New Constructions of Reusable Designated-Verifier NIZKs

Alex Lombardi[1*], Willy Quach[2†], Ron D. Rothblum[3‡], Daniel Wichs[2§], and David J. Wu[4¶]

[1] MIT, Cambridge, MA
[2] Northeastern University, Boston, MA
[3] Technion, Haifa, Israel
[4] University of Virginia, Charlottesville, VA

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

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

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

*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) and the Technion Hiroshi Fujiwara cyber security research center and the Israel cyber directorate.

§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.

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

1 Introduction

Zero-knowledge proofs [28] 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 [4] 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 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 [4,51,23,50,26], the Diffie-Hellman assumption over bilinear groups [11,32], optimal hardness of the learning with errors (LWE) assumption[5] [10], and circular-secure fully homomorphic encryption [12]. We also have such NIZKs in the random-oracle model [24]. However, we are lacking constructions from several standard assumptions, most notably the computational or decisional Diffie-Hellman assumptions (CDH, DDH), the plain learning with errors (LWE) assumption, and the learning parity with noise (LPN) assumption.

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

While reusable DV-NIZKs appear to be a non-trivial relaxation of standard NIZKs,[6] 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 [14,35,46] constructed such DV-NIZKs under the CDH assumption. However, it was left as an open problem whether such DV-NIZKs can be constructed under LWE or LPN.

We note that the work [37] constructed an orthogonal notion of reusable "designated-prover" NIZKs (DP-NIZK) under LWE, where the trusted third party generates a CRS together with a secret key that is given to the prover and needed to generate proofs. In addition, the work [8] constructed "preprocessing NIZKs" (PP-NIZK), in which the trusted third party generates both a secret proving key

[5] This means that no polynomial-time attacker can break LWE with any probability better than random guessing.

[6] The public verifiability of traditional NIZKs immediately implies reusable soundness.

and a secret verification key, under variants of the LPN assumption over large fields.

Malicious-designated-verifier NIZK. We also consider a strengthening of DV-NIZKs to the malicious-designated-verifier model (MDV-NIZKs) introduced by [46]. 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 [46] 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 secret-key encryption scheme satisfying a weak form of key-dependent message (KDM) security Both components can be instantiated under any of the CDH/LWE/LPN assumptions, so we obtain constructions of DV-NIZKs under these assumptions. In particular, we obtain the following theorem:

Theorem 1.1 (informal). *Assuming the existence of public-key encryption and secret-key encryption that is KDM-secure with respect to projections (see Definition 2.12), there exist reusable designated-verifier NIZK arguments for* NP*. In particular, there exist reusable DV-NIZKs under either the CDH assumption, the LWE assumption, or the LPN assumption with noise rate* $O(\frac{1}{\sqrt{n}})$*.*

We then show how to construct reusable *malicious* DV-NIZKs from any (receiver-extractable) 2-round oblivious transfer (OT) in the common random string model and the same form of KDM-secure SKE. This yields instantiations of MDV-NIZKs under the CDH/LWE/LPN assumptions using the OT constructions of [45,21], as summarized by the following theorem.

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

More generally, we give a compiler converting any Σ-protocol (or even more generally, any "zero-knowledge PCP" [36,34]) 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 [49,31] or lattices [30,6], but *single-key* ABE can be constructed from any public-key encryption scheme [48,29]. While we are unable to construct our variant of ABE (i.e., one that satisfies our function-hiding property) from an arbitrary public-key encryption (PKE) scheme, we show how to construct it by additionally relying on KDM-secure SKE, using a technique recently developed in [39,38]. However, in addition to this construction, we outline an alternate approach for building single-key ABE with our function-hiding property (using the standard lattice-based ABE [6]) in the full version of this paper [40]. As a result, we believe that our new notion may be helpful in order to construct DV-NIZKs from other assumptions in the future. Note that if one could construct DV-NIZKs from any semantically-secure PKE scheme, it would show that semantically-secure PKE generically implies CCA-secure PKE (via the Naor-Yung paradigm [42]), 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 [14,35,46]. In particular, those works relied on the hidden-bits model from [23] and used a variant of the Cramer-Shoup hash-proof system under CDH [16,17] to instantiate the hidden bits for a designated verifier. Unfortunately, we do not have good hash-proof systems under LWE/LPN and so it does not appear that these techniques can be used when starting from "noisy assumptions" (among other concerns). As we describe below, we take a vastly different approach and do not rely on the hidden bits model. One disadvantage of our results is that, while [14,35,46] achieve *statistically* sound (M)DV-NIZK proofs, we only get argument systems with *computational* soundness[7].

