$ , then the challenger samples  $\text{crs} \leftarrow \text{zkPCP.Setup}(1^\lambda)$ . If  $b = 1$ , the challenger samples  $(\text{st}_{\mathcal{S}}, \text{crs}) \leftarrow \mathcal{S}_1(1^\lambda)$ . It gives the common reference string  $\text{crs}$  to  $\mathcal{A}$ .
- **Challenge:** The adversary  $\mathcal{A}$  outputs a statement  $x \in \{0, 1\}^n$ , a witness  $w \in \{0, 1\}^h$  and a collection of  $\ell$  indices  $q_1, \dots, q_\ell \in [m]$ . If  $\mathcal{R}(x, w) = 0$ , then the challenger aborts and outputs 0. Otherwise, if  $b = 0$ , the challenger computes  $(\text{aux}, \pi) \leftarrow \text{zkPCP.Prove}(\text{crs}, x, w)$  and replies to  $\mathcal{A}$  with  $(\text{aux}, \pi_{q_1}, \dots, \pi_{q_\ell})$ . If  $b = 1$ , the challenger replies with  $(\text{aux}, \pi_1, \dots, \pi_\ell) \leftarrow \mathcal{S}_2(\text{st}_{\mathcal{S}}, x, q_1, \dots, q_\ell)$ .
- **Output:** At the end of the experiment,  $\mathcal{A}$  outputs a bit  $b' \in \{0, 1\}$ , which is also the output of the experiment.

We say that  $\text{zkPCP}$  provides (semi-malicious) zero-knowledge if for all PPT (semi-malicious) adversaries, there exists a PPT simulator  $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$  such that

$$|\Pr[\text{Expt}_{\text{zk}, \mathcal{A}, \mathcal{S}}(\lambda, 0) = 1] - \Pr[\text{Expt}_{\text{zk}, \mathcal{A}, \mathcal{S}}(\lambda, 1) = 1]| = \text{negl}(\lambda).$$

- **Trapdoor generation:** There exist PPT algorithms ( $\text{zkPCP.TrapGen}$ ,  $\text{zkPCP.BadRandomness}$ ) defined as follows:

- $\text{zkPCP.TrapGen}(1^\lambda) \rightarrow (\text{crs}, \text{td})$ : On input the security parameter  $\lambda$ , the trapdoor-generation algorithm outputs a simulated reference string  $\text{crs}$  and a trapdoor  $\text{td}$ .
- $\text{zkPCP.BadRandomness}(\text{crs}, \text{td}, x, \text{aux}) \rightarrow r$ : On input a common reference string  $\text{crs}$ , a trapdoor  $\text{td}$ , an instance  $x \in \{0, 1\}^n$  and auxiliary proof data  $\text{aux}$ , the bad-challenge-computation algorithm outputs some randomness  $r$  (for the  $\text{zkPCP.Query}$  algorithm).

Moreover, these two algorithms satisfy the following two properties:

- **CRS indistinguishability:** The common reference string output by `zkPCP.Setup` and `zkPCP.TrapGen` algorithms are computationally indistinguishable:

$$\{\text{crs} \leftarrow \text{zkPCP.Setup}(1^\lambda) : \text{crs}\} \stackrel{c}{\approx} \{(\text{crs}, \text{td}) \leftarrow \text{zkPCP.TrapGen}(1^\lambda) : \text{crs}\}.$$

- **Correctness:** For every instance  $x \notin \mathcal{L}$  and every setting of the auxiliary data `aux`,

$$\Pr[\text{zkPCP.BadRandomness}(\text{crs}, \text{td}, x, \text{aux}) = f(\text{crs}, x, \text{aux})] = 1,$$

where  $(\text{crs}, \text{td}) \leftarrow \text{zkPCP.TrapGen}(1^\lambda)$ .

**Remark A.2** (Instantiating Trapdoor Zero-Knowledge PCPs). We can construct trapdoor zero-knowledge PCPs from any “instance-independent” trapdoor  $\Sigma$ -protocol as defined by [CLW18]. For example, as suggested in [CLW18], we can use Blum’s protocol for graph Hamiltonicity [Blu86]. This yields a trapdoor zero-knowledge PCP for NP in the common random string model.

**Adaptive soundness via trapdoor zero-knowledge PCPs.** If we instantiate Construction 4.1 using a trapdoor zero-knowledge PCP, it is straightforward to show that the resulting DV-NIZK is adaptively sound. Note that we first need to modify the construction so that `dvNIZK.Setup` also includes the CRS for the trapdoor zero-knowledge PCP, and `dvNIZK.Prove` includes the auxiliary data `aux` output by the zero-knowledge PCP prover (in the clear). With these modifications, adaptive soundness roughly follows from the fact that for any false statement  $x \notin \mathcal{L}$ , there is only one setting of the PCP verifier’s randomness  $r$  that causes the verifier to accept. Since the randomness is derived from a pseudorandom function (and assumed to be drawn from a super-polynomial space), this event happens with negligible probability. We give the formal statement in the following theorem:

**Theorem A.3** (Adaptive Soundness). *If PRF is a secure PRF, ABSFE satisfies strong key-hiding, and `zkPCP` is a trapdoor zero-knowledge PCP (where the randomness complexity  $\rho$  of `zkPCP.Query` satisfies  $\rho = \omega(\log \lambda)$ ), then `dvNIZK` from Construction 4.1 (where we modify `dvNIZK.Setup` to include the common reference string output by `zkPCP.Setup` and `dvNIZK.Prove` includes the auxiliary information output by `zkPCP.Prove` as part of the proof) satisfies adaptive computational soundness.*

*Proof.* Before proving the theorem, we first show that without loss of generality, an adversary breaking the soundness of a DV-NIZK can be assumed not to make verification queries on its eventual challenge statement; this is true even for adaptive soundness.

**Lemma A.4** (No Verification Queries on the Challenge Statement). *A DV-NIZK argument system is adaptively (resp., non-adaptively) sound if and only if it is adaptively (resp., non-adaptively) sound against adversaries that are further required not to make verification queries on their eventual challenge statements.*

*Proof.* Suppose that some PPT adversary  $\mathcal{A}$  breaks (either adaptive or non-adaptive) soundness of some scheme `dvNIZK` (for a language  $\mathcal{L}$ ) with non-negligible probability. We define a new adversary  $\mathcal{A}'$  as follows:

- At the beginning of the security game,  $\mathcal{A}'$  chooses a uniformly random pair of indices  $i^*, j^* \stackrel{R}{\leftarrow} [Q]$  (where  $Q$  is a bound on the number of verification queries  $\mathcal{A}$  makes).

- Algorithm  $\mathcal{A}'$  starts running algorithm  $\mathcal{A}$ . For an index  $i \in [Q]$ , let  $(x_i, \pi_i)$  denote the  $i^{\text{th}}$  verification query  $\mathcal{A}$  makes.
- Whenever  $\mathcal{A}$  makes a verification query on statement  $(x_i, \pi_i)$  where  $i \geq i^*$  and  $x_i = x_{i^*}$ , algorithm  $\mathcal{A}'$  replies to  $\mathcal{A}$  with  $\perp$ . Otherwise,  $\mathcal{A}'$  forwards  $(x_i, \pi_i)$  to its verification oracle and replies to  $\mathcal{A}$  with the oracle's response.
- At the end of the experiment,  $\mathcal{A}'$  outputs  $(x_{j^*}, \pi_{j^*})$ . If  $\mathcal{A}$  makes fewer than  $j^*$  queries, then  $\mathcal{A}'$  outputs whatever  $\mathcal{A}$  outputs.

By construction, with non-negligible probability,  $\mathcal{A}'$  will guess a pair  $(i^*, j^*)$  such that with non-negligible probability over the randomness of an honest execution of  $\mathcal{A}$ , the following hold:

- $\mathcal{A}$  is successful on some instance  $x \notin \mathcal{L}$ ,
- $i^*$  is the first oracle query involving the statement  $x$ , and
- $j^*$  is the first oracle query involving the statement  $x$  on which the verifier accepts.

Thus, we see that  $\mathcal{A}'$  breaks the soundness of  $\text{dvNIZK}$  with non-negligible probability without ever making a verification query on its challenge instance.  $\square$

The proof of Theorem A.3 now follows by a similar sequence of hybrid experiments as the proof of non-adaptive soundness (Theorem 4.3):

- **Hyb<sub>0</sub>**: This is the real adaptive soundness experiment. At the beginning of the game, the challenger samples  $k \xleftarrow{\mathbb{R}} \mathcal{K}$ ,  $\text{zkPCP.crs} \leftarrow \text{zkPCP.Setup}(1^\lambda)$ ,  $\text{ABSFE.crs} \leftarrow \text{ABSFE.Setup}(1^\lambda)$ , and  $(\text{pk}', \text{sk}') \leftarrow \text{ABSFE.KeyGen}(\text{ABSFE.crs}, k)$ . It gives  $\text{crs} = (\text{zkPCP.crs}, \text{ABSFE.crs})$  and  $\text{pk} = \text{pk}'$  to  $\mathcal{A}$  and sets  $\text{sk} = (k, \text{sk}')$ . When the adversary makes a verification query  $(x, \pi)$ , the challenger replies with  $\text{dvNIZK.Verify}(\text{crs}, \text{sk}, x, \pi)$ . At the end of the experiment, the adversary outputs a statement  $x^*$  and a proof  $\pi^*$ . The output of the experiment is 1 if  $x^* \notin \mathcal{L}$  and  $\text{dvNIZK.Verify}(\text{crs}, \text{sk}, x^*, \pi^*) = 1$  and the adversary does not make any oracle queries on  $x^*$ . Otherwise, the output is 0.

