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 plain *learning with errors (LWE)* assumption or the Diffie-Hellman (CDH/DDH) 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. In this work, we give a new construction using generic primitives that can be instantiated under CDH or LWE.

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. In this work, we give a new construction using generic primitives that can be instantiated under DDH or LWE.

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 (NIZK) proof, which consists of a single message sent by the prover to the verifier. Although such

*Email: 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.

‡Email: rothblum@cs.technion.ac.il. Supported in part by the Israeli Science Foundation (Grant No. 1262/18).

§Email: 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. Part of this work was done while visiting the Technion.

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+01, Gol11], the Diffie-Hellman assumption over bilinear groups [CHK03, GOS06], optimal hardness of the learning with errors (LWE) assumption[1] [CCH+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) and the plain learning-with-errors (LWE) 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,[2] 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, KNYY19, 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.

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.

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.

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 recently introduced primitive called a *hinting PRG* [KW18b]. Both

[1] This means that no polynomial-time attacker can break LWE with any probability better than random guessing.
[2] The public verifiability of traditional NIZKs immediately implies reusability.

components can be instantiated under the CDH or LWE 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 hinting PRGs, there exist reusable designated-verifier NIZK arguments for* NP*. In particular, there exist reusable DV-NIZKs under either the CDH or the LWE assumption.*

We then show how to construct reusable MDV-NIZKs from any (receiver-extractable) 2-round oblivious transfer (OT) in the common random string model and a hinting PRG. This yields instantiations of MDV-NIZKs under the DDH or LWE assumptions. This yields the following theorem:

Theorem 1.2 (informal)**.** *Assuming the existence of "receiver-extractable 2-message OT" (see Definition 2.14) and hinting PRGs, there exist reusable* malicious *designated-verifier NIZK arguments for* NP*. In particular, there exist reusable MDV-NIZKs under either the DDH or the LWE assumption.*

More generally, we give a compiler converting any Σ-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$^+$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 single-key ABE (i.e., one that satisfies our function-hiding property) from an arbitrary public-key encryption scheme, we show how to construct it by additionally relying on hinting PRGs. However, in addition to this construction, we outline two alternate approaches yielding single-key ABE with our function-hiding property (without using hinting PRGs). 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 public-key encryption (PKE), 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, KNYY19, 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 and so it does not appear that these techniques can be used to get LWE-based instantiations. 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, KNYY19, QRW19] achieve statistically sound (M)DV-NIZK *proofs*, we only get computationally sound *arguments*.

1.2 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 Σ-protocol [CDS94] with 1-bit challenges for an NP-complete language, such as Blum's protocol for graph

Hamiltonicity [Blu86]. Recall that a Σ-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 λ (security parameter) independent copies of the following base scheme in parallel:

- **Setup:** The common reference string consists of PKE public keys, $(\mathsf{pk}_0, \mathsf{pk}_1)$. The verifier's secret verification key (b, sk_b) consists of a random bit b along with the secret key sk_b for the corresponding public key pk_b.
- **Proof generation:** On input a statement x and a witness w, the prover P first computes the first message a of the Σ-protocol. Then, the prover computes responses (z_0, z_1) for both possible challenge bits $b \in \{0,1\}$, respectively, and outputs $(a, \mathsf{ct}_0 = \mathsf{Encrypt}(\mathsf{pk}_0, z_0), \mathsf{ct}_1 = \mathsf{Encrypt}(\mathsf{pk}_1, z_1))$.
- **Proof verification:** Given a proof $(a, \mathsf{ct}_0, \mathsf{ct}_1)$, and verification key (b, sk_b), the verifier computes $z = \mathsf{Decrypt}(\mathsf{sk}_b, \mathsf{ct}_b)$ 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 Σ-protocol transcripts (a, b, z_b) without knowing a witness. It can create the ciphertext ct_b by encrypting z_b and can put an arbitrary dummy value in the "other" ciphertext 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 Σ-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 λ 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 ct_0 of the i^{th} copy of the protocol, the adversary learns the verifier's bit b in the i^{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 Σ-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 $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ allows for the encryption of a message m under public parameters pp with respect to an *attribute* x resulting in a ciphertext ct. The ciphertext ct can be decrypted using a secret key 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 sk_f. In this work, we focus on schemes satisfying semantic security in the presence of a *single* secret key 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 $\mathsf{Decrypt}(\mathsf{sk}_f, \cdot)$ does not reveal anything about the function f beyond whether 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 ct, and require that an oracle call of the form $\mathsf{Decrypt}(\mathsf{sk}_f, \mathsf{ct})$ can be simulated using the master secret key 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 E.

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

- A Σ-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 Σ-protocols as a special case).
- An ABE scheme $\mathsf{ABE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ 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 λ (security parameter) independent copies of the following base scheme in parallel.

- **Setup:** The common reference string consists of the public parameters pp for ABE. The verifier's secret verification key (k, sk_f) consists of a PRF key k along with an ABE secret key sk_f for evaluating the function

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

- **Proof generation:** On input a statement x and a witness w, the prover P computes the first message a in the Σ-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 $(\mathsf{ct}_0, \mathsf{ct}_1)$; the prover sends $(a, \mathsf{ct}_0, \mathsf{ct}_1)$ to the verifier.
- **Proof verification:** The verifier first computes $y = \mathsf{PRF}(k, x)$. Then, the verifier decrypts ct_y using its secret verification key 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[3] $x \notin \mathcal{L}$ 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 $y = \mathsf{PRF}(k, x)$ 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 $\mathsf{PRF}(k, x')$ without revealing any additional info about k. But, by the special soundness of the Σ-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 λ invocations, which only happens with negligible probability.

Moreover, we claim that the DV-NIZK is zero-knowledge. In an honestly-generated proof $\pi = (a, \mathsf{ct}_0, \mathsf{ct}_1)$, on instance x, the verifier can compute the response z_y for $y = \mathsf{PRF}(k, x)$, but the response z_{1-y} is computationally hidden by semantic security of ABE. This means that π 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 Σ-protocol.

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

Constructing function-hiding ABE. We now describe several ways 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). We describe three candidate approaches here. In the body of the paper, we will focus primarily on the third candidate based on hinting PRGs for two main reasons: (1) it enables instantiations from CDH or LWE (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 DDH or LWE.

- **Lattice-based ABE:** First, we observe that a simple variant of the lattice-based ABE construction from [BGG+14] satisfies our notion of function-hiding. Namely, we can modify the construction [BGG+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

[3] 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* Σ*-protocol* [CLW18] (see Remark 4.4).

scheme. This high-level idea of leveraging *randomness recovery* is a common theme across all of our constructions. We provide additional details in Appendix C.

- **Circular-secure encryption:** There is a natural candidate construction of a function-hiding ABE scheme using two ingredients: (1) a single-key ABE scheme $\mathsf{ABE} = (\mathsf{ABE.Setup}, \mathsf{ABE.KeyGen}, \mathsf{ABE.Encrypt}, \mathsf{ABE.Decrypt})$, and (2) a secret-key encryption scheme $\mathsf{SKE} = (\mathsf{SKE.Setup}, \mathsf{SKE.Encrypt}, \mathsf{SKE.Decrypt})$. The candidate scheme is defined as follows: to encrypt a message m using public parameters pp and attribute x, sample a SKE-secret key k and output

$$\mathsf{ct} = \Big(\mathsf{ct}_1 := \mathsf{ABE.Encrypt}(\mathsf{pp}, x, k; r), \mathsf{ct}_2 := \mathsf{SKE.Encrypt}(k, m \| r)\Big),$$

  where r denotes the randomness used to encrypt k. To decrypt using a secret key sk_f, first decrypt ct_1 to obtain k; then, decrypt ct_2 to obtain $m \| r$ and output m if and only if $\mathsf{ct}_1 = \mathsf{ABE.Encrypt}(\mathsf{pp}, x, k; r)$. In Appendix D, we describe and analyze this approach for specific instantiations of ABE and SKE (which in turn can be based on either DDH or LWE).

- **Public-key encryption and hinting PRGs:** Following the approach of [KW18b], 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 *hinting PRG* [KW18b], which is a PRG $G\colon \{0,1\}^n \to \{0,1\}^\ell \times \{0,1\}^{\ell n}$ satisfying a natural form of "circular security." Namely, it is required that for $(z_0, z_1, \ldots, z_n) \leftarrow G(s)$, the distribution of (z_0, Z) is pseudorandom, where $Z \in (\{0,1\}^\ell)^{2 \times n}$ is a 2-by-n matrix where $Z_{i,s_i} = z_i$ and $Z_{i,1-s_i}$ sampled uniformly at random (from $\{0,1\}^\ell$). In other words, Z contains "hints" about the secret PRG seed. As shown in [KW18b], hinting PRGs can be constructed using techniques from the laconic OT literature[4] [CDG+17, DG17, BLSV18, DGHM18, GH18], and hence, give instantiations from CDH and LWE.

  In fact, the exact construction of *CCA-secure* ABE in [KW18b] 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 their construction that suffices to construct function-hiding ABE and then analyze its security.

  A high-level description of (our simplification of) the [KW18b] construction is as follows: given any (single-key) ABE scheme ABE, a hinting PRG G, a public-key encryption scheme PKE, and a commitment scheme Com, consider a modified encryption algorithm of the form:

  - **Input:** ABE public parameters pp, PKE public key pk, commitment common reference string crs, attribute x, and message m.
  - **Encryption algorithm:**
    1. Sample a hinting PRG seed s.
    2. Compute (z_0, Z) from the hinting PRG distribution (using $G(s)$ and additional randomness).
    3. Output $\mathsf{ct}_0 = z_0 \oplus m$, a commitment $\mathsf{ct}_1 \leftarrow \mathsf{Com}(\mathsf{crs}, s; \rho)$ to the seed s, and a joint encryption $\mathsf{ct}_2 = \mathsf{Encrypt}(\mathsf{pp}, \mathsf{pk}, x, (s, \rho); Z)$ of the seed s and commitment randomness ρ.

[4] These techniques yield hinting PRGs in the common reference string model, which suffice for our purposes.

This "joint encryption algorithm" encrypts some bits of the commitment randomness ρ using ABE and some bits of ρ using PKE. The bits of the seed s are "encrypted" based on which bits of ρ are encrypted under which scheme. To argue semantic security, we proceed in three steps:

- Switch the commitment crs to an "equivocal mode" so that ct_1 can be explained as a commitment to *any string* (with an appropriate choice of randomness).
- Show that (in equivocal mode) ct_2 can be simulated (using ρ and Z) without knowing s.
- At this point, ct_0 is guaranteed to hide m by invoking hinting PRG security (i.e., (z_0, Z) is indistinguishable from a uniformly random pair).

To obtain function-hiding, we show that the ciphertext ct_2 can be decrypted in two equivalent ways: (1) by using the "honest" decryption algorithm with the ABE secret key sk_f; and (2) using *the* PKE *secret key associated with* pk (while outputting a message only when $f(x) = 1$). Semantic security of the scheme is guaranteed to hold in the presence of 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 clearly hides $f(x')$ for any attribute x' not queried by the adversary, function-hiding follows. Combined with the instantiations of hinting PRGs from CDH or LWE [KW18b] and the fact that single-key ABE follows from PKE [SS10, GVW12], this approach gives a single-key function-hiding ABE scheme from either the CDH assumption or the LWE assumption. We describe this construction (specifically, its generalization) 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 assumptions like CDH or LWE. 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 $(\mathsf{pp}, \mathsf{msk})$ 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 $F\colon \mathcal{X} \times \mathcal{Y} \to \{0,1\}$. 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 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 pk_y should both hide y and be *reusable* for arbitrarily many protocol executions. In addition, we are interested in security even against malicious receivers that choose 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 pk_y.

Similarly to our use of ABE in the generic compiler above, an AB-SFE scheme can be used to compile a Σ-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 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 $\mathsf{PRF}(k,x) = b$. 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 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: first, we combine a form of malicious-secure 2-message OT [PVW08] with garbled circuits to obtain a form of AB-SFE in which the receiver input y is hidden *without* access to the decryption oracle. Then, we apply the [KW18b] hinting PRG transformation to obtain an AB-SFE scheme that has receiver input-hiding with decryption queries. This allows for new instantiations of MDV-NIZK from either DDH or LWE (Theorem 1.2). We provide a formal definition of AB-SFE in Section 3, and the full construction and analysis in Section 5.

2 Preliminaries

We write λ to denote a security parameter. We say that a function f is negligible in λ, denoted $\mathsf{negl}(\lambda)$, 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 $\mathsf{poly}(\lambda)$ to denote a function bounded by a (fixed) polynomial in λ. We say that two families of distributions $\mathcal{D}_1 = \{\mathcal{D}_{1,\lambda}\}_{\lambda\in\mathbb{N}}$ and $\mathcal{D}_2 = \{\mathcal{D}_{2,\lambda}\}_{\lambda\in\mathbb{N}}$ are computationally indistinguishable if no PPT adversary can distinguish samples from $\mathcal{D}_1$ and $\mathcal{D}_2$ except with negligible probability, and we denote this by writing $\mathcal{D}_1 \overset{c}{\approx} \mathcal{D}_2$. We write $\mathcal{D}_1 \overset{s}{\approx} \mathcal{D}_2$ to denote that $\mathcal{D}_1$ and $\mathcal{D}_2$ are statistically indistinguishable (i.e., the statistical distance between $\mathcal{D}_1$ and $\mathcal{D}_2$ is bounded by a negligible function).

For an integer $n \geq 1$, we write $[n]$ to denote the set of integers $\{1,\ldots,n\}$. For a finite set S, we write $x \xleftarrow{R} S$ to denote that x is sampled uniformly at random from S. For a distribution $\mathcal{D}$, we write $x \leftarrow \mathcal{D}$ to denote that x is sampled from $\mathcal{D}$. For two finite sets $\mathcal{X}$ and $\mathcal{Y}$, we write $\mathsf{Funs}[\mathcal{X},\mathcal{Y}]$ 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 $\mathsf{PRF}\colon \mathcal{K}\times\mathcal{X}\to\mathcal{Y}$ such that for all PPT adversaries $\mathcal{A}$,

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

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

- $\mathsf{PKE.KeyGen}(1^\lambda) \to (\mathsf{pk},\mathsf{sk})$: On input the security parameter λ, the key-generation algorithm outputs a public key pk and a secret key sk.