Application to reusable non-interactive secure computation. We note that MDV-NIZKs can be used to obtain new solutions to the problem of reusable non-interactive secure computation (rNISC) [13]. In this setting, there is a public function f and a receiver (Rachel) publishes a "query" using her secret input x. Later a sender (Sam) can send a "response" using his secret input y and ensure that Rachel only learns $f(x, y)$. We further want Rachel's query to be reusable so that Sam can send many different responses with various values y_i and have Rachel learn $f(x, y_i)$ without compromising security. The main difficulty is that a malicious Sam can send malformed responses and, by observing whether Rachel aborts or not, can potentially learn information about her input x. Previously, we had instantiations of rNISC (in the CRS model) using 2-round (malicious) oblivious transfer (OT) *and* NIZKs, or more recently, via a black-box use of oblivious linear-function evaluation (OLE) [13]. However, we had no constructions

[7] Our construction is also computational zero-knowledge. None of the recent constructions of DV-NIZKs satisfy statistical zero knowledge.

under many standard assumptions, including any of CDH/DDH, LPN or LWE. It turns out that we can easily use MDV-NIZKs instead of standard NIZKs (along with 2-round malicious OT) to solve this problem. In particular, Rachel sends OT queries corresponding to her input x as well as the public-key of an MDV-NIZK. Sam then creates a garbled circuit for $f(\cdot, y)$ with his input y hard-coded, and sends the labels via the OT responses; in addition he encrypts y (under a public key in the CRS) and proves that he computed the garbled circuit and the OT responses correctly and consistently with the encrypted y. We can simulate Sam's view (including Rachel's output) by checking the MDV-NIZK to decide if Rachel aborts or not; if the MDV-NIZK verifies then we can extract y from the encryption and be sure that Rachel correctly outputs $f(x, y)$. Using our instantiations of MDV-NIZKs along with known constructions of 2-round OT from [45,21], we get instantiations of rNISC under CDH, LPN or LWE.

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 [43]. The [43] construction relies on a Σ-protocol [15] with 1-bit challenges for an NP-complete language, such as Blum's protocol for graph Hamiltonicity [3]. 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 [43] 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 [43], 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 [48,29].

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 the full version of this paper [40].

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

- A Σ-protocol [15] with 1-bit challenges for an NP-complete language $\mathcal{L}$, such as Blum's protocol for graph Hamiltonicity [3]. (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[8] $x \notin \mathcal{L}$ and an adversary that makes arbitrary verification queries and eventually produces an accepting proof for x. First, without loss of generality, we claim that

[8] 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 [5] (see Remark 2.4) or by relying on a *trapdoor Σ-protocol* [12] (see Remark 4.4).

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 two ways[9] to construct a (single-key) ABE scheme that satisfies our function-hiding property. Combined with our compiler above, this suffices to construct DV-NIZKs (i.e., the results from Theorem 1.1). In the body of the paper, we will focus on the second candidate based on KDM-secure SKE for two main reasons: (1) it enables instantiations from CDH/LWE/LPN (and correspondingly, DV-NIZKs from these assumptions); and (2) it readily generalizes to notions beyond ABE, which as we discuss in greater detail below, enables constructions of *MDV-NIZKs* from CDH/LWE/LPN.

- **Lattice-based ABE:** First, we observe that a simple variant of the lattice-based ABE construction from [6] satisfies our notion of function-hiding. Namely, we can modify the construction [6] so that the decryption algorithm (with either the master secret key or a function key) can *fully recover* the encryption randomness used to construct a particular ciphertext, and in doing so, verify that a ciphertext is well-formed (i.e., could be output by the honest encryption algorithm). If the scheme supports this randomness recovery property, function-privacy essentially follows from (perfect) correctness of the underlying scheme. This high-level idea of leveraging *randomness recovery* is a common theme in our constructions. We provide additional details in the full version of this paper [40].
- **PKE and KDM-secure SKE:** Following the approach of [39,38], we show that any single-key ABE scheme can be used to construct an ABE scheme satisfying function-hiding with respect to decryption queries. The amplification procedure additionally requires the existence of a secret-key encryption

[9] We refer to a previous version of this work [41] for an additional approach based on lossy trapdoor functions.

scheme SKE that is KDM-secure for a simple class of functions. As shown in [7,2,19,9,20], such secret-key encryption schemes can be constructed from the CDH/LWE/LPN assumptions, and hence, give instantiations of function-hiding ABE from CDH/LWE/LPN.

In fact, the exact construction of *CCA-secure* ABE in [39] (and the modification introduced in [38]) can also be shown to satisfy function-hiding. However, as noted above, CCA-security does not generically imply our notion of function-hiding or vice versa. In this work, we describe a simplified variant of the [38] compiler that suffices to construct function-hiding ABE and then analyze its security.