The further condition that the adversary does not make oracle queries on  $x^*$  is without loss of generality by Lemma A.4.

- **Hyb<sub>1</sub>**: Same as **Hyb<sub>0</sub>**, except at the end of the experiment, after the adversary outputs its statement  $x^*$  and proof  $\pi^* = (\text{aux}^*, \text{ct}_1^*, \dots, \text{ct}_m^*)$ , the output of the experiment is 1 if  $f(\text{crs}, x^*, \text{aux}^*) = \text{PRF}(k, x^*)$  (and the adversary makes no oracle queries on  $x^*$ ).

We claim that for all adversaries  $\mathcal{A}$ ,  $\Pr[\text{Hyb}_0(\mathcal{A}) = 1] \leq \Pr[\text{Hyb}_1(\mathcal{A}) = 1]$ . By construction, **Hyb<sub>0</sub>** and **Hyb<sub>1</sub>** only differ in how the output is computed. Suppose **Hyb<sub>0</sub>**( $\mathcal{A}$ ) = 1. We show that **Hyb<sub>1</sub>**( $\mathcal{A}$ ) also outputs 1 in this case. If **Hyb<sub>0</sub>**( $\mathcal{A}$ ) = 1, then it must be the case that  $x^* \notin \mathcal{L}$  and  $\text{dvNIZK.Verify}(\text{crs}, \text{sk}, x^*, \pi^*) = 1$ . By construction, the verification algorithm first computes  $(\text{st}_{x^*}, q_1, \dots, q_\ell) \leftarrow \text{zkPCP.Query}(x^*; \text{PRF}(k, x^*))$ , decrypts  $\text{ct}_{q_1}^*, \dots, \text{ct}_{q_\ell}^*$  to obtain values  $s_1, \dots, s_\ell \in \Sigma$ , and finally outputs  $\text{zkPCP.Verify}(\text{zkPCP.crs}, \text{st}_{x^*}, \text{aux}^*, s_1, \dots, s_\ell)$ . Since  $x^* \notin \mathcal{L}$ , by soundness of  $\text{zkPCP}$ , there is only one setting of the randomness to  $\text{zkPCP.Query}$  that can cause  $\text{zkPCP.Verify}$  to accept: namely, if  $\text{PRF}(k, x^*) = f(\text{crs}, x^*, \text{aux}^*)$ . In this case, the output in **Hyb<sub>1</sub>** is also 1, and the claim follows.

In the following, we show that  $\Pr[\text{Hyb}_1(\mathcal{A}) = 1] = \text{negl}(\lambda)$ , which upper bounds the probability that the adversary breaks soundness.

- **Hyb<sub>2</sub>**: Same as **Hyb<sub>1</sub>**, except the challenger computes  $(\text{zkPCP.crs}, \text{td}) \leftarrow \text{zkPCP.TrapGen}$  when constructing **crs** and it uses  $\text{zkPCP.BadRandomness}(\text{zkPCP.crs}, \text{td}, x^*, \text{aux}^*)$  to compute  $f(\text{crs}, x^*, \text{aux}^*)$  when computing the output of the experiment. In particular, the output of this experiment can be *efficiently* computed.

This is computationally indistinguishable from **Hyb<sub>1</sub>** by the trapdoor-generation properties of **zkPCP**.

- **Hyb<sub>3</sub>**: Same as **Hyb<sub>2</sub>**, except the challenger uses the simulator  $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$  for the strong key-hiding game for **ABSFE** to construct **ABSFE.crs**, the public key **pk**, and to implement the decryption algorithm when responding to the verification queries. In particular, this is the analog of **Hyb<sub>1</sub>** in the proof of Theorem 4.3.

As in the proof of Theorem 4.3, this is indistinguishable from **Hyb<sub>2</sub>** from by strong key-hiding security of **ABSFE**.

- **Hyb<sub>4</sub>**: Same as **Hyb<sub>3</sub>**, except the challenger samples a random function  $g \xleftarrow{\mathcal{R}} \text{Funs}[\{0, 1\}^n, \{0, 1\}^\rho]$  at the beginning of the experiment. Whenever it needs to compute  $\text{PRF}(k, x)$ , it instead computes  $g(x)$ . This is the analog of **Hyb<sub>2</sub>** in the proof of Theorem 4.3.

As in the proof of Theorem 4.3, this is indistinguishable from **Hyb<sub>3</sub>** by security of **PRF**.

In hybrid **Hyb<sub>4</sub>**, the output is 1 only if  $g(x^*) = \text{zkPCP.BadRandomness}(\text{zkPCP.crs}, \text{td}, x^*, \text{aux}^*)$ . Since the function  $g$  is uniformly random (and in particular, independent of  $\text{zkPCP.BadRandomness}$ ) and the adversary makes no oracle calls on  $x^*$  (thus revealing no information about  $g(x^*)$  in the experiment), this relation holds with probability  $1/2^\rho = \text{negl}(\lambda)$ . Thus, with all but negligible probability, the output of **Hyb<sub>4</sub>**( $\mathcal{A}$ ) will be 0, and the claim follows.  $\square$

## B Receiver-Extractable OT from DDH or LWE

In this section, we prove the LWE instantiation in Theorem 2.19 using the OT construction based on dual-mode encryption from [PVW08]. In a dual-mode encryption scheme, there is a setup algorithm that outputs a CRS in one of two computationally indistinguishable modes: a “messy” mode and a “decryption” mode. When using a dual-mode encryption scheme to construct an OT protocol, the properties of the messy mode are used to obtain extractability against malicious receivers, while the decryption mode is used to obtain extractability against malicious senders. In [PVW08], the LWE-based instantiation of dual-mode encryption satisfies a weaker property in the decryption mode that only ensures security in a setting where the CRS is used for an a priori bounded number of OTs. Here, we show that if we only require extractability against malicious receivers, the properties of the decryption mode still suffice to prove the weaker requirement that the receiver’s choice bit is computationally hidden from the sender.

This is done in two steps. First, we recall that we can build a *messy* encryption scheme satisfying some key-hiding property, which is a relaxation of the dual-mode encryption of [PVW08], starting from either LWE or DDH. Then we show that such a scheme yields a receiver-extractable OT.

We start by defining key-hiding messy encryption schemes, which are a relaxation of the dual mode encryption of [PVW08], where the only mode used is the messy mode, and where the properties of the decryption mode are relaxed.

**Definition B.1** (Key-Hiding Messy Encryption). A *key-hiding messy encryption scheme* with message space  $\{0, 1\}^k$  is a tuple of PPT algorithms ( $\text{SetupMessy}$ ,  $\text{KeyGen}$ ,  $\text{Encrypt}$ ,  $\text{Decrypt}$ ,  $\text{FindMessy}$ ) with the following syntax:

- $\text{SetupMessy}(1^\lambda) \rightarrow (\text{crs}, \text{td})$ : Given the security parameter  $\lambda$ , the setup algorithm outputs a common reference string  $\text{crs}$  along with a trapdoor  $\text{td}$ .
- $\text{KeyGen}(\text{crs}, b) \rightarrow (\text{pk}, \text{sk})$ : Given a reference string  $\text{crs}$  and a branch  $b \in \{0, 1\}$ , the key-generation algorithm outputs a public key  $\text{pk}$  and a secret key  $\text{sk}$ .
- $\text{Encrypt}(\text{crs}, \text{pk}, b, m) \rightarrow \text{ct}$ : Given a reference string  $\text{crs}$ , a public key  $\text{pk}$ , a branch  $b \in \{0, 1\}$  and a message  $m \in \{0, 1\}^k$ , the encryption algorithm outputs a ciphertext  $\text{ct}$ .
- $\text{Decrypt}(\text{crs}, \text{sk}, \text{ct}) \rightarrow m$ : Given a reference string  $\text{crs}$ , a secret key  $\text{sk}$  and a ciphertext  $\text{ct}$ , the decryption algorithm outputs a message  $m$ .
- $\text{FindMessy}(\text{td}, \text{pk}) \rightarrow b$ : Given a trapdoor  $\text{td}$  and a (possibly malformed) public key  $\text{pk}$ , the find-messy algorithm outputs a branch  $b \in \{0, 1\}$ .

We require the following properties to hold:

- **Completeness for decryptable branch:** For all  $m \in \{0, 1\}^k$  and  $b \in \{0, 1\}$ :

$$\Pr[\text{Decrypt}(\text{crs}, \text{sk}, \text{Encrypt}(\text{crs}, \text{pk}, b, m)) = m] = 1,$$

where  $(\text{crs}, \text{td}) \leftarrow \text{SetupMessy}(1^\lambda)$  and  $(\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(\text{crs}, b)$ .

- **Trapdoor identification of a messy branch:** With overwhelming probability over  $(\text{crs}, \text{td}) \leftarrow \text{SetupMessy}(1^\lambda)$ , it holds for all (possibly malformed)  $\text{pk}$  and all messages  $m_0, m_1 \in \{0, 1\}^k$ ,

$$\text{Encrypt}(\text{crs}, \text{pk}, b, m_0) \stackrel{s}{\approx} \text{Encrypt}(\text{crs}, \text{pk}, b, m_1),$$

where  $b \leftarrow \text{FindMessy}(\text{td}, \text{pk})$ .