- $\mathsf{PKE.Encrypt}(\mathsf{pk}, m) \to \mathsf{ct}$: On input a public key pk and a message $m \in \mathcal{M}$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{PKE.Decrypt}(\mathsf{sk}, \mathsf{ct}) \to m/\perp$: On input a secret key st and a ciphertext 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 $m \in \mathcal{M}$,

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

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

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

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

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-generations algorithms are combined, and the public key 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 $\mathsf{dvNIZK} = (\mathsf{dvNIZK.Setup}, \mathsf{dvNIZK.KeyGen}, \mathsf{dvNIZK.Prove}, \mathsf{dvNIZK.Verify})$ with the following properties:

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

Moreover, dvNIZK should satisfy the following properties:

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

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

- **Soundness:** We consider two variants of soundness:
  - **Non-adaptive soundness:** For all $x \notin \mathcal{L}$, all PPT adversaries $\mathcal{A}$,

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

    where $\mathsf{crs} \leftarrow \mathsf{dvNIZK.Setup}(1^\lambda)$ and $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{dvNIZK.KeyGen}(\mathsf{crs})$.
  - **Adaptive soundness:** For all PPT adversaries $\mathcal{A}$,

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

    where $(\mathsf{crs}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{dvNIZK.Setup}(1^\lambda)$ and $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{dvNIZK.KeyGen}(\mathsf{crs})$.
- **Zero-knowledge:** For all PPT adversaries $\mathcal{A}$, there exists a PPT simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ such that

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

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

Definition 2.4 (Malicious Designated-Verifier NIZKs [QRW19])**.** Let 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 $\mathsf{ExptReal}_\mathcal{A}(\lambda)$ and $\mathsf{ExptSim}_{\mathcal{A},\mathcal{S}}(\lambda)$ as follows:

- **Setup:** In $\mathsf{ExptReal}_\mathcal{A}(\lambda)$, the challenger samples $\mathsf{crs} \leftarrow \mathsf{dvNIZK.Setup}(1^\lambda)$ and in $\mathsf{ExptSim}_{\mathcal{A},\mathcal{S}}(\lambda)$, the challenger samples $(\mathsf{st}_\mathcal{S}, \overline{\mathsf{crs}}) \leftarrow \mathcal{S}_1(1^\lambda)$. In $\mathsf{ExptReal}_\mathcal{A}(\lambda)$, the challenger gives crs to $\mathcal{A}$, while in $\mathsf{ExptSim}_{\mathcal{A},\mathcal{S}}(\lambda)$, the challenger gives $\overline{\mathsf{crs}}$ to $\mathcal{A}$. Then, $\mathcal{A}$ outputs a public key 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 $\mathsf{ExptReal}_\mathcal{A}(\lambda)$, the challenger replies with $\pi \leftarrow \mathsf{dvNIZK.Prove}(\mathsf{crs}, \mathsf{pk}, x, w)$, and in $\mathsf{ExptSim}_{\mathcal{A},\mathcal{S}}(\lambda)$, the challenger replies with $\overline{\pi} \leftarrow \mathcal{S}_2(\mathsf{st}_\mathcal{S}, \mathsf{pk}, x)$.
- **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 dvNIZK provides *zero-knowledge against malicious verifiers* if for all PPT adversaries $\mathcal{A}$, there exists an efficient simulator $\mathcal{S}$ such that $\mathsf{ExptReal}_\mathcal{A}(\lambda) \overset{c}{\approx} \mathsf{ExptSim}_{\mathcal{A},\mathcal{S}}(\lambda)$. If dvNIZK satisfies this property (in addition to completeness and soundness), then we say that 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 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 $\mathcal{R}\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ be an NP relation and $\mathcal{L} \subseteq \{0,1\}^n$ be the associated language. A *non-adaptive, ℓ-query zero-knowledge PCP (with alphabet Σ) for* $\mathcal{L}$ is a tuple of algorithms $\mathsf{zkPCP} = (\mathsf{zkPCP.Prove}, \mathsf{zkPCP.Query}, \mathsf{zkPCP.Verify})$ with the following properties:

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

Moreover, zkPCP should satisfy the following properties:

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

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

  where $\pi \leftarrow \mathsf{zkPCP.Prove}(x, w)$ and $(\mathsf{st}_x, q_1, \ldots, q_\ell) \leftarrow \mathsf{zkPCP.Query}(x)$.
- **Soundness:** For all $x \notin \mathcal{L}$, all proof strings $\pi \in \Sigma^m$,

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

  where $(\mathsf{st}_x, q_1, \ldots, q_\ell) \leftarrow \mathsf{zkPCP.Query}(x)$.
- **Zero-knowledge:** For all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists an efficient simulator $\mathcal{S}$ such that

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

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

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 ℓ bits of the PCP, including subsets that would never be output by $\mathsf{zkPCP.Query}$. 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 $\mathsf{zkPCP.Query}$ (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

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

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

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 $\mathsf{zkPCP.Query}$ 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 ℓ-query zero-knowledge PCP for $\mathcal{L}$ with alphabet $\Sigma = \{0, 1, 2\}$ and $\ell = \mathsf{poly}(n)$.*

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, Σ-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 $\mathcal{F} = \{f \colon \mathcal{X} \to \{0,1\}\}$ is a tuple of algorithms $\mathsf{ABE} = (\mathsf{ABE.Setup}, \mathsf{ABE.KeyGen}, \mathsf{ABE.Encrypt}, \mathsf{ABE.Decrypt})$ with the following properties:

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

- $\mathsf{ABE.Encrypt}(\mathsf{pp}, x, m) \to \mathsf{ct}_{x,m}$: On input the public parameters pp, an attribute $x \in \mathcal{X}$, and a message $m \in \mathcal{M}$, the encryption algorithm outputs a ciphertext $\mathsf{ct}_{x,m}$.
- $\mathsf{ABE.Decrypt}(\mathsf{pp}, \mathsf{sk}, \mathsf{ct}) \to (x, m)$: On input the public parameters pp, a secret key sk (which could be the master secret key), and a ciphertext 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 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 $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{ABE.Setup}(1^\lambda)$,

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

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

Definition 2.12 (Security)**.** Let ABE be an ABE scheme over an attribute space $\mathcal{X}$, message space $\mathcal{M}$, and function family $\mathcal{F}$. For a security parameter λ and an adversary $\mathcal{A}$, we define the ABE security experiment $\mathsf{Expt}^{\mathsf{ABE}}_{\mathcal{A}}(\lambda, b)$ as follows. The challenger begins by sampling $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{ABE.Setup}(1^\lambda)$ and gives 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 $\mathsf{sk}_f \leftarrow \mathsf{ABE.KeyGen}(\mathsf{pp}, \mathsf{msk}, f)$.
- **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 $\mathsf{ct} \leftarrow \mathsf{ABE.Encrypt}(\mathsf{pp}, x, m_b)$.

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 ABE is *secure* if for all efficient and admissible adversaries $\mathcal{A}$,

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

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

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

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

- **Real decryption oracle:** On input pp, sk_f, $\mathsf{ct} \in \{0,1\}^t$, the real decryption oracle $\mathcal{O}_1(\mathsf{pp}, \mathsf{sk}_f, \mathsf{ct})$ outputs $\mathsf{ABE.Decrypt}(\mathsf{pp}, \mathsf{sk}_f, \mathsf{ct})$.
- **Ideal decryption oracle:** On input pp, msk, and a string $\mathsf{ct} \in \{0,1\}^t$, the ideal decryption oracle $\mathcal{O}_2(\mathsf{pp}, \mathsf{msk}, \mathsf{ct})$ outputs $\mathcal{S}^{f(\cdot)}(\mathsf{pp}, \mathsf{msk}, \mathsf{ct})$. 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 $\mathsf{OT} = (\mathsf{OT.Setup}, \mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT.Receive})$ satisfying the following properties:

- $\mathsf{OT.Setup}(1^\lambda) \to \mathsf{crs}$: On input the security parameter λ, the setup algorithm outputs a *common reference string* crs. If $\mathsf{OT.Setup}$ outputs a uniformly random string, we say that the OT scheme is in the *common random string* model.
- $\mathsf{OT}_1(\mathsf{crs}, b; r) \to M^{(1)}$: On input the common reference string crs and a choice bit $b \in \{0,1\}$, 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.
- $\mathsf{OT}_2(\mathsf{crs}, M^{(1)}, m_0, m_1) \to M^{(2)}$: On input the common reference string crs, the OT first message $M^{(1)}$ and the sender's messages $m_0, m_1 \in \{0,1\}^k$, OT_2 outputs the the second OT message (from sender to receiver).
- $\mathsf{OT.Receive}(\mathsf{crs}, M^{(2)}, b, r) \to m$: On input the common reference string 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 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:

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

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

Definition 2.16 (Computational Receiver Security)**.** An OT scheme OT satisfies (computational) receiver privacy if

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

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

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

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

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

- **Sender privacy in extraction mode:** For a security parameter λ and an adversary $\mathcal{A}$, let $\mathsf{Expt}^{\mathsf{OT\text{-}Ext}}_{\mathsf{OT},\mathcal{A}}(\lambda, b)$ be the following experiment:
  - **Setup:** The challenger samples $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{OT.SetupExt}(1^\lambda)$ and gives crs to the adversary.
  - **Initial message:** The adversary picks $M^{(1)}$ and sends it to the challenger. The challenger computes $\beta \leftarrow \mathsf{OT.Ext}(\mathsf{td}, M^{(1)})$ 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 $\widetilde{m}_\beta = m_\beta$ and $\widetilde{m}_{1-\beta} = m_{1-\beta}$ if $b = 0$ and $\widetilde{m}_{1-\beta} = 0^k$ if $b = 1$, and sends $M^{(2)} \leftarrow \mathsf{OT}_2(\mathsf{crs}, M^{(1)}, \widetilde{m}_0, \widetilde{m}_1)$ 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 $\mathcal{A}$:

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

Remark 2.18 (Batch OT)**.** In the following we will assume that OT_1 takes as input an ℓ-bit choice string $y \in \{0,1\}^\ell$ for some $\ell = \mathsf{poly}(\lambda)$ (rather than just a single choice bit $b \in \{0,1\}$), and that the OT_2 algorithm takes as input 2ℓ 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 $\mathsf{OT}_1(\mathsf{crs}, y)$ 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 $\mathsf{OT.Ext}$ 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 ℓ independent instances of the basic 1-out-of-2 OT scheme in parallel (where the new CRS is the concatenation of ℓ *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] imply receiver-extractable 2-message (batch) OTs from both the DDH as well as the LWE assumptions. We state this in the following theorem:

Theorem 2.19 (Receiver-Extractable 2-Message OT from DDH and LWE [PVW08])**.** *Assuming either DDH or LWE with a polynomial modulus-to-noise ratio (and polynomial modulus), there exists a receiver-extractable 2-message 1-out-of-2 OT for k-bit messages (for any fixed $k = \mathsf{poly}(\lambda)$) in the common random string model.*

Remark 2.20 (Statistical Sender Security)**.** The OT scheme from Theorem 2.19 actually achieves a stronger notion of extractability against *unbounded* malicious receivers. This means that the CRS indistinguishability and sender privacy properties from Definition 2.17 hold *statistically*, even against unbounded adversaries.

2.5 Garbling Scheme

Definition 2.21 (Garbling Scheme [Yao86, LP09, BHR12])**.** A garbling scheme for Boolean circuits consists of two algorithms $\mathsf{Yao} = (\mathsf{Yao.Garble}, \mathsf{Yao.Eval})$ with the following properties:

- $\mathsf{Yao.Garble}(1^\lambda, C) \to (\widetilde{C}, \overline{\mathsf{lab}})$: On input a security parameter λ, a Boolean circuit C on n-bit inputs, the garbling algorithm outputs a garbled circuit $\widetilde{C}$ and a collection of labels $\overline{\mathsf{lab}} = \{\mathsf{lab}_{i,b}\}_{i\in[n],b\in\{0,1\}}$ where $\mathsf{lab}_{i,b} \in \{0,1\}^\lambda$ for all $i \in [n]$ and $b \in \{0,1\}$.

- $\mathsf{Yao.Eval}(\widetilde{C}, \overrightarrow{\mathsf{lab}}) \to y$: On input a garbled circuit $\widetilde{C}$ and a collection of labels $\overrightarrow{\mathsf{lab}} = \{\mathsf{lab}_{i,x_i}\}_{i\in[n]}$, the evaluation algorithm outputs a string y.

A garbling scheme should satisfy the following two properties:

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

$$\Pr\left[C(x) = \mathsf{Yao.Eval}(\widetilde{C}, \{\mathsf{lab}_{i,x_i}\}_{i\in[n]})\right] = 1,$$

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

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

$$\left(\widetilde{C}, \{\mathsf{lab}_{i,x_i}\}_{i\in[n]}\right) \overset{c}{\approx} \mathcal{S}\left(1^{|C|}, 1^n, C(x)\right),$$

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

Theorem 2.22 (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 2.23 (Non-Interactive Equivocable Commitment [CIO98])**.** A non-interactive equivocable commitment scheme $\mathsf{Com} = (\mathsf{Com.Setup}, \mathsf{Com.Commit})$ with message space $\mathcal{M}$ is defined as follows:

- $\mathsf{Com.Setup}(1^\lambda) \to \mathsf{crs}$: On input the security parameter λ, the setup algorithm outputs a common reference string crs. If $\mathsf{Com.Setup}$ outputs a uniformly random string, we say that the commitment scheme is in the *common random string* model.

- $\mathsf{Com.Commit}(\mathsf{crs}, m; r) \to c$: On input a reference string crs, a message $m \in \mathcal{M}$, and randomness $r \in \{0,1\}^\lambda$, the commit algorithm outputs a commitment c.