We now provide a description of (our simplification of) the [39,38] construction. Take any (single-key) ABE scheme ABE, a secret-key encryption scheme SKE, a public-key encryption scheme PKE, an equivocable commitment scheme Com, and consider the following modified ABE scheme:

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

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

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

Following [39,38], we provide some high-level intuition for this encryption algorithm. For a fixed pair (i,b), call a ciphertext $\mathsf{ct}_{i,b}$ "good" with respect to commitment com_i if there exists commitment randomness ρ_i such that $\mathsf{com}_i = \mathsf{Com}(\mathsf{crs}, b; \rho_i)$ and $\mathsf{ct}_{i,b}$ is a well-formed encryption of ρ_i. Then, given a qualified secret key sk_f, an honestly-generated matrix $M = \{\mathsf{ct}_{i,b}\}$ encodes the SKE-secret key s: we have $s_i = b$ if and only if $\mathsf{ct}_{i,b}$ is "good," so s_i

can be identified by decrypting $\mathsf{ct}_{i,0}$ (using sk_f) and checking whether the underlying message is $\perp$.[10]

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

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

- Switch the commitment crs to an "equivocal mode" so that $\mathsf{com} = (\mathsf{com}_i)_{i\in[\lambda]}$ can be explained as a commitment to *any string* (with an appropriate choice of randomness).
- Show that (in equivocal mode) $M = \{\mathsf{ct}_{i,b}\}$ can be simulated (using ρ and $\{R_{i,b}\}$) without knowing s.
- At this point, ct_0 is guaranteed to hide m by invoking KDM-security of SKE.

To argue function-hiding, we show that the ciphertext ct 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 (and 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 hides $f(x')$ for any attribute x' not queried by the adversary, function-hiding follows.

Combined with the instantiations of KDM-secure SKE from various assumptions [7,2,19,9,20] and the fact that single-key ABE follows from PKE [48,29], this approach gives a single-key function-hiding ABE scheme from any of the CDH/LWE/LPN assumptions. For our LPN instantiation, we require noise rate $1/\sqrt{n}$ to instantiate the public-key encryption scheme [1]. We describe this construction and its analysis in Section 5.

Obtaining malicious security. So far, we have shown how to construct DV-NIZKs from function-hiding single-key ABE and provided several instantiations of the latter object from concrete assumptions. However, the DV-NIZKs obtained

[10] In the actual decryption procedure, a more sophisticated mechanism is employed to identify s in order to handle malformed ciphertexts.

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. We say AB-SFE schemes satisfying this property are "key-hiding". 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 is therefore an MDV-NIZK. Specifically, in our construction, we replace the ABE scheme with an AB-SFE scheme with respect to the function F where $F((x,b),k) = 1$ if and only if $\mathsf{PRF}(k,x) = b$. 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:

- **Constructing weak key-hiding AB-SFE.** First, we combine a form of malicious-secure 2-message OT [45,21] with garbled circuits to obtain an AB-SFE scheme that satisfies *weak key-hiding*. Namely, the receiver's input y is hidden to an adversary that does *not* have access to the decryption oracle. We describe this construction in Section 5.2.

– **Amplifying weak key-hiding to strong key-hiding.** Then, we apply the [39,38] transformation to the weak key-hiding AB-SFE scheme from above to obtain an AB-SFE scheme that satisfies strong key-hiding where the receiver's input y is hidden even in the presence of the decryption oracle. This allows for new instantiations of MDV-NIZK from any of the CDH/LWE/LPN assumptions (Theorem 1.2). Our LPN-based instantiation requires noise rate $n^{-(\frac{1}{2}+\epsilon)}$ for some $\epsilon > 0$ in order to implement the [21] OT protocol. We describe this construction in Section 5.3.

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

1.3 Recent Related Work

In this section, we describe several recent works that are directly related to this work. Several of these works [38,21] have yielded new instantiations of our general framework for constructing designated-verifier NIZKs (relative to a preliminary version of this work [41]).

NIZKs from LWE. In a concurrent and independent work, Peikert and Shiehian [44] construct NIZKs from the plain LWE assumption, which in particular yields reusable (M)DV-NIZKs from LWE. While the [44] NIZK has the major advantage of being publicly verifiable, we note that our usage of LWE only relies on plain Regev (public-key) encryption [47] rather than more complex lattice-based primitives.

KDM-Secure SKE and hinting PRGs. In a concurrent work, Kitagawa, Matsuda and Tanaka [38] modify the "signaling technique" of [39] with the goal of constructing CCA-secure encryption (similarly to [39]). The [38] modification of [39] can be plugged into our construction of (M)DV-NIZKs to obtain our LPN-based instantiation of DV-NIZKs.

2-Message OT from CDH/LPN. In another concurrent work, Döttling, Garg, Hajiabadi, Masny, and Wichs [21] construct 2-round OT from the CDH/LPN assumptions. Their construction can be directly combined with our generic transformations to obtain the CDH/LPN-based instantiations of MDV-NIZKs in this paper.