- **Key-hiding:** The following distributions are computationally indistinguishable:

$$(\text{crs}, \text{pk}_0) \stackrel{c}{\approx} (\text{crs}, \text{pk}_1),$$

where  $(\text{crs}, \text{td}) \leftarrow \text{SetupMessy}(1^\lambda)$ ,  $(\text{pk}_0, \text{sk}_0) \leftarrow \text{KeyGen}(\text{crs}, 0)$ , and  $(\text{pk}_1, \text{sk}_1) \leftarrow \text{KeyGen}(\text{crs}, 1)$ .

We now show that the dual-mode encryption schemes of [PVW08] induce key-hiding messy encryption schemes with a *uniformly random* CRS.

**Theorem B.2** (Key-Hiding Messy Encryption from DDH and LWE [PVW08]). *Assuming DDH or LWE (with polynomial modulus-to-noise ratio), there exists a key-hiding messy encryption scheme. Furthermore, the CRS output by SetupMessy is a uniformly random string.*

*Proof.* The constructions are the messy modes of the dual-mode encryption schemes from [PVW08] (specifically, the DDH-based construction from [PVW08, §5.3] and the LWE-based construction from [PVW08, §7.3]). Both of these constructions satisfy completeness and trapdoor identification, which are required of any dual-mode encryption scheme. To show key-hiding, we use the following property that the same constructions satisfy (which is also proven in [PVW08]):

- There exists an algorithm  $(\text{crs}', \text{pk}', \text{sk}'_0, \text{sk}'_1) \leftarrow \text{SetupDec}(1^\lambda)$  such that for all  $b \in \{0, 1\}$ ,

$$(\text{crs}, \text{pk}, \text{sk}_b) \stackrel{c}{\approx} (\text{crs}', \text{pk}', \text{sk}'_b),$$

where  $(\text{crs}, \text{td}) \leftarrow \text{SetupMessy}(1^\lambda)$ ,  $(\text{pk}, \text{sk}_b) \leftarrow \text{KeyGen}(\text{crs}, b)$ , and  $(\text{crs}', \text{pk}', \text{sk}'_0, \text{sk}'_1) \leftarrow \text{SetupDec}(1^\lambda)$ .

Key-hiding follows from this property by a simple hybrid argument. Finally, we note that the output of the setup for the extraction mode of the DDH dual-mode scheme in [PVW08] is uniformly random; and the one from LWE is statistically close to uniform. We note that in the latter case, replacing it with a completely uniform string preserves perfect security.  $\square$

**OT from messy encryption.** Next, we show that the OT induced by the messy encryption scheme (similarly to [PVW08]) is receiver-extractable (as defined in Section 2.4).

**Construction B.3** (Receiver-Extractable 2-Message OT). Let  $\lambda$  be a security parameter, and  $(\text{SetupMessy}, \text{KeyGen}, \text{Encrypt}, \text{Decrypt}, \text{FindMessy})$  be a key-hiding messy encryption scheme. Define the following OT scheme  $\text{OT} = (\text{OT.Setup}, \text{OT}_1, \text{OT}_2, \text{OT.Receive})$ :

- $\text{OT.Setup}(1^\lambda)$ : Compute  $(\text{crs}, \text{td}) \leftarrow \text{SetupMessy}(1^\lambda)$  and output  $\text{crs}$ .
- $\text{OT}_1(\text{crs}, b; r)$ : Output  $M^{(1)} = \text{pk}$  where  $(\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(\text{crs}, b; r)$ .
- $\text{OT}_2(\text{crs}, M^{(1)}, m_0, m_1)$ : Output  $M^{(2)} = (\text{ct}_0, \text{ct}_1)$  where  $\text{ct}_0 \leftarrow \text{Encrypt}(\text{crs}, M^{(1)}, 0, m_0)$  and  $\text{ct}_1 \leftarrow \text{Encrypt}(\text{crs}, M^{(1)}, 1, m_1)$ .
- $\text{OT.Receive}(\text{crs}, M^{(2)}, b, r)$ : Parse  $M^{(2)}$  as  $(\text{ct}_0, \text{ct}_1)$ . Compute  $(\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(\text{crs}, b; r)$  and output  $m \leftarrow \text{Decrypt}(\text{crs}, \text{sk}, \text{ct}_b)$ .

**Claim B.4.** *Assuming  $(\text{SetupMessy}, \text{KeyGen}, \text{Encrypt}, \text{Decrypt}, \text{FindMessy})$  is a key-hiding messy encryption scheme, then Construction B.3 achieves correctness (Definition 2.15), receiver security (Definition 2.16) and extractability against malicious receivers (Definition 2.17).*

*Proof.* Correctness follows from completeness for the decryptable branch, and receiver security follows from key-hiding. For extractability against malicious receivers, we use the trapdoor identification of a messy branch property and define  $\text{OT.SetupExt}$  and  $\text{OT.Ext}$  as follows:

- $\text{OT.SetupExt}(1^\lambda)$ : Output  $(\text{crs}, \text{td}) \leftarrow \text{SetupMessy}(1^\lambda)$ .
- $\text{OT.Ext}(\text{td}, M^{(1)})$ : Compute  $b \leftarrow \text{FindMessy}(\text{td}, M^{(1)})$  and output  $1 - b$ .

In particular, with overwhelming probability over  $(\text{crs}, \text{td}) \leftarrow \text{SetupMessy}(1^\lambda)$ , then for all  $M^{(1)}$ , if  $\beta \leftarrow \text{FindMessy}(\text{crs}, M^{(1)})$ , then for all  $m_0, m_1$ ,  $\text{OT}_2(\text{crs}, M^{(1)}, m_0, m_1)$  statistically hides  $m_\beta$ .  $\square$

Combining the above with Theorem B.2, we obtain the following corollary:

**Corollary B.5.** *Assuming DDH (resp., LWE with polynomial modulus and polynomial modulus-to-noise ratio), there exists a receiver-extractable 2-message OT scheme in the common random string model.*

## C Lattice-Based ABE with Weak Function-Privacy

In this section, we recall the lattice-based ABE construction from [BGG<sup>+</sup>14] and show that a simple variant of it satisfies our notion of weak function-hiding (Definition 2.13). By Remark 5.4, this implies an AB-SFE scheme that satisfies weak message-hiding and strong key-hiding. This gives a construction of DV-NIZKs from the LWE assumption (via Construction 4.1). However, the scheme does not provide strong message-hiding, and thus, does not directly imply a MDV-NIZK via our constructions.

### C.1 Lattice Preliminaries

We begin by providing some basic background on lattice-based cryptography.

**Notation.** Throughout this section, we use bold uppercase letters (e.g.,  $\mathbf{A}, \mathbf{B}$ ) to denote matrices and bold lowercase letters (e.g.,  $\mathbf{u}, \mathbf{v}$ ) to denote vectors. We will use the infinity norm for vectors and matrices. Specifically, for a vector  $\mathbf{u}$ , we write  $\|\mathbf{u}\|$  to denote the maximum absolute value of an element in  $\mathbf{u}$  and  $\|\mathbf{A}\|$  to denote the maximum absolute value of an element in  $\mathbf{A}$ . For  $x \in \mathbb{Z}_q$ , we write  $\lfloor x \rfloor_2$  to denote the modular rounding function [BPR12] that outputs  $\lfloor 2/q \cdot x \rfloor \in \mathbb{Z}_2$ .

**Learning with errors.** The learning with errors (LWE) assumption was introduced by Regev [Reg05]. In the same work, Regev showed that solving LWE in the *average case* is as hard as (quantumly) approximating several standard lattice problems in the *worst case*. We state the assumption below.

**Definition C.1** (Learning with Errors [Reg05]). Fix a security parameter  $\lambda$  and integers  $n = n(\lambda)$ ,  $m = m(\lambda)$ ,  $q = q(\lambda)$ , and an error (or noise) distribution  $\chi = \chi(\lambda)$  over the integers. Then, the (decisional) learning with errors (LWE) assumption  $\text{LWE}_{n,m,q,\chi}$  states that for  $\mathbf{A} \xleftarrow{\mathbb{R}} \mathbb{Z}_q^{n \times m}$ ,  $\mathbf{s} \xleftarrow{\mathbb{R}} \mathbb{Z}_q^n$ ,  $\mathbf{e} \leftarrow \chi^m$ , and  $\mathbf{u} \xleftarrow{\mathbb{R}} \mathbb{Z}_q^m$ , the following two distributions are computationally indistinguishable:

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

When the error distribution  $\chi$  is  $B$ -bounded (oftentimes, a discrete Gaussian distribution), and under mild conditions on the modulus  $q$ , the  $\text{LWE}_{n,m,q,\chi}$  assumption is true assuming various worst-case lattice problems such as GapSVP and SIVP on an  $n$ -dimensional lattice are hard to approximate within a factor of  $\tilde{O}(n \cdot q/B)$  by a quantum algorithm [Reg05]. Similar reductions of LWE to the *classical* hardness of approximating worst-case lattice problems are also known [Pei09, ACPS09, MM11, MP12, BLP<sup>+</sup>13].

