Stronger Notions and Constructions for Multi-Designated Verifier Signatures [⋆]

Ivan Damgård[1], Helene Haagh[1], Rebekah Mercer[2], Anca Nitulescu[1], Claudio Orlandi[1], and Sophia Yakoubov[1,3]

[1] Aarhus University {ivan, haagh, anca, orlandi}@cs.au.dk,
[2] rebekah@o1labs.org, O(1) Labs, USA
[3] sonka@bu.edu, Boston University

Abstract. Off-the-Record (OTR) messaging is a two-party message authentication protocol that also provides plausible deniability: there is no record that can later convince a third party what messages were actually sent. To extend OTR to group messaging we need to consider issues that are not present in the 2-party case. In group OTR (as in two-party OTR), the sender should be able to authenticate (or sign) his messages so that group members can verify who sent a message (that is, signatures should be *unforgeable*, even by group members). Also as in the two-party case, we want the off-the-record property: even if some verifiers are corrupt and collude, they should not be able to prove the authenticity of a message to any outsider. Finally, we need *consistency*, meaning that a corrupt sender cannot create confusion in the group as to what he said: if any group member accepts a signature, then all of them do.

To achieve these properties it is natural to consider Multi-Designated Verifier Signatures (MDVS), which intuitively seem to target exactly the properties we require. However, existing literature defines and builds only limited notions of MDVS, where (a) the off-the-record property (referred to as *source hiding*) only holds when *all* verifiers could conceivably collude, and (b) the consistency property is not considered.

The contributions of this paper are two-fold: stronger definitions for MDVS, and new constructions meeting those definitions. We strengthen source-hiding to support any subset of corrupt verifiers, and give the first formal definition of consistency.

We give several constructions of our stronger notion of MDVS: one from generic standard primitives such as pseudorandom functions, pseudorandom generators, key agreement and NIZKs; one from specific instances of these primitives (for concrete efficiency); and one from functional encryption. The third construction requires an involved trusted setup step — including verification keys derived from a master secret — but this trusted setup buys us verifier-identity-based signing, for which such trusted setup is unavoidable. Additionally, in the third construction, the signature size can be made smaller by assuming a bound on colluding verifiers.

[⋆] This research was supported by: the Concordium Blockhain Research Center, Aarhus University, Denmark; the Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM); the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation programme under grant agreement No 669255 (MPCPRO) and No 803096 (SPEC); the Danish Independent Research Council under Grant-ID DFF-6108-00169 (FoCC); the NSF MACS project.

Table of Contents

1 Introduction

Encrypted and authenticated messaging has experienced widespread adoption in recent years, due to the attractive combination of properties offered by, for example, the Signal protocol [Mar13]. With so many conversations happening over the internet, there is a growing need for protocols offering security to conversation participants. Encryption can be used to guarantee privacy of message contents, but authenticating messages while maintaining the properties of an in person conversation is more involved. There are two properties of in person conversations related to authenticity that we wish to emulate in the context of digital conversations:

– *Unforgeability*, meaning that the receiver should be convinced that the message actually came from the sender in question, and
– *Off-the-record* or *deniability*, meaning that the receiver cannot later prove to a third party that the message came from the sender.

Off-the-record (OTR) messaging offers a solution to this in the two-party case, enabling authentication of messages such that participants can convincingly deny having made certain statements, or even having taken part in the conversation at all [BGB04]. The protocol deals with encrypted messages accompanied by a message authentication code (MAC) constructed with a shared key. MACs work well in two-party conversations, because for parties S(ender) and R(eciever) with a shared secret key, a MAC attests 'this message comes from S or R'. MACs provide unforgeability, since a party R receiving a message authenticated with such a MAC knows that if this MAC verifies, the message came from S. MACs provide off-the-record (deniable) communication as R cannot convince a third party that a message and MAC originally came from S (since R could have produced it just as easily). More generally, tools that provide unforgeable, off-the-record two-party communication are known as *Designated Verifier Signatures* (DVSs, proposed by [JSI96] and [Cha96]).

When there are multiple recipients, for example in group messaging, the situation becomes more complicated. DVSs have been extended to the multiparty setting under the name of *Multi-Designated Verifier Signatures (MDVSs)* (we give a number of references in Figure 1). One might hope that these schemes would work for off-the record group messaging; however, it turns out that existing MDVS definitions and schemes do not have the properties one would naturally ask for. In the following section, we give a motivating example illustrating which properties we should actually ask from an MDVS scheme, and we explain how existing schemes fall short of providing them.

1.1 A Motivating Example for MDVS

Imagine a government official Sophia who wants to blow the whistle on some corrupt government activity; e.g., perhaps her colleague, $\mathcal{A}$aron, accepted a bribe. She wants to send a message describing this corruption to Robert, Rachel and Rebekah, who are all Reporters at national newspapers.

Naturally, Sophia wants the Reporters to be convinced that she is the true sender of the message. Otherwise, they would have no reason to believe — or print — the story.

Goal 1 (Unforgeability) *It is vital that each of the* Reporters *be able to authenticate that the message came from* Sophia.

In order to achieve unforgeability, Sophia produces a signature σ using an MDVS scheme, and attaches it to her message. (In such a scheme, each sender has a private signing key and each recipient has a private verification key.) However, blowing the whistle and reporting on $\mathcal{A}$aron's corrupt activity could put Sophia in danger. If any of Robert, Rachel or Rebekah could use σ demonstrate to $\mathcal{A}$aron that Sophia blew the whistle on him, she could lose her position, or face other grave consequences.

Goal 2 (Source-Hiding / Off-the-Record) *It is vital that the* Reporters *be unable to prove to an outsider ($\mathcal{A}$aron) that the message came from* Sophia.

One way to guarantee that the Reporters cannot link Sophia to the message is to require that the Reporters can *simulate* a signature σ themselves. Then, if they try to implicate Sophia by showing σ to $\mathcal{A}$aron, he would have no reason to believe them; as far as he is concerned, the Reporters could have produced σ to try to frame Sophia.

All previous constructions only support off-the-record in the limited sense that *all* of the Reporters must collaborate in order to produce a simulated signature.[4] However, this is insufficient. Suppose, for instance, that $\mathcal{A}$aron knows Rachel was undercover — and thus unreachable — for the entire time between the bribery taking place, and Robert and Rebekah bringing σ to $\mathcal{A}$aron. Then he would conclude that Rachel could not have collaborated in simulating σ, and so it must be genuine. Even with the off-the-record definition used in prior works, it is still possible that some subset of the Reporters would be able to implicate Sophia in the eyes of $\mathcal{A}$aron. We therefore need a stronger off-the-record defintion.

Contribution 1 (Off-the-record For Any Subset) *We give a stronger definition of the off-the-record property, where* any *subset of* Reporters *must be able to simulate a signature. A simulation looks like a genuine signature to an outsider, even given the verification keys of the subset that produced it (as well as a number of other signatures that are guaranteed to be genuine).*

Under our stronger definition, no set of Reporters is able to use σ to provably tie Sophia to the message *even if $\mathcal{A}$aron has side information about communication amongst the* Reporters as well as guaranteed-to-be-genuine signatures.

[4] One previous work [Tia12] achieves off-the-record when $\mathcal{C}$ consists of a single verifier. However, in this construction a simulated signature created by a malicious verifier will look like a real signature for all other designated verifers, violating unforgeability.

Remark 1. (The Tension Between Off-The-Record and Unforgeability) Note that, if Rachel did not participate in Robert and Rebekah's signature simulation (e.g. if she was undercover at the time), she will later be able to distinguish the simulation from a real signature produced by Sophia. Otherwise, Robert and Rebekah would have succeeded in producing a forgery that fools Rachel.

This means that under a sufficiently strong model of attack, we cannot have unforgeability and off-the-record at the same time. Namely, suppose $\mathcal{A}aron$ first gets a signature σ from Robert and Rebekah, while preventing them from communicating with Rachel. Then he coerces Rachel into giving him her secret verification key. By the unforgeability property, he can use this key to tell if σ is a simulation. (Note that $\mathcal{A}$aron will be able to tell whether Rachel gives him her true verification key, since he may have other signatures from Sophia that he knows are genuine that he can use to test it. So, she has no choice but to hand over her real verification key.)

Given this observation, we choose to explore the model where the secret keys of all coerced/corrupted verifiers (but not honest ones) can be used to simulate a signature, as this is the strongest model of attack in which both unforgeability and off-the-record can be achieved. As we shall see, even in this model, achieving both properties requires highly non-trivial constructions and implies a lower bound on the size of signatures.

Finally, let us fast forward to the moment when Robert, Rachel and Rebekah receive Sophia's message. They want to print this high-profile story as soon as possible, but of course they want to be sure they won't make themselves look foolish by printing the story if their colleagues — the other well-respected Reporters listed as recipients — don't believe it actually came from Sophia. The concern here is that Sophia could be dishonest and her actual goal could be to discredit the Reporters. Hence we need another property — consistency, or designated verifier transferability.

Goal 3 (Consistency / Designated Verifier Transferability) *It is desirable that, even if Sophia is malicious, if one of the Reporters can authenticate that the message came from Sophia, all of them can.*

Contribution 2 *We provide the first formal definition of consistency.*

Now that we have covered the basic storyline, let us consider a few possible plot-twists. First, what if $\mathcal{A}$aron is tapping the wires connecting the government building to the outside world? Then he will see Sophia's message — together with her signature σ — as she sends it to the Reporters. In such a situation, we would want the signature σ not to give Sophia— or the Reporters— away.

Goal 4 (Privacy of Identities) *It is desirable that σ shouldn't reveal Sophia's or the Reporters' identities* [5]*. When only the signer's — Sophia's — identity is hidden, this property is called* privacy of signer identity (PSI)*.*

[5] Note that privacy of identities is related to — but very different from — off-the-record. Neither of these definitions is strictly stronger than the other. *Privacy of identities* is weaker in that it assumes that none of the Reporters help in identifying Sophia as the sender, while *off-the-record* makes no such assumptions. However, *privacy of identities* is stronger in that it requires that σ alone reveal nothing about Sophia's identity to anyone other than the Reporters; *off-the-record* allows such leakage, as long as it is not provable. .

Next, what if, at the time at which Sophia has the opportunity to send out her message, she cannot look up Rebekah's public key securely — perhaps because Rebekah has not yet set up an account on the secure messaging system Sophia uses? Then, it would be ideal for Sophia to need nothing other than Rebekah's identity (and some global public parameters) in order to include her as a designated verifier. Rebekah would then be able to get the appropriate key from a trusted authority such as the International Press Institute[6] (having proved that she is, in fact, Rebekah), and would be able to use that key to verify Sophia's signature.

Goal 5 (Verifier-Identity-Based (VIB) Signing) *It is desirable that* Sophia *should only need the* Reporters*' identities, not their public keys, in order to produce her designated verifier signature.*

Contribution 3 *We give the first three constructions that achieve unforgeability, off-the-record with any-subset simulation, and consistency. One of them additionally achieves privacy of identities and verifier-identity-based signing.*

The third construction, which additionally achieves privacy of identities and verifier-identity-based signing, may, at first glance, seem strictly better; however, the price it pays is two-fold. It uses functional encryption (which requires strong computational assumptions), and it requires an involved trusted setup in which a master secret is used to derive verifier keys. Note that such a trusted setup is clearly necessary in order to achieve verifier-identity-based signing.

In contrast, our first two constructions can be instantiated either in the random oracle model, or with a common reference string — in both cases avoiding the need for a master secret key. They use only standard primitives such as pseudorandom functions, pseudorandom generators, key agreement and NIZKs. The first construction uses these primitives in a black-box way; the second construction uses specific instances of these primitives, for concrete efficiency.

In the following subsections, we give an overview of previous work and then discuss our results in more detail.

1.2 Flavors of Multi-Designated Verifier Signatures

There are many ways to define MDVS and its properties. Figure 1 summarizes the approaches taken by prior work, compared to our own.

There are several different flavors of verification. In some MDVS schemes, even a single designated verifier cannot link a signature to the signer; the designated verifiers need to work together in order to verify a signature. Thus, we have two notions of verification: *local* verification and *cooperative* verification (where *all* designated verifiers need to cooperate in order to verify the signature).