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 $\mathcal{Y}$. In the the full version of this paper [40], we also review the definitions of core cryptographic primitives such as pseudorandom functions, public-key encryption, (receiver-extractable) 2-message oblivious transfer, garbling schemes, and non-interactive equivocable commitments.

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 2.1 (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* 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* $\mathcal{A}$*,*

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

    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* $\mathcal{A}$*,*

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

    for $(\mathsf{crs}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{dvNIZK.Setup}(1^\lambda)$ *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 2.2 (Malicious Designated-Verifier NIZKs [46]). *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* $\mathsf{ExptReal}_{\mathcal{A}}(\lambda) \stackrel{c}{\approx} \mathsf{ExptSim}_{\mathcal{A},\mathcal{S}}(\lambda)$*. 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 2.3 (Reusability of the CRS with Many Public Keys). The zero-knowledge property of Definition 2.2 only provides (multi-theorem) zero-knowledge with respect to a *single* maliciously-generated public key pk. Using the "OR trick" transformation from [23], 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 2.4 (Adaptive Soundness via Complexity Leveraging). Using the standard technique of complexity leveraging [5], 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 2.5 (Zero-Knowledge PCP [36,34]). *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 2.5 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 2.6 (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*

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

for $(\mathsf{st}_{\mathcal{A}}, x, w, r) \leftarrow \mathcal{A}_1(1^n)$, $(q_1, \ldots, q_\ell) \leftarrow \mathsf{zkPCP.Query}(x; r)$, $\pi \leftarrow \mathsf{zkPCP.Prove}(x, w)$, 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. [34], the original zero-knowledge protocol by Goldreich et al. [27] 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 2.7 (Semi-Malicious Zero-Knowledge PCP [27]). *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 [3] 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. [33]. 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 2.8 (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 2.9 (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*

$$\mathsf{ABE.Decrypt}\big(\mathsf{pp}, \mathsf{ABE.KeyGen}(\mathsf{pp}, \mathsf{msk}, f), \mathsf{ABE.Encrypt}(\mathsf{pp}, x, m)\big) = (x, m)$$

with probability 1*.*

Definition 2.10 (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 2.11 (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 KDM-Secure Secret-Key Encyryption

Definition 2.12 (One-Time KDM-Secure SKE). *A secret-key encryption (SKE) scheme* $\mathsf{SKE} = (\mathsf{SKE.Encrypt}, \mathsf{SKE.Decrypt})$ *is said to be* one-time KDM

secure *for a function class $\mathcal{F}$ (with many-bit outputs) if for every function $f \in \mathcal{F}$, the following two distributions are computationally indistinguishable:*

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

Remark 2.13 (KDM-Secure SKE Constructions for Projection Functions). We say a function $f\colon \{0,1\}^\lambda \to \{0,1\}^m$ is a *projection function* if each bit of $f(s)$ depends on at most one bit of s. As in [38], we consider the class $\mathcal{F} = \mathcal{F}_{\text{proj}}$ of projection functions. Secret-key encryption schemes that are KDM-secure for the class of projection functions can be constructed from the CDH [7,9], LWE (with polynomial modulus) [2,9], and constant-noise LPN [2] assumptions.

3 Attribute-Based Secure Function Evaluation

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

Definition 3.1 (Attribute-Based Secure Function Evaluation). *An attribute-based secure function evaluation (AB-SFE) scheme for a function $F\colon \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ with message space $\mathcal{M}$ consists of a tuple of PPT algorithms* $\mathsf{ABSFE} = (\mathsf{ABSFE.Setup}, \mathsf{ABSFE.KeyGen}, \mathsf{ABSFE.Encrypt}, \mathsf{ABSFE.Decrypt})$ *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 $m \in \mathcal{M} \cup \{\perp\}$.*

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* $\mathcal{A}$*,*

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

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 CRS distributions 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 2.11), 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 the full version of this paper [40], 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 2.5). 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.*
- *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 the completeness, soundness, and zero-knowledge theorems for Construction 4.1, but defer the proofs to the full version of this paper [40].

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

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.*

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 2.4, 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" [12]. We refer the reader to the full version of this paper [40] (Remark 4.4 and Appendix A) for more details.