**The gadget matrix.** We use the gadget matrix [MP12]  $\mathbf{G} = \mathbf{g} \otimes \mathbf{I}_n \in \mathbb{Z}_q^{n \times n \lceil \log q \rceil}$ , where  $\mathbf{g} = (1, 2, 4, \dots, 2^{\lceil \log q \rceil - 1})$  is the powers-of-two vector. We define the bit-decomposition function  $\mathbf{G}^{-1}: \mathbb{Z}_q^{n \times m} \rightarrow \mathbb{Z}_q^{n \lceil \log q \rceil \times m}$  which expands each entry  $x \in \mathbb{Z}_q$  in the input matrix into a column of dimension  $\lceil \log q \rceil$  consisting of the bits of the binary representation of  $x$ . To simplify notation, we always assume that  $\mathbf{G}$  has width  $m = \Theta(n \log q)$ . Note that this is without loss of generality since we can always extend  $\mathbf{G}$  by appending zero columns. For any matrix  $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ , we have that  $\mathbf{G} \cdot \mathbf{G}^{-1}(\mathbf{A}) = \mathbf{A}$ .

**Lattice trapdoors and lattice sampling.** Although LWE is believed to be hard, with some auxiliary trapdoor information, the problem becomes easy. Lattice trapdoors have been used in a wide variety of context and are studied extensively in the literature [Ajt99, GPV08, AP09, CHKP10, ABB10, MP12, BGG<sup>+</sup>14, LW15]. For convenience, we will use the notion of gadget trapdoors from [MP12].



**Theorem C.2** (Gadget Trapdoors and Lattice Sampling [ABB10, MP12]). *Fix a security parameter  $\lambda$ , lattice parameters  $n, m, q, \beta$ , where  $m = O(n \log q)$ . Then, there exists a tuple of efficient algorithms (TrapGen, Invert, SampleLeft, SampleRight):*

- $\text{TrapGen}(1^\lambda) \rightarrow (\mathbf{A}, \mathbf{R})$ : *On input the security parameter  $\lambda$ , the trapdoor generation algorithm outputs a matrix  $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$  and a gadget trapdoor  $\mathbf{R} \in \mathbb{Z}_q^{m \times m}$ .*
- $\text{Invert}(\mathbf{A}, \mathbf{R}, \mathbf{v}) \rightarrow \mathbf{s}/\perp$ : *On input a matrix  $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ , a matrix  $\mathbf{R} \in \mathbb{Z}_q^{m \times m}$  (a gadget trapdoor for  $\mathbf{A}$ ), and a vector  $\mathbf{a} \in \mathbb{Z}_q^m$ , the inversion algorithm outputs a vector  $\mathbf{s} \in \mathbb{Z}_q^n$  or  $\perp$ .*
- $\text{SampleLeft}(\mathbf{A}, \mathbf{B}, \mathbf{R}, \mathbf{V}, \beta) \rightarrow \mathbf{U}$ : *On input matrices  $\mathbf{A}, \mathbf{B} \in \mathbb{Z}_q^{n \times m}$ , a matrix  $\mathbf{R} \in \mathbb{Z}_q^{m \times m}$  (a gadget trapdoor of  $\mathbf{A}$ ), a target matrix  $\mathbf{V} \in \mathbb{Z}_q^{n \times m}$ , and a norm bound  $\beta$ , SampleLeft returns a matrix  $\mathbf{U} \in \mathbb{Z}_q^{2m \times m}$ .*
- $\text{SampleRight}(\mathbf{A}, \mathbf{B}, \mathbf{T}, \mathbf{V}, \beta) \rightarrow \mathbf{U}$ : *On input matrices  $\mathbf{A}, \mathbf{B} \in \mathbb{Z}_q^{n \times m}$ , a matrix  $\mathbf{T} \in \mathbb{Z}_q^{m \times m}$ , a target matrix  $\mathbf{V} \in \mathbb{Z}_q^{n \times m}$ , and a norm bound  $\beta$ , SampleRight returns a matrix  $\mathbf{U} \in \mathbb{Z}_q^{2m \times m}$ .*

Moreover, the above algorithms satisfy the following properties:

- If we sample  $(\mathbf{A}, \mathbf{R}) \leftarrow \text{TrapGen}(1^\lambda)$ , then  $\mathbf{A} \stackrel{s}{\approx} \text{Uniform}(\mathbb{Z}_q^{n \times m})$ ,  $\mathbf{R}$  is full-rank,  $\mathbf{AR} = \mathbf{G}$ , and  $\|\mathbf{R}\| \leq \beta$ .
- If  $\mathbf{a} = \mathbf{s}^\top \mathbf{A} + \mathbf{e}^\top$  for some  $\mathbf{s}, \mathbf{e} \in \mathbb{Z}_q^n$  where  $\|\mathbf{e}\| \leq q/(4m \|\mathbf{R}\|)$  and  $\mathbf{AR} = \mathbf{G}$ , then

$$\Pr [\text{Invert}(\mathbf{A}, \mathbf{R}, \mathbf{a}) = \mathbf{s}] = 1.$$

- The SampleLeft and SampleRight algorithms satisfy the following properties. For any rank- $n$  matrices  $\mathbf{A}, \mathbf{B} \in \mathbb{Z}_q^{n \times m}$  and any target matrix  $\mathbf{V} \in \mathbb{Z}_q^{n \times m}$ , the following properties hold:
  - Let  $\mathbf{R} \in \mathbb{Z}_q^{m \times m}$  be any matrix satisfying  $\mathbf{AR} = \mathbf{G}$  and  $\|\mathbf{R}\| \cdot \omega(m\sqrt{\log m}) \leq \beta_2 \leq q$ . Then, for  $\mathbf{U}_0 \leftarrow \text{SampleLeft}(\mathbf{A}, \mathbf{B}, \mathbf{R}, \mathbf{V}, \beta_2)$ , we have that  $[\mathbf{A} \mid \mathbf{B}] \cdot \mathbf{U}_0 = \mathbf{V}$  and  $\|\mathbf{U}_0\| \leq \beta_2$ .
  - Let  $\mathbf{T} \in \mathbb{Z}_q^{m \times m}$  be any matrix satisfying  $\mathbf{AT} + y\mathbf{G} = \mathbf{B}$  for some  $0 \neq y \in \mathbb{Z}_q$  and  $\|\mathbf{T}\| \cdot \omega(m\sqrt{\log m}) \leq \beta_2 \leq q$ . Then, for  $\mathbf{U}_1 \leftarrow \text{SampleRight}(\mathbf{A}, \mathbf{B}, \mathbf{T}, \mathbf{V}, \beta_2)$ , we have that  $[\mathbf{A} \mid \mathbf{B}] \cdot \mathbf{U}_1 = \mathbf{V}$  and  $\|\mathbf{U}_1\| \leq \beta_2$ .
  - The distributions of  $\mathbf{U}_0, \mathbf{U}_1$  as defined above are statistically indistinguishable.

**Randomness extraction.** We also rely on a generalization of the leftover hash lemma [HILL99] due to Dodis et. al [DRS04]. Specifically, we use the following special case from [ABB10]:

**Theorem C.3** (Leftover Hash Lemma [HILL99, DRS04, ABB10]). *Suppose that  $m > (n+1) \log q + \omega(\log n)$  and that  $q > 2$  is prime. Take  $\mathbf{R} \stackrel{\mathbb{R}}{\leftarrow} \{\pm 1\}^{m \times k}$  where  $k = k(n)$  is polynomial in  $n$ . Sample  $\mathbf{A} \stackrel{\mathbb{R}}{\leftarrow} \mathbb{Z}_q^{n \times m}$  and  $\mathbf{B} \stackrel{\mathbb{R}}{\leftarrow} \mathbb{Z}_q^{n \times k}$ . Then, for all vectors  $\mathbf{e} \in \mathbb{Z}_q^m$ , the following two distributions are statistically indistinguishable:  $(\mathbf{A}, \mathbf{AR}, \mathbf{R}^\top \mathbf{e})$  and  $(\mathbf{A}, \mathbf{B}, \mathbf{R}^\top \mathbf{e})$ .*

**Matrix embeddings.** We review the matrix encoding scheme from Boneh et al. [BGG<sup>+</sup>14] that is the basis of their ABE scheme. At a high-level, a matrix encoding scheme allows one to embed a sequence of input bits  $x_1, \dots, x_\rho \in \{0, 1\}$  into matrices  $\mathbf{A}_1, \dots, \mathbf{A}_\rho \in \mathbb{Z}_q^{n \times m}$ . Moreover, it is possible to homomorphically evaluate any circuit (of *a priori* bounded depth) over the encoded input bits. The specific implementation of the homomorphic operations are non-essential to this work, so we just give the basic schematic that we need in the theorem below:

**Theorem C.4** (Matrix Embeddings [BGG<sup>+</sup>14]). *Fix a security parameter  $\lambda$ , and lattice parameters  $n, m, q$ . Then, there exist a pair of efficiently-computable algorithms ( $\text{Eval}_{\text{pk}}, \text{Eval}_{\text{ct}}$ ) with the following syntax:*

- $\text{Eval}_{\text{pk}}(C, \mathbf{A}_1, \dots, \mathbf{A}_\ell) \rightarrow \mathbf{A}_C$ : On input a circuit  $C: \{0, 1\}^\ell \rightarrow \{0, 1\}$ , and a set of matrices  $\mathbf{A}_1, \dots, \mathbf{A}_\ell \in \mathbb{Z}_q^{n \times m}$ , the  $\text{Eval}_{\text{pk}}$  algorithm outputs a matrix  $\mathbf{A}_C \in \mathbb{Z}_q^{n \times m}$ .
- $\text{Eval}_{\text{ct}}(C, x, \mathbf{A}_1, \dots, \mathbf{A}_\ell, \mathbf{a}_1, \dots, \mathbf{a}_\ell) \rightarrow \mathbf{a}_{C,x}$ : On input a circuit  $C: \{0, 1\}^\ell \rightarrow \{0, 1\}$ , an input  $x \in \{0, 1\}^\ell$ , a set of matrices  $\mathbf{A}_1, \dots, \mathbf{A}_\ell \in \mathbb{Z}_q^{n \times m}$ , and a set of vectors  $\mathbf{a}_1, \dots, \mathbf{a}_\ell \in \mathbb{Z}_q^m$ , the  $\text{Eval}_{\text{ct}}$  algorithm outputs a vector  $\mathbf{a}_{C,x} \in \mathbb{Z}_q^m$ .