Moreover, Com should satisfy the following properties:

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

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

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

- **Equivocation:** There exists PPT algorithms $(\mathsf{Com.SetupEquiv}, \mathsf{Com.Equivocate})$ with the following properties:

  - $\mathsf{Com.SetupEquiv}(1^\lambda) \to (\overline{\mathsf{crs}}, \bar{c}, \mathsf{td})$: On input the security parameter λ, the setup algorithm outputs a common reference string $\overline{\mathsf{crs}}$, a commitment $\bar{c}$, and a trapdoor td.

– $\mathsf{Com.Equivocate}(\mathsf{td}, m, c) \to r$: On input a trapdoor 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

$$(\mathsf{crs}, c, r) \overset{c}{\approx} (\overline{\mathsf{crs}}, \bar{c}, \bar{r}),$$

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

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

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

Theorem 2.24 (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 Hinting PRGs

Definition 2.25 (Hinting PRG [KW18b])**.** A hinting PRG $\mathsf{HPRG} = (\mathsf{HPRG.Setup}, \mathsf{HPRG.Eval})$ with seed length $n = n(\lambda)$ and stretch $\ell = \ell(\lambda)$ is defined as follows:

- $\mathsf{HPRG.Setup}(1^\lambda)$: Outputs a common reference string crs. We say that HPRG is in the *common random string model* if $\mathsf{HPRG.Setup}(1^\lambda)$ outputs a uniformly random string.
- $\mathsf{HPRG.Eval}(\mathsf{crs}, s, i)$: On input a common reference string crs, a seed $s \in \{0,1\}^n$, and an index $i \in \{0, \ldots, n\}$, the evaluation algorithm outputs a block $z_i \in \{0,1\}^\ell$.

We say HPRG is secure if for $\mathsf{crs} \leftarrow \mathsf{HPRG.Setup}(1^\lambda)$, $s \overset{\mathrm{R}}{\leftarrow} \{0,1\}^n$, $z_0 \leftarrow \mathsf{HPRG.Eval}(\mathsf{crs}, s, 0)$, $Z_{i,s_i} \leftarrow \mathsf{HPRG.Eval}(\mathsf{crs}, s, i)$, $Z_{i,1-s_i} \overset{\mathrm{R}}{\leftarrow} \{0,1\}^\ell$, $r_0 \overset{\mathrm{R}}{\leftarrow} \{0,1\}^\ell$, and $R_{i,b} \overset{\mathrm{R}}{\leftarrow} \{0,1\}^\ell$ for all $i \in [n]$ and $b \in \{0,1\}$, the following distributions are computationally indistinguishable:

$$\{(\mathsf{crs}, z_0, \{Z_{i,b}\}_{i\in[n], b\in\{0,1\}})\} \overset{c}{\approx} \{(\mathsf{crs}, r_0, \{R_{i,b}\}_{i\in[n], b\in\{0,1\}})\}.$$

Theorem 2.26 (Hinting PRGs from CDH or LWE [KW18b])**.** *Under the CDH assumption (resp., the LWE assumption with any super-polynomial modulus-to-noise ratio), there exists a family of hinting PRGs. Moreover, the public parameters in this HPRG family (i.e., the output of* $\mathsf{HPRG.Setup}$*) are uniformly random.*

Remark 2.27 (Uniformly Random Parameters vs. Pseudorandom Parameters)**.** Note that the constructions of hinting PRGs from CDH (resp., LWE) of [KW18b] uses a common reference string that is only pseudorandom under the DDH assumption (resp., the LWE assumption). By a standard hybrid argument, we can replace the pseudorandom parameters with uniformly random parameters while ensuring security under the DDH assumption (resp., the LWE assumption).

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 $F\colon \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ with message space $\mathcal{M}$ consists of a tuple $\mathsf{ABSFE} = (\mathsf{ABSFE.Setup}, \mathsf{ABSFE.KeyGen}, \mathsf{ABSFE.Encrypt}, \mathsf{ABSFE.Decrypt})$ of PPT algorithms with the following properties:

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

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

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

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

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 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 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 $\mathsf{crs} \leftarrow \mathsf{ABSFE.Setup}(1^\lambda)$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{ABSFE.KeyGen}(\mathsf{crs}, y)$ and gives crs, pk, 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 $\mathsf{ct} \leftarrow \mathsf{ABSFE.Encrypt}(\mathsf{crs}, \mathsf{pk}, x, m_b)$ and gives ct to $\mathcal{A}$.

- **Output:** The adversary $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$.

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

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

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

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

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

- **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 $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{ABSFE.SetupExt}(1^\lambda)$ and gives crs to $\mathcal{A}$.
  - **Public key selection:** The adversary chooses a public key pk. The challenger computes $y \leftarrow \mathsf{ABSFE.Ext}(\mathsf{td}, \mathsf{pk})$ 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 $\mathsf{ct} \leftarrow \mathsf{ABSFE.Encrypt}(\mathsf{crs}, \mathsf{pk}, x, m_b)$ and gives ct to $\mathcal{A}$.
  - **Output:** The adversary $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$.

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

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 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 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 $\mathcal{A}$,

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

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

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:

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

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

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

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 E, 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 λ 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 $\mathsf{zkPCP} = (\mathsf{zkPCP.Prove}, \mathsf{zkPCP.Query}, \mathsf{zkPCP.Verify})$ be an efficient ℓ-query, non-adaptive, zero-knowledge PCP (with alphabet Σ) 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 for $\mathsf{zkPCP.Query}$.
- Let $\mathsf{PRF}\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^\rho$ be a pseudorandom function (Definition 2.1).
- Let $F\colon (\{0,1\}^n \times [m]) \times \mathcal{K} \to \{0,1\}$ be the function

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

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

- Let $\mathsf{ABSFE} = (\mathsf{ABSFE.Setup}, \mathsf{ABSFE.KeyGen}, \mathsf{ABSFE.Encrypt}, \mathsf{ABSFE.Decrypt})$ 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 $\mathsf{dvNIZK} = (\mathsf{dvNIZK.Setup}, \mathsf{dvNIZK.KeyGen}, \mathsf{dvNIZK.Prove}, \mathsf{dvNIZK.Verify})$ for $\mathcal{L}$ as follows:

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

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

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

Proof. Follows immediately by completeness of zkPCP and correctness of ABSFE. □

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

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

- Hyb_0: This is the real soundness experiment. At the beginning of the game, the challenger samples $k \xleftarrow{R} \mathcal{K}$, $\mathsf{crs} \leftarrow \mathsf{ABSFE.Setup}(1^\lambda)$, and $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{ABSFE.KeyGen}(\mathsf{crs}, k)$. It gives crs and $\mathsf{pk} = \mathsf{pk}'$ to the adversary and sets $\mathsf{sk} = (k, \mathsf{sk}')$. When the adversary makes a verification query (x, π), the challenger replies with $\mathsf{dvNIZK.Verify}(\mathsf{crs}, \mathsf{sk}, x, \pi)$. At the end of the experiment, the adversary outputs a proof π^*. The output of the experiment is 1 if $\mathsf{dvNIZK.Verify}(\mathsf{crs}, \mathsf{sk}, x^*, \pi^*) = 1$ and 0 otherwise.
- Hyb_1: Same as Hyb_0, except the challenger uses the simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ for the strong key-hiding game for 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 $(\mathsf{st}_{\mathcal{S}}, \overline{\mathsf{crs}}, \overline{\mathsf{pk}}) \leftarrow \mathcal{S}_1(1^\lambda)$, and gives $\overline{\mathsf{crs}}$, $\overline{\mathsf{pk}}$ to the adversary. When simulating verification queries $\mathsf{dvNIZK.Verify}(\mathsf{crs}, \mathsf{sk}, x, \pi)$, the challenger proceeds as follows:
  - Compute $(\mathsf{st}_x, q_1, \ldots, q_\ell) \leftarrow \mathsf{zkPCP.Query}(x; \mathsf{PRF}(k, x))$.
  - Write $\pi = (\mathsf{ct}_1, \ldots, \mathsf{ct}_m)$. For each $j \in [\ell]$, it computes $s_j \leftarrow \mathcal{S}_2(\mathsf{st}, (x, q_j), \mathsf{ct}_{q_j}, 1)$.

– The challenge replies with $\mathsf{zkPCP.Verify}(\mathsf{st}_x, s_1, \ldots, s_\ell)$.

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

This is computationally indistinguishable from Hyb_0 by strong key-hiding security of ABSFE. In particular, we use the fact that by construction, $F((x, q_j), k) = 1$ for each verification query.

- Hyb_2: Same as Hyb_1, except the challenger samples a random function $f \xleftarrow{R} \mathsf{Funs}[\{0,1\}^n, \{0,1\}^\rho]$ at the beginning of the experiment. Then, whenever it needs to compute $\mathsf{PRF}(k, x)$, it instead computes $f(x)$.

  This is computationally indistinguishable from Hyb_1 by security of PRF.

- Hyb_3: Same as Hyb_2, except whenever the challenger needs to compute $\mathsf{dvNIZK.Verify}(\mathsf{crs}, \mathsf{sk}, x, \pi)$ for some $x \notin \mathcal{L}$, the challenger sets the output to 0.

  This is statistically indistinguishable from Hyb_2 by soundness of zkPCP. Namely, in Hyb_2, when the challenger verifies a proof π for a statement $x \notin \mathcal{L}$, it does so by first sampling $(\mathsf{st}_x, q_1, \ldots, q_\ell) \leftarrow \mathsf{zkPCP.Prove}(x; r)$ where r is *uniformly* random, and then computes $\mathsf{zkPCP.Verify}(\mathsf{st}_x, s_1, \ldots, s_\ell)$ for some collection of bits $s_1, \ldots, s_\ell$ (independent of r). By soundness of zkPCP, the challenger will reject with all but negligible probability.

By construction, for any proof π for a false statement $x \notin \mathcal{L}$, the challenger in Hyb_3 rejects with probability 1, and soundness follows. □

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 Σ-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 Σ-protocol, there is exactly a single challenge that would cause the verifier to accept.
- **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 Σ-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 Σ-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 Σ-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* 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 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 $\mathsf{ABSFE.SetupExt}$ and $\mathsf{ABSFE.Ext}$ 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 λ, the simulator runs $(\overline{\mathsf{crs}}, \mathsf{td}) \leftarrow \mathsf{ABSFE.SetupExt}(1^\lambda)$. It also samples randomness $r \xleftarrow{R} \{0,1\}^\tau$ that it will use for the $\mathsf{ABSFE.Ext}$ algorithm (where τ is a bound on the number of bits of randomness $\mathsf{ABSFE.Ext}$ takes). Finally, it outputs $\overline{\mathsf{crs}}$ and a state $\mathsf{st}_{\mathcal{S}} = (\mathsf{td}, r)$. 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.

- $\mathcal{S}_2(\mathsf{st}_{\mathcal{S}}, \mathsf{pk}, x)$: On input the simulation state $\mathsf{st}_{\mathcal{S}} = (\mathsf{td}, r)$, a public key pk, and a statement x, the simulator begins by extracting a key $k \leftarrow \mathsf{ABSFE.Ext}(\mathsf{td}, \mathsf{pk}; r)$. Then, it computes $(\mathsf{st}_x, q_1, \ldots, q_\ell) \leftarrow \mathsf{zkPCP.Prove}(x; \mathsf{PRF}(k, x))$ and $(\tilde{\pi}_1^{(\mathsf{PCP})}, \ldots, \tilde{\pi}_\ell^{(\mathsf{PCP})}) \leftarrow \mathcal{S}^{(\mathsf{PCP})}(x, q_1, \ldots, q_\ell)$. For $i \in [m]$, the simulator takes $z_i \leftarrow \tilde{\pi}_j^{(\mathsf{PCP})}$ 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 $\mathsf{ct}_i \leftarrow \mathsf{ABSFE.Encrypt}(\mathsf{crs}, \mathsf{pk}, (x, i), z_i)$ and outputs the simulated proof $\overline{\pi} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_m)$.

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

- Hyb_0: This is $\mathsf{ExptReal}_{\mathcal{A}}(\lambda)$. Namely, at the beginning of the game, the challenger samples $\mathsf{crs} \leftarrow \mathsf{ABSFE.Setup}(1^\lambda)$ and gives crs to the adversary. The adversary $\mathcal{A}$ then outputs a public key pk. When $\mathcal{A}$ makes a verification query on (x, w) where $\mathcal{R}(x, w) = 1$, the challenger replies with $\mathsf{dvNIZK.Prove}(\mathsf{crs}, \mathsf{pk}, x, w)$. At the end of the experiment, algorithm $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$, which is also the output of the experiment.

- Hyb_1: Same as Hyb_0, except we replace the real common reference string $\mathsf{crs} \leftarrow \mathsf{ABSFE.Setup}(1^\lambda)$ with a simulated one $\overline{\mathsf{crs}}$ where $(\overline{\mathsf{crs}}, \mathsf{td}) \leftarrow \mathsf{ABSFE.SetupExt}(1^\lambda)$.

  This is computationally indistinguishable from Hyb_0 by the CRS indistinguishability of ABSFE (from Definition 3.4).

- Hyb_2: Same as Hyb_1, except the challenger implements the adversary's verification queries using the following modified procedure. First, after $\mathcal{A}$ outputs a public key pk, the challenger computes $k \leftarrow \mathsf{ABSFE.Ext}(\mathsf{td}, \mathsf{pk})$. 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 $\pi^{(\mathsf{PCP})} \leftarrow \mathsf{zkPCP.Prove}(x, w)$. It additionally computes $(\mathsf{st}_x, q_1, \ldots, q_\ell) \leftarrow \mathsf{zkPCP.Query}(x; \mathsf{PRF}(k, x))$ and for each $i \in [m]$, it sets $z_i \leftarrow \pi_i^{(\mathsf{PCP})}$ if $i = q_j$ for some $j \in [\ell]$ and $z_i \leftarrow 0$ otherwise. It then computes $\mathsf{ct}_i \leftarrow \mathsf{ABSFE.Encrypt}(\mathsf{crs}, \mathsf{pk}, (x, i), z_i)$ for each $i \in [m]$ and outputs the proof $\pi = (\mathsf{ct}_1, \ldots, \mathsf{ct}_m)$.