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).*

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

5 Constructing AB-SFE Schemes

In this section, we describe several approaches to construct AB-SFE schemes satisfying different flavors of message-hiding and key-hiding. First, in Section 5.1, we show how to build weak message-hiding AB-SFE from any single-key ABE scheme. In Section 5.2, we show how to construct AB-SFE schemes with strong message-hiding (and weak key-hiding) from receiver-extractable OT. Then, in Section 5.3, we show how to generically boost an AB-SFE scheme satisfying weak key-hiding into one that satisfies strong key-hiding (Definition 3.7) via a KDM-secure secret-key encryption scheme (while preserving weak/strong message-hiding). Combining the constructions in Section 5.2 and 5.3, we obtain AB-SFE schemes that provide both strong message-hiding and strong key-hiding (which

suffice to realize our strongest notion of MDV-NIZK via Construction 4.1). Finally, in Section 5.4, we describe how to instantiate the different building blocks from the CDH, DDH, or LWE assumptions.

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

As noted in Section 1.2, an AB-SFE scheme can be viewed as a generalization of a single-key ABE scheme. In the full version of this paper [40], we describe two simple constructions of AB-SFE schemes from single-key ABE schemes (which are in turn implied by public-key encryption [48,29]). Both of these schemes provide *weak message-hiding*.

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. The resulting scheme satisfies weak key-hiding, and we show how to amplify key-hiding security in Section 5.3.

Construction 5.1 (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 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 2-message batch OT scheme that is receiver-extractable with k-bit messages with batch size ℓ (see the full version of this paper [40] for the formal definitions), 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* $\mathsf{Yao.Eval}(\widetilde{C}, \overrightarrow{\mathsf{lab}})$.

We state the properties of Construction 5.1 in the following theorem, but defer the proof to the full version of this paper [40].

Theorem 5.2 (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 5.1 satisfies strong message-hiding and weak key-hiding.*

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

In the full version of this paper [40], we show how to generically upgrade weak key-hiding to strong key-hiding via KDM-secure secret-key encryption (Definition 2.12) and an equivocable non-interactive commitment scheme [18]. 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 5.3 (Recovery from Randomness [39]). *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 5.4 (Recovery from Randomness [39]). It is straightforward to upgrade any PKE (resp., AB-SFE) scheme to have the recovery from randomness property. As noted in [39], 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 5.5 (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 5.3, Remark 5.4). 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 (Remark 5.4).*
- *Let* $\ell = \ell(\lambda)$ *be a bound on the number of bits of randomness* $\mathsf{PKE.Encrypt}$ *and* $\mathsf{ABSFE.Encrypt}$ *use.*

– *Let* SKE *denote a secret-key encryption scheme with message-space* $\mathcal{M} \times \{0,1\}^{\ell\lambda}$ *that is one-time KDM-secure for the class of projection functions (Definition 2.12, Remark 2.13).*
– *Let* Com *be a non-interactive equivocable commitment scheme with message space* $\{0,1\}$.

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

– $\mathsf{Aug.Setup}(1^\lambda)$*: Sample* $\mathsf{ABSFE.crs} \leftarrow \mathsf{ABSFE.Setup}(1^\lambda)$, $(\mathsf{PKE.pk}, \mathsf{PKE.sk}) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$*, and for each* $i \in [\lambda]$, $\mathsf{Com.crs}_i \leftarrow \mathsf{Com.Setup}(1^\lambda)$*. It outputs the common reference string* $\mathsf{crs} = (\mathsf{ABSFE.crs}, \mathsf{PKE.pk}, \{\mathsf{Com.crs}_i\}_{i\in[\lambda]})$.
– $\mathsf{Aug.KeyGen}(\mathsf{crs}, y)$*: On input* $\mathsf{crs} = (\mathsf{ABSFE.crs}, \mathsf{PKE.pk}, \{\mathsf{Com.crs}_i\}_{i\in[\lambda]})$*, sample a key-pair* $(\mathsf{ABSFE.pk}, \mathsf{ABSFE.sk}) \leftarrow \mathsf{ABSFE.KeyGen}(\mathsf{ABSFE.crs}, y)$ *and output the public key* $\mathsf{pk} = \mathsf{ABSFE.pk}$ *and the secret key* $\mathsf{sk} = (y, \mathsf{ABSFE.pk}, \mathsf{ABSFE.sk})$.
– $\mathsf{Aug.Encrypt}(\mathsf{crs}, \mathsf{pk}, x, m)$*: Parse* $\mathsf{crs} = (\mathsf{ABSFE.crs}, \mathsf{PKE.pk}, \{\mathsf{Com.crs}_i\}_{i\in[\lambda]})$ *and* $\mathsf{pk} = \mathsf{ABSFE.pk}$*, and proceed as follows:*
  - *Sample a secret key* $s \xleftarrow{\text{R}} \{0,1\}^\lambda$ *for* SKE.
  - *For every* $i \in [\lambda]$*, sample* $\rho_i \xleftarrow{\text{R}} \{0,1\}^\lambda$ *and compute* $c_i \leftarrow \mathsf{Com.Commit}(\mathsf{Com.crs}_i, s_i; \rho_i)$.
  - *For every* $i \in [\lambda]$*, define* $M_{i,s_i} = \rho_i$ *and* $M_{i,1-s_i} = \bot$*. Then, sample* $R_{i,0}, R_{i,1} \xleftarrow{\text{R}} \{0,1\}^\ell$ *and construct the ciphertexts*

$$\begin{aligned}\mathsf{ct}_{i,0} &\leftarrow \mathsf{ABSFE.Encrypt}(\mathsf{ABSFE.crs}, \mathsf{ABSFE.pk}, x, M_{i,0}; R_{i,0})\\ \mathsf{ct}_{i,1} &\leftarrow \mathsf{PKE.Encrypt}(\mathsf{PKE.pk}, M_{i,1}; R_{i,1}).\end{aligned}$$