Recall that the off-the-record property states that an outsider cannot determine whether a given signature was created by the signer or simulated by the

[6] This trusted authority can also be distributed; perhaps the master secret is secret-shared across several different institutions, who must collaborate in order to produce a secret verification key.

designated verifiers. We have three flavors of such simulateability: *one* designated verifier (out of n) can by himself simulate a signature (as done by [Tia12])[7], *all* designated verifiers need to collude in order to simulate a signature (all other works on MDVS), or *any subset* of the designated verifiers can simulate a signature (this paper). Of course, the simulated signature should remain indistinguishable from a real one even in the presence of the secrets held by the simulating parties.

There is also the standard security property of signature schemes, which is *unforgeability*; no one (except the signer) should be able to construct a signature that any verifier will accept as a valid signature from that signer. There are two flavors of unforgeability. The first is *weak* unforgeability, where designated verifiers can forge, but others cannot. The second is *strong* unforgeability, where a designated verifier can distinguish between real signatures and signatures simulated by other verifiers; that is, even other designated verifiers cannot fool a verifier into accepting a simulated signature.[8]

Schemes	PSI	Verification	Simulation	Unforgeability	Signature Size
[JSI96,LV04,LSMP07] [Cho08,Ver08,ZAYS12]	No	Local	All	Weak	$O(1)$
Our work, from standard primitives	No	Local	Any subset $\mathcal{C}$	Strong	$O(\lvert\mathcal{D}\rvert)$
[NSM05,Cho06]	Yes	All	All	Weak	$O(\lvert\mathcal{D}\rvert)$
[MW08,SHCL08,Cha11]	Yes	All	All	Weak	$O(1)$
[SKS06,LV07,Ver08,ZAYS12]	Yes	Local	All	Weak	$O(\lvert\mathcal{D}\rvert)$
[Tia12]	Yes	Local	One	Weak	$O(1)$
Our work, from FE	Yes	Local	Any subset $\mathcal{C}$ of size up to t	Strong	$O(t)$

Fig. 1. Existing MDVS constructions and their properties. Let $\mathcal{D}$ be the set of designated verifiers, and $t \leq |\mathcal{D}|$ be an upper bound on the set of colluding designated verifiers $\mathcal{C} \subseteq \mathcal{D}$.

1.3 Our Contributions

We propose formal definitions of all the relevant security properties of MDVS in the strongest flavor, including the definition of off-the-record with any-subset simulation. We also give the first formal (game based) definition of consistency,

[7] If *only* one designated verifier can simulate a signature, it must be distinguishable from a real signature by other verifiers (by the strong unforgeability property). Two colluding verifiers would be able to prove to an outsider that a given signature is not a simulation by showing that it verifiers for both of them. So, any-subset simulation gives strictly stronger off-the-record guarantees than one-verifier simulation.

[8] Note that when all designated verifiers are needed for the simulation, then a designated verifier will be able to distinguish a simulation from a real signature based on whether he participated in the simulation of the signature. However, if this is the only way he can distinguish, then the signature scheme has *weak* unforgeability, since the simulated signature is still a valid forgery.

where a corrupt signer can collude with some of the designated verifiers to create an inconsistent signature.

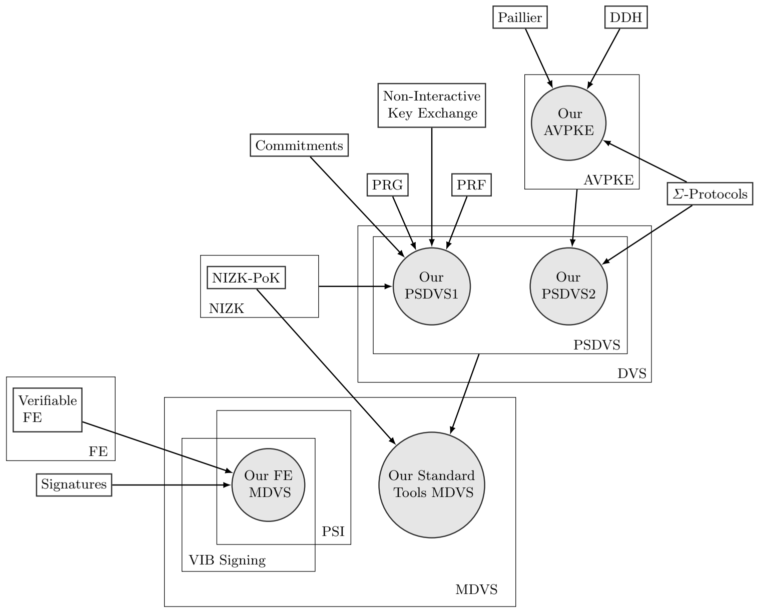

Fig. 2. Our MDVS constructions and building blocks.

We then give several different constructions of MDVS that achieve these properties, including local verification, off-the-record with any-subset simulation, and strong unforgeability. Our constructions, and the tools they require, are mapped out in Figure 2. In particular, these are the first constructions that combine any-subset simulation and with strong unforgeability, as described in Figure 1. We get these results at the expense of signature sizes that are larger than in some of the earlier constructions. However, this is unavoidable, as shown in Theorem 1 below.

Theorem 1. *Any MDVS with any-subset simulation and strong unforgeability must have signature size $\Omega(|\mathcal{D}|)$.*

Remark 2. It may seem from the table that our functional encryption based scheme contradicts the theorem, but this is not the case. It can be instantiated such that signatures can be simulated by collusions up to a certain maximal size t, and then signatures will be of size $\Omega(|\mathcal{C}|)$. However, if we want *any* subset to be able to simulate, the signature size is $\Omega(|\mathcal{D}|)$, in accordance with the theorem.

Proof. Imagine that we give all the verifiers' keys to a sender and a receiver; the sender can now encode an arbitrary subset $\mathcal{C} \subseteq \mathcal{D}$ by letting $\mathcal{C}$ construct a simulated signature σ on some default message, and sending it to the receiver. The receiver can infer $\mathcal{C}$ from σ: by strong unforgeability, all verifiers' keys outside $\mathcal{C}$ will reject σ, whereas keys in $\mathcal{C}$ will accept, since we require the simulation to look convincing even given the secret keys in $\mathcal{C}$. It follows that σ must consist of enough bits to determine $\mathcal{C}$, which is $\log_2(2^{|\mathcal{D}|}) = |\mathcal{D}|$. $\square$

Why First Ideas Fail

Using MACs Black-box usage of a standard MAC scheme cannot help us combine unforgeability with consistency.[9] There are two straightforward ways to use a standard MAC scheme in this context: sharing a MAC key among the entire group, and sharing MAC keys pairwise. Sharing a single key does not provide the desired notion of unforgeability, since any member of the group can forge messages from any other member. Sharing keys pairwise does not provide the desired notion of consistency. If recipients R_1 and R_2 are the chosen recipients of a message, and R_1 receives a message he accepts as coming from S, he cannot be sure that R_2 would also accept that message: If S is corrupt, he could include a valid MAC for R_1 and an invalid MAC for R_2.

Using Proofs of Knowledge A standard technique for making designated verifier signatures for a single verifier is to start from an interactive protocol that proves knowledge of either the signer's or the verifier's secret key, and turn this into a signature scheme using the Fiat-Shamir paradigm. It may seem natural to try to build an MDVS from this. However, it turns out to be challenging to achieve strong unforgeability using this technique; a signature cannot consist of a proof of knowledge of the signer's or one of the verifiers' secret keys, since any verifier will be able to convince other verifiers to accept a signature that did not come from the signer. For the same reason, a signature cannot consist of a proof of knowledge of the signer's secret key or some subset of the verifiers' secret keys.

MDVS from Standard Primitives Our first class of MDVS constructions is based only on standard primitives. With one exception specified below, all of these constructions can be instantiated in the random oracle model with no trusted setup. (Without random oracles, we would need to set up a common reference string.)

The idea is that the signer creates a DVS signature for each verifier individually, and then proves the consistency of those signatures.[10]. To support such proofs, we define a new primitive called Publicly Simulatable Designated Verifier Signatures (PSDVS) in Section 3.1, which is a single-verifier DVS equipped with extra properties. We then show, in Section 3.2, that a PSDVS together with a non-interactive zero knowledge proof of knowledge (NIZK-PoK) imply an MDVS for any number of signers and verifiers. Finally, we give some constructions of PSDVS. Our first PSDVS construction (in Section 3.3) uses only generic

[9] Note that our construction from standard primitives does make use of MAC schemes; however, it does so in a complex, non-black-box way.

[10] Simply proving that all of the signatures verify would violate the off-the-record property; instead, the signer proves that either all of the signatures are real, or they are all simulated, as described in Section 3

tools, namely psedudorandom functions, non-interactive key exchange (such as Diffie-Hellman), and non-interactive zero-knowledge proofs of knowledge. Our second PSDVS construction (in Section 3.4) aims at better concrete efficiency. It is based on DDH, strong RSA and Paillier encryption, is secure in the random oracle model, and requires a constant number of exponentiations for all operations. This scheme requires the trusted generation of an RSA modulus so that the factorization remains unknown. We also sketch a variant that requires no trusted setup, is secure in the random oracle model, and only requires (a variant of) the DDH assumption. However, this version requires double discrete log proofs, and therefore requires a non-constant number of exponentiations.

In order to support one of our constructions in which the signer sends an encrypted MAC key, we introduce a new tool we call Authenticated and Verifiable Encryption (AVPKE), which may be of independent interest. This is a variant of Paillier encryption with built-in authentication, and as such it is related to the known primitive "signcryption" [Zhe97]. However, our AVPKE scheme has the additional property that we can give efficient zero-knowledge proofs involving the encrypted message, using the algebraic properties of Paillier encryption.

To sign in our PSDVS schemes, the signer and verifiers first must establish a shared symmetric key k. In some cases they can do this non-interactively, using their secret and public keys, while in other cases the signer must send an encrypted key alongside the signature. After this, the signer sends a MAC on the message under key k; this MAC is based on a pseudorandom function.

MDVS from Functional Encryption. Our last construction is based on Verifiable Functional Encryption (VFE). It has the advantages of additionally meeting the privacy of identities and verifier-identity-based signing properties. Additionally, it can be set up to have smaller signatures if we are willing to make a stronger assumption on the number of colluding verifiers. Namely, the signature size is $O(t)$, where t is the size of the largest number of colluding verifiers we want to tolerate. The downsides are that, with current state of the art, VFE requires non-standard computational assumptions. We also need a trusted setup for generating keys; however, this is unavoidable if we wish to achieve verifier-identity-based signing.

Remark 3. If we are going to put a bound on the size of a collusion, it may seem we can use bounded collusion FE, which can be realized from standard assumptions [GVW12,AV19], and then there is no need for our other constructions from standard primitives. However, this is not true. Bounded collusion FE requires us to fix the bound on collusion size at key generation time; a bound that may later turn out to be too small. Additionally, ciphertext sizes in bounded collusion FE depend on the bound; thus, choosing a large bound to make sure we can handle the application implies a cost in efficiency. The MDVS signature sizes would depend on some upper bound on number of corrupt parties in the *system*, as opposed to on the number of recipients for the signature in question, which may be orders of magnitude smaller.

In a nutshell, the idea behind the functional encryption based construction is to do the proof of knowledge of one of the relevant secret keys "inside the ciphertext". In a little more detail, the idea is to encrypt a list of t *standard* signatures, where t is the maximal size of collusion we want to protect against (that is, $t \geq |\mathcal{C}|$), and the MDVS signature will simply be this ciphertext. To sign, the signer will generate their own standard signature σ_{S} on the message, and then encrypt a list a signatures consisting of σ_{S} followed by $t-1$ dummy values. To verify a signature, a verifier R gets a functional decryption key that will look at the list of signatures inside the ciphertext and output accept or reject. It will accept if the list contains a valid signature from S or a valid signature from R. Now, if a corrupt set of verifiers $\mathcal{C}$ wants to simulate a signature, they will all sign the message and encrypt the list of these signatures. By security of the encryption scheme, this looks like a real signature, and will indeed verify under all verification keys belonging to verifiers in $\mathcal{C}$. However, no honest verifier will accept it as a signature from S, so we have strong unforgeability.