  This is computationally indistinguishable from Hyb_1 by ciphertext indistinguishability of ABSFE (from Definition 3.4). Namely, the only difference between Hyb_1 and Hyb_2 is that in Hyb_2, instead of computing the ciphertexts as $\mathsf{ct}_i \leftarrow \mathsf{ABSFE.Encrypt}(\mathsf{crs}, \mathsf{pk}, (x, i), \pi_i^{(\mathsf{PCP})})$

when $F((x,i),k) = 0$, the challenger instead computes $\mathsf{ct}_i \leftarrow \mathsf{ABSFE.Encrypt}(\mathsf{crs}, \mathsf{pk}, (x,i), 0)$. In the formal reduction, after the zero-knowledge adversary outputs its (possibly maliciously-generated) public key, the ciphertext-indistinguishability adversary forwards 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 pk (see Remark 3.5).

- Hyb_3: Same as Hyb_2, except the challenger simulates the bits of the PCP using $\mathcal{S}^{(\mathsf{PCP})}$ instead of computing them via $\mathsf{zkPCP.Prove}$. Specifically, on a verification query (x,w) where $\mathcal{R}(x,w) = 1$, after computing $(\mathsf{st}_x, q_1, \ldots, q_\ell) \leftarrow \mathsf{zkPCP.Prove}(x; \mathsf{PRF}(k,x))$, the challenger constructs a simulated PCP $(\tilde{\pi}_1^{(\mathsf{PCP})}, \ldots, \tilde{\pi}_\ell^{(\mathsf{PCP})}) \leftarrow \mathcal{S}^{(\mathsf{PCP})}(x, q_1, \ldots, q_\ell)$. For each $i \in [m]$, it sets $z_i = \tilde{\pi}_i^{(\mathsf{PCP})}$ if $i = q_j$ for some $j \in [\ell]$ and $z_i = 0$ otherwise. The output is then computed as in Hyb_2. Notably, this is the $\mathsf{ExptSim}_{\mathcal{A},\mathcal{S}}(\lambda)$ experiment.

  This is computationally indistinguishable from Hyb_2 by zero-knowledge of 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. □

Remark 4.6 (DV-NIZKs in the Common Random String Model)**.** If the public parameters of ABSFE (i.e., the output of $\mathsf{ABSFE.Setup}$) 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 hinting PRGs [KW18b] (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. In conjunction with Construction 4.1, we obtain DV-NIZKs from the CDH assumption, and MDV-NIZKs (in the common *random* string model) from the DDH or the LWE assumptions.

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

As noted in Section 1.2, 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 $F\colon \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ 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:

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

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 $(\mathsf{crs}, \mathsf{pk} = \mathsf{pp})$ is generated independently of y. Finally, weak message-hiding follows immediately via semantic security of ABE. □

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. □

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 $F\colon \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ 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 $C_{x,m}\colon \mathcal{Y} \to \mathcal{M} \cup \{\bot\}$ be a circuit that on input y' outputs m if $F(x,y') = 1$ and $\bot$ otherwise. Let $\ell = \mathsf{poly}(\lambda)$ be a bound on the bit-length of elements in $\mathcal{Y}$.

- Let $\mathsf{Yao} = (\mathsf{Yao.Garble}, \mathsf{Yao.Eval})$ be a garbling scheme (Definition 2.21) that supports the circuit class $\mathcal{C} = \{x \in \mathcal{X}, m \in \mathcal{M} : C_{x,m}\}$.
- Let $\mathsf{OT} = (\mathsf{OT.Setup}, \mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT.Receive})$ be a receiver-extractable 2-message batch OT scheme on k-bit messages with batch size ℓ (Definition 2.14, Remark 2.18), where $k = \mathsf{poly}(\lambda)$ 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:

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

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 receive privacy of OT.

Strong message-hiding. We first define the extraction algorithms $(\mathsf{ABSFE.SetupExt}, \mathsf{ABSFE.Ext})$:

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

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_0: This is the ciphertext indistinguishability game for ABSFE. Namely, the challenger begins by sampling $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{OT.SetupExt}(1^\lambda)$ and gives crs to $\mathcal{A}$. The adversary chooses a public key pk and the challenger replies with $y \leftarrow \mathsf{OT.Ext}(\mathsf{td}, \mathsf{pk})$. Finally, when the adversary makes a challenge query (x, m_0, m_1), the challenger garbles $(\widetilde{C}, \{\mathsf{lab}_{i,b}\}_{i\in[n],b\in\{0,1\}}) \leftarrow \mathsf{Yao.Garble}(1^\lambda, C_{x,m_b})$ and replies with the ciphertext $\mathsf{ct}'_b = (\widetilde{C}, \mathsf{OT}_2(\mathsf{crs}, \mathsf{pk}, \{\mathsf{lab}_{i,b}\}_{i\in[n],b\in\{0,1\}}))$.
- Hyb_1: Same as Hyb_0 except the challenger instead constructs the challenge ciphertext as $\mathsf{ct}'_b \leftarrow (\widetilde{C}, \mathsf{OT}_2(\mathsf{crs}, \mathsf{pk}, \{(\mathsf{lab}_{i,y_i}, \mathsf{lab}_{i,y_i})\}_{i\in[n]}))$.

  Hybrids Hyb_0 and Hyb_1 are computationally indistinguishable by sender privacy of OT in extraction mode.

- Hyb_2: Same as Hyb_1, except instead of sampling $(\widetilde{C}, \{\mathsf{lab}_{i,b}\}_{i\in[n],b\in\{0,1\}}) \leftarrow \mathsf{Yao.Garble}(1^\lambda, C_{x,m_b})$, the challenger uses the simulator $\mathcal{S}$ for Yao to sample the garbled circuit. Namely, the challenger samples $(\widetilde{C}, \{\mathsf{lab}_i\}_{i\in[n]}) \leftarrow \mathsf{Sim}(1^t, 1^n, 0)$, where t is a bound on the size of the circuit in $\mathcal{C}$, and sets $\mathsf{ct}'_b \leftarrow (\widetilde{C}, \mathsf{OT}_2(\mathsf{crs}, \mathsf{pk}, \{(\mathsf{lab}_i, \mathsf{lab}_i)\}_{i\in[n]}))$.

  Hybrids Hyb_1 and Hyb_2 are computationally indistinguishable by security of Yao.

Adversary $\mathcal{A}$'s view in hybrid Hyb_2 does not depend on the challenge bit b, and the claim follows. □

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 hinting PRGs (Definition 2.25) [KW18b] and an equivocable non-interactive commitment scheme (Definition 2.23) [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 $\mathsf{PKE} = (\mathsf{PKE.KeyGen}, \mathsf{PKE.Encrypt}, \mathsf{PKE.Decrypt})$ with message space $\mathcal{M}$ satisfies the *recover from randomness property* if there exists an efficient algorithm $\mathsf{PKE.Recover}$ with the following property:

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

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

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 $F\colon \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ 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 $\mathsf{PKE.Encrypt}$ and $\mathsf{ABSFE.Encrypt}$ use.
- Let HPRG be a hinting PRG with seed length λ and stretch ℓ (Definition 2.25).
- Let Com be a non-interactive equivocable commitment scheme with message space $\{0,1\}$ (Definition 2.23).

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

- $\mathsf{Aug.Setup}(1^\lambda)$: Sample $\mathsf{ABSFE.crs} \leftarrow \mathsf{ABSFE.Setup}(1^\lambda)$, $(\mathsf{PKE.pk}, \mathsf{PKE.sk}) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$, $\mathsf{HPRG.crs} \leftarrow \mathsf{HPRG.Setup}(1^\lambda)$, and for each $i \in [\lambda]$, $\mathsf{Com.crs}_i \leftarrow \mathsf{Com.Setup}(1^\lambda)$. It outputs the common reference string $\mathsf{crs} = (\mathsf{ABSFE.crs}, \mathsf{PKE.pk}, \mathsf{HPRG.crs}, \{\mathsf{Com.crs}_i\}_{i\in[\lambda]})$.

- $\mathsf{Aug.KeyGen}(\mathsf{crs}, y)$: Parse $\mathsf{crs} = (\mathsf{ABSFE.crs}, \mathsf{PKE.pk}, \mathsf{HPRG.crs}, \{\mathsf{Com.crs}_i\}_{i\in[\lambda]})$, construct $\mathsf{ABSFE.pk} \leftarrow \mathsf{ABSFE.KeyGen}(\mathsf{ABSFE.crs}, y)$ and output the public key $\mathsf{pk} = \mathsf{ABSFE.pk}$ and the secret key $\mathsf{sk} = (y, \mathsf{ABSFE.pk})$.

- $\mathsf{Aug.Encrypt}(\mathsf{crs}, \mathsf{pk}, x, m)$: Parse $\mathsf{crs} = (\mathsf{ABSFE.crs}, \mathsf{PKE.pk}, \mathsf{HPRG.crs}, \{\mathsf{Com.crs}_i\}_{i\in[\lambda]})$ and $\mathsf{pk} = \mathsf{ABSFE.pk}$, and proceed as follows:

  - Sample a seed $s \xleftarrow{R} \{0,1\}^\lambda$ and compute $(z_0, \{Z_{i,b}\}_{i\in[\lambda],b\in\{0,1\}})$ as in the HPRG security experiment (Definition 2.25). In particular, $Z_{i,1-s_i}$ is uniform over $\{0,1\}^\ell$ and the other components are pseudorandom.
  - For every $i \in [\lambda]$, sample $\rho_i \xleftarrow{R} \{0,1\}^\lambda$ and compute $c_i \leftarrow \mathsf{Com.Commit}(\mathsf{Com.crs}_i, s_i; \rho_i)$.
  - For every $i \in [\lambda]$, define $M_{i,s_i} = \rho_i$ and $M_{i,1-s_i} = \bot$. Then, construct the ciphertexts

$$\mathsf{ct}_{i,0} \leftarrow \mathsf{ABSFE.Encrypt}(\mathsf{ABSFE.crs}, \mathsf{ABSFE.pk}, x, M_{i,0}; Z_{i,0})$$
$$\mathsf{ct}_{i,1} \leftarrow \mathsf{PKE.Encrypt}(\mathsf{PKE.pk}, M_{i,1}; Z_{i,1}).$$

  - Let $\mathsf{ct}_0 \leftarrow z_0 \oplus m$ and output $\mathsf{ct} = \big(\mathsf{ct}_0, (c_i, \mathsf{ct}_{i,0}, \mathsf{ct}_{i,1})_{i\in[\lambda]}\big)$.

- $\mathsf{Aug.Decrypt}(\mathsf{crs}, \mathsf{sk}, x, \mathsf{ct})$: Parse $\mathsf{crs} = (\mathsf{ABSFE.crs}, \mathsf{PKE.pk}, \mathsf{HPRG.crs}, \{\mathsf{Com.crs}_i\}_{i\in[\lambda]})$, $\mathsf{sk} = (y, \mathsf{ABSFE.pk})$, and $\mathsf{ct} = \big(\mathsf{ct}_0, (c_i, \mathsf{ct}_{i,0}, \mathsf{ct}_{i,1})_{i\in[\lambda]}\big)$, and proceed as follows:

  1. If $F(x, y) = 0$, output $\bot$.
  2. For every $i \in [\lambda]$, compute $\rho'_i \leftarrow \mathsf{ABSFE.Decrypt}(\mathsf{ABSFE.crs}, \mathsf{sk}, x, \mathsf{ct}_{i,0})$. If $\rho'_i \neq \bot$ and $\mathsf{Com.Commit}(\mathsf{Com.crs}_i, 0; \rho'_i) = c_i$, set $s'_i = 0$; otherwise, set $s'_i = 1$.
  3. For every $i \in [\lambda]$, let $z'_i \leftarrow \mathsf{HPRG.Eval}(\mathsf{HPRG.crs}, s', i)$, and perform the following checks:
     - If $s'_i = 0$, then check if $\mathsf{Recover}(\mathsf{ABSFE.crs}, \mathsf{ABSFE.pk}, \mathsf{ct}_{i,0}, z'_i) = (x, \rho'_i)$, and output $\bot$ if the check fails.
     - If $s'_i = 1$, then compute $\rho'_i \leftarrow \mathsf{PKE.Recover}(\mathsf{PKE.pk}, \mathsf{ct}_{i,1}, z'_i)$, and check if $c_i = \mathsf{Com.Commit}(\mathsf{Com.crs}_i, 1; \rho'_i)$. Output $\bot$ if this check fails.
  4. If all checks pass, output $\mathsf{ct}_0 \oplus \mathsf{HPRG.Eval}(\mathsf{HPRG.crs}, s', 0)$.

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

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

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

- $\mathcal{S}_2(\mathsf{st}_{\mathcal{S}}, x, \mathsf{ct}, \beta)$: Parse $\mathsf{st}_{\mathcal{S}} = (\overline{\mathsf{crs}}, \mathsf{ABSFE.}\overline{\mathsf{pk}}, \mathsf{PKE.sk})$, $\overline{\mathsf{crs}} = (\mathsf{ABSFE.}\overline{\mathsf{crs}}, \mathsf{PKE.pk}, \mathsf{HPRG.crs}, \{\mathsf{Com.crs}_i\}_{i\in[\lambda]})$ and $\mathsf{ct} = \left(\mathsf{ct}_0, (c_i, \mathsf{ct}_{i,0}, \mathsf{ct}_{i,1})_{i\in[\lambda]}\right)$, and proceed as follows:

  1. If $\beta = 0$, output $\perp$.
  2. For every $i \in [\lambda]$, decrypt $\rho_i' \leftarrow \mathsf{PKE.Decrypt}(\mathsf{PKE.sk}, \mathsf{ct}_{i,1})$. If $\rho_i' \neq \perp$ and moreover, $\mathsf{Com.Commit}(\mathsf{Com.crs}_i, 1; \rho_i') = c_i$, set $s_i' = 1$; otherwise, set $s_i' = 0$.
  3. For every $i \in [\lambda]$, let $z_i' \leftarrow \mathsf{HPRG.Eval}(\mathsf{HPRG.crs}, s', i)$. If $s_i' = 0$, then check if $\mathsf{Recover}(\mathsf{ABSFE.crs}, \mathsf{ABSFE.}\overline{\mathsf{pk}}, \mathsf{ct}_{i,0}, z_i') = (x, \rho_i')$, and output $\perp$ if the check fails. If $s_i' = 1$, then check if $\mathsf{Recover}(\mathsf{PKE.pk}, \mathsf{ct}_{i,1}, z_i') = \rho_i'$, and output $\perp$ if the check fails. Finally, if $c_i \neq \mathsf{Com.Commit}(\mathsf{Com.crs}_i, s_i', \rho_i')$ for any $i \in [\lambda]$, output $\perp$.
  4. Output $\mathsf{ct}_0 \oplus \mathsf{HPRG.Eval}(\mathsf{HPRG.crs}, s', 0)$.