Moreover, the algorithms ( $\text{Eval}_{\text{pk}}, \text{Eval}_{\text{ct}}$ ) satisfy the following properties. There exists a fixed function  $B(\beta, m, d) = \beta \cdot m^{O(d)}$  such that for all  $\mathbf{A}_1, \dots, \mathbf{A}_\ell \in \mathbb{Z}_q^{n \times m}$ ,  $x \in \{0, 1\}^\ell$ , and Boolean circuits  $C: \{0, 1\}^\ell \rightarrow \{0, 1\}$  of depth  $d$ , let  $\mathbf{A}_C := \text{Eval}_{\text{pk}}(C, \mathbf{A}_1, \dots, \mathbf{A}_\ell)$ . Then, the following hold:

- Let  $\mathbf{a}_1, \dots, \mathbf{a}_\ell \in \mathbb{Z}_q^m$  be vectors of the form

$$\mathbf{a}_i = \mathbf{s}^\top (\mathbf{A}_i + x_i \cdot \mathbf{G}) + \mathbf{e}_i^\top \quad \forall i \in [\ell],$$

for some vector  $\mathbf{s} \in \mathbb{Z}_q^n$  and where  $\|\mathbf{e}_i\| \leq \beta$  for all  $i \in [\ell]$ . Then, if we compute the vector  $\mathbf{a}_{C,x} \leftarrow \text{Eval}_{\text{ct}}(C, x, \mathbf{A}_1, \dots, \mathbf{A}_\ell, \mathbf{a}_1, \dots, \mathbf{a}_\ell)$ , we have

$$\mathbf{a}_{C,x} = \mathbf{s}^\top (\mathbf{A}_C + C(x) \cdot \mathbf{G}) + \mathbf{e}_{C,x}^\top,$$

for an error vector  $\|\mathbf{e}_{C,x}\| \leq B(\beta, m, d) = \beta \cdot m^{O(d)}$ .

- If there exist matrices  $\mathbf{R}_i \in \mathbb{Z}_q^{n \times m}$  where  $\mathbf{A}_i = \mathbf{A}\mathbf{R}_i - x_i\mathbf{G}$  and  $\|\mathbf{R}_i\| \leq \beta$  for all  $i \in [\ell]$ , then

$$\mathbf{A}_C = \mathbf{A}\mathbf{R}_C - C(x) \cdot \mathbf{G},$$

where  $\mathbf{R}_C$  can be efficiently computed from  $(C, x, \mathbf{A}, \mathbf{R}_1, \dots, \mathbf{R}_\ell)$  and  $\|\mathbf{R}_C\| \leq B(\beta, m, d) = \beta \cdot m^{O(d)}$ .

## C.2 Weak Function-Hiding ABE from Lattices

In this section, we describe a simple variant of the ABE scheme from Boneh et al. [BGG<sup>+</sup>14] where the decryption operation additionally recovers the encryption randomness and checks well-formedness of the ciphertext. This is conceptually very similar to our construction of strong key-hiding AB-SFE schemes from KDM-secure secret-key encryption. We recall the construction here:

**Construction C.5** (Weak Function-Hiding ABE [BGG<sup>+</sup>14, adapted]). Let  $\lambda$  be a security parameter and let  $(n, m, q, \chi)$  be lattice parameters, where  $\chi$  is a  $\beta$ -bounded distribution. Let  $\beta_1, \beta_2 < q$  be additional noise parameters. Let  $\mathcal{X} = \{0, 1\}^\ell$  be the attribute space and  $\mathcal{M} = \{0, 1\}$  be the message space. Let  $\mathcal{C}$  be a class of Boolean circuits of depth at most  $d$  from  $\mathcal{X} \rightarrow \{0, 1\}$ . In the following, we will use the convention that a secret key  $\text{sk}_C$  for a circuit  $C$  should successfully decrypt a ciphertext  $\text{ct}$  with attribute  $x$  if  $C(x) = 0$ . Then, the ABE scheme  $\Pi_{\text{ABE}} = (\text{ABE.Setup}, \text{ABE.KeyGen}, \text{ABE.Encrypt}, \text{ABE.Decrypt})$  with attribute space  $\mathcal{X}$ , message space  $\mathcal{M}$ , and function class  $\mathcal{C}$  is defined as follows:

- **ABE.Setup**( $1^\lambda$ ): On input the security parameter  $\lambda$ , the setup algorithm samples  $(\mathbf{A}, \mathbf{R}) \leftarrow \text{TrapGen}(1^\lambda)$ ,  $\mathbf{A}_1, \dots, \mathbf{A}_\ell \stackrel{\mathcal{R}}{\leftarrow} \mathbb{Z}_q^{n \times m}$  and  $\mathbf{u} \leftarrow \mathbb{Z}_q^n$ . It outputs the public parameters  $\text{pp} = (\mathbf{A}, \mathbf{A}_1, \dots, \mathbf{A}_\ell, \mathbf{u})$  and the master secret key  $\text{msk} = \mathbf{R}$ .
- **ABE.KeyGen**( $\text{pp}, \text{msk}, C$ ): On input the public parameters  $\text{pp} = (\mathbf{A}, \mathbf{A}_1, \dots, \mathbf{A}_\ell, \mathbf{u})$ , the master secret key  $\text{msk} = \mathbf{R}$  and a circuit  $C: \{0, 1\}^\ell \rightarrow \{0, 1\}$ , the key-generation algorithm computes  $\mathbf{A}_C \leftarrow \text{Eval}_{\text{pk}}(C, \mathbf{A}_1, \dots, \mathbf{A}_\ell)$  and samples  $\mathbf{R}_C \leftarrow \text{SampleLeft}(\mathbf{A}, \mathbf{A}_C, \mathbf{R}, \mathbf{G}, \beta_2)$ . It outputs the secret key  $\text{sk} = (C, \mathbf{R}_C)$ .
- **ABE.Encrypt**( $\text{pp}, x, m$ ): On input the public parameters  $\text{pp} = (\mathbf{A}, \mathbf{A}_1, \dots, \mathbf{A}_\ell, \mathbf{u})$ , an attribute  $x \in \{0, 1\}^\ell$ , and a message  $m \in \{0, 1\}$ , the encryption algorithm samples  $\mathbf{s} \stackrel{\mathcal{R}}{\leftarrow} \mathbb{Z}_q^n$ ,  $\mathbf{e} \leftarrow \chi^m$ ,  $e_u \leftarrow \chi$ , and  $\mathbf{R}_1, \dots, \mathbf{R}_\ell \stackrel{\mathcal{R}}{\leftarrow} \{\pm 1\}^{m \times m}$ . It computes  $\mathbf{a} \leftarrow \mathbf{s}^\top \mathbf{A} + \mathbf{e}^\top$ ,  $\mathbf{a}_i \leftarrow \mathbf{s}^\top (\mathbf{A}_i + x_i \cdot \mathbf{G}) + \mathbf{e}^\top \mathbf{R}_i$  for each  $i \in [\ell]$ , and  $a_u \leftarrow \mathbf{s}^\top \mathbf{u} + m \cdot \lfloor q/2 \rfloor + e_u$ , and outputs the ciphertext  $\text{ct} = (x, \mathbf{a}, \mathbf{a}_1, \dots, \mathbf{a}_\ell, a_u)$ .
- **ABE.Decrypt**( $\text{pp}, \text{sk}, \text{ct}$ ): On input the public parameters  $\text{pp} = (\mathbf{A}, \mathbf{A}_1, \dots, \mathbf{A}_\ell, \mathbf{u})$ , a secret key  $\text{sk}$ , and a ciphertext  $\text{ct} = (x, \mathbf{a}, \mathbf{a}_1, \dots, \mathbf{a}_\ell, a_u)$ , the decryption algorithm first uses the trapdoor (in the secret key  $\text{sk}$ ) to try and recover part of the encryption randomness  $\mathbf{s} \in \mathbb{Z}_q^n$ :
  - If  $\text{sk} = \mathbf{R} \in \mathbb{Z}_q^{m \times m}$  (i.e.,  $\text{sk}$  has the structure of the master secret key), then the decryption algorithm computes  $\mathbf{s} \leftarrow \text{Invert}(\mathbf{A}, \mathbf{R}, \mathbf{a})$ .
  - If  $\text{sk} = (C, \mathbf{R}_C)$  for some  $C \in \mathcal{C}$  and  $\mathbf{R}_C \in \mathbb{Z}_q^{m \times m}$ , (i.e.,  $\text{sk}$  has the structure of a function key), then the decryption algorithm first checks that  $C(x) = 0$ . If not, then the algorithm outputs  $\perp$ . Otherwise, it computes  $\mathbf{A}_C \leftarrow \text{Eval}_{\text{pk}}(C, \mathbf{A}_1, \dots, \mathbf{A}_\ell)$ ,  $\mathbf{a}_{C,x} \leftarrow \text{Eval}_{\text{ct}}(C, x, \mathbf{A}_1, \dots, \mathbf{A}_\ell, \mathbf{a}_1, \dots, \mathbf{a}_\ell)$  and finally  $\mathbf{s} \leftarrow \text{Invert}([\mathbf{A} \mid \mathbf{A}_C], \mathbf{R}_C, [\mathbf{a} \mid \mathbf{a}_{C,x}])$ .
  - If  $\text{sk}$  does not have one of these two forms, the algorithm outputs  $\perp$ .