2 Multi-Designated Verifier Signatures

MDVS Algorithms A *multi-designated verifier signature* (MDVS) scheme is defined by the following probabilistic polynomial-time algorithms:

$\mathsf{Setup}(1^\kappa) \to (pp, msk)$**:** On input the security parameter $\kappa \in \mathbb{N}$, outputs public parameters pp and the master secret key msk.

$\mathsf{SignKeyGen}(pp, msk) \to (spk, ssk)$**:** On input the public parameter pp and the master secret key msk, outputs the public key spk and secret key ssk for a signer.

$\mathsf{VerKeyGen}(pp, msk) \to (vpk, vsk)$**:** On input the public parameter pp and the master secret key msk, outputs the public key vpk and secret key vsk for a verifier.

$\mathsf{Sign}(pp, ssk_i, \{vpk_j\}_{j\in\mathcal{D}}, m) \to \sigma$**:** On input the public parameters pp, a secret signing key ssk_i, the public keys of the designated verifiers $\{vpk_j\}_{j\in\mathcal{D}}$, and a message m, outputs a signature σ.

$\mathsf{Verify}(pp, spk_i, vsk_j, \{vpk_{j'}\}_{j'\in\mathcal{D}}, m, \sigma) \to d$**:** On input the public parameters pp, a public verification key spk_i, a secret key vsk_j of a verifier such that $j \in \mathcal{D}$, the public keys of the designated verifiers $\{vpk_j\}_{j\in\mathcal{D}}$, a message m, and a signature σ, outputs a boolean decision d: $d = 1$ (accept) or $d = 0$ (reject).

$\mathsf{Sim}(pp, spk_i, \{vpk_j\}_{j\in\mathcal{D}}, \{vsk_j\}_{j\in\mathcal{C}}, m) \to \sigma'$**:** On input public parameters pp, a public verification key spk_i, the public keys of the designated verifiers $\{vpk_j\}_{j\in\mathcal{D}}$, the secret keys of the corrupt designated verifiers $\{vsk_j\}_{j\in\mathcal{C}}$, and a message m, outputs a simulated signature σ'.

The different algorithms take many different inputs, which are not all needed for all of our constructions. For instance, the constructions based on standard primitives (Section 3) do not need a master secret key; they allow key pairs for signers and verifiers to be generated locally. Additionally, some of our constructions do not use the signers' and verifiers' public keys in all of the algorithms in which they appear as inputs above. Thus, to simplify the notation we exclude these inputs in later sections whenever they are not needed.

MDVS Properties Let σ be a signature from signer i on message m and designated for verifiers $\mathcal{D}$. We ask for the following (informal) properties:

Correctness: All verifiers $j \in \mathcal{D}$ are able to verify an honestly generated signature σ.

Consistency: If there exists one verifier $j \in \mathcal{D}$ that accepts the signature σ, then all other designated verifiers (i.e. all $j' \in \mathcal{D} \setminus \{j\}$) also accept the signature.

Unforgeability: An adversary without knowledge of the secret key ssk_i for signer i cannot create a signature σ' that is accepted by any designated verifier as a signature from signer i.

Off-The-Record: Given a signature σ, any malicious subset of the designated verifiers $\mathcal{C} \subseteq \mathcal{D}$ cannot convince any outsider that σ is a signature from signer i (i.e. the malicious set could have simulated the signature themselves).

(Optionally) Privacy of Identities: Any outsider (without colluding with any designated verifiers) cannot determine the identity of the signer and/or the identities of the designated verifiers.

(Optionally) Verifier-Identity-Based Signing: The signer should be able to produce a signature for a set of designated verifiers without requiring any information about them apart from their identities. In other words, we should have $vpk_j = j$ for a verifier with identity j.

Throughout our formal definitions we use the following six oracles:

Signer Key Generation Oracle: $\mathcal{O}_{SK}(i)$

1. If a signer key generation query has previously been performed for i, look up and return the previously generated key.
2. Otherwise, output and store $(spk_i, ssk_i) \leftarrow \mathsf{SignKeyGen}(pp, msk)$.

Verifier Key Generation Oracle: $\mathcal{O}_{VK}(j)$

1. If a verifier key generation query has previously been performed for j, look up and return the previously generated key.
2. Otherwise, output and store $(vpk_j, vsk_j) \leftarrow \mathsf{VerKeyGen}(pp, msk)$.

Public Signer Key Generation Oracle: $\mathcal{O}_{SPK}(i)$

1. $(spk_i, ssk_i) \leftarrow \mathcal{O}_{SK}(i)$.
2. Output spk_i.

Public Verifier Key Generation Oracle: $\mathcal{O}_{VPK}(j)$

1. $(vpk_j, vsk_j) \leftarrow \mathcal{O}_{VK}(j)$.
2. Output vpk_j.

Signing Oracle: $\mathcal{O}_{S}(i, \mathcal{D}, m)$

1. $(spk_i, ssk_i) \leftarrow \mathcal{O}_{SK}(i)$.
2. For all $j \in \mathcal{D}$: $vpk_j \leftarrow \mathcal{O}_{VPK}(j)$.
3. Output $\sigma \leftarrow \mathsf{Sign}(pp, ssk_i, \{vpk_j\}_{j \in \mathcal{D}}, m)$.

Verification Oracle: $\mathcal{O}_{V}(i, j, \mathcal{D}, m, \sigma)$

1. $spk_i \leftarrow \mathcal{O}_{SPK}(i)$.
2. $(vpk_j, vsk_j) \leftarrow \mathcal{O}_{VK}(j)$.
3. Output $d \leftarrow \mathsf{Verify}(pp, spk_i, vsk_j, \{vpk_{j'}\}_{j' \in \mathcal{D}}, m, \sigma)$.

Definition 1 (Correctness). *Let $\kappa \in \mathbb{N}$ be the security parameter, and let* $\mathsf{MDVS} = (\mathsf{Setup}, \mathsf{SignKeyGen}, \mathsf{VerKeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Sim})$ *be an MDVS scheme.* MDVS *is* correct *if for all signer identities i, messages m, verifier identity sets $\mathcal{D}$ and $j \in \mathcal{D}$, it holds that*

$$\mathsf{Pr}\left[\mathsf{Verify}(pp, spk_i, vsk_j, \{vpk_{j'}\}_{j' \in \mathcal{D}}, m, \sigma) \neq 1\right] = 0,$$

where the inputs to Verify *are generated as follows:*

– $(pp, msk) \leftarrow \mathsf{Setup}(1^\kappa)$*;*
– $(spk_i, ssk_i) \leftarrow \mathsf{SignKeyGen}(pp, msk, i)$*;*
– $(vpk_j, vsk_j) \leftarrow \mathsf{VerKeyGen}(pp, msk, j)$ *for $j \in \mathcal{D}$;*
– $\sigma \leftarrow \mathsf{Sign}(pp, ssk_i, \{vpk_j\}_{j \in \mathcal{D}}, m)$*.*

In Definition 1, we require that all the designated verifiers can verify the signature, without considering what happens for parties that are not designated verifiers (i.e. parties who should not be able to verify the signature). Parties that are not designated verifiers are accounted for by the off-the-record property.

Definition 2 (Consistency). *Let $\kappa \in \mathbb{N}$ be the security parameter, and let* $\mathsf{MDVS} = (\mathsf{Setup}, \mathsf{SignKeyGen}, \mathsf{VerKeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Sim})$ *be an MDVS scheme. Consider the following game between a challenger and an adversary $\mathcal{A}$:*

$\mathsf{Game}^{con}_{\mathsf{MDVS},\mathcal{A}}(\kappa)$

```
1. $(pp, msk) \leftarrow \mathsf{Setup}(1^\kappa)$
2. $(m^*, i^*, \mathcal{D}^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{SK}, \mathcal{O}_{VK}, \mathcal{O}_{SPK}, \mathcal{O}_{VPK}, \mathcal{O}_V}(pp)$
```

We say that $\mathcal{A}$ wins the game if there exist verifiers $j_0, j_1 \in \mathcal{D}^*$ such that:

$$\mathsf{Verify}(pp, spk_{i^*}, vsk_{j_0}, \{vpk_{j'}\}_{j' \in \mathcal{D}^*}, m^*, \sigma^*) = 0,$$
$$\mathsf{Verify}(pp, spk_{i^*}, vsk_{j_1}, \{vpk_{j'}\}_{j' \in \mathcal{D}^*}, m^*, \sigma^*) = 1,$$

where all keys are the honestly generated outputs of the key generation oracles, and $\mathcal{O}_{VK}$ is never queried on j_0 or j_1.
MDVS *is* consistent *if, for all PPT adversaries $\mathcal{A}$,*

$$\mathsf{adv}^{con}_{\mathsf{MDVS},\mathcal{A}}(\kappa) = \Pr\left[\mathcal{A} \textit{ wins } \mathsf{Game}^{con}_{\mathsf{MDVS},\mathcal{A}}(\kappa)\right] \leq \mathsf{negl}(\kappa).$$

Definition 2 states that even a valid signer (i.e. someone who knows a secret signing key) cannot create an inconsistent signature that will be accepted by some designated verifiers and rejected by others. By the correctness property, an honestly generated signature is accepted by all designated verifiers. By design, corrupt designated verifiers can construct an inconsistent signature, since some verifiers will accept it (i.e. those verifiers that created it), while the remaining honest designated verifiers will reject the simulated signature. Thus, we need to ask for $j \neq j_0, j_1$ for all queries j to the oracle $\mathcal{O}_{VK}$.

Definition 3 (Existential Unforgeability). *Let $\kappa \in \mathbb{N}$ be the security parameter, and let* $\mathsf{MDVS} = (\mathsf{Setup}, \mathsf{SignKeyGen}, \mathsf{VerKeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Sim})$ *be an MDVS scheme. Consider the following game between a challenger and an adversary $\mathcal{A}$:*

$$\mathsf{Game}^{euf}_{\mathsf{MDVS},\mathcal{A}}(\kappa)$$

```
1. $(pp, msk) \leftarrow \mathsf{Setup}(1^\kappa)$
2. $(m^*, i^*, \mathcal{D}^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{SK}, \mathcal{O}_{VK}, \mathcal{O}_{SPK}, \mathcal{O}_{VPK}, \mathcal{O}_S}(pp)$
```

We say that $\mathcal{A}$ wins the game if we have all of the following:

- *for all queries i to oracle $\mathcal{O}_{SK}$, it holds that $i^* \neq i$;*
- *for all queries $(i, \mathcal{D}, m)$ to oracle $\mathcal{O}_S$ that result in signature σ, it holds that $(i^*, \mathcal{D}^*, m^*) \neq (i, \mathcal{D}, m)$;*
- *there exists a verifier $j' \in \mathcal{D}^*$ such that for all queries j to oracle $\mathcal{O}_{VK}$, it holds that $j' \neq j$ and*

$$\mathsf{Verify}(pp, spk_{i^*}, vsk_{j'}, \{vpk_{j''}\}_{j'' \in \mathcal{D}^*}, m^*, \sigma^*) = 1,$$

where all keys are honestly generated outputs of the key generation oracles.

MDVS *is* existentially unforgeable *if, for all PPT adversaries $\mathcal{A}$,*

$$\mathsf{adv}^{euf}_{\mathsf{MDVS},\mathcal{A}}(\kappa) = \Pr\left[\mathcal{A} \text{ wins } \mathsf{Game}^{euf}_{\mathsf{MDVS},\mathcal{A}}(\kappa)\right] \leq \mathsf{negl}(\kappa).$$

Definition 3 states that an adversary cannot create a signature that any honest verifier will accept as coming from a signer whose secret signing key the adversary does not know. The adversary will always get the public keys of the involved parties, i.e. signer with identity i^* and the designated verifiers $\mathcal{D}$, through the key generation oracles. He is also allowed to obtain the secret keys of every party except the signer i^* and at least one designated verifier. The reason why we need at least one honest verifier is that corrupt verifiers can create a simulated signature that will look like a real signature with respect to their own verifier secret keys. However, this simulation will be rejected by any honest designated verifier, i.e. the simulation will be a valid forgery for the corrupt verifiers, but not for the honest verifiers.