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

- Hyb_0: This is the real distribution where $\mathsf{crs} \leftarrow \mathsf{Aug.Setup}(1^\lambda)$, and $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Aug.KeyGen}(\mathsf{crs}, y)$. The oracle queries are handled by computing $\mathsf{Aug.Decrypt}(\mathsf{crs}, \mathsf{sk}, x, \mathsf{ct})$. At the end of the experiment, $\mathcal{A}$ outputs a bit, which is the output of the experiment, denoted $\mathsf{Hyb}_0(\mathcal{A})$.

- Hyb_1: Same as Hyb_0, except we use $\mathcal{S}_2$ to implement the decryption queries (but leaves crs, pk unchanged). Namely, in this experiment, the challenger still samples $\mathsf{crs} \leftarrow \mathsf{Aug.Setup}(1^\lambda)$ and $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Aug.KeyGen}(\mathsf{crs}, y)$. It also sets $\mathsf{st}_{\mathcal{S}} = (\mathsf{crs}, \mathsf{ABSFE.pk}, \mathsf{PKE.sk})$. When responding to the adversary's oracle queries (x, ct), the challenger computes $\mathcal{S}_2(\mathsf{st}_{\mathcal{S}}, x, \mathsf{ct}, F(x, y))$.

  We show in Lemma 5.12 that Hyb_0 and Hyb_1 are computationally indistinguishable.

- Hyb_2: Same as Hyb_1, except the simulator uses $\mathcal{S}_{\mathsf{weak}}$ to sample the components $\mathsf{ABSFE.crs}$ and $\mathsf{ABSFE.pk}$ in crs, pk, and $\mathsf{st}_{\mathcal{S}}$. This is the simulated distribution.

  This is computationally indistinguishable from Hyb_1 since ABSFE satisfies weak key-hiding.

Lemma 5.12. *If* Com *is statistically binding, then for all adversaries* $\mathcal{A}$*, the outputs of* $\mathsf{Hyb}_0(\mathcal{A})$ *and* $\mathsf{Hyb}_1(\mathcal{A})$ *are statistically indistinguishable.*

Proof. Since Hyb_0 and 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, ct) where $\mathsf{ct} = \left(\mathsf{ct}_0, \left(c_i, \mathsf{ct}_{i,0}, \mathsf{ct}_{i,1}\right)_{i\in[\lambda]}\right)$. First, if $F(x, y) = 0$, then in both Hyb_0 and 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 Hyb_0 and Hyb_1, or outputs $\perp$ in both experiments.

- Suppose the challenger sets $s'_i = 0$ in Hyb_0 (without aborting). This means the challenger obtained ρ'_i and z'_i where $\mathsf{ct}_{i,0} = \mathsf{ABSFE.Encrypt}(\mathsf{ABSFE.crs}, \mathsf{ABSFE.pk}, \rho'_i; z'_i)$ and in addition, $c_i = \mathsf{Com.Commit}(\mathsf{Com.crs}_i, 0; \rho'_i)$. Since the commitment scheme is statistically binding, with overwhelming probability over the choice of $\mathsf{Com.crs}_i$, there does not exist ρ^* such that $c_i = \mathsf{Com.Commit}(\mathsf{Com.crs}_i, 1, \rho^*)$. Hence, the challenger in Hyb_1 will also set $s'_i = 0$ (and moreover, the randomness recovery check for index i passes).
- Suppose the challenger sets $s'_i = 1$ in Hyb_0 (without aborting). This means that the challenger must have recovered ρ'_i and z'_i such that $\mathsf{ct}_{i,1} = \mathsf{PKE.Encrypt}(\mathsf{PKE.pk}, \rho'_i; z'_i)$ and $c_i = \mathsf{Com.Commit}(\mathsf{Com.crs}_i, 1; \rho'_i)$. By correctness of PKE, the challenger in Hyb_1 also sets $s'_i = 1$ (and moreover, the randomness recovery check for index i passes).

Thus, we see that if the challenger successfully recovers s' in Hyb_0, then with overwhelming probability, it successfully recovers the same s' in Hyb_1 (and applies the same set of consistency checks in both experiments). By an entirely analogous argument, the converse also holds, and we conclude that with overwhelming probability, the challenger responds identically to each decryption query in Hyb_0 and Hyb_1. The claim then follows by a union bound over the total number of decryption queries the adversary makes. □

We conclude that the outputs of Hyb_0 and Hyb_2 are computationally indistinguishable. □

Theorem 5.13 (Message-Hiding)**.** *If* ABSFE *satisfies weak message-hiding (resp., strong message-hiding),* PKE *is semantically secure,* HPRG *is secure, 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 $(\mathsf{ABSFE.SetupExt}, \mathsf{ABSFE.Ext})$ with the properties from Definition 3.4. We use these to define the extraction algorithms $(\mathsf{Aug.SetupExt}, \mathsf{Aug.Ext})$:

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

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

- Hyb_0: This is the ciphertext indistinguishability game for Aug.
- Hyb_1: Same as Hyb_0, except the challenger substitutes equivocable commitments for real commitments. Namely, at the beginning of the game, the challenger samples for each $i \in [\lambda]$, $(\mathsf{Com.}\overline{\mathsf{crs}}_i, \bar{c}_i, \mathsf{td}_i) \leftarrow \mathsf{Com.SetupEquiv}(1^\lambda)$, and uses $\mathsf{Com.}\overline{\mathsf{crs}}_i$ in place of $\mathsf{Com.crs}_i$ in the crs and $\bar{c}_i$ in place of c_i when constructing the challenge ciphertext. In addition, when constructing the challenge ciphertexts, the challenger instead computes $\rho_i \leftarrow \mathsf{Com.Equivocate}(\mathsf{td}_i, s_i, \bar{c}_i)$.

  This is computationally indistinguishable from Hyb_0 by the equivocation property of Com.

- Hyb_2: Same as Hyb_1, except that when computing the challenge ciphertext, instead of setting $M_{i,1-s_i} = \bot$, the challenger computes $\rho'_i \leftarrow \mathsf{Com.Equivocate}(\mathsf{td}_i, 1-s_i, c_i)$ and sets $M_{i,1-s_i} = \rho'_i$.

  This is computationally indistinguishable from Hyb_1 by semantic security of PKE and strong message-hiding security of ABSFE (where the challenge public key pk, the extracted value $y \leftarrow \mathsf{ABSFE.Ext}(\mathsf{td}, \mathsf{pk})$ 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, $Z_{i,1-s_i}$ is uniformly random over $\{0,1\}^\ell$, and thus $\mathsf{ct}_{i,1-s_i}$ are honestly-generated encryptions of $M_{i,1-s_i}$ under either PKE or ABSFE.

- Hyb_3: Same as Hyb_2, except that when computing the challenge ciphertext, the challenger simply sets $M_{i,b} \leftarrow \mathsf{Com.Equivocate}(\mathsf{td}_i, b, c_i)$ 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_3. Namely in Hyb_2, $M_{i,s_i} = \rho_i \leftarrow \mathsf{Com.Equivocate}(\mathsf{td}_i, s_i, c_i)$ and $M_{i,1-s_i} = \rho'_i \leftarrow \mathsf{Com.Equivocate}(\mathsf{td}_i, 1-s_i, c_i)$.

- Hyb_4: Same as Hyb_3 except when constructing the challenge ciphertext, instead of computing z_0 and $Z_{i,b}$ as in the HPRG security experiment, the challenge instead computes $z_0, Z_{i,b} \xleftarrow{\mathrm{R}} \{0,1\}^\ell$ for all $i \in [\lambda]$ and $b \in \{0,1\}$.

  This is computationally indistinguishable from Hyb_3 by security of HPRG.

Finally, any adversary's view in Hyb_4 is independent of the message m, and the claim follows. □

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 message-hiding from either the DDH or the LWE assumptions. Moreover, the resulting AB-SFE scheme has *uniformly random* public parameters, thus yielding a designated-verifier NIZK 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 DDH or the LWE assumption (with polynomial modulus and modulus-to-noise ratio). By Theorem 2.22, 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 DDH or the LWE assumption (with polynomial modulus-to-noise ratio) in the common random string model.

- By Theorem 2.26, there exists a hinting PRG under the CDH assumption or the LWE assumption (with super-polynomial modulus-to-noise ratio). Moreover, by Remark 2.27, under the DDH assumption or the LWE assumption (with super-polynomial modulus-to-noise ratio), the public parameters of the hinting PRG scheme are uniformly random.

- By Theorem 2.24, there exists a non-interactive equivocable commitment scheme from one-way functions in the common random string model.

- There exist public-key encryption schemes with pseudorandom (or uniformly random) public keys from the CDH assumption [Gam84] or the LWE assumption (with polynomial modulus-to-noise ratio) [Reg05]. Because we only use the associated secret key in the proof of security, we can replace the public key $\mathsf{PKE.pk}$ from Construction 5.9 with a uniformly random string, while maintaining security (by a standard hybrid argument) and perfect correctness.

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

Corollary 5.14 (Weak Message-Hiding, Strong Key-Hiding AB-SFE from CDH)**.** *Assuming polynomial hardness of the CDH assumption, there exists an AB-SFE scheme with that satisfies strong key-hiding and weak message-hiding.*

Corollary 5.15 (Strong Message-Hiding, Strong Key-Hiding AB-SFE from DDH or LWE)**.** *Assuming polynomial hardness of either DDH or LWE (with a super-polynomial modulus-to-noise ratio), 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 5.16 (Designated-Verifier NIZKs from CDH)**.** *Assuming polynomial hardness of CDH, 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 5.17 (Malicious-Designated-Verifier NIZKs from DDH or LWE)**.** *Assuming polynomial hardness of either DDH or LWE (with a super-polynomial modulus-to-noise ratio), 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 for many helpful discussions.

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.

[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.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *STOC*, pages 103–112, 1988.

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

[BL88] Josh Cohen Benaloh and Jerry Leichter. Generalized secret sharing and monotone functions. In *CRYPTO*, pages 27–35, 1988.

[BLP+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+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.

[CDG+17] Chongwon Cho, Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Antigoni Polychroniadou. Laconic oblivious transfer and its applications. In *CRYPTO*, pages 33–65, 2017.

[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. Towards non-interactive zero-knowledge proofs from CDH and LWE. 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. Non-interactive zero knowledge 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.

[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.

[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.

[GGH19] Sanjam Garg, Romain Gay, and Mohammad Hajiabadi. New techniques for efficient trapdoor functions and applications. In *EUROCRYPT*, 2019.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *FOCS*, pages 464–479, 1984.

[GH18] Sanjam Garg and Mohammad Hajiabadi. Trapdoor functions from the Computational Diffie-Hellman assumption. In *CRYPTO*, pages 362–391, 2018.

[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.

[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 Yaos protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.

[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.

[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.

[PW08] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. In *STOC*, pages 187–196, 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.

[RS09] Alon Rosen and Gil Segev. Chosen-ciphertext security via correlated products. In *TCC*, pages 419–436, 2009.

[SCO+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 Σ-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 aux together with the PCP π. The verifier is given the auxiliary data aux in addition to its chosen bits of π. This is just for notational convenience since we can always include aux as part of the PCP, and have the verifier always read the bits corresponding to aux.
- We allow the prover and verifier algorithms to take a common reference (or random) string crs as additional input.
- For every choice of common reference string crs, every false statement $x \notin \mathcal{L}$, every choice of auxiliary data aux, and every proof string $\pi \in \Sigma^m$, there exists exactly one setting of the randomness $r = f(\mathsf{crs}, x, \mathsf{aux})$ such that $\mathsf{zkPCP.Verify}(\mathsf{crs}, \mathsf{aux}, \mathsf{st}_x, \pi_{q_1}, \ldots, \pi_{q_\ell}) = 1$, where $(\mathsf{st}_x, q_1, \ldots, q_\ell) \leftarrow \mathsf{zkPCP.Query}(\mathsf{crs}, x)$. Note that f does not have to be efficiently-computable, but becomes efficiently computable with knowledge of a trapdoor (associated with the crs).

We define the notion more formally below:

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

- $\mathsf{zkPCP.Setup}(1^\lambda) \to \mathsf{crs}$: On input a security parameter λ, the setup algorithm outputs a common reference string crs.
- $\mathsf{zkPCP.Prove}(\mathsf{crs}, x, w) \to (\mathsf{aux}, \pi)$: On input the common reference string crs, a statement $x \in \{0,1\}^n$, and a witness $w \in \{0,1\}^h$, the prove algorithm outputs some auxiliary data aux and a proof $\pi \in \Sigma^m$.
- $\mathsf{zkPCP.Query}(\mathsf{crs}, x) \to (\mathsf{st}_x, q_1, \ldots, q_\ell)$: On input the common reference string crs, a statement $x \in \{0,1\}^n$, the query-generation algorithm outputs a verification state st_x and ℓ query indices $q_1, \ldots, q_\ell \in [m]$.
- $\mathsf{zkPCP.Verify}(\mathsf{crs}, \mathsf{st}_x, \mathsf{aux}, s_1, \ldots, s_\ell) \to \{0,1\}$: On input the common reference string crs, a verification state st_x, auxiliary proof data aux, and a set of responses $s_1, \ldots, s_\ell \in \Sigma$, the verify algorithm outputs a bit $b \in \{0,1\}$.