If the candidate key  $\mathbf{s} = \perp$ , then the algorithm outputs  $\perp$ . Otherwise, it sets  $m \leftarrow \lfloor a_u - \mathbf{s}^\top \mathbf{u} \rfloor_2$ . Next, the decryption algorithm applies the following verification procedure:

1. Compute  $\mathbf{e}^\top \leftarrow \mathbf{a} - \mathbf{s}^\top \mathbf{A}$ ,  $\mathbf{e}_i^\top \leftarrow \mathbf{a}_i - \mathbf{s}^\top (\mathbf{A}_i + x_i \mathbf{G})$ , and  $e_u \leftarrow a_u - \mathbf{s}^\top \mathbf{u} - m \cdot \lfloor q/2 \rfloor$ .
2. Check that  $\|\mathbf{e}\| \leq \beta_1$ ,  $\|\mathbf{e}_i\| \leq \beta_1$  for all  $i \in [\ell]$ , and  $|e_u| \leq \beta_1$ . If any check fails, output  $\perp$ .
3. If all checks pass, output the pair  $(x, m)$ .

**Correctness and security analysis.** Correctness and security of this scheme follow by essentially the same arguments from [BGG<sup>+</sup>14, §4]. To show that the scheme satisfies weak function-privacy, we first show that it in fact satisfies a stronger notion of *consistency*: namely, that all honestly-generated decryption keys behave identically to the master secret key when decrypting valid ciphertexts (satisfying the predicate). As we show below, any consistent ABE scheme immediately satisfies our notion of weak function hiding.

**Definition C.6** (Consistency). Let ABE be an ABE scheme, and let  $t = t(\lambda)$  be a bound on the length of ciphertexts in ABE. We say that ABE is *consistent* if for  $(\text{pp}, \text{msk}) \leftarrow \text{ABE.Setup}(1^\lambda)$ , and all strings  $\text{ct} \in \{0, 1\}^t$ , the following properties hold:

- For all  $f \in \mathcal{F}$ , if  $\text{sk}_f \leftarrow \text{ABE.KeyGen}(\text{pp}, \text{msk}, f)$  and  $\text{ABE.Decrypt}(\text{pp}, \text{sk}_f, \text{ct}) = (x, m)$ , then  $f(x) = 1$ .

- If  $\text{ABE.Decrypt}(\text{pp}, \text{sk}, \text{ct}) = (x, m)$  where  $\text{sk} = \text{msk}$  or  $\text{sk}$  is output by  $\text{ABE.KeyGen}(\text{pp}, \text{msk}, f)$  for some function  $f \in \mathcal{F}$ , then  $\Pr[\text{ABE.Decrypt}(\text{pp}, \text{msk}, \text{ct}) = (x, m)] = 1$  and for all  $g \in \mathcal{F}$  where  $g(x) = 1$ ,

$$\Pr[\text{sk}_g \leftarrow \text{ABE.KeyGen}(\text{pp}, \text{msk}, g) : \text{ABE.Decrypt}(\text{pp}, \text{sk}_g, \text{ct}) = (x, m)] = 1.$$

**Lemma C.7** (Consistency Implies Weak Function Hiding). *Let ABE be a consistent ABE scheme. Then, ABE satisfies weak function hiding.*

*Proof.* We begin by constructing a weak function hiding simulator  $\mathcal{S}^{f(\cdot)}(\text{pp}, \text{msk}, \text{ct})$  as follows:

1. On input the public parameters  $\text{pp}$ , the master secret key  $\text{msk}$ , and a ciphertext  $\text{ct}$ , the simulator computes  $(x, m) \leftarrow \text{ABE.Decrypt}(\text{pp}, \text{msk}, \text{ct})$ . If  $\text{ABE.Decrypt}(\text{pp}, \text{msk}, \text{ct})$  outputs  $\perp$ , then the simulator also outputs  $\perp$ .
2. Otherwise, the simulator makes a single oracle query to  $f$  on input  $x$ . If  $f(x) = 1$ , then the simulator outputs  $(x, m)$ , and otherwise, it outputs  $\perp$ .

To complete the proof, we show that the simulator *perfectly* simulates the real distribution. Let  $t = t(\lambda)$  be the length of a ciphertext in ABE. Then, take any function  $f \in \mathcal{F}$ , any string  $\text{ct} \in \{0, 1\}^t$ ,  $(\text{pp}, \text{msk}) \leftarrow \text{ABE.Setup}(1^\lambda)$ , and  $\text{sk}_f \leftarrow \text{ABE.KeyGen}(\text{pp}, \text{msk}, f)$ . We consider two possibilities:

- Suppose  $\text{ABE.Decrypt}(\text{pp}, \text{sk}_f, \text{ct}) = (x, m)$  for some  $x \in \mathcal{X}$  and  $m \in \mathcal{M}$ . By consistency of ABE, this means that  $f(x) = 1$  and  $\text{ABE.Decrypt}(\text{pp}, \text{msk}, \text{ct}) = (x, m)$ . By construction, the simulator outputs  $(x, m)$  in this case.
- Suppose  $\text{ABE.Decrypt}(\text{pp}, \text{sk}_f, \text{ct}) = \perp$ . If  $\text{ABE.Decrypt}(\text{pp}, \text{msk}, \text{ct}) = \perp$ , then the simulator also outputs  $\perp$ . Suppose instead that  $\text{ABE.Decrypt}(\text{pp}, \text{msk}, \text{ct}) = (x, m)$  for some  $x \in \mathcal{X}$  and  $m \in \mathcal{M}$ . We consider two possibilities. If  $f(x) = 1$ , then by consistency of ABE,  $\text{ABE.Decrypt}(\text{pp}, \text{sk}_f, \text{ct}) = (x, m)$ , which is a contradiction. Alternatively if  $f(x) = 0$ , then the simulator output  $\perp$ , which matches the behavior of the real decryption algorithm.

Since the output of the simulator on every ciphertext  $\text{ct} \in \{0, 1\}^t$  is distributed identically as the output of the real decryption algorithm, weak function hiding holds.  $\square$

**Theorem C.8** (Consistency). *Let ABE be the ABE scheme from Construction C.5. If  $4m\beta\beta_1 < q$  and  $4m\beta_2 \cdot B(\beta_1, m, d) < q$  where  $B(\beta_1, m, d) = \beta_1 \cdot m^{O(d)}$  is the bound from Theorem C.4, then ABE is consistent (and thus, is also weak function hiding).*

*Proof.* We begin by showing the following lemma on the statistical properties of ABE:

**Lemma C.9.** *Take  $(\text{pp}, \text{msk}) \leftarrow \text{ABE.Setup}(1^\lambda)$  where  $\text{pp} = (\mathbf{A}, \mathbf{A}_1, \dots, \mathbf{A}_\ell, \mathbf{u})$  and  $\text{msk} = \mathbf{R}$ . Take any  $\mathbf{a}, \mathbf{a}_1, \dots, \mathbf{a}_\ell \in \mathbb{Z}_q^m$  and  $a_u \in \mathbb{Z}_q$  such that there exist  $\mathbf{s} \in \mathbb{Z}_q^n$ ,  $x \in \{0, 1\}^\ell$ ,  $\mathbf{e}, \mathbf{e}_1, \dots, \mathbf{e}_\ell \in \mathbb{Z}_q^m$ , and  $e_u \in \mathbb{Z}_q$  with the following properties:*

- $\|\mathbf{e}\|, \|\mathbf{e}_1\|, \dots, \|\mathbf{e}_\ell\|, |e_u| \leq \beta_1$ .
- $\mathbf{a} = \mathbf{s}^\top \mathbf{A} + \mathbf{e}^\top$ ,  $\mathbf{a}_i = \mathbf{s}^\top (\mathbf{A}_i + x_i \mathbf{G}) + \mathbf{e}_i^\top$  for all  $i \in [\ell]$ , and  $a_u = \mathbf{s}^\top \mathbf{u} + e_u + m \cdot \lfloor q/2 \rfloor$  for some  $m \in \{0, 1\}$ .