Definition 4 (Off-The-Record). *Let $\kappa \in \mathbb{N}$ be the security parameter, let* $\mathsf{MDVS} = (\mathsf{Setup}, \mathsf{SignKeyGen}, \mathsf{VerKeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Sim})$ *be an MDVS scheme, and let t be an upper bound on the number of verifiers an adversary $\mathcal{A}$ can corrupt. Consider the following game between a challenger and an adversary $\mathcal{A}$, where all keys are honestly generated outputs of the key generation oracles:*

$$\mathsf{Game}^{otr}_{\mathsf{MDVS},\mathsf{Sim},\mathcal{A}}(\kappa)$$

```
1. $(pp, msk) \leftarrow \mathsf{Setup}(1^\kappa)$
2. $(i^*, \mathcal{D}^*, m^*, \mathcal{C}^*) \leftarrow \mathcal{A}^{\mathcal{O}_{SK}, \mathcal{O}_{VK}, \mathcal{O}_{SPK}, \mathcal{O}_{VPK}, \mathcal{O}_S, \mathcal{O}_V}(pp)$
3. $b \leftarrow \{0, 1\}$
4. $\sigma_0 \leftarrow \mathsf{Sign}(pp, ssk_{i^*}, \{vpk_j\}_{j \in \mathcal{D}^*}, m^*)$
5. $\sigma_1 \leftarrow \mathsf{Sim}(pp, spk_{i^*}, \{vpk_j\}_{j \in \mathcal{D}^*}, \{vsk_j\}_{j \in \mathcal{C}^*}, m^*)$
6. $b' \leftarrow \mathcal{A}^{\mathcal{O}_{SK}, \mathcal{O}_{VK}, \mathcal{O}_{SPK}, \mathcal{O}_{VPK}, \mathcal{O}_S, \mathcal{O}_V}(\sigma_b)$
```

We say that $\mathcal{A}$ wins the game if $b' = b$, and all of the following hold:

- *$|\mathcal{C}^*| \leq t$ and $\mathcal{C}^* \subseteq \mathcal{D}^*$;*
- *for all queries i to oracle $\mathcal{O}_{SK}$ it holds that $i^* \neq i$;*
- *for all queries j to oracle $\mathcal{O}_{VK}$ it holds that $j \notin \mathcal{D}^* \backslash \mathcal{C}^*$;*
- *for all queries $(i, j, \mathcal{D}, m, \sigma)$ to $\mathcal{O}_V$ it holds that $\sigma_b \neq \sigma$.*

*We say that an MDVS scheme is t-*off-the-record *if, for all PPT adversaries $\mathcal{A}$,*

$$\mathsf{adv}^{otr}_{\mathsf{MDVS},\mathsf{Sim},\mathcal{A}}(\kappa) = \Pr\left[\mathcal{A} \text{ wins } \mathsf{Game}^{otr}_{\mathsf{MDVS},\mathsf{Sim},\mathcal{A}}(\kappa)\right] - \frac{1}{2} \leq \mathsf{negl}(\kappa).$$

If a scheme supports $t = |\mathcal{D}|$, we say that it is off-the-record.

Definition 4 states that any adversary that corrupts a subset (of size t) of the designated verifiers $\mathcal{C}^*$ cannot determine whether the received signature was created by real signer i^* or simulated by the corrupt verifiers $\mathcal{C}^*$. The adversary is not allowed to see the secret keys for the designated verifiers that are in $\mathcal{D}^* \backslash \mathcal{C}^*$. If the adversary was allowed to get secret keys of additional parties in $\mathcal{D}^*$ (which are not in $\mathcal{C}^*$), then he would be able to distinguish trivially, since any honest designated verifiers (i.e. any $j \in \mathcal{D}^* \backslash \mathcal{C}^*$) can distinguish simulated signatures from real signatures (from the unforgeability property).

Definition 5 (Privacy of Identities). *Let $\kappa \in \mathbb{N}$ be the security parameter, and let* $\mathsf{MDVS} = (\mathsf{Setup}, \mathsf{SignKeyGen}, \mathsf{VerKeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Sim})$ *be an MDVS scheme. Consider the following game between a challenger and an adversary $\mathcal{A}$, where all keys are the honestly generated outputs of the key generation oracles:*

$\mathsf{Game}^{pri}_{\mathsf{MDVS},\mathcal{A}}(\kappa)$

```
1. $(pp, msk) \leftarrow \mathsf{Setup}(1^\kappa)$
2. $(m^*, i_0, i_1, \mathcal{D}_0, \mathcal{D}_1) \leftarrow \mathcal{A}^{\mathcal{O}_{SK}, \mathcal{O}_{VK}, \mathcal{O}_{SPK}, \mathcal{O}_{VPK}, \mathcal{O}_S, \mathcal{O}_V}(pp)$
3. $b \leftarrow \{0, 1\}$
4. $\sigma^* \leftarrow \mathsf{Sign}(pp, ssk_{i_b}, \{vpk_j\}_{j \in \mathcal{D}_b}, m^*)$
5. $b' \leftarrow \mathcal{A}^{\mathcal{O}_{SK}, \mathcal{O}_{VK}, \mathcal{O}_{SPK}, \mathcal{O}_{VPK}, \mathcal{O}_S, \mathcal{O}_V}(\sigma^*)$
```

We say that $\mathcal{A}$ wins the game if $b = b'$, and all of the following hold:

- *$|\mathcal{D}_0| = |\mathcal{D}_1|$;*
- *for all queries i to $\mathcal{O}_{SK}$, it holds that $i \notin \{i_0, i_1\}$;*
- *for all gueries j to $\mathcal{O}_{VK}$, it holds that $j \notin \mathcal{D}_0 \cup \mathcal{D}_1$;*
- *for all queries $(i, j, \mathcal{D}, m, \sigma)$ to $\mathcal{O}_V$, it holds that $\sigma^* \neq \sigma$.*

MDVS *has* privacy of identities *if, for all PPT adversaries $\mathcal{A}$,*

$$\mathsf{adv}^{pri}_{\mathsf{MDVS},\mathcal{A}}(\kappa) = \Pr\left[\mathcal{A} \text{ wins } \mathsf{Game}^{pri}_{\mathsf{MDVS},\mathcal{A}}(\kappa)\right] - \frac{1}{2} \leq \mathsf{negl}(\kappa).$$

We say that MDVS *has additional properties as follows:*

- *privacy of the signer's identity (PSI) if we make the restriction that $\mathcal{D}_0 = \mathcal{D}_1$;*
- *privacy of the designated verifiers' identities (PVI) if we make the restriction that $i_0 = i_1$.*

Definition 5 states that an adversary cannot distinguish between signatures from two different signers (PSI) if he does not know the secret key of any of the signers or designated verifiers (as designated verifiers *are* allowed to identify the signer). Furthermore, it should not help him to see other signatures that he knows are from the signers in question.

In addition, if we vary the verifier sets ($\mathcal{D}_0 \neq \mathcal{D}_1$), then the MDVS scheme has privacy of designated verifier's identities (PVI), which means that any outsider without knowledge of any secret keys cannot distinguish between signatures meant for different verifiers.

Definition 6 (Verifier-Identity-Based Signing). *We say that an MDVS scheme has* verifier-identity-based signing *if for honestly generated verifier keys* (vsk_j, vpk_j) *for verifier with identity* j*, we have* $vpk_j = j$.

Note that, in order to achieve verifier-identity-based signing, verifier key generation must require a master secret key msk. Otherwise, any outsider would be able to generate a verification key for verifier j, and use it to verify signatures meant only for that verifier.

Relation to Previous Definitions Our definition of MDVS is consistent with previous work in this area, but with some differences. Our MDVS syntax closely follows the one introduced by [LV04], but we allow for a master secret key in the case where the keys are generated by a trusted party (like in our construction based on functional encryption). Our security definitions are adapted from those in [LV04,ZAYS12] to capture the flexibility introduced by allowing any subset of designated verifiers to simulate a signature, thus providing better deniability properties. Finally, we formalize consistency as an additional and desirable requirement.

3 Standard Primitive-Based MDVS Constructions

In this section we show how to create an MDVS scheme that uses only standard primitives, such as key exchange, commitments, pseudorandom functions and generators, and non-interactive zero knowledge proofs.

On a high level, one way to build an MDVS is for the signer to use a separate DVS with each verifier; the MDVS signature would then consist of a vector of individual DVS signatures. This gives us almost everything we need — the remaining issue is consistency. Each verifier can verify one of the DVS signatures, but is not convinced that all of the other verifiers will come to the same conclusion.

A solution to this consistency issue is to include as part of the MDVS signature a zero knowledge proof that all of the DVS signatures verify. However, this introduces a new issue with off-the-record. Now, a colluding set of verifiers will not be able to simulate a signature unless *all* of the verifiers collude. In order to produce such a convincing zero knowledge proof as part of the signature, they would need to forge signatures for the other verifiers in the underlying DVS scheme, which they should not be able to do.

So, instead of using a zero knowledge proof of knowledge that all of the DVS signatures verify, we use a proof that *either* all of the DVS signatures verify, *or they are all simulated.* Then, a corrupt set of verifiers can simulate all of the underlying DVS signatures — with the caveat that the signatures they simulate for themselves should be convincing simulations even in the presence of their secret keys — and, instead of proving that all of the signatures verify, they prove that all of the signatures are simulations.

In order to support such proofs, in Section 3.1 we introduce a new primitive called a Provably Simulatable DVS (PSDVS). Then, in Section 3.2 we show how to compose PSDVS instances into an MDVS. In Section 3.3 we build a PSDVS out of generic standard primitives. In Section 3.4 we build a more efficient PSDVS out of concrete instantiations of those primitives.

3.1 New Primitive: Provably Simulatable Designated-Verifier Signatures (PSDVS)

Designated Verifier Signatures (DVS) have a simulation algorithm Sim which is used to satisfy the off-the-record property (the single-verifier equivalent of Definition 4). Given the signer's public key, the verifier's secret key and a message m, Sim should return a signature which is indistinguishable from a real signature. A *Provably Simulatable DVS (PSDVS)*, in addition to correctness and existential unforgeability, must have two notions of transcript simulation: *public* simulation and *verifier* simulation. Furthermore, for each of these, it should be possible to produce a zero knowledge proof that the signature produced is a simulation. Additionally, it should be possible to similarly prove that a real signature is, in fact, real. This makes a PSDVS well suited for use in an MDVS which uses a zero knowledge proof of knowledge to enforce consistency.[11]

More formally, a PSDVS consists of the standard DVS algorithms Setup, $\mathsf{SignKeyGen}$, $\mathsf{VerKeyGen}$, Sign, Verify, as well as five additional algorithms: $\mathsf{RealSigVal}$ to validate real signatures, and $\mathsf{PubSigSim}$, $\mathsf{PubSigVal}$, $\mathsf{VerSigSim}$ and $\mathsf{VerSigVal}$ to simulate signatures and to validate such simulations.

Definition 7. *A PSDVS must satisfy the standard notions of correctness and existential unforgeability. Additionally, it should satisfy* PubSigSim indstinguishability *(Definition 8),* PubSigSim correctness *(Definition 9),* PubSigSim soundness *(Definition 10),* VerSigSim indstinguishability *(Definition 11),* VerSigSim correctness *(Definition 12),* VerSigSim soundness *(Definition 13),* provable signing correctness *(Definition 14), and* provable signing soundness *(Definition 15).*

Provable Public Simulation As in *PSI* (Definition 5), anyone should be able to produce a signature that is indistinguishable from a real signature. Additionally, the party simulating the signature should be able to produce a proof that this is *not* a real signature. This proof will be incorporated into the MDVS proof of consistency; the colluding verifiers, when producing a simulation, need to prove that all underlying PSDVS signatures are real, or that they are all fake.

[11] While these additional properties allow the composition of PSDVS into an MDVS, they are not useful when PSDVS is used on its own.

In other words, we require two additional algorithms, as follows:

1. $\mathsf{PubSigSim}(pp, spk, vpk, m) \to (\sigma, \pi)$
2. $\mathsf{PubSigVal}(pp, spk, vpk, m, \sigma, \pi) \to d \in \{0, 1\}$

The colluding verifiers will produce a public simulation in the underlying PSDVS for verifiers outside their coalition, and use PubSigSim to prove that this simulation is not a real signature. π will not be explicitly included in the proof of "the underlying PSDVS signatures are all real or all fake," of course, as it would give away the fact that all underlying signatures are fake, as opposed to all being real; rather, it will be wrapped in a larger zero knowledge proof.