  - *Let* $\mathsf{ct}_0 \leftarrow \mathsf{SKE.Encrypt}(s, (m, (R_{i,s_i})_{i\in[\lambda]}))$.
  - *Output* $\mathsf{ct} = \big(\mathsf{ct}_0, (c_i, \mathsf{ct}_{i,0}, \mathsf{ct}_{i,1})_{i\in[\lambda]}\big)$.
– $\mathsf{Aug.Decrypt}(\mathsf{crs}, \mathsf{sk}, x, \mathsf{ct})$*: On input* $\mathsf{crs} = (\mathsf{ABSFE.crs}, \mathsf{PKE.pk}, \{\mathsf{Com.crs}_i\}_{i\in[\lambda]})$*, a secret key* $\mathsf{sk} = (y, \mathsf{ABSFE.pk}, \mathsf{ABSFE.sk})$*, and a ciphertext* $\mathsf{ct} = \big(\mathsf{ct}_0, (c_i, \mathsf{ct}_{i,0}, \mathsf{ct}_{i,1})_{i\in[\lambda]}\big)$*, proceed as follows:*
  1. *If* $F(x,y) = 0$*, output* $\bot$.
  2. *For every* $i \in [\lambda]$*, compute* $\rho'_i \leftarrow \mathsf{ABSFE.Decrypt}(\mathsf{ABSFE.crs}, \mathsf{ABSFE.sk}, x, \mathsf{ct}_{i,0})$*. 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. *Compute* $(m', (r'_i)_{i\in[\lambda]}) \leftarrow \mathsf{SKE.Decrypt}(s', \mathsf{ct}_0)$.
  4. *For every* $i \in [\lambda]$*, perform the following checks:*
     - *If* $s'_i = 0$*, then check if* $\mathsf{Recover}(\mathsf{ABSFE.crs}, \mathsf{ABSFE.pk}, \mathsf{ct}_{i,0}, r'_i) = (x, \tilde{\rho}_i)$ *for some* $\tilde{\rho}_i$*, and output* $\bot$ *if the check fails.*
     - *If* $s'_i = 1$*, then compute* $\tilde{\rho}_i \leftarrow \mathsf{PKE.Recover}(\mathsf{PKE.pk}, \mathsf{ct}_{i,1}, r'_i)$ *and output* $\bot$ *if the recovery procedure fails.*
     - *Finally, check if* $c_i = \mathsf{Com.Commit}(\mathsf{Com.crs}_i, s'_i; \tilde{\rho}_i)$*. Output* $\bot$ *if this check fails.*
  5. *If all checks pass, output* m'.

5.4 Instantiations

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

The resulting weak message-hiding AB-SFE instantiations correspondingly yield DV-NIZKs. Moreover, the resulting strong message-hiding AB-SFE schemes have *uniformly random* public parameters, thus yielding designated-verifier NIZKs with security against malicious verifiers in the common *random* string model. We instantiate each building block as follows (with more details in the full version of this paper [40]):

- There exists a receiver-extractable 2-message batch OT scheme in the common random string model under the CDH/LWE/LPN assumptions (with the parameters specified above). There exists a garbling scheme from one-way functions. Thus, by Theorem 5.2, we obtain an AB-SFE scheme with strong message-hiding and weak key-hiding under the CDH/LWE/LPN assumptions in the common random string model.
- There exist public-key encryption schemes with pseudorandom (or uniformly random) public keys from the CDH assumption [25], the LWE assumption [47], or the LPN assumption [1] with the parameters specified above. Because we only use the associated secret key in the proof of security, we can replace the public key $\mathsf{PKE.pk}$ from Construction 5.5 with a uniformly random string, while maintaining security (by a standard hybrid argument) and perfect correctness.
- From Section 5.1, there exists an AB-SFE scheme with weak message-hiding and weak key-hiding under any assumption implying PKE.
- By Remark 2.13, there exists a KDM-secure secret-key encryption scheme for projection functions under the CDH assumption, the LWE assumption, or the LPN assumption with constant noise rate.
- There exists a non-interactive equivocable commitment scheme from one-way functions in the common random string model.