Let  $\text{ct} = (x, \mathbf{a}, \mathbf{a}_1, \dots, \mathbf{a}_\ell, a_u)$ . If  $4m\beta\beta_1 < q$  and  $4m\beta_2 \cdot B(\beta_1, m, d) < q$  where  $B(\beta_1, m, d) = \beta_1 \cdot m^{O(d)}$  is the bound from Theorem C.4, then the following hold:

- $\Pr [\text{ABE.Decrypt}(\text{pp}, \text{msk}, \text{ct}) = (x, m)] = 1$ .
- For any  $C \in \mathcal{C}$  where  $C(x) = 0$ ,

$$\Pr [\text{ABE.Decrypt}(\text{pp}, \text{ABE.KeyGen}(\text{pp}, \text{msk}, C), \text{ct}) = (x, m)] = 1.$$

*Proof.* We first analyze the randomness recovery step in the ABE.Decrypt function:

- Since  $(\mathbf{A}, \mathbf{R})$  is sampled using TrapGen, by Theorem C.2, we have that  $\|\mathbf{R}\| \leq \beta$ . Next, by assumption,  $\|\mathbf{e}\| \leq \beta_1 < q/(4m\beta)$ , so we can again appeal to Theorem C.2 and conclude that  $\text{Invert}(\mathbf{A}, \mathbf{R}, \mathbf{a}) = \mathbf{s}$ .
- Take  $C \in \mathcal{C}$  where  $C(x) = 0$  and let  $\text{sk}_C = (C, \mathbf{R}_C) \leftarrow \text{KeyGen}(\text{pp}, \text{msk}, C)$ ,  $\mathbf{A}_C \leftarrow \text{Eval}_{\text{pk}}(C, \mathbf{A}_1, \dots, \mathbf{A}_\ell)$ ,  $\mathbf{a}_{C,x} \leftarrow \text{Eval}_{\text{ct}}(C, x, \mathbf{A}_1, \dots, \mathbf{A}_\ell, \mathbf{a}_1, \dots, \mathbf{a}_\ell)$ . By definition of KeyGen, we have that  $\mathbf{R}_C \leftarrow \text{SampleLeft}(\mathbf{A}, \mathbf{A}_C, \mathbf{R}, \mathbf{G}, \beta_2)$ . By Theorem C.2  $[\mathbf{A} \mid \mathbf{A}_C]\mathbf{R}_C = \mathbf{G}$  and moreover,  $\|\mathbf{R}_C\| \leq \beta_2$ . In addition, by Theorem C.4 and the fact that  $C(x) = 0$ , we have that  $\mathbf{a}_{C,x} = \mathbf{s}^\top \mathbf{A}_C + \mathbf{e}_{C,x}^\top$ , where  $\|\mathbf{e}_{C,x}\| \leq B(\beta_1, m, d)$ . This means that  $[\mathbf{a} \mid \mathbf{a}_{C,x}] = \mathbf{s}^\top [\mathbf{A} \mid \mathbf{A}_C] + [\mathbf{e}^\top \mid \mathbf{e}_{C,x}^\top]^\top$ . Since  $4m\beta_2 \cdot B(\beta_1, m, d) < q$ ,

$$\|\mathbf{e}_{C,x}\| \leq B(\beta_1, m, d) \leq q/(4m\beta_2) \leq q/(4m\|\mathbf{R}_C\|).$$

We now appeal to Theorem C.2 and conclude that  $\text{Invert}([\mathbf{A} \mid \mathbf{A}_C], \mathbf{R}_C, [\mathbf{a} \mid \mathbf{a}_{C,x}]) = \mathbf{s}$ .

Next,  $a_u - \mathbf{s}^\top \mathbf{u} = m \cdot \lfloor q/2 \rfloor + e_u$  and since  $|e_u| \leq \beta_1 < q/4$ , it follows that  $\lfloor a_u - \mathbf{s}^\top \mathbf{u} \rfloor_2 = m$ . Finally, by assumption, all of the error terms are bounded by  $\beta_1$  by assumption. Thus,  $\text{ABE.Decrypt}(\text{pp}, \text{msk}, \text{ct}) = (x, m)$  and  $\text{ABE.Decrypt}(\text{pp}, \text{sk}_C, \text{ct}) = (x, m)$ .  $\square$

To see that ABE is consistent, take  $(\text{pp}, \text{msk}) \leftarrow \text{ABE.Setup}(1^\lambda)$ , and consider any ciphertext  $\text{ct} = (x, \mathbf{a}, \mathbf{a}_1, \dots, \mathbf{a}_\ell, a_u)$  where  $x \in \{0, 1\}^\ell$ ,  $\mathbf{a}, \mathbf{a}_1, \dots, \mathbf{a}_\ell \in \mathbb{Z}_q^m$ , and  $a_u \in \mathbb{Z}_q$ . Suppose that  $\text{ABE.Decrypt}(\text{pp}, \text{sk}, \text{ct}) = (x, m)$  for some string  $\text{sk}$  (could be the master secret key  $\text{msk}$ , an key output by  $\text{ABE.KeyGen}$ , or even an arbitrary string). By construction of the decryption function, this is only possible if there exists  $\mathbf{s} \in \mathbb{Z}_q^n$ ,  $\mathbf{e}, \mathbf{e}_1, \dots, \mathbf{e}_\ell \in \mathbb{Z}_q^m$  and  $e_u \in \mathbb{Z}_q$  such that  $\mathbf{a} = \mathbf{s}^\top \mathbf{A} + \mathbf{e}^\top$ ,  $\mathbf{a}_i = \mathbf{s}^\top (\mathbf{A}_i + x_i \mathbf{G}) + \mathbf{e}_i^\top$  for all  $i \in [\ell]$ , and  $a_u = \mathbf{s}^\top \mathbf{u} + m \cdot \lfloor q/2 \rfloor + e_u$ , and where  $\|\mathbf{e}\|, \|\mathbf{e}_i\|, |e_u| \leq \beta_1$  for all  $i \in [\ell]$ . Consistency now follows by Lemma C.9.  $\square$

## D Strong Key-Hiding AB-SFE from PKE and DV-NIZK

In this section, we prove a converse to Construction 4.1, showing that (assuming the existence of public-key encryption) AB-SFE is actually *necessary* to construct reusable DV-NIZK arguments. This establishes an equivalence between (strong key-hiding) AB-SFE and (adaptively-sound) reusable DV-NIZK + PKE.

**Theorem D.1** (AB-SFE from PKE and DV-NIZK). *Suppose that public-key encryption exists, and that adaptively-sound reusable DV-NIZKs exist. Then, there exists an AB-SFE scheme that is strong key-hiding and weak message-hiding.*

*Proof.* To prove the theorem, we make use of Remark 5.4; namely, it suffices to construct a single-key ABE scheme satisfying weak function-hiding (Definition 2.13). Constructing such a scheme is intuitively simple: augment any single-key ABE scheme (which follows from any public-key encryption scheme [SS10, GVW12]) with a “DV-NIZK argument of well-formedness.” This is conceptually very similar to the Naor-Yung construction of CCA-secure encryption from CPA-secure encryption [NY90], except our objective is showing function-hiding rather than CCA-security. More formally, we construct such a weak function-hiding ABE scheme as follows:

**Construction D.2** (Weak Function-Hiding ABE). Let  $\text{ABE} = (\text{ABE.Setup}, \text{ABE.KeyGen}, \text{ABE.Encrypt}, \text{ABE.Decrypt})$  denote a single-key ABE scheme, and let  $\text{dvNIZK} = (\text{dvNIZK.Setup}, \text{dvNIZK.KeyGen}, \text{dvNIZK.Prove}, \text{dvNIZK.Verify})$  denote an adaptively-sound reusable DV-NIZK argument system. We construct a single-key ABE scheme  $\text{Aug}$  as follows:

- $\text{Aug.Setup}(1^\lambda)$ : Sample  $(\text{ABE.pp}, \text{ABE.msk}) \leftarrow \text{ABE.Setup}(1^\lambda)$ ,  $\text{crs} \leftarrow \text{dvNIZK.Setup}(1^\lambda)$ , and  $(\text{pk}, \text{sk}) \leftarrow \text{dvNIZK.KeyGen}(\text{crs})$ . Output the public parameters  $\text{pp} = (\text{ABE.pp}, \text{crs}, \text{pk})$  and the master secret key  $\text{msk} = (\text{ABE.msk}, \text{sk})$ .
- $\text{Aug.KeyGen}(\text{msk}, f)$ : Parse  $\text{msk} = (\text{ABE.msk}, \text{sk})$ , compute  $\text{ABE.sk}_f \leftarrow \text{ABE.KeyGen}(\text{ABE.msk}, f)$ , and output  $\text{sk}_f = (\text{ABE.sk}_f, \text{sk})$ .
- $\text{Aug.Encrypt}(\text{pp}, x, m)$ : On input the public parameters  $\text{pp} = (\text{ABE.pp}, \text{crs}, \text{pk})$ , an attribute  $x$ , and a message  $m$ , proceed as follows:
  - Compute  $\text{ABE.ct} \leftarrow \text{ABE.Encrypt}(\text{ABE.pp}, x, m; \rho)$  for  $\rho$  sampled uniformly at random.
  - Compute  $\pi \leftarrow \text{dvNIZK.Prove}(\text{crs}, \text{pk}, \mathbf{x}, (m, \rho))$ , where  $\mathbf{x} = (\text{ABE.pp}, \text{ABE.ct}, x)$  is the statement
$$\exists (m', \rho') : \text{ABE.ct} = \text{ABE.Encrypt}(\text{ABE.pp}, x, m'; \rho'), \quad (\text{D.1})$$
and  $(m, \rho)$  is the witness for this statement.
  - Output the ciphertext  $\text{ct} = (x, \text{ABE.ct}, \pi)$ .
- $\text{Aug.Decrypt}(\text{pp}, \text{sk}_f, \text{ct})$ : On input the public parameters  $\text{pp} = (\text{ABE.pp}, \text{crs}, \text{pk})$ , the decryption key  $\text{sk}_f = (\text{ABE.sk}_f, \text{sk})$ , and the ciphertext  $\text{ct} = (x, \text{ABE.ct}, \pi)$ , proceed as follows:
  - Call  $\text{dvNIZK.Verify}(\text{crs}, \text{sk}, \mathbf{x}, \pi)$  where  $\mathbf{x} = (\text{ABE.pp}, \text{ABE.ct}, x)$  is the statement from Eq. (D.1). Output  $\perp$  if this verification fails.
  - Output  $\text{ABE.Decrypt}(\text{ABE.pp}, \text{ABE.sk}_f, \text{ABE.ct})$ .