Moreover, zkPCP should satisfy the following properties:

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

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

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

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

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

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

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

  - **Setup:** If $b = 0$, then the challenger samples $\mathsf{crs} \leftarrow \mathsf{zkPCP.Setup}(1^\lambda)$. If $b = 1$, the challenger samples $(\mathsf{st}_{\mathcal{S}}, \mathsf{crs}) \leftarrow \mathcal{S}_1(1^\lambda)$. It gives the common reference string 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 ℓ indices $q_1, \ldots, q_\ell \in [m]$. If $\mathcal{R}(x, w) = 0$, then the challenger aborts and outputs 0. Otherwise, if $b = 0$, the challenger computes $(\mathsf{aux}, \pi) \leftarrow \mathsf{zkPCP.Prove}(\mathsf{crs}, x, w)$ and replies to $\mathcal{A}$ with $(\mathsf{aux}, \pi_{q_1}, \ldots, \pi_{q_\ell})$. If $b = 1$, the challenger replies with $(\mathsf{aux}, \pi_1, \ldots, \pi_\ell) \leftarrow \mathcal{S}_2(\mathsf{st}_{\mathcal{S}}, x, q_1, \ldots, q_\ell)$.
  - **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 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

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

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

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

  Moreover, these two algorithms satisfy the following two properties:

  - **CRS indistinguishability:** The common reference string output by $\mathsf{zkPCP.Setup}$ and $\mathsf{zkPCP.TrapGen}$ algorithms are computationally indistinguishable:

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

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

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

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

Remark A.2 (Instantiating Trapdoor Zero-Knowledge PCPs)**.** We can construct trapdoor zero-knowledge PCPs from any "instance-independent" trapdoor Σ-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 $\mathsf{dvNIZK.Setup}$ also includes the CRS for the trapdoor zero-knowledge PCP, and $\mathsf{dvNIZK.Prove}$ 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* ρ *of* $\mathsf{zkPCP.Query}$ *satisfies* $\rho = \omega(\log \lambda)$*), then* dvNIZK *from Construction 4.1 (where we modify* $\mathsf{dvNIZK.Setup}$ *to include the common reference string output by* $\mathsf{zkPCP.Setup}$ *and* $\mathsf{dvNIZK.Prove}$ *includes the auxiliary information output by* $\mathsf{zkPCP.Prove}$ *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 $i^*, j^* \xleftarrow{R} [Q]$ (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, π_i) denote the i^{th} verification query $\mathcal{A}$ makes.
- Whenever $\mathcal{A}$ makes a verification query on statement (x_i, π_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, π_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^*}, π_{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 dvNIZK with non-negligible probability without ever making a verification query on its challenge instance. □

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_0: This is the real adaptive soundness experiment. At the beginning of the game, the challenger samples $k \xleftarrow{R} \mathcal{K}$, $\mathsf{zkPCP.crs} \leftarrow \mathsf{zkPCP.Setup}(1^\lambda)$, $\mathsf{ABSFE.crs} \leftarrow \mathsf{ABSFE.Setup}(1^\lambda)$, and $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{ABSFE.KeyGen}(\mathsf{ABSFE.crs}, k)$. It gives $\mathsf{crs} = (\mathsf{zkPCP.crs}, \mathsf{ABSFE.crs})$ and $\mathsf{pk} = \mathsf{pk}'$ to $\mathcal{A}$ and sets $\mathsf{sk} = (k, \mathsf{sk}')$. When the adversary makes a verification query (x, π), the challenger replies with $\mathsf{dvNIZK.Verify}(\mathsf{crs}, \mathsf{sk}, x, \pi)$. At the end of the experiment, the adversary outputs a statement x^* and a proof π^*. The output of the experiment is 1 if $x^* \notin \mathcal{L}$ and $\mathsf{dvNIZK.Verify}(\mathsf{crs}, \mathsf{sk}, x^*, \pi^*) = 1$ *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_1: Same as Hyb_0, except at the end of the experiment, after the adversary outputs its statement x^* and proof $\pi^* = (\mathsf{aux}^*, \mathsf{ct}_1^*, \ldots, \mathsf{ct}_m^*)$, the output of the experiment is 1 if $f(\mathsf{crs}, x^*, \mathsf{aux}^*) = \mathsf{PRF}(k, x^*)$ (and the adversary makes no oracle queries on x^*).

  We claim that for all adversaries $\mathcal{A}$, $\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] \leq \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]$. By construction, Hyb_0 and Hyb_1 only differ in how the output is computed. Suppose $\mathsf{Hyb}_0(\mathcal{A}) = 1$. We show that $\mathsf{Hyb}_1(\mathcal{A})$ also outputs 1 in this case. If $\mathsf{Hyb}_0(\mathcal{A}) = 1$, then it must be the case that $x^* \notin \mathcal{L}$ and $\mathsf{dvNIZK.Verify}(\mathsf{crs}, \mathsf{sk}, x^*, \pi^*) = 1$. By construction, the verification algorithm first computes $(\mathsf{st}_{x^*}, q_1, \ldots, q_\ell) \leftarrow \mathsf{zkPCP.Query}(x^*; \mathsf{PRF}(k, x^*))$, decrypts $\mathsf{ct}_{q_1}^*, \ldots, \mathsf{ct}_{q_\ell}^*$ to obtain values $s_1, \ldots, s_\ell \in \Sigma$, and finally outputs $\mathsf{zkPCP.Verify}(\mathsf{zkPCP.crs}, \mathsf{st}_{x^*}, \mathsf{aux}^*, s_1, \ldots, s_\ell)$. Since $x^* \notin \mathcal{L}$, by soundness of zkPCP, there is only one setting of the randomness to $\mathsf{zkPCP.Query}$ that can cause $\mathsf{zkPCP.Verify}$ to accept: namely, if $\mathsf{PRF}(k, x^*) = f(\mathsf{crs}, x^*, \mathsf{aux}^*)$. In this case, the output in Hyb_1 is also 1, and the claim follows.

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

- Hyb_2: Same as Hyb_1, except the challenger computes $(\mathsf{zkPCP.crs}, \mathsf{td}) \leftarrow \mathsf{zkPCP.TrapGen}$ when constructing crs and it uses $\mathsf{zkPCP.BadRandomness}(\mathsf{zkPCP.crs}, \mathsf{td}, x^*, \mathsf{aux}^*)$ to compute $f(\mathsf{crs}, x^*, \mathsf{aux}^*)$ when computing the output of the experiment. In particular, the output of this experiment can be *efficiently* computed.

  This is computationally indistinguishable from Hyb_1 by the trapdoor-generation properties of zkPCP.

- Hyb_3: Same as Hyb_2, except the challenger uses the simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ for the strong key-hiding game for ABSFE to construct $\mathsf{ABSFE.crs}$, the public key pk, and to implement the decryption algorithm when responding to the verification queries. In particular, this is the analog of Hyb_1 in the proof of Theorem 4.3.

  As in the proof of Theorem 4.3, this is indistinguishable from Hyb_2 from by strong key-hiding security of ABSFE.

- Hyb_4: Same as Hyb_3, except the challenger samples a random function $g \xleftarrow{R} \mathsf{Funs}[\{0,1\}^n, \{0,1\}^\rho]$ at the beginning of the experiment. Whenever it needs to compute $\mathsf{PRF}(k, x)$, it instead computes $g(x)$. This is the analog of Hyb_2 in the proof of Theorem 4.3.

  As in the proof of Theorem 4.3, this is indistinguishable from Hyb_3 by security of PRF.

In hybrid Hyb_4, the output is 1 only if $g(x^*) = \mathsf{zkPCP.BadRandomness}(\mathsf{zkPCP.crs}, \mathsf{td}, x^*, \mathsf{aux}^*)$. Since the function g is uniformly random (and in particular, independent of $\mathsf{zkPCP.BadRandomness}$) and the adversary makes no oracle calls on x^* (thus revealing no information about $g(x^*)$ in the experiment), this relation holds with probability $1/2^\rho = \mathsf{negl}(\lambda)$. Thus, with all but negligible probability, the output of $\mathsf{Hyb}_4(\mathcal{A})$ will be 0, and the claim follows. □

B Receiver-Extractable OT from DDH or LWE

In this section, we prove 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 $(\mathsf{SetupMessy}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{FindMessy})$ with the following syntax:

- $\mathsf{SetupMessy}(1^\lambda) \to (\mathsf{crs}, \mathsf{td})$: Given the security parameter λ, the setup algorithm outputs a common reference string crs along with a trapdoor td.

- $\mathsf{KeyGen}(\mathsf{crs}, b) \to (\mathsf{pk}, \mathsf{sk})$: Given a reference string crs and a branch $b \in \{0,1\}$, the key-generation algorithm outputs a public key pk and a secret key sk.
- $\mathsf{Encrypt}(\mathsf{crs}, \mathsf{pk}, b, m) \to \mathsf{ct}$: Given a reference string crs, a public key pk, a branch $b \in \{0,1\}$ and a message $m \in \{0,1\}^k$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Decrypt}(\mathsf{crs}, \mathsf{sk}, \mathsf{ct}) \to m$: Given a reference string crs, a secret key sk and a ciphertext ct, the decryption algorithm outputs a message m.
- $\mathsf{FindMessy}(\mathsf{td}, \mathsf{pk}) \to b$: Given a trapdoor td and a (possibly malformed) public key 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 $b \in \{0,1\}$:

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

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

$$\mathsf{Encrypt}(\mathsf{crs}, \mathsf{pk}, b, m_0) \overset{s}{\approx} \mathsf{Encrypt}(\mathsf{crs}, \mathsf{pk}, b, m_1),$$

  where $b \leftarrow \mathsf{FindMessy}(\mathsf{td}, \mathsf{pk})$.
- **Key-hiding**: The following distributions are computationally indistinguishable:

$$(\mathsf{crs}, \mathsf{pk}_0) \overset{c}{\approx} (\mathsf{crs}, \mathsf{pk}_1),$$

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

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* $\mathsf{SetupMessy}$ *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 $(\mathsf{crs}', \mathsf{pk}', \mathsf{sk}'_0, \mathsf{sk}'_1) \leftarrow \mathsf{SetupDec}(1^\lambda)$ such that for all $b \in \{0,1\}$,

$$(\mathsf{crs}, \mathsf{pk}, \mathsf{sk}_b) \overset{c}{\approx} (\mathsf{crs}', \mathsf{pk}', \mathsf{sk}'_b),$$

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

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. □

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 λ be a security parameter, and $(\mathsf{SetupMessy}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{FindMessy})$ be a key-hiding messy encryption scheme. Define the following OT scheme $\mathsf{OT} = (\mathsf{OT.Setup}, \mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT.Receive})$:

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

Claim B.4. *Assuming* $(\mathsf{SetupMessyKeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{FindMessy})$ *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 extratability against malicious receivers, we use the trapdoor identification of a messy branch property and define $\mathsf{OT.SetupExt}$ and $\mathsf{OT.Ext}$ as follows:

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

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

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+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 $\lfloor x \rceil_2$ to denote the modular rounding function [BPR12] that outputs $\lfloor 2/q \cdot x \rceil \in \mathbb{Z}_2$.

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 λ 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 $\mathsf{LWE}_{n,m,q,\chi}$ states that for $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times m}$, $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$, $\mathbf{e} \leftarrow \chi^m$, and $\mathbf{u} \xleftarrow{R} \mathbb{Z}_q^m$, the following two distributions are computationally indistinguishable:

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

When the error distribution χ is B-bounded (oftentimes, a discrete Gaussian distribution), and under mild conditions on the modulus q, the $\mathsf{LWE}_{n,m,q,\chi}$ 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+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 $\mathbf{g} = (1, 2, 4, \ldots, 2^{\lceil \log q \rceil - 1})$ is the powers-of-two vector. We define the bit-decomposition function $\mathbf{G}^{-1} \colon \mathbb{Z}_q^{n \times m} \to \mathbb{Z}_q^{n\lceil \log q \rceil \times m}$ which expands each entry $x \in \mathbb{Z}_q$ in the input matrix into a column of 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+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 λ, lattice parameters n, m, q, β, where $m = O(n \log q)$. Then, there exists a tuple of efficient algorithms* $(\mathsf{TrapGen}, \mathsf{Invert}, \mathsf{SampleLeft}, \mathsf{SampleRight})$*:*

- $\mathsf{TrapGen}(1^\lambda) \to (\mathbf{A}, \mathbf{R})$*: On input the security parameter λ, the trapdoor generation algorithm outputs a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and a gadget trapdoor $\mathbf{R} \in \mathbb{Z}_q^{m \times m}$.*

- $\mathsf{Invert}(\mathbf{A}, \mathbf{R}, \mathbf{v}) \to \mathbf{s}/\perp$: *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* $\perp$*.*
- $\mathsf{SampleLeft}(\mathbf{A}, \mathbf{B}, \mathbf{R}, \mathbf{V}, \beta) \to \mathbf{U}$: *On input matrices* $\mathbf{A}, \mathbf{B} \in \mathbb{Z}_q^{n \times m}$*, a matrix* $\mathbf{R} \in \mathbb{Z}_q^{m \times m}$ *(a gadget trapdoor of* $\mathbf{A}$*), a target matrix* $\mathbf{V} \in \mathbb{Z}_q^{n \times m}$*, and a norm bound* β*,* SampleLeft *returns a matrix* $\mathbf{U} \in \mathbb{Z}_q^{2m \times m}$*.*
- $\mathsf{SampleRight}(\mathbf{A}, \mathbf{B}, \mathbf{T}, \mathbf{V}, \beta) \to \mathbf{U}$: *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* β*,* SampleRight *returns a matrix* $\mathbf{U} \in \mathbb{Z}_q^{2m \times m}$*.*

Moreover, the above algorithms satisfy the following properties:

- *If we sample* $(\mathbf{A}, \mathbf{R}) \leftarrow \mathsf{TrapGen}(1^\lambda)$*, then* $\mathbf{A} \overset{s}{\approx} \mathsf{Uniform}(\mathbb{Z}_q^{n \times m})$*,* $\mathbf{R}$ *is full-rank,* $\mathbf{AR} = \mathbf{G}$*, and* $\|\mathbf{R}\| \leq \beta$*.*
- *If* $\mathbf{a} = \mathbf{s}^\mathsf{T}\mathbf{A} + \mathbf{e}^\mathsf{T}$ *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*

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

- *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* $\mathbf{U}_0 \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{B}, \mathbf{R}, \mathbf{V}, \beta_2)$*, 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* $\mathbf{U}_1 \leftarrow \mathsf{SampleRight}(\mathbf{A}, \mathbf{B}, \mathbf{T}, \mathbf{V}, \beta_2)$*, 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* $\mathbf{R} \xleftarrow{\mathrm{R}} \{\pm 1\}^{m \times k}$ *where* $k = k(n)$ *is polynomial in* n*. Sample* $\mathbf{A} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times m}$ *and* $\mathbf{B} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times k}$*. Then, for all vectors* $\mathbf{e} \in \mathbb{Z}_q^m$*, the following two distributions are statistically indistinguishable:* $(\mathbf{A}, \mathbf{AR}, \mathbf{R}^\mathsf{T}\mathbf{e})$ *and* $(\mathbf{A}, \mathbf{B}, \mathbf{R}^\mathsf{T}\mathbf{e})$*.*

Matrix embeddings. We review the matrix encoding scheme from Boneh et al. [BGG+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 $x_1, \ldots, x_\rho \in \{0,1\}$ into matrices $\mathbf{A}_1, \ldots, \mathbf{A}_\rho \in \mathbb{Z}_q^{n \times m}$. 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+14])**.** *Fix a security parameter* λ*, and lattice parameters* n, m, q*. Then, there exist a pair of efficiently-computable algorithms* $(\mathsf{Eval}_{\mathsf{pk}}, \mathsf{Eval}_{\mathsf{ct}})$ *with the following syntax:*

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

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

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

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

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

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

  for an error vector $\|\mathbf{e}_{C,x}\| \le B(\beta, m, d) = \beta \cdot m^{O(d)}$*.*
- *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* $\|\mathbf{R}_i\| \le \beta$ *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* $(C, x, \mathbf{A}, \mathbf{R}_1, \ldots, \mathbf{R}_\ell)$ *and* $\|\mathbf{R}_C\| \le B(\beta, m, d) = \beta \cdot m^{O(d)}$*.*

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+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 via hinting PRGs. We recall the construction here:

Construction C.5 (Weak Function-Hiding ABE [BGG+14, adapted])**.** Let λ be a security parameter and let (n, m, q, χ) be lattice parameters, where χ is a β-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 $\mathcal{X} \to \{0,1\}$. In the following, we will use the convention that a secret key sk_C for a circuit C should successfully decrypt a ciphertext ct with attribute x if $C(x) = 0$. Then, the ABE scheme $\Pi_{\mathsf{ABE}} = (\mathsf{ABE.Setup}, \mathsf{ABE.KeyGen}, \mathsf{ABE.Encrypt}, \mathsf{ABE.Decrypt})$ with attribute space $\mathcal{X}$, message space $\mathcal{M}$, and function class $\mathcal{C}$ is defined as follows:

- $\mathsf{ABE.Setup}(1^\lambda)$: On input the security parameter λ, the setup algorithm samples $(\mathbf{A}, \mathbf{R}) \leftarrow \mathsf{TrapGen}(1^\lambda)$, $\mathbf{A}_1, \ldots, \mathbf{A}_\ell \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times m}$ and $\mathbf{u} \leftarrow \mathbb{Z}_q^n$. It outputs the public parameters $\mathsf{pp} = (\mathbf{A}, \mathbf{A}_1, \ldots, \mathbf{A}_\ell, \mathbf{u})$ and the master secret key $\mathsf{msk} = \mathbf{R}$.

- $\mathsf{ABE.KeyGen}(\mathsf{pp}, \mathsf{msk}, C)$: On input the public parameters $\mathsf{pp} = (\mathbf{A}, \mathbf{A}_1, \ldots, \mathbf{A}_\ell, \mathbf{u})$, the master secret key $\mathsf{msk} = \mathbf{R}$ and a circuit $C\colon \{0,1\}^\ell \to \{0,1\}$, the key-generation algorithm computes $\mathbf{A}_C \leftarrow \mathsf{Eval}_{\mathsf{pk}}(C, \mathbf{A}_1, \ldots, \mathbf{A}_\ell)$ and samples $\mathbf{R}_C \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{A}_C, \mathbf{R}, \mathbf{G}, \beta_2)$. It outputs the secret key $\mathsf{sk} = (C, \mathbf{R}_C)$.

- $\mathsf{ABE.Encrypt}(\mathsf{pp}, x, m)$: On input the public parameters $\mathsf{pp} = (\mathbf{A}, \mathbf{A}_1, \ldots, \mathbf{A}_\ell, \mathbf{u})$, an attribute $x \in \{0,1\}^\ell$, and a message $m \in \{0,1\}$, the encryption algorithm samples $\mathbf{s} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$, $\mathbf{e} \leftarrow \chi^m$, $e_u \leftarrow \chi$, and $\mathbf{R}_1, \ldots, \mathbf{R}_\ell \xleftarrow{\mathrm{R}} \{\pm 1\}^{m \times m}$. It computes $\mathbf{a} \leftarrow \mathbf{s}^\mathsf{T}\mathbf{A} + \mathbf{e}^\mathsf{T}$, $\mathbf{a}_i \leftarrow \mathbf{s}^\mathsf{T}(\mathbf{A}_i + x_i \cdot \mathbf{G}) + \mathbf{e}^\mathsf{T}\mathbf{R}_i$ for each $i \in [\ell]$, and $a_u \leftarrow \mathbf{s}^\mathsf{T}\mathbf{u} + m \cdot \lfloor q/2 \rceil + e_u$, and outputs the ciphertext $\mathsf{ct} = (x, \mathbf{a}, \mathbf{a}_1, \ldots, \mathbf{a}_\ell, a_u)$.

- $\mathsf{ABE.Decrypt}(\mathsf{pp}, \mathsf{sk}, \mathsf{ct})$: On input the public parameters $\mathsf{pp} = (\mathbf{A}, \mathbf{A}_1, \ldots, \mathbf{A}_\ell, \mathbf{u})$, a secret key sk, and a ciphertext $\mathsf{ct} = (x, \mathbf{a}, \mathbf{a}_1, \ldots, \mathbf{a}_\ell, a_u)$, the decryption algorithm first uses the trapdoor (in the secret key sk) to try and recover part of the encryption randomness $\mathbf{s} \in \mathbb{Z}_q^n$:

  - If $\mathsf{sk} = \mathbf{R} \in \mathbb{Z}_q^{m \times m}$ (i.e., sk has the structure of the master secret key), then the decryption algorithm computes $\mathbf{s} \leftarrow \mathsf{Invert}(\mathbf{A}, \mathbf{R}, \mathbf{a})$.
  - If $\mathsf{sk} = (C, \mathbf{R}_C)$ for some $C \in \mathcal{C}$ and $\mathbf{R}_C \in \mathbb{Z}_q^{m \times m}$, (i.e., 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 $\mathbf{A}_C \leftarrow \mathsf{Eval}_{\mathsf{pk}}(C, \mathbf{A}_1, \ldots, \mathbf{A}_\ell)$, $\mathbf{a}_{C,x} \leftarrow \mathsf{Eval}_{\mathsf{ct}}(C, x, \mathbf{A}_1, \ldots, \mathbf{A}_\ell, \mathbf{a}_1, \ldots, \mathbf{a}_\ell)$ and finally $\mathbf{s} \leftarrow \mathsf{Invert}([\mathbf{A} \mid \mathbf{A}_C], \mathbf{R}_C, [\mathbf{a} \mid \mathbf{a}_{C,x}])$.
  - If 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 $m \leftarrow \lfloor a_u - \mathbf{s}^\mathsf{T}\mathbf{u} \rceil_2$. Next, the decryption algorithm applies the following verification procedure:

  1. Compute $\mathbf{e}^\mathsf{T} \leftarrow \mathbf{a} - \mathbf{s}^\mathsf{T}\mathbf{A}$, $\mathbf{e}_i^\mathsf{T} \leftarrow \mathbf{a}_i - \mathbf{s}^\mathsf{T}(\mathbf{A}_i + x_i\mathbf{G})$, and $e_u \leftarrow a_u - \mathbf{s}^\mathsf{T}\mathbf{u} - m \cdot \lfloor q/2 \rceil$.
  2. Check that $\|\mathbf{e}\| \le \beta_1$, $\|\mathbf{e}_i\| \le \beta_1$ for all $i \in [\ell]$, and $|e_u| \le \beta_1$. 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+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 $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{ABE.Setup}(1^\lambda)$, and all strings $\mathsf{ct} \in \{0,1\}^t$, the following properties hold:

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

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

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

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 $\mathcal{S}^{f(\cdot)}(\mathsf{pp}, \mathsf{msk}, \mathsf{ct})$ as follows:

1. On input the public parameters pp, the master secret key msk, and a ciphertext ct, the simulator computes $(x, m) \leftarrow \mathsf{ABE.Decrypt}(\mathsf{pp}, \mathsf{msk}, \mathsf{ct})$. If $\mathsf{ABE.Decrypt}(\mathsf{pp}, \mathsf{msk}, \mathsf{ct})$ 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 $\mathsf{ct} \in \{0,1\}^t$, $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{ABE.Setup}(1^\lambda)$, and $\mathsf{sk}_f \leftarrow \mathsf{ABE.KeyGen}(\mathsf{pp}, \mathsf{msk}, f)$. We consider two possibilities:

- Suppose $\mathsf{ABE.Decrypt}(\mathsf{pp}, \mathsf{sk}_f, \mathsf{ct}) = (x, m)$ for some $x \in \mathcal{X}$ and $m \in \mathcal{M}$. By consistency of ABE, this means that $f(x) = 1$ and $\mathsf{ABE.Decrypt}(\mathsf{pp}, \mathsf{msk}, \mathsf{ct}) = (x, m)$. By construction, the simulator outputs (x, m) in this case.
- Suppose $\mathsf{ABE.Decrypt}(\mathsf{pp}, \mathsf{sk}_f, \mathsf{ct}) = \perp$. If $\mathsf{ABE.Decrypt}(\mathsf{pp}, \mathsf{msk}, \mathsf{ct}) = \perp$, then the simulator also outputs $\perp$. Suppose instead that $\mathsf{ABE.Decrypt}(\mathsf{pp}, \mathsf{msk}, \mathsf{ct}) = (x, m)$ for some $x \in \mathcal{X}$ and $m \in \mathcal{M}$. We consider two possibilities. If $f(x) = 1$, then by consistency of ABE, $\mathsf{ABE.Decrypt}(\mathsf{pp}, \mathsf{sk}_f, \mathsf{ct}) = (x, m)$, 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 $\mathsf{ct} \in \{0,1\}^t$ is distributed identically as the output of the real decryption algorithm, weak function hiding holds. □

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

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

Let $\mathsf{ct} = (x, \mathbf{a}, \mathbf{a}_1, \ldots, \mathbf{a}_\ell, a_u)$*. 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:*

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

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

Proof. We first analyze the randomness recovery step in the $\mathsf{ABE.Decrypt}$ function:

- Since $(\mathbf{A}, \mathbf{R})$ is sampled using $\mathsf{TrapGen}$, 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 $\mathsf{Invert}(\mathbf{A}, \mathbf{R}, \mathbf{a}) = \mathbf{s}$.
- Take $C \in \mathcal{C}$ where $C(x) = 0$ and let $\mathsf{sk}_C = (C, \mathbf{R}_C) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, \mathsf{msk}, C)$, $\mathbf{A}_C \leftarrow \mathsf{Eval}_{\mathsf{pk}}(C, \mathbf{A}_1, \ldots, \mathbf{A}_\ell)$, $\mathbf{a}_{C,x} \leftarrow \mathsf{Eval}_{\mathsf{ct}}(C, x, \mathbf{A}_1, \ldots, \mathbf{A}_\ell, \mathbf{a}_1, \ldots, \mathbf{a}_\ell)$. By definition of KeyGen, we have that $\mathbf{R}_C \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{A}_C, \mathbf{R}, \mathbf{G}, \beta_2)$. 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 $\mathbf{a}_{C,x} = \mathbf{s}^\mathsf{T}\mathbf{A}_C + \mathbf{e}_{C,x}^\mathsf{T}$, where $\|\mathbf{e}_{C,x}\| \leq B(\beta_1, m, d)$. This means that $[\mathbf{a} \mid \mathbf{a}_{C,x}] = \mathbf{s}^\mathsf{T}[\mathbf{A} \mid \mathbf{A}_C] + [\mathbf{e}^\mathsf{T} \mid \mathbf{e}_{C,x}^\mathsf{T}]^\mathsf{T}$. 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 $\mathsf{Invert}([\mathbf{A} \mid \mathbf{A}_C], \mathbf{R}_C, [\mathbf{a} \mid \mathbf{a}_{C,x}]) = \mathbf{s}$.

Next, $a_u - \mathbf{s}^\mathsf{T}\mathbf{u} = m \cdot \lfloor q/2 \rceil + e_u$ and since $|e_u| \leq \beta_1 < q/4$, it follows that $\lfloor a_u - \mathbf{s}^\mathsf{T}\mathbf{u} \rceil_2 = m$. Finally, by assumption, all of the error terms are bounded by β_1 by assumption. Thus, $\mathsf{ABE.Decrypt}(\mathsf{pp}, \mathsf{msk}, \mathsf{ct}) = (x, m)$ and $\mathsf{ABE.Decrypt}(\mathsf{pp}, \mathsf{sk}_C, \mathsf{ct}) = (x, m)$. □

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

D Function-Hiding ABE from Circular-Secure Encryption (via Lossy Trapdoor Functions)

In this section, we informally discuss an alternative approach to constructing function-hiding (single-key) ABE based on some form of trapdoor functions and circular-secure encryption.