Definition 8 (PubSigSim Indistinguishability). *We say that the PSDVS has* PubSigSim Indistinguishability *if* PubSigSim *produces a signature σ that is indistinguishable from real. More formally, an adversary should not be able to win the following game with probability non-negligibly more than half:*

$$\mathsf{Game}^{PubSigSim\text{-}Ind}_{\mathsf{PVDVS},\mathcal{A}}(\kappa)$$

```
1. $pp \leftarrow \mathsf{Setup}(1^\kappa)$
2. $(spk, ssk) \leftarrow \mathsf{SignKeyGen}(pp)$
3. $(vpk, vsk) \leftarrow \mathsf{VerKeyGen}(pp)$
4. $m^* \leftarrow \mathcal{A}^{\mathcal{O}_S,\mathcal{O}_V}(spk, vpk)$
5. $b \leftarrow \{0, 1\}$
6. $\sigma_0 \leftarrow \mathsf{Sign}(pp, ssk, vpk, m^*)$
7. $(\sigma_1, \pi) \leftarrow \mathsf{PubSigSim}(pp, spk, vpk, m^*)$
8. $b' \leftarrow \mathcal{A}^{\mathcal{O}_S,\mathcal{O}_V}(pp, spk, vpk, m^*, \sigma_b)$
```

We say that $\mathcal{A}$ wins the PubSigSim-Ind *game if $b = b'$ and for all queries (m, σ) to $\mathcal{O}_V$, it holds that $(m, \sigma) \neq (m^*, \sigma_b)$.*

Definition 9 (PubSigSim Correctness). *We say that the PSDVS has* PubSigSim Correctness *if for all $pp \leftarrow \mathsf{Setup}(1^\kappa)$; $(spk, ssk) \leftarrow \mathsf{SignKeyGen}(pp)$; $(vpk, vsk) \leftarrow \mathsf{VerKeyGen}(pp)$; $m \in \{0, 1\}^*$; $(\sigma, \pi) \leftarrow \mathsf{PubSigSim}(pp, spk, vpk, m)$;*

$$\Pr[\mathsf{PubSigVal}(pp, spk, vpk, m, \sigma, \pi) = 1] = 1.$$

Definition 10 (PubSigSim Soundness). *We say that the PSDVS has* PubSigSim Soundness *if it is hard to construct a signature σ which is accepted by the verifier algorithm and at the same time can be proven to be a simulated signature. More formally, an adversary should not be able to win the following game with non-negligible probability:*

$$\mathsf{Game}^{PubSigSim\text{-}Sound}_{\mathsf{PVDVS},\mathcal{A}}(\kappa)$$

```
1. $pp \leftarrow \mathsf{Setup}(1^\kappa)$
2. $(spk, ssk) \leftarrow \mathsf{SignKeyGen}(pp)$
3. $(vpk, vsk) \leftarrow \mathsf{VerKeyGen}(pp)$
4. $(m^*, \sigma^*, \pi^*) \leftarrow \mathcal{A}(pp, ssk, spk, vpk)$
```

We say that $\mathcal{A}$ wins the PubSigSim-Sound *game if* $\mathsf{Verify}(pp, vsk, m^*, \sigma^*) = 1$ *and* $\mathsf{PubSigVal}(pp, spk, vpk, m^*, \sigma^*, \pi^*) = 1$.

Provable Verifier Simulation As in *off-the-record* (Definition 4), a verifier should be able to produce a signature that is indistinguishable from a real signature, *even given its secret key*. Additionally, the verifier should be able to produce a proof that the signature is *not* a real signature (that is, that the verifier, and not the signer, produced it). This proof will be incorporated into the MDVS proof of consistency.

In other words, we require two additional algorithms, as follows:

1. $\mathsf{VerSigSim}(pp, spk, vpk, vsk, m) \rightarrow (\sigma, \pi)$
2. $\mathsf{VerSigVal}(pp, spk, vpk, m, \sigma, \pi) \rightarrow d \in \{0, 1\}$

The colluding verifiers will produce a verifier simulation in the underlying PSDVS for verifiers *inside* their coalition, and use $\mathsf{VerSigSim}$ to prove that this simulation is not a real signature.

Definition 11 (VerSigSim Indistinguishability). *We say that the PSDVS has* VerSigSim Indistinguishability *if* $\mathsf{VerSigSim}$ *produces a signature* σ *that is indistinguishable from real. More formally, an adversary should not be able to win the following game with probability non-negligibly more than half:*

$$\mathsf{Game}^{VerSigSim\text{-}Ind}_{\mathsf{PVDVS},\mathcal{A}}(\kappa)$$

```
1. $pp \leftarrow \mathsf{Setup}(1^\kappa)$
2. $(spk, ssk) \leftarrow \mathsf{SignKeyGen}(pp)$
3. $(vpk, vsk) \leftarrow \mathsf{VerKeyGen}(pp)$
4. $m^* \leftarrow \mathcal{A}^{\mathcal{O}_S}(pp, spk, vpk, vsk)$
5. $b \leftarrow_\$ \{0, 1\}$
6. $\sigma_0 \leftarrow \mathsf{Sign}(pp, ssk, vpk, m^*)$
7. $(\sigma_1, \pi) \leftarrow \mathsf{VerSigSim}(pp, spk, vsk, m^*)$
8. $b' \leftarrow \mathcal{A}^{\mathcal{O}_S}(pp, spk, vpk, vsk, m^*, \sigma_b)$
```

We say that $\mathcal{A}$ *wins the* VerSigSim-Ind *game if* $b = b'$.

Definition 12 (VerSigSim Correctness). *We say that the PSDVS has* VerSigSim Correctness *if for all* $pp \leftarrow \mathsf{Setup}(1^\kappa)$, $(spk, ssk) \leftarrow \mathsf{SignKeyGen}(pp)$, $(vpk, vsk) \leftarrow \mathsf{VerKeyGen}(pp)$, $m \in \{0, 1\}^*$, $(\sigma, \pi) \leftarrow \mathsf{VerSigSim}(pp, spk, vpk, vsk, m)$,

$$\Pr[\mathsf{VerSigVal}(pp, spk, vpk, m, \sigma, \pi) = 1] = 1.$$

Definition 13 (VerSigSim Soundness). *We say that the PSDVS has* VerSigSim Soundness *if the signer is not able to produce* σ *and* π *that pass the validation check* $\mathsf{VerSigVal}$*, i.e.* π *is a proof that* σ *was not produced by the signer. More formally, an adversary should not be able to win the following game with non-negligible probability:*

$$\mathsf{Game}^{VerSigSim\text{-}Sound}_{\mathsf{PVDVS},\mathcal{A}}(\kappa)$$

```
1. $pp \leftarrow \mathsf{Setup}(1^\kappa)$
2. $(spk, ssk) \leftarrow \mathsf{SignKeyGen}(pp)$
3. $(vpk, vsk) \leftarrow \mathsf{VerKeyGen}(pp)$
4. $(m^*, \sigma^*, \pi^*) \leftarrow \mathcal{A}(pp, ssk, spk, vpk)$
```

$\mathcal{A}$ *wins the* VerSigSim-Sound *game if* $\mathsf{VerSigVal}(pp, spk, vpk, m^*, \sigma^*, \pi^*) = 1$.

Provable Signing Lastly, we require a provable variant of signing, so that the signer is able to produce a proof that a signature is real. In other words, we require the signing algorithm $\mathsf{Sign}(pp, spk, ssk, vpk, m) \to (\sigma, \pi)$ to output π as well. We also require one additional validation algorithm, as follows:

$$\mathsf{RealSigVal}(pp, spk, vpk, m, \sigma, \pi) \to d \in \{0, 1\}$$

Definition 14 (Provable Signing Correctness). *We say that the PSDVS has* Provable Signing Correctness *if* $\forall pp \leftarrow \mathsf{Setup}(1^\kappa)$, $(spk, ssk) \leftarrow \mathsf{SignKeyGen}(pp)$, $(vpk, vsk) \leftarrow \mathsf{VerKeyGen}(pp)$, $m \in \{0,1\}^*$, $(\sigma, \pi) \leftarrow \mathsf{Sign}(pp, spk, ssk, vpk, m)$,

$$\Pr[\mathsf{RealSigVal}(pp, spk, vpk, m, \sigma, \pi) = 1] = 1.$$

Definition 15 (Provable Signing Soundness). *We say that the PSDVS has* Provable Signing Soundness *if the proof of correctness π produced by* Sign *does not verify unless σ verifies. More formally, an adversary should not be able to win the following game with non-negligible probability:*

$$\mathsf{Game}^{Sign\text{-}Sound}_{\mathsf{PVDVS},\mathcal{A}}(\kappa)$$

1. $pp \leftarrow \mathsf{Setup}(1^\kappa)$
2. $(spk, ssk) \leftarrow \mathsf{SignKeyGen}(pp)$
3. $(vpk, vsk) \leftarrow \mathsf{VerKeyGen}(pp)$
4. $(m^*, \sigma^*, \pi^*) \leftarrow \mathcal{A}(pp, ssk, spk, vpk)$

We say that $\mathcal{A}$ wins the Sign-Sound *game if* $\mathsf{RealSigVal}(pp, spk, vpk, m^*, \sigma^*, \pi^*) = 1$ *and* $\mathsf{Verify}(pp, spk, vsk, m^*, \sigma^*) = 0$.

Note that none of these proofs π are parts of the signature. If included in the signature, such proofs would allow an adversary to distinguish a simulation from a real signature.

3.2 Standard Primitive-Based MDVS Construction

Given a PSDVS, as defined in Section 3.1, we can build an MDVS. The transformation is straightforward: the signer uses the PSDVS to sign a message for each verifier, and proves consistency using a non-interactive zero knowledge proof of knowledge. The proof of consistency will claim that either all of the PSDVS signatures verify, or all of them are simulated. Construction 1 describes this transformation.

Construction 1 *Let* $\mathsf{PSDVS} = (\mathsf{Setup}, \mathsf{SignKeyGen}, \mathsf{VerKeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{RealSigVal}, \mathsf{PubSigSim}, \mathsf{PubSigVal}, \mathsf{VerSigSim}, \mathsf{VerSigVal})$ *be a provably simulatable designated verifier signature scheme, and* $\mathsf{NIZK\text{-}PoK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *be a non-interactive zero knowledge proof of knowledge system and $\mathcal{R}_{cons}$ a relation that we will define later in the protocol.*

$\mathsf{Setup}(1^\kappa)$**:**

1. $crs \leftarrow \mathsf{NIZK\text{-}PoK.Setup}(1^\kappa, \mathcal{R}_{cons})$.
2. $\mathsf{PSDVS}.pp \leftarrow \mathsf{PSDVS.Setup}(1^\kappa)$.

Output $(crs, \mathsf{PSDVS}.pp)$ as the public parameters pp.

$\mathsf{SignKeyGen}(pp)$**:** $(spk_i, ssk_i) \leftarrow \mathsf{PSDVS.SignKeyGen}(\mathsf{PSDVS}.pp)$.

Output (spk_i, ssk_i) as signer i's public/secret key pair.

$\mathsf{VerKeyGen}(pp)$**:** $(vpk_j, vsk_j) \leftarrow \mathsf{PSDVS.VerKeyGen}(\mathsf{PSDVS}.pp)$.

Output (vpk_j, vsk_j) as verifier j's public/secret key pair.