Correctness of this ABE scheme follows by completeness of the underlying DV-NIZK and correctness of the underlying single-key ABE. We now show that this scheme is semantically secure and satisfies weak function-hiding.

**Lemma D.3.** *If ABE is semantically secure and dvNIZK is zero-knowledge, then the scheme Aug is semantically secure.*

*Proof.* We want to prove that for every function  $f$  and attribute  $x$  such that  $f(x) = 0$ , we have that

$$(\text{pp}, \text{sk}_f, \text{Aug.Encrypt}(\text{pp}, x, 0)) \stackrel{c}{\approx} (\text{pp}, \text{sk}_f, \text{Aug.Encrypt}(\text{pp}, x, 1)),$$

where  $(\mathbf{pp}, \mathbf{sk}_f)$  are generated according to the honest setup and key-generation algorithms. In our analysis, we will rely on the zero-knowledge simulator  $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$  for  $\text{dvNIZK}$ . First, we define a hybrid setup algorithm  $\text{HybridSetup}(1^\lambda)$  that outputs  $\widetilde{\mathbf{pp}} = (\text{ABE.pp}, \widetilde{\text{crs}}, \widetilde{\text{pk}})$  and  $\widetilde{\text{msk}} = (\text{ABE.msk}, \widetilde{\text{sk}})$  for  $(\text{st}_{\mathcal{S}}, \widetilde{\text{crs}}, \widetilde{\text{pk}}, \widetilde{\text{sk}}) \leftarrow \mathcal{S}_1(1^\lambda)$ . We also define hybrid encryption algorithm  $\text{HybridEncrypt}(\mathbf{pp}, x, m)$  defined as follows:

- Interpret  $\mathbf{pp} = (\text{ABE.pp}, \text{crs}, \text{pk})$ .
- Compute  $\text{ABE.ct} = \text{ABE.Encrypt}(\text{ABE.pp}, x, m; \rho)$  for  $\rho$  sampled uniformly at random.
- Compute  $\pi \leftarrow \mathcal{S}_2(\text{st}_{\mathcal{S}}, \mathbf{x})$ , where  $\mathbf{x} = (\text{ABE.pp}, \text{ABE.ct}, x)$  is the statement from Eq. (D.1).
- Output  $\text{ct} = (x, \text{ABE.ct}, \pi)$ .

Now, by zero-knowledge of  $\text{dvNIZK}$ , we have that for all  $(f, x, m)$ ,

$$(\mathbf{pp}, \mathbf{sk}_f, \text{Aug.Encrypt}(\mathbf{pp}, x, m)) \stackrel{c}{\approx} (\widetilde{\mathbf{pp}}, \widetilde{\mathbf{sk}}_f, \text{Aug.HybridEncrypt}(\widetilde{\mathbf{pp}}, x, m)),$$

where  $\widetilde{\mathbf{sk}}_f = (\text{ABE.sk}_f, \widetilde{\text{sk}})$ . Moreover, we have that

$$(\widetilde{\mathbf{pp}}, \widetilde{\mathbf{sk}}_f, \text{Aug.HybridEncrypt}(\widetilde{\mathbf{pp}}, x, 0)) \stackrel{c}{\approx} (\widetilde{\mathbf{pp}}, \widetilde{\mathbf{sk}}_f, \text{Aug.HybridEncrypt}(\widetilde{\mathbf{pp}}, x, 1))$$

by the semantic security of ABE (as  $\text{HybridEncrypt}$  ciphertexts can be produced in a black-box way from ABE ciphertexts), and the claim follows.  $\square$

**Lemma D.4.** *If  $\text{dvNIZK}$  is adaptively-sound, then  $\text{Aug}$  is weakly function-hiding.*

*Proof.* We define a weak function hiding simulator  $\mathcal{S}^{f(\cdot)}(\mathbf{pp}, \text{msk}, \text{ct})$  as follows:

1. On input the public parameters  $\mathbf{pp} = (\text{ABE.pp}, \text{crs}, \text{pk})$ , the master secret key  $\text{msk} = (\text{ABE.msk}, \text{sk})$ , and a ciphertext  $\text{ct} = (x, \text{ABE.ct}, \pi)$ , the simulator calls  $\text{dvNIZK.Verify}(\text{crs}, \text{sk}, \mathbf{x}, \pi)$  as in the honest decryption algorithm, and outputs  $\perp$  if this verification fails.
2. Otherwise, the simulator makes a single oracle query to  $f$  on input  $x$ . If  $f(x) = 1$ , then the simulator computes and outputs  $\text{ABE.Decrypt}(\text{ABE.pp}, \text{ABE.msk}, \text{ct})$ .

We now show that oracle access to this simulator is computationally indistinguishable from oracle access to the honest decryption algorithm. Suppose that for some function  $f$ , some PPT adversary  $\mathcal{A}^{\mathcal{O}}$  can distinguish between oracle access to the honest decryption algorithm  $\text{Aug.Decrypt}(\mathbf{pp}, \mathbf{sk}_f, \cdot)$  with key  $\mathbf{sk}_f$  and oracle access to  $\mathcal{S}^{f(\cdot)}(\mathbf{pp}, \text{msk}, \cdot)$ . This means that (with non-negligible probability)  $\mathcal{A}$  must make an oracle query on a ciphertext  $\text{ct} = (x, \text{ABE.ct}, \pi)$  of the following form:

- The statement  $\mathbf{x} = (\text{ABE.pp}, \text{ABE.ct}, x)$  is false, and
- The DV-NIZK verifier accepts the proof  $(\mathbf{x}, \pi)$ .

This is because for every ciphertext  $\text{ct} = (x, \text{ABE.ct}, \pi)$  associated with a *true* statement  $\mathbf{x}$ , the outputs of  $\text{ABE.Decrypt}(\text{ABE.pp}, \text{ABE.sk}_f, \text{ABE.ct})$  and  $\text{ABE.Decrypt}(\text{ABE.pp}, \text{ABE.msk}, \text{ABE.ct})$  are identical (provided that  $f(x) = 1$ , which is an enforced condition) by correctness of ABE. Moreover, for every query  $(x, \text{ABE.ct}, \pi)$  on which the proof is rejected, the two oracles again behave identically.

We use the above observation to break adaptive soundness of  $\text{dvNIZK}$ . Namely, we define an adversary  $\mathcal{B}$  breaking the soundness of  $\text{dvNIZK}$  as follows:

1. On input  $(\text{crs}, \text{pk})$ , algorithm  $\mathcal{B}$  samples  $(\text{ABE.pp}, \text{ABE.msk}) \leftarrow \text{ABE.Setup}(1^\lambda)$ ,  $\text{ABE.sk}_f \leftarrow \text{ABE.KeyGen}(\text{ABE.msk}, f)$ , and constructs  $(\text{pp}, \text{msk})$  as in Construction D.2.
2. Choose a random index  $i^* \xleftarrow{\text{R}} [Q]$ , where  $Q$  is a bound on the number of queries  $\mathcal{A}$  makes. Let  $\text{ct}_i = (x_i, \text{ABE.ct}_i, \pi_i)$  be the  $i^{\text{th}}$  decryption query  $\mathcal{A}$  makes.
3. Start running algorithm  $\mathcal{A}$  on input  $\text{pp} = (\text{ABE.pp}, \text{crs}, \text{pk})$ . Whenever  $\mathcal{A}$  makes a decryption query on a ciphertext  $\text{ct}_i = (x_i, \text{ABE.ct}_i, \pi_i)$ , algorithm  $\mathcal{B}$  makes a verification query on the statement  $\mathbf{x}_i = (\text{ABE.pp}, \text{ABE.ct}_i, x_i)$  and proof  $\pi_i$ . If the proof fails to verify, algorithm  $\mathcal{B}$  replies with  $\perp$ , and otherwise with  $\text{ABE.Decrypt}(\text{ABE.pp}, \text{ABE.sk}_f, \text{ABE.ct}_i)$ , exactly as in the real scheme.
4. At the end of the experiment, output the statement  $\mathbf{x}_{i^*} = (\text{ABE.pp}, \text{ABE.ct}_{i^*}, x_{i^*})$  and  $\pi_{i^*}$  where  $\text{ct}_{i^*} = (x_{i^*}, \text{ABE.ct}_{i^*}, \pi_{i^*})$ .

By construction,  $\mathcal{B}$  will output an accepting proof on a false statement with non-negligible probability, contradicting the adaptive soundness of  $\text{dvNIZK}$ . Thus, we conclude that if  $\text{dvNIZK}$  is adaptively-sound, then  $\text{Aug}$  is weakly function-hiding.  $\square$

Thus, the augmented ABE scheme from Construction D.2 is a single-key ABE scheme that satisfies weak function hiding. By Remark 5.4, this yields an AB-SFE scheme that satisfies weak message-hiding and strong key-hiding.  $\square$