To recall, there is a natural candidate construction of a function-hiding ABE scheme fhABE using two ingredients: (1) a single-key ABE scheme $\mathsf{ABE} = (\mathsf{ABE.Setup}, \mathsf{ABE.KeyGen}, \mathsf{ABE.Encrypt}, \mathsf{ABE.Decrypt})$, and (2) a secret-key encryption scheme $\mathsf{SKE} = (\mathsf{SKE.Setup}, \mathsf{SKE.Encrypt}, \mathsf{SKE.Decrypt})$. The candidate scheme is defined as follows: to encrypt a message m using public parameters pp and attribute x, sample a SKE-secret key s and output

$$\mathsf{ct} = \Big(\mathsf{ct}_1 := \mathsf{ABE.Encrypt}(\mathsf{pp}, x, s; r), \mathsf{ct}_2 := \mathsf{SKE.Encrypt}(s, m\|r)\Big),$$

where r denotes the randomness used to encrypt s. To decrypt using a secret key sk_f, first decrypt ct_1 to obtain s; then, decrypt ct_2 to obtain $m\|r$ and output m if and only if $\mathsf{ct}_1 = \mathsf{ABE.Encrypt}(\mathsf{pp}, x, s; r)$.

In fact, for *any* choice of ABE and SKE, the scheme fhABE is (weak) function-hiding. Indeed, any secret key sk_f qualified to decrypt ct obtains a certificate that ct_1 is well-formed; this implies that any other qualified $\mathsf{sk}_{f'}$ will successfully decrypt ct (and obtain the same plaintext), which is

sufficient to imply function-hiding. In particular, fhABE is consistent (Definition C.6), which as shown in Lemma C.7, implies (weak) function-hiding.

However, it is unclear how to show that fhABE is (in general) semantically secure—indeed, proving this would imply a construction of DV-NIZK (and hence CCA-secure public key encryption) from any public-key encryption scheme.

On the other hand, we can prove that fhABE is secure when special-purpose instantiations of ABE and SKE are used. In particular, letting $\mathsf{TDF} = \{F_{\mathsf{pk}}, (\mathsf{pk}, \mathsf{tk}) \leftarrow \mathsf{KeyGen}(1^n)\}$ denote a family of trapdoor functions, we consider the following implementation of an ABE scheme for formulas of a bounded depth $d = O(\log(\lambda))$.

- $\mathsf{ABE.Setup}(1^\lambda)$: Sample D TDF pairs $(\mathsf{pk}_i, \mathsf{tk}_i)_{i\in[D]} \leftarrow \mathsf{KeyGen}(1^n)$ (for $D = \mathsf{poly}(\lambda)$ and $n = \mathsf{poly}(\lambda)$ to be chosen later) and output $(\mathsf{pp} = (\mathsf{pk}_i), \mathsf{msk} = (\mathsf{tk}_i))$.
- $\mathsf{ABE.Encrypt}(\mathsf{pp}, x, m)$: Compute a secret sharing $(s_i \in \{0,1\}^n)_{i\in\mathsf{gates}(U_x)}$ of m (using the information-theoretic scheme of [BL88]) according to the access structure U_x, a universal formula (monotone except on the leaves) for evaluation of formulas of depth d. Letting $D = |\mathsf{gates}(U_x)|$, output $(\mathsf{ct}_i = F_{\mathsf{pk}_i}(s_i))_{i\in[D]}$.
- $\mathsf{ABE.KeyGen}(\mathsf{msk}, f)$: Outputs $(\mathsf{tk}_i)_{i:f_i=1}$, where f_i denotes the ith bit of the binary representation of the function f that is compatible with U_x.
- $\mathsf{ABE.Decrypt}(\mathsf{sk}_f, \mathsf{ct})$: Invert each ct_i for which $f_i = 1$, using td_i, to obtain s_i. Then, reconstructs m using the collection $(s_i)_{i:f_i=1}$, and output $\perp$ if reconstruction fails.

The scheme ABE as defined above does not satisfy standard semantic security. However, it can be shown to satisfy a notion of security for messages m chosen uniformly at random, provided that TDF is *correlated-input secure* [RS09].

Lemma D.1 (Informal)**.** *Suppose that for all $N \le D$, TDF satisfies the property that for all collections of affine offsets $(a_i)_{i\in[N]}$, and for all $(\mathsf{pk}_i, \mathsf{td}_i) \leftarrow \mathsf{TDF.KeyGen}(1^n)$, and for $m \xleftarrow{R} \{0,1\}^n$, it is computationally hard to recover m given*

$$(F_{\mathsf{pk}_i}(m + a_i))_{i\in[N]}.$$

Then, it is computationally hard to recover m given $(\mathsf{pp}, \mathsf{ABE.Encrypt}(\mathsf{pp}, x, m), \mathsf{sk}_f)$, provided that $f(x) = 0$ and m is sampled uniformly at random.

The above notion of correlated input security follows, for example, from sufficiently lossiness of a TDF family [PW08]. In fact, this property also suffices to show that the scheme fhABE above is semantically secure, provided that the scheme SKE is circular-secure[5] in the presence of a small amount of leakage.

Theorem D.2 (Informal)**.** *Suppose that TDF with security parameter $n = n(\lambda)$ is lossy [PW08] with lossiness parameter $\frac{n(\lambda)}{\lambda D(\lambda)}$ and SKE is KDM-secure for $\mathbb{F}_2$-affine functions of the secret key in the presence of leakage with rate $1/\lambda$. Then, the scheme fhABE as implemented above is semantically secure.*

[5]Technically, we require KDM-security for affine functions of the secret key.

Theorem D.2 allows for the instantiation of function-hiding (single-key) ABE from the DDH or LWE assumptions using the lossy trapdoor functions of [PW08] and the encryption schemes of [BHHO08, BLSV18]. Moreover, by assuming slightly stronger properties on SKE, one can make do with a form of *correlated-input secure trapdoor functions* [RS09] instead of lossy trapdoor functions. This allows for a CDH-based instantiation of TDF using the trapdoor functions of [GGH19].

With the above instantiations, the scheme fhABE can be combined with any PRF in NC^1 (which exists under either DDH or LWE) to obtain a DV-NIZK. Ingredients such as lossy TDFs and correlated-input secure TDFs are reminiscent of constructions of CCA-secure encryption [PW08, RS09], which is unsurprising given the connection between DV-NIZK and CCA-secure encryption [NY90].

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

- $\mathsf{Aug.Setup}(1^\lambda)$: Sample $(\mathsf{ABE.pp}, \mathsf{ABE.msk}) \leftarrow \mathsf{ABE.Setup}(1^\lambda)$, $\mathsf{crs} \leftarrow \mathsf{dvNIZK.Setup}(1^\lambda)$, and $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{dvNIZK.KeyGen}(\mathsf{crs})$. Output the public parameters $\mathsf{pp} = (\mathsf{ABE.pp}, \mathsf{crs}, \mathsf{pk})$ and the master secret key $\mathsf{msk} = (\mathsf{ABE.msk}, \mathsf{sk})$.
- $\mathsf{Aug.KeyGen}(\mathsf{msk}, f)$: Parse $\mathsf{msk} = (\mathsf{ABE.msk}, \mathsf{sk})$, compute $\mathsf{ABE.sk}_f \leftarrow \mathsf{ABE.KeyGen}(\mathsf{ABE.msk}, f)$, and output $\mathsf{sk}_f = (\mathsf{ABE.sk}_f, \mathsf{sk})$.
- $\mathsf{Aug.Encrypt}(\mathsf{pp}, x, m)$: On input the public parameters $\mathsf{pp} = (\mathsf{ABE.pp}, \mathsf{crs}, \mathsf{pk})$, an attribute x, and a message m, proceed as follows:
  - Compute $\mathsf{ABE.ct} \leftarrow \mathsf{ABE.Encrypt}(\mathsf{ABE.pp}, x, m; \rho)$ for ρ sampled uniformly at random.

– Compute $\pi \leftarrow \mathsf{dvNIZK.Prove}(\mathsf{crs}, \mathsf{pk}, \mathbf{x}, (m, \rho))$, where $\mathbf{x} = (\mathsf{ABE.pp}, \mathsf{ABE.ct}, x)$ is the statement

$$\exists\, (m', \rho') : \mathsf{ABE.ct} = \mathsf{ABE.Encrypt}(\mathsf{ABE.pp}, x, m'; \rho'), \tag{E.1}$$

and (m, ρ) is the witness for this statement.

– Output the ciphertext $\mathsf{ct} = (x, \mathsf{ABE.ct}, \pi)$.

- $\mathsf{Aug.Decrypt}(\mathsf{pp}, \mathsf{sk}_f, \mathsf{ct})$: On input the public parameters $\mathsf{pp} = (\mathsf{ABE.pp}, \mathsf{crs}, \mathsf{pk})$, the decryption key $\mathsf{sk}_f = (\mathsf{ABE.sk}_f, \mathsf{sk})$, and the ciphertext $\mathsf{ct} = (x, \mathsf{ABE.ct}, \pi)$, proceed as follows:

  – Call $\mathsf{dvNIZK.Verify}(\mathsf{crs}, \mathsf{sk}, \mathbf{x}, \pi)$ where $\mathbf{x} = (\mathsf{ABE.pp}, \mathsf{ABE.ct}, x)$ is the statement from Eq. (E.1). Output $\perp$ if this verification fails.

  – Output $\mathsf{ABE.Decrypt}(\mathsf{ABE.pp}, \mathsf{ABE.sk}_f, \mathsf{ABE.ct})$.

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 E.3. *If* ABE *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

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

where $(\mathsf{pp}, \mathsf{sk}_f)$ 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 dvNIZK. First, we define a hybrid setup algorithm $\mathsf{HybridSetup}(1^\lambda)$ that outputs $\widetilde{\mathsf{pp}} = (\mathsf{ABE.pp}, \widetilde{\mathsf{crs}}, \widetilde{\mathsf{pk}})$ and $\widetilde{\mathsf{msk}} = (\mathsf{ABE.msk}, \widetilde{\mathsf{sk}})$ for $(\mathsf{st}_{\mathcal{S}}, \widetilde{\mathsf{crs}}, \widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) \leftarrow \mathcal{S}_1(1^\lambda)$. We also define hybrid encryption algorithm $\mathsf{HybridEncrypt}(\mathsf{pp}, x, m)$ defined as follows:

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

Now, by zero-knowledge of dvNIZK, we have that for all (f, x, m),

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

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

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

by the semantic security of ABE (as HybridEncrypt ciphertexts can be produced in a black-box way from ABE ciphertexts), and the claim follows. □

Lemma E.4. *If* dvNIZK *is adaptively-sound, then* Aug *is weakly function-hiding.*

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

1. On input the public parameters $\mathsf{pp} = (\mathsf{ABE.pp}, \mathsf{crs}, \mathsf{pk})$, the master secret key $\mathsf{msk} = (\mathsf{ABE.msk}, \mathsf{sk})$, and a ciphertext $\mathsf{ct} = (x, \mathsf{ABE.ct}, \pi)$, the simulator calls $\mathsf{dvNIZK.Verify}(\mathsf{crs}, \mathsf{sk}, \mathbf{x}, \pi)$ 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 $\mathsf{ABE.Decrypt}(\mathsf{ABE.pp}, \mathsf{ABE.msk}, \mathsf{ct})$.

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 $\mathsf{Aug.Decrypt}(\mathsf{pp}, \mathsf{sk}_f, \cdot)$ with key sk_f and oracle access to $\mathcal{S}^{f(\cdot)}(\mathsf{pp}, \mathsf{msk}, \cdot)$. This means that (with non-negligible probability) $\mathcal{A}$ must make an oracle query on a ciphertext $\mathsf{ct} = (x, \mathsf{ABE.ct}, \pi)$ of the following form:

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

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

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

1. On input $(\mathsf{crs}, \mathsf{pk})$, algorithm $\mathcal{B}$ samples $(\mathsf{ABE.pp}, \mathsf{ABE.msk}) \leftarrow \mathsf{ABE.Setup}(1^\lambda)$, $\mathsf{ABE.sk}_f \leftarrow \mathsf{ABE.KeyGen}(\mathsf{ABE.msk}, f)$, and constructs $(\mathsf{pp}, \mathsf{msk})$ as in Construction E.2.

2. Choose a random index $i^* \xleftarrow{R} [Q]$, where Q is a bound on the number of queries $\mathcal{A}$ makes. Let $\mathsf{ct}_i = (x_i, \mathsf{ABE.ct}_i, \pi_i)$ be the i^{th} decryption query $\mathcal{A}$ makes.

3. Start running algorithm $\mathcal{A}$ on input $\mathsf{pp} = (\mathsf{ABE.pp}, \mathsf{crs}, \mathsf{pk})$. Whenever $\mathcal{A}$ makes a decryption query on a ciphertext $\mathsf{ct}_i = (x_i, \mathsf{ABE.ct}_i, \pi_i)$, algorithm $\mathcal{B}$ makes a verification query on the statement $\mathbf{x}_i = (\mathsf{ABE.pp}, \mathsf{ABE.ct}_i, x_i)$ and proof π_i. If the proof fails to verify, algorithm $\mathcal{B}$ replies with $\perp$, and otherwise with $\mathsf{ABE.Decrypt}(\mathsf{ABE.pp}, \mathsf{ABE.sk}_f, \mathsf{ABE.ct}_i)$, exactly as in the real scheme.

4. At the end of the experiment, output the statement $\mathbf{x}_{i^*} = (\mathsf{ABE.pp}, \mathsf{ABE.ct}_{i^*}, x_{i^*})$ and π_{i^*} where $\mathsf{ct}_{i^*} = (x_{i^*}, \mathsf{ABE.ct}_{i^*}, \pi_{i^*})$.

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

Thus, the augmented ABE scheme from Construction E.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. □