$\mathsf{Sign}(pp, ssk_i, \{vpk_j\}_{j\in\mathcal{D}}, m)$**:**

1. For every verifier $j \in \mathcal{D}$, compute a signature and proof of signature validity as $(\sigma_j, \pi_j) \leftarrow \mathsf{PSDVS.Sign}(\mathsf{PSDVS}.pp, ssk_i, vpk_j, m)$.
2. Create a proof π of consistency, i.e a proof of knowledge of $\{\pi_j\}_{j\in\mathcal{D}}$ such that either all signatures are real (as demonstrated by $\{\pi_j\}_{j\in\mathcal{D}}$), or all signatures are fake (as could be demonstrated by the proofs produced by $\mathsf{PSDVS.PubSigSim}$ or $\mathsf{PSDVS.VerSigSim}$).
   More formally, for this $\mathsf{NIZK\text{-}PoK}$ we define a relation for a statement $u = (\mathsf{PSDVS}.pp, spk_i, \{vpk_j\}_{j\in\mathcal{D}}, \{\sigma_j\}_{j\in\mathcal{D}})$ and the witness $w = \{\pi_j\}_{j\in\mathcal{D}}$:

$$\begin{aligned}\mathcal{R}_{cons} = \Big\{ & u = (\mathsf{PSDVS}.pp, spk_i, \{vpk_j\}_{j\in\mathcal{D}}, \{\sigma_j\}_{j\in\mathcal{D}}), w = \{\pi_j\}_{j\in\mathcal{D}} : \\ & \Big(\bigwedge_{j\in\mathcal{D}} \mathsf{PSDVS.RealSigVal}(\mathsf{PSDVS}.pp, spk_i, vpk_j, m, \sigma_j, \pi_j) = 1\Big) \bigvee \\ & \Big(\bigwedge_{j\in\mathcal{D}} \big(\mathsf{PSDVS.VerSigVal}(\mathsf{PSDVS}.pp, spk_i, vpk_j, m, \sigma_j, \pi_j) = 1 \vee \\ & \mathsf{PSDVS.PubSigVal}(\mathsf{PSDVS}.pp, spk_i, vpk_j, m, \sigma_j, \pi_j) = 1\big)\Big)\Big\} \end{aligned} \tag{1}$$

   Let $\pi \leftarrow \mathsf{NIZK\text{-}PoK.Prove}(crs, u = (\mathsf{PSDVS}.pp, spk_i, \{vpk_j\}_{j\in\mathcal{D}}, \{\sigma_j\}_{j\in\mathcal{D}}), w = \{\pi_j\}_{j\in\mathcal{D}})$.
3. $\sigma = (\{\sigma_j\}_{j\in\mathcal{D}}, \pi)$.

Output σ as the signature.

$\mathsf{Verify}(pp, spk_i, vsk_j, m, \sigma = (\{\sigma_j\}_{j\in\mathcal{D}}, \pi))$**:**

1. Let $d_\pi \leftarrow \mathsf{NIZK\text{-}PoK.Verify}(crs, u = (\mathsf{PSDVS}.pp, spk_i, \{vpk_j\}_{j\in\mathcal{D}}, \{\sigma_j\}_{j\in\mathcal{D}}), \pi)$.
2. Let $d \leftarrow \mathsf{PSDVS.Verify}(\mathsf{PSDVS}.pp, spk_i, vsk_j, m, \sigma_j) \wedge d_\pi$.

Output d as the verification decision.

$\mathsf{Sim}(pp, spk_i, \{vpk_j\}_{j\in\mathcal{D}}, \{vsk_j\}_{j\in\mathcal{C}}, m)$**:**

1. For $j \in \mathcal{D} \cap \mathcal{C}$: $(\sigma_j, \pi_j) \leftarrow \mathsf{VerSigSim}(\mathsf{PSDVS}.pp, spk_i, vpk_j, vsk_j, m)$.
2. For $j \in \mathcal{D} \backslash \mathcal{C}$: $(\sigma_j, \pi_j) \leftarrow \mathsf{PubSigSim}(\mathsf{PSDVS}.pp, spk_i, vpk_j, m)$.
3. Use these signatures and proofs to produce the NIZK π for relation $\mathcal{R}_{cons}$.
4. $\sigma = (\{\sigma_j\}_{j\in\mathcal{D}}, \pi)$.

Output σ as the signature.

Theorem 2. *Assume* PSDVS *is a secure provably simulatable designated verifier signature scheme and* $\mathsf{NIZK\text{-}PoK}$ *is a secure non-interactive zero knowledge proof of knowledge system. Then Construction 1 is a correct and secure MDVS scheme (without privacy of identities (Definition 5)).*

Proof. Correctness is apparent by inspection. We show consistency, unforgeability and off-the-record separately.

Claim. Construction 1 is consistent, as per Definition 2.

Assume that Construction 1 is inconsistent; then there exists an adversary $\mathcal{A}$ that can produce a message m^* and signature $\sigma^* = (\{\sigma^*_j\}_{j\in\mathcal{D}^*}, \pi^*)$ such that σ^* verifies for some, but not all, of the intended recipients $\mathcal{D}^*$. We can then use $\mathcal{A}$ to create another adversary $\mathcal{B}$ that can break either the security of the underlying PSDVS, or the security of the $\mathsf{NIZK\text{-}PoK}$.

$\mathcal{B}$ receives pp, ssk, spk, vpk regardless of whether it's playing the *PubSigSim-Sound, VerSigSim-Sound* or *Sign-Sound* games.

It randomly chooses identities to assign the given signer and verifier keys to; it generates the other signer and verifier keys honestly. It answers signing oracle queries honestly, since in all these cases it has the signer secret key. It answers key generation keys honestly as well, unless asked for ssk or the secret key corresponding to vpk; then, it aborts. However, since at least one signer and one verifier secret key must remain unqueried by $\mathcal{A}$, the probability of an abort is not overwhelming. Eventually, it gets $\mathcal{D}^*$ and $(m^*, \sigma^* = (\{\sigma^*_j\}_{j\in\mathcal{D}}, \pi^*))$ from $\mathcal{A}$; by assumption, with non-negligible probability, σ^* is inconsistent.

Assume without loss of generality that $j_0, j_1 \in \mathcal{D}^*$, $\mathsf{Verify}(pp, vsk_{j_0}, m^*, \sigma^*) = 0$, and $\mathsf{Verify}(pp, vsk_{j_1}, m^*, \sigma^*) = 1$. Assume, also without loss of generality, that $\mathsf{NIZK\text{-}PoK.Verify}$ is deterministic; then, the decision d_π regarding the validity of the zero knowledge proof of knowledge π^* must be the same for verifiers R_{j_0} and R_{j_1}, and so it must be that $d_\pi = 1$. It follows that in order for $\mathsf{Verify}(pp, vsk_{j_0}, m^*, \sigma^*) = 0$ we need $\mathsf{PSDVS.Verify}(\mathsf{PSDVS}.pp, vsk_{j_0}, m^*, \sigma^*_{j_0}) = 0$, and in order for $\mathsf{Verify}(pp, vsk_{j_1}, m^*, \sigma^*) = 1$ we need $\mathsf{PSDVS.Verify}(\mathsf{PSDVS}.pp, vsk_{j_1}, m^*, \sigma^*_{j_1}) = 1$. Then, if $d_\pi = 1$, either π^* violates the soundness of $\mathsf{NIZK\text{-}PoK}$, or one of the following must be true:

1. $\mathsf{PSDVS.RealSigVal}(\mathsf{PSDVS}.pp, spk, vpk_{j_0}, m^*, \sigma^*_{j_0}, \pi_{j_0}) = 1$. If this is the case, then $\mathcal{B}$ returns $(m^*, \sigma^*_{j_0}, \pi_{j_0})$ (the last of which is extractable from the knowledge soundness property of π^*) as a break of Provable Signing Soundness of the PSDVS, since we know that $\mathsf{PSDVS.Verify}(\mathsf{PSDVS}.pp, vsk_{j_0}, m^*, \sigma^*_{j_0}) = 0$.
2. $\mathsf{PSDVS.VerSigVal}(\mathsf{PSDVS}.pp, spk, vpk_{j_1}, m^*, \sigma^*_{j_1}, \pi_{j_1}) = 1$. If this is the case, then $\mathcal{B}$ returns $(m^*, \sigma^*_{j_1}, \pi_{j_1})$ (the last of which is extractable from the knowledge soundness property of π^*) as a break of VerSigSim Soundness, since the adversary was not given the secret key corresponding to vpk.
3. $\mathsf{PSDVS.PubSigVal}(\mathsf{PSDVS}.pp, spk, vpk_{j_1}, m^*, \sigma^*_{j_1}, \pi_{j_1}) = 1$. If this is the case, then $\mathcal{B}$ returns $(m^*, \sigma^*_{j_1}, \pi_{j_1})$ (the last of which is extractable from the knowledge soundness property of π^*) as a break of PubSigSim Soundness, since we know that $\mathsf{PSDVS.Verify}(\mathsf{PSDVS}.pp, vsk_{j_1}, m^*, \sigma^*_{j_1}) = 1$.

Claim. Construction 1 is existentially unforgeable, as per Definition 3.

This holds since any forgery of Construction 1 either includes a forgery of the underlying PSDVS, or includes a fresh proof of knowledge on a statement for which the adversary does not have a witness, which is impossible by the knowledge soundness property of our NIZK proof of knowledge.

Claim. Construction 1 is off-the-record, as per Definition 4.

The simulation algorithm Sim, described above, is defined to use public and verifier simulation to produce the individual PSDVS signatures, and to prove that all PSDVS signatures are simulations instead of proving that they all verify.

Below we describe a sequence of games; in Game 0, it is impossible for the adversary to distinguish $b = 0$ from $b = 1$, since its view in the two cases are identically distributed. In each subsequent game, the advantage of the adversary is at most negligibly greater than in the previous one; in the final game, the adversary will find itself playing exactly the game described in Definition 4.

Game 0**:** This gives $\mathcal{A}$ a real signature no matter what the value of b is.
$\mathcal{A}$ can have no advantage in this game.

Game 1**:** This game is the same as the previous game, except that if $b = 1$, the $\mathsf{NIZK\text{-}PoK}$ is simulated (and thus does not require the witnesses $\{\pi_j\}_{j\in\mathcal{D}^*}$).
If $\mathcal{A}$ distinguishes $b = 0$ from $b = 1$ with non-negligibly greater advantage than in the previous game, $\mathcal{B}$ can use $\mathcal{A}$ to break the zero-knowledge property of the $\mathsf{NIZK\text{-}PoK}$.

Game $2.j$ **for** $j \in \mathcal{D}^* \cap \mathcal{C}^*$**:** This game is the same as the previous game, except that if $b = 1$, R_j's portion of the signature is replaced with a verifier simulation; that is, $(\sigma_j, \pi_j) \leftarrow \mathsf{VerSigSim}(pp, spk_{i^*}, vpk_j, vsk_j, m^*)$.
If $\mathcal{A}$ distinguishes $b = 0$ from $b = 1$ with non-negligibly greater advantage than in the previous game, $\mathcal{B}$ can use $\mathcal{A}$ to break the VerSigSim indistinguishability property.
Note that the statement the $\mathsf{NIZK\text{-}PoK}$ is proving no longer holds; however, the $\mathsf{NIZK\text{-}PoK}$ simulator must still produce a simulated $\mathsf{NIZK\text{-}PoK}$ that is indistinguishable from a real one, since otherwise the $\mathsf{NIZK\text{-}PoK}$ simulator would be useable to distinguish a signature produced by $\mathsf{VerSigSim}$ from a signature produced by Sign.

Game $3.j$ **for** $j \in \mathcal{D}^* \backslash \mathcal{C}^*$**:** This game is the same as the previous game, except that if $b = 1$, R_j's portion of the signature is replaced with a public simulation; that is, $(\sigma_j, \pi_j) \leftarrow \mathsf{PubSigSim}(pp, spk_{i^*}, vpk_j, m^*)$.
If $\mathcal{A}$ distinguishes $b = 0$ from $b = 1$ with non-negligibly greater advantage than in the previous game, $\mathcal{B}$ can use $\mathcal{A}$ to break the PubSigSim indistinguishability property.

Game 4**:** This game is the same as the previous game, except that if $b = 1$, we replace the simulated π with a real proof; since all the individual signatures are now simulated, such a valid proof can be computed again. Note that now, if $b = 1$, we are executing exactly the simulation procedure Sim described above, and thus this is exactly the game described in Definition 4.
If $\mathcal{A}$ distinguishes $b = 0$ from $b = 1$ with non-negligibly greater advantage than in the previous game, $\mathcal{B}$ can use $\mathcal{A}$ to break the zero-knowledge property of the $\mathsf{NIZK\text{-}PoK}$.

3.3 Standard Primitive-Based PSDVS Construction

We can build a PSDVS from a special message authentication code (MAC) which looks uniformly random without knowledge of the secret MAC key — such a MAC can be built from any pseudorandom function. A signature on a message m will be a MAC on (m, t), where t is some random tag. Proving that the signature is real simply involves proving knowledge of a MAC key that is consistent with the MAC and some global public commitment to the MAC key. A public proof that the signature is simulated and does not verify would involve proving that the MAC was pseudorandomly generated. A verifier's proof that the signature is simulated would involve proving that the tag was generated in a way that only the verifier could use (e.g. from a PRF to which only the verifier knows the key).