Remark 5.6 (Almost-All-Keys Perfect Decryption Correctness). Some of the PKE/OT schemes above (such as the PKE scheme of [1]) do not actually satisfy perfect decryption correctness. However, the transformation of [22] shows that these encryption schemes can be modified to satisfy the following "almost-all-keys perfect correctness" property: with probability $1 - \mathsf{negl}(\lambda)$ over the randomness of $\mathsf{PKE.KeyGen}(\cdot)$, decryption is perfectly correct with probability 1 over the choice of encryption randomness. Encryption schemes satisfying this notion of almost-all-keys perfect correctness suffice for all of the constructions in this paper.

Instantiations. Combining the above primitives in Construction 5.5, we now obtain the following corollaries. In all cases, we only rely on polynomial hardness of the underlying assumption.

Corollary 5.7 (Weak Message-Hiding, Strong Key-Hiding AB-SFE from LPN). *Assuming polynomial hardness of the LPN assumption with noise rate $O(\frac{1}{\sqrt{n}})$, there exists an AB-SFE scheme with that satisfies strong key-hiding and weak message-hiding.*

Corollary 5.8 (Strong Message-Hiding, Strong Key-Hiding AB-SFE from CDH/LWE/LPN). *Assuming polynomial hardness of either CDH, LWE, or LPN with noise rate $n^{-(\frac{1}{2}+\varepsilon)}$ for any $\varepsilon > 0$, there exists an AB-SFE scheme with uniformly random public parameters that satisfies strong key-hiding and strong message-hiding security.*

Combining Theorem 2.5 now with Construction 4.1 (and Remarks 4.4 and 4.6), we obtain the following instantiations of designated-verifier NIZKs:

Corollary 5.9 (Designated-Verifier NIZKs from LPN). *Assuming polynomial hardness of the LPN assumption with noise rate $O(\frac{1}{\sqrt{n}})$, there exists a designated-verifier NIZK argument for* NP *that is adaptively sound and provides computational zero-knowledge in the common reference string model.*

Corollary 5.10 (Malicious-Designated-Verifier NIZKs from CDH/LWE /LPN). *Assuming polynomial hardness of either CDH, LWE, or LPN with noise rate $n^{-(\frac{1}{2}+\varepsilon)}$ for any $\varepsilon > 0$, there exists a designated-verifier NIZK argument for* NP *that is adaptively sound and provides computational zero-knowledge against malicious verifiers in the common random string model.*

Acknowledgments

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

References

1. M. Alekhnovich. More on average case vs approximation complexity. In *FOCS*, pages 298–307, 2003.
2. B. Applebaum, D. Cash, C. Peikert, and A. Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *CRYPTO*, pages 595–618, 2009.
3. M. 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.
4. M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *STOC*, pages 103–112, 1988.
5. D. Boneh and X. Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *EUROCRYPT*, pages 223–238, 2004.
6. D. Boneh, C. Gentry, S. Gorbunov, S. Halevi, V. Nikolaenko, G. Segev, V. Vaikuntanathan, and D. Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *EUROCRYPT*, pages 533–556, 2014.

7. D. Boneh, S. Halevi, M. Hamburg, and R. Ostrovsky. Circular-secure encryption from decision diffie-hellman. In *CRYPTO*, pages 108–125, 2008.
8. E. Boyle, G. Couteau, N. Gilboa, and Y. Ishai. Compressing vector OLE. In *ACM CCS*, pages 896–912, 2018.
9. Z. Brakerski, A. Lombardi, G. Segev, and V. Vaikuntanathan. Anonymous IBE, leakage resilience and circular security from new assumptions. In *EUROCRYPT*, pages 535–564, 2018.
10. R. Canetti, Y. Chen, J. Holmgren, A. Lombardi, G. N. Rothblum, and R. D. Rothblum. Fiat-Shamir from simpler assumptions. *IACR Cryptology ePrint Archive*, 2018:1004, 2018.
11. R. Canetti, S. Halevi, and J. Katz. A forward-secure public-key encryption scheme. *IACR Cryptology ePrint Archive*, 2003:83, 2003.
12. R. Canetti, A. Lombardi, and D. Wichs. Fiat-Shamir: From practice to theory, part ii (NIZK and correlation intractability from circular-secure FHE). *IACR Cryptology ePrint Archive*, 2018:1248, 2018.
13. M. Chase, Y. Dodis, Y. Ishai, D. Kraschewski, T. Liu, R. Ostrovsky, and V. Vaikuntanathan. Reusable non-interactive secure computation. In *CRYPTO*, 2019.
14. G. Couteau and D. Hofheinz. Designated-verifier pseudorandom generators, and their applications. In *EUROCRYPT*, 2019.
15. R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, pages 174–187, 1994.
16. R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO*, pages 13–25, 1998.
17. R. Cramer and V. Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *EUROCRYPT*, pages 45–64, 2002.
18. G. D. Crescenzo, Y. Ishai, and R. Ostrovsky. Non-interactive and non-malleable commitment. In *STOC*, pages 141–150, 1998.
19. N. Döttling and S. Garg. Identity-based encryption from the Diffie-Hellman assumption. In *CRYPTO*, pages 537–569, 2017.
20. N. Döttling, S. Garg, M. Hajiabadi, and D. Masny. New constructions of identity-based and key-dependent message secure encryption schemes. In *PKC*, pages 3–31, 2018.
21. N. Döttling, S. Garg, M. Hajiabadi, D. Masny, and D. Wichs. Two-round oblivious transfer from CDH or LPN. *IACR Cryptology ePrint Archive*, 2019, 2019.
22. C. Dwork, M. Naor, and O. Reingold. Immunizing encryption schemes from decryption errors. In *EUROCRYPT*, pages 342–360, 2004.
23. U. Feige, D. Lapidot, and A. Shamir. Multiple non-interactive zero knowledge proofs under general assumptions. *SIAM J. Comput.*, 29(1):1–28, 1999.
24. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.
25. T. E. Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*, pages 10–18, 1984.
26. O. 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.
27. O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In *FOCS*, pages 174–187, 1986.
28. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