Of course, this is not ideal, since MACs require knowledge of a shared key; in order to use MACs, we would need to set up shared keys between every possible pair of signer and verifier. However, we can get around this using non-interactive key exchange (NIKE). Each signer and verifier publishes a public key, and any pair of them can agree on a shared secret key by simply using their own secret key and the other's public key.

Construction 2 describes this construction in more detail.

Construction 2 *Let:*

- $\mathsf{COMM} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ *be a commitment scheme,*
- $\mathsf{PRF} = (\mathsf{KeyGen}, \mathsf{Compute})$ *be a length-preserving pseudorandom function,*
- PRG *be a length-doubling pseudorandom generator,*
- $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *be a non-interactive zero knowledge proof system, and*
- $\mathsf{NIKE} = (\mathsf{KeyGen}, \mathsf{KeyExtract}, \mathsf{KeyMatch})$ *be a non-interactive key exchange protocol.* $\mathsf{KeyMatch}$ *is an additional algorithm that checks if a public key and a secret key match.* $\mathsf{KeyMatch}$ *is not typically defined as a part of a* NIKE *scheme; however, such an algorithm always exists.*

We consider the public parameters for the underlying primitives $\mathsf{COMM}, \mathsf{PRF}, \mathsf{PRG}, \mathsf{NIKE}$ *together with the three common reference strings* (crs_1, crs_2, crs_3) *corresponding to the relations* $\mathcal{R}_1, \tilde{\mathcal{R}}_2, \tilde{\mathcal{R}}_3$ [12] *necessary to compute* NIZK *proofs as part of the public parameters of the PSDVS. Note that relations denoted* $\tilde{\mathcal{R}}$ *refer to statements of fake-ness, whereas relations denoted* $\mathcal{R}$ *refer to statements of real-ness.*

$\mathsf{Setup}(1^\kappa)$**:**

1. $crs_i \leftarrow \mathsf{NIZK.Setup}(1^\kappa, \mathcal{R}_i), i = 1, 2, 3.$
2. $ck \leftarrow \mathsf{COMM.Setup}(1^\kappa).$

Output $(\{crs_1, crs_2, crs_3\}, ck)$ as the public parameters pp.

$\mathsf{SignKeyGen}(pp)$**:**

1. $(\mathsf{NIKE}.pk_{\mathsf{S}}, \mathsf{NIKE}.sk_{\mathsf{S}}) \leftarrow \mathsf{NIKE.KeyGen}(1^\kappa).$
2. $ssk = \mathsf{NIKE}.sk_{\mathsf{S}}.$

[12] The three relations will be defined later in the protocol description.

3. $spk = \mathsf{NIKE}.pk_{\mathsf{S}}$.

Output ssk as the signer's secret key and spk as the signer's public key.

$\mathsf{VerKeyGen}(pp)$**:**

1. $(\mathsf{NIKE}.pk_{\mathsf{R}}, \mathsf{NIKE}.sk_{\mathsf{R}}) \leftarrow \mathsf{NIKE}.\mathsf{KeyGen}(1^{\kappa})$.
2. $k_{\mathsf{R}} \leftarrow \mathsf{PRF}.\mathsf{KeyGen}(1^{\kappa})$. (Informally, this key will be used by the verifier to simulate signatures using $\mathsf{VerSigSim}$.)
3. Choose randomness (i.e. decommitment value) r_{R} at random.
4. $c_{\mathsf{R}} = \mathsf{COMM}.\mathsf{Commit}(ck, k_{\mathsf{R}}; r_{\mathsf{R}})$. (Informally, this commitment will be used by the verifier to support its proofs of fake-ness.)
5. $vsk = (\mathsf{NIKE}.sk_{\mathsf{R}}, k_{\mathsf{R}}, r_{\mathsf{R}})$.
6. $vpk = (\mathsf{NIKE}.pk_{\mathsf{R}}, c_{\mathsf{R}})$.

Output vsk as the verifier's secret key and vpk as the verifier's public key.

$\mathsf{Sign}(pp, ssk = \mathsf{NIKE}.sk_{\mathsf{S}}, vpk = (\mathsf{NIKE}.pk_{\mathsf{R}}, c_{\mathsf{R}}), m)$**:**

The signer computes a shared key with the designated verifier and proceeds to sign the message m:

1. $k_{shared} = \mathsf{NIKE}.\mathsf{KeyExtract}(\mathsf{NIKE}.sk_{\mathsf{S}}, \mathsf{NIKE}.pk_{\mathsf{R}})$. (Informally, this key will be used as a MAC key.)
2. Choose t at random.
3. $\sigma = (\sigma_1, \sigma_2) \leftarrow (t, \mathsf{PRF}_{k_{shared}}((m, t)))$.
4. $\pi \leftarrow \mathsf{NIZK}.\mathsf{Prove}(crs_1, u, w)$ where $u = ((\sigma_1, \sigma_2), \mathsf{NIKE}.pk_{\mathsf{S}}, \mathsf{NIKE}.pk_{\mathsf{R}}, m)$ and $w = (\mathsf{NIKE}.sk_{\mathsf{S}}, k_{shared}))$

   We define the relation $\mathcal{R}_1$ indexed by NIKE public parameters and PRF for a statement u and witness w:

$$\begin{aligned}\mathcal{R}_1 = \{(u = (\sigma_1, \sigma_2, \mathsf{NIKE}.pk_{\mathsf{S}}, \mathsf{NIKE}.pk_{\mathsf{R}}, m), w = (\mathsf{NIKE}.sk_{\mathsf{S}}, k_{shared})) :\\ \mathsf{KeyMatch}(\mathsf{NIKE}.pk_{\mathsf{S}}, \mathsf{NIKE}.sk_{\mathsf{S}}) = 1\\ \wedge\ k_{shared} = \mathsf{NIKE}.\mathsf{KeyExtract}(\mathsf{NIKE}.sk_{\mathsf{S}}, \mathsf{NIKE}.pk_{\mathsf{R}})\\ \wedge\ \sigma_2 = \mathsf{PRF}_{k_{shared}}((m, \sigma_1))\}\end{aligned}$$

Output σ as the signature, and π as the proof of real-ness.

$\mathsf{Verify}(pp, spk = \mathsf{NIKE}.pk_{\mathsf{S}}, vsk = (\mathsf{NIKE}.sk_{\mathsf{R}}, k_{\mathsf{R}}, r_{\mathsf{R}}), m, \sigma = (\sigma_1, \sigma_2))$**:**

1. $k_{shared} = \mathsf{NIKE}.\mathsf{KeyExtract}(\mathsf{NIKE}.sk_{\mathsf{R}}, \mathsf{NIKE}.pk_{\mathsf{S}})$. (Informally, this key will be used as a MAC key.)
2. If $\mathsf{PRF}_{k_{shared}}((m, \sigma_1)) = \sigma_2$, set $d = 1$. Otherwise, set $d = 0$.

Output d as the verification decision.

$\mathsf{RealSigVal}(pp, spk, vpk, m, \sigma, \pi)$**:**

Output $d \leftarrow \mathsf{NIZK}.\mathsf{Verify}(crs_1, \sigma, \pi)$ as the validation decision.

$\mathsf{PubSigSim}(pp, m)$**:**

1. Choose a PRG seed seed.
2. Choose σ_1 and σ_2 pseudorandomly by running PRG on seed.
3. $\sigma \leftarrow (\sigma_1, \sigma_2)$.
4. Let $\pi \leftarrow \mathsf{NIZK}.\mathsf{Prove}(crs_2, u = \sigma, w = \mathsf{seed})$.

   We define the relation $\tilde{\mathcal{R}}_2$ indexed by the PRG for a statement $u = (\sigma = (\sigma_1, \sigma_2))$ and the witnesses $w = \mathsf{seed}$:

$$\tilde{\mathcal{R}}_2 = \{(u = \sigma; w = \mathsf{seed}) : u = \mathsf{PRG}(w)\} \tag{2}$$

Output σ as the simulated signature, and π as the proof of fake-ness.

$\mathsf{PubSigVal}(pp, spk, vpk, m, \sigma = (\sigma_1, \sigma_2), \pi)$**:**

Output $d \leftarrow \mathsf{NIZK.Verify}(crs_2, \sigma, \pi)$ as the validation decision.

$\mathsf{VerSigSim}(pp, spk = \mathsf{NIKE}.pk_{\mathsf{S}}, vpk = (\mathsf{NIKE}.pk_{\mathsf{R}}, c_{\mathsf{R}}), vsk = (\mathsf{NIKE}.sk_{\mathsf{R}}, k_{\mathsf{R}}, r_{\mathsf{R}}), m)$**:**

The verifier can fake a signature using its PRF key k_{R}.

1. $k_{shared} = \mathsf{NIKE.KeyExtract}(\mathsf{NIKE}.sk_{\mathsf{R}}, \mathsf{NIKE}.pk_{\mathsf{S}})$.
2. Choose r at random.
3. $t \leftarrow \mathsf{PRF}_{k_{\mathsf{R}}}(r)$.
4. $\sigma \leftarrow (t, \mathsf{PRF}_{k_{shared}}((m, t)))$.
5. Let $\pi \leftarrow \mathsf{NIZK.Prove}(crs_3, u = (c_{\mathsf{R}}, \sigma_1), w = (k_{\mathsf{R}}, r_{\mathsf{R}}, r))$.
   We define the relation $\tilde{\mathcal{R}}_3$ indexed by the NIKE public parameters and PRF for statements u and witnesses w:

$$\tilde{\mathcal{R}}_3 = \{(u = (c_{\mathsf{R}}, \sigma_1), w = (k_{\mathsf{R}}, r_{\mathsf{R}}, r)) : \\ k_{\mathsf{R}} = \mathsf{COMM.Open}(c_{\mathsf{R}}, r_{\mathsf{R}}) \wedge \ \sigma_1 = \mathsf{PRF}_{k_{\mathsf{R}}}(r)\}$$

Output σ as the simulated signature and π as the proof of fake-ness.

$\mathsf{VerSigVal}(pp, spk, vpk, m, \sigma, \pi)$**:**

Output $d \leftarrow \mathsf{NIZK.Verify}(crs_3, (c_{\mathsf{R}}, \sigma_1), \pi)$ as the validation decision.

Theorem 3. *If the schemes* COMM, PRF, PRG, NIZK, NIKE *are secure, then Construction 2 is a correct and secure PSDVS scheme as per Definition 7.*

Proof. Correctness. It is straightforward to verify that any honestly generated signature will pass the verification test.

Existential Unforgeability. We can reduce the unforgeability of the PVDVS to the pseudorandomness of the underlying PRF. Suppose there exists a forger $\mathcal{A}$ having non-negligible advantage in winning the existential unforgeability game 3 for a single signer and a single verifier (see Definition 3). We can use this forger to build a distinguisher $\mathcal{B}$ that is able to break the pseudo-randomness property of the PRF. $\mathcal{B}$ runs $\mathcal{A}$ and simulate the signing queries m of $\mathcal{A}$ by picking a random tag t and forwarding the query (m, t) to its evaluation oracle that outputs either the evaluation $\mathsf{PRF}(m, t)$, either a truly random value. In the first case, the forger $\mathcal{A}$ has the same view as in the game 3. Given that $\mathcal{A}$ is able to forge a signature on a fresh message m^* in the first case, but not in the second, then this implies that the adversary $\mathcal{B}$ can distinguish between the two, breaking the pseudo-randomness of the PRF.

PubSigSim Indistinguishability (Definition 8). By the security property of the PRG (indistinguishability from real randomness), the advantage of an adversary $\mathcal{A}$ in the game $\mathsf{Game}^{PubSigSim\text{-}Sound}_{\mathsf{PVDVS},\mathcal{A}}(\kappa)$ is bounded by the advantage of an adversary $\mathcal{B}$ in distinguishing between PRG and a truly random generator. Note that we can apply the property only for the first half of the signature, adversary $\mathcal{A}$ should not be able to distinguish between a random t and σ_1 generated as $(\sigma_1, \sigma_2) \leftarrow \mathsf{PRG}(\mathsf{seed})$.