29. S. Gorbunov, V. Vaikuntanathan, and H. Wee. Functional encryption with bounded collusions via multi-party computation. In *CRYPTO*, pages 162–179, 2012.
30. S. Gorbunov, V. Vaikuntanathan, and H. Wee. Attribute-based encryption for circuits. In *STOC*, pages 545–554, 2013.
31. V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM CCS*, pages 89–98, 2006.
32. J. Groth, R. Ostrovsky, and A. Sahai. Non-interactive Zaps and new techniques for NIZK. In *CRYPTO*, pages 97–111, 2006.
33. Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge from secure multiparty computation. In *STOC*, pages 21–30, 2007.
34. Y. Ishai, M. Mahmoody, and A. Sahai. On efficient zero-knowledge PCPs. In *TCC*, pages 151–168, 2012.
35. S. Katsumata, R. Nishimaki, S. Yamada, and T. Yamakawa. Designated verifier/prover and preprocessing NIZKs from Diffie-Hellman assumptions. In *EUROCRYPT*, 2019.
36. J. Kilian, E. Petrank, and G. Tardos. Probabilistically checkable proofs with zero knowledge. In *STOC*, pages 496–505, 1997.
37. S. Kim and D. J. Wu. Multi-theorem preprocessing NIZKs from lattices. In *CRYPTO*, pages 733–765, 2018.
38. F. Kitagawa, T. Matsuda, and K. Tanaka. Cca security and trapdoor functions via key-dependent-message security. *IACR Cryptology ePrint Archive*, 2019:291, 2019.
39. V. Koppula and B. Waters. Realizing chosen ciphertext security generically in attribute-based encryption and predicate encryption. *IACR Cryptology ePrint Archive*, 2018:847, 2018.
40. A. Lombardi, W. Quach, R. D. Rothblum, D. Wichs, and D. J. Wu. New constructions of reusable designated-verifier NIZKs. *IACR Cryptology ePrint Archive*, 2019:242, 2019.
41. A. Lombardi, W. Quach, R. D. Rothblum, D. Wichs, and D. J. Wu. New constructions of reusable designated-verifier nizks. *IACR Cryptology ePrint Archive*, 2019, 2019. Preliminary Version.
42. M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *STOC*, pages 427–437, 1990.
43. R. Pass, a. shelat, and V. Vaikuntanathan. Construction of a non-malleable encryption scheme from any semantically secure one. In *CRYPTO*, pages 271–289, 2006.
44. C. Peikert and S. Shiehian. Noninteractive zero knowledge for NP from (plain) learning with errors. In *CRYPTO*, 2019.
45. C. Peikert, V. Vaikuntanathan, and B. Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO*, pages 554–571, 2008.
46. W. Quach, R. D. Rothblum, and D. Wichs. Reusable designated-verifier NIZKs for all NP from CDH. In *EUROCRYPT*, 2019.
47. O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93, 2005.
48. A. Sahai and H. Seyalioglu. Worry-free encryption: functional encryption with public keys. In *ACM CCS*, pages 463–472, 2010.
49. A. Sahai and B. Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.
50. A. D. Santis, G. D. Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai. Robust non-interactive zero knowledge. In *CRYPTO*, pages 566–598, 2001.
51. A. D. Santis, S. Micali, and G. Persiano. Non-interactive zero-knowledge proof systems. In *CRYPTO*, pages 52–72, 1987.