PubSigSim Correctness (Definition 9). By the completeness of the NIZK scheme, any proof π generated honestly by running the PubSigSim, i.e $\pi \leftarrow$

$\mathsf{NIZK.Prove}(crs_2, u = \sigma, w)$ will be validated by $\mathsf{PubSigVal}(pp, spk, vpk, m, \sigma, \pi)$ that runs $\mathsf{NIZK.Verify}(crs_2, u = \sigma, \pi)$ algorithm.

PubSigSim Soundness (Definition 10). Suppose there is an adversary $\mathcal{A}$ that wins the game of *PubSigSim-Sound* game with non negligible probability. Then, from an output (m^*, σ^*, π^*) of $\mathcal{A}(ssk, spk, vpk)$ we have from the soundness of the NIZK, since PubSigVal outputs 1, that there is a value seed such that $\sigma^* = (\sigma_1^*, \sigma_2^*) \leftarrow \mathsf{PRG}(\mathsf{seed})$ and also $\sigma^* = (\sigma_1^*, \mathsf{PRF}(m^*, \sigma_1^*))$ from the verification check of the signature $\mathsf{Verify}(pp, spk, vsk, m^*, \sigma^*) = 1$. This implies there exists a collision $\mathsf{PRG}(\mathsf{seed}) = (\sigma_1^*, \mathsf{PRF}(m^*, \sigma_1^*))$ breaking the pseudorandomness of the underlying primitives PRG and PRF. In the first case, given a signature $\sigma^* = (\sigma_1^*, \mathsf{PRF}(m^*, \sigma_1^*))$ that verifies, $\mathcal{A}$ should not be able to find a preimage seed^* for σ^* with respect to the PRG with advantage significantly better than for a truly random function, without breaking the pseudorandomness of the PRG. Otherwise, from computing an output of the pseudorandom generator $\mathsf{PRG}(\mathsf{seed}) = (\sigma_1^*, \sigma_2^*)$, $\mathcal{A}$ should not be able to find a (fixed prefix) preimage (t, m^*) of PRF such that $t = \sigma_1^*$. This is indeed infeasable without breaking the pseudorandom property of the PRF.

VerSigSim Indistinguishability (Definition 11). This follows from the pseudorandomness of our PRF. Remark that both a real signature and a verifier-simulated signature pass the verification test, the only difference is in how the tag t is generated, truly random, or as $t \leftarrow \mathsf{PRF}_{k_\mathsf{R}}(r)$.

VerSigSim Correctness (Definition 12). As in the case of PubSigSim correctness, this holds by considering the completeness of the NIZK scheme, since an honnest proof generated by VerSigSim, will be validated by VerSigVal that simply runs NIZK.Verify algorithm.

VerSigSim Soundness (Definition 13.) This follows from the properties of the underlying COMM and NIZK schemes. Consider an adversary $\mathcal{A}$ that is able to win the game of VerSigSim-Sound, meaning that it produces a couple (σ, π) validated by VerSigVal. Then, if the NIZK scheme is assumed to be sound, the following should hold: $\mathcal{A}$ is able to compute an opening k_R of c_R, breaking the hidding of the commitment scheme COMM or $\mathcal{A}$ is able to find a preimage r for $\mathsf{PRF}_{k_\mathsf{R}}$, i.e $\sigma_1 = \mathsf{PRF}_{k_\mathsf{R}}(r)$ which breaks the pseudorandomness of the PRF.

Provable Signing Correctness (Definition 14). As in the case of PubSigSim and VerSigSim, this follows by definition of RealSigVal and the completeness of the NIZK.

Provable Signing Soundness (Definition 15). This holds if we assume soundness of the NIZK proof togheter with NIKE security properties and pseudorandomness of the PRF. Assuming that the NIZK is sound, then an adversary winning the game RealSigVal, is able either to break the soundness of the NIKE scheme or to find a preimage of the PRF. Finding a preimage is infeasible, given the pseudorandomness property of the PRF.

3.4 DDH and Paillier-Based PSDVS Construction

The goal of this section is to construct a PSDVS scheme based on DDH and the security of Paillier encryption. The idea in the PSDVS construction is that the authenticator for a message m will be $\mathsf{H}(m, t)^k$ in a group G where t is a nonce, k is a key known to both parties and H is a hash function modeled as a random oracle. The construction requires that certain properties of the key can be proved in zero-knowledge, and we can do this efficiently using standard Σ-protocols because the key is in the exponent. However, naive use of this idea would mean that a sender needs to store a key for every verifier he talks to, and the set-up must generate correlated secret keys for the parties. To get around this, we will instead let the sender choose k on the fly and send it to the verifier, encrypted using a new variant of Paillier encryption. In the following subsection we describe and prove this new encryption scheme, and then we specify the actual PSDVS construction. Paillier-style encryption comes in handy since its algebraic properties are useful in making our zero knowledge proofs efficient.

Paillier-based Authenticated and Verifiable Encryption An authenticated and verifiable encryption scheme (AVPKE) involves a sender S and a receiver R. Such a scheme comes with the following polynomial time algorithms:

$\mathsf{Setup}(1^\kappa) \to pp$**:** A probabilistic algorithm for setup which outputs public parameters.
$\mathsf{KeyGen}_\mathsf{S}(pp) \to (sk_\mathsf{S}, pk_\mathsf{S})$**:** A probabilistic sender key generation algorithm.
$\mathsf{KeyGen}_\mathsf{R}(pp) \to (sk_\mathsf{R}, pk_\mathsf{R})$**:** A probabilistic receiver key generation algorithm.
$\mathsf{Enc}_{pp,sk_\mathsf{S},pk_\mathsf{R}}(k) \to c$**:** A probabilistic encryption algorithm for message k.
$\mathsf{Dec}_{pp,sk_\mathsf{R},pk_\mathsf{S}}(c) \to \{k, \perp\}$**:** A decryption algorithm that outputs either reject or a message.

We require, of course, that $\mathsf{Dec}_{pp,sk_\mathsf{R},pk_\mathsf{S}}(\mathsf{Enc}_{pp,sk_\mathsf{S},pk_\mathsf{R}}(k)) = k$ for all messages k.

Intuitively, the idea is that given only the receiver public key pk_R and his own secret key sk_S, the sender S can encrypt a message k in such a way that on receiving the ciphertext, R can check that k comes from S, no third party knows k and finally, the encryption is verifiable in that it allows S to efficiently prove in zero-knowledge that k satisfies certain properties.

To help understand our concrete construction of an AVPKE scheme, we recall that standard Paillier encryption of a message k under the public key n is defined as $(n+1)^k v^n \bmod n^2$ where $v \in_\mathsf{R} \mathbb{Z}_n^*$. In [DJ03], it was suggested that first, v can be chosen as $\pm \hat{g}^s$ for a random s and a $\hat{g}$ of large order modulo n – or equivalently, a random number of Jacobi symbol 1 mod n. This is not a security problem, as the Jacobi symbol of v can be efficiently computed from the ciphertext anyway. Further, they suggested that v can be chosen similarly as in El-Gamal encryption, if the sender sends along a random power of $\hat{g}$. They also showed that the resulting encryption scheme is still CPA secure under the same assumption. In this way all users can share the same modulus, which comes in very handy in our setting.

We add an authentication mechanism to this encryption scheme and get the following AVPKE scheme.

Construction 3 *Let:*

– Ggen *be a* Group Generator*, a probabilistic polynomial time algorithm which on input* 1^κ *outputs the description of a cyclic group* G *and a generator* g*,* *such that the order of G is a random κ-bit RSA modulus n, which is the product of so-called safe primes. (That is, $n = pq$ where $p = 2p' + 1, q = 2q' + 1$ and p', q' are also primes.) Finally, we need the algorithm to output an element $\hat{g} \in \mathbb{Z}_n^*$ of order $p'q'$.*

– $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *be a simulation-sound non-interactive zero knowledge proof system. In this section, we will use Σ-protocols made non-interactive using the Fiat-Shamir heuristic, so in this case* Setup *is empty and there is no common reference string.*

Ggen *can be constructed using standard techniques. For instance, first generate n using standard techniques, then repeatedly choose a small random number r until $P = 2rn + 1$ is a prime. Let g' be a generator of $\mathbb{Z}_P^*$. Then let G be the subgroup of $\mathbb{Z}_P^*$ generated by $g = g'^{2r} \bmod P$.*[13] *Finally, to construct the element $\hat{g}$, let $u \in_{\mathsf{R}} \mathbb{Z}_n$ and set $\hat{g} = u^2 \bmod n$. Indeed, this is a random square, and since the subgroup of squares modulo n has only large prime factors in its order (p' and q'), a random element is a generator with overwhelming probability*[14].

$\mathsf{Setup}(1^\kappa)$**:** *Run* Ggen *to generate a modulus n and $\hat{g} \in \mathbb{Z}_n^*$ as explained above. Output $pp = (n, \hat{g})$.*

$\mathsf{KeyGen}_\mathsf{S}(pp)$**:** *Pick $sk_\mathsf{S} \in_\mathsf{R} \mathbb{Z}_n$, and set $pk_\mathsf{S} = \hat{g}^{sk_\mathsf{S}}$. Output $(sk_\mathsf{S}, pk_\mathsf{S})$.*

$\mathsf{KeyGen}_\mathsf{R}(pp)$**:** *Pick $\alpha_1, \alpha_2 \in_\mathsf{R} \mathbb{Z}_n$, set $sk_\mathsf{R} = (\alpha_1, \alpha_2)$, and set $pk_\mathsf{R} = (\beta_1, \beta_2) = (\hat{g}^{\alpha_1}, \hat{g}^{\alpha_2})$.*

The public key values are statistically indistinguishable from random elements in the group generated by $\hat{g}$ since n is a sufficiently good "approximation" to the order $p'q'$ of $\hat{g}$.

$\mathsf{Enc}_{pp,sk_\mathsf{S},pk_\mathsf{R}}(k; r, b_1, b_2)$**:**

1. *The randomness should have been picked as follows: $r \in_\mathsf{R} \mathbb{Z}_n$ and $b_1, b_2 \in_\mathsf{R} \{0, 1\}$.*
2. *Set $c_1 = (-1)^{b_1} \hat{g}^r \bmod n$.*
3. *Set $c_2 = (n + 1)^k ((-1)^{b_2} \beta_1^{sk_\mathsf{S}} \beta_2^r \bmod n)^n \bmod n^2$.*
4. *Let π_{valid} be a non-interactive zero-knowledge proof of knowledge wherein given public data $(n, \hat{g}, (c_1, c_2))$, the prover shows knowledge of a witness $w = (k, r, b_1, v)$ such that $c_1 = (-1)^{b_1} \hat{g}^r$ and $c_2 = (n + 1)^k v^n \bmod n^2$. An honest prover can use $v = (-1)^{b_2} \beta_1^{sk_\mathsf{S}} \beta_2^r \bmod n$. The factor $(-1)^{b_1}$ is only in the ciphertext for technical reasons: it allows π_{valid} to be efficient.*

Output $c = (c_1, c_2, \pi_{valid})$.

$\mathsf{Dec}_{pp,sk_\mathsf{R}=(\alpha_1,\alpha_2),pk_\mathsf{S}}(c = (c_1, c_2, \pi_{valid}))$**:**

1. *Check that c_1, c_2 have Jacobi symbol 1 modulo n, and check π_{valid}. Output reject if either check fails.*

[13] The group G will be more prominently used in the construction of the PSDVS scheme.

[14] This set-up need to keep the factorization of n secret. Hence, to avoid relying on a trusted party, the parties can use an interactive protocol to generate n securely, there are several quite efficient examples in the literature.

2. *Let $u = pk_{\mathsf{S}}^{\alpha_1} c_1^{\alpha_2} \bmod n$ and check that $(c_2 u^{-n})^n \bmod n^2 = \pm 1$. $\mathbb{Z}_{n^2}^*$ contains a unique subgroup of order n, generated by $n+1$. So here we are verifying that – up to a sign difference – $c_2 u^{-n} \bmod n^2$ is in the subgroup generated by $n+1$. If the check fails, output reject.*
3. *Otherwise, compute k such that $(n+1)^k = \pm c_2 u^{-n} \bmod n^2$.*[15]

An AVPKE scheme should allow anyone to make "fake" ciphertexts that look indistinguishable from real encryptions, given only the system parameters. Furthermore, the receiver R should be able to use his own secret key sk_{R} and the public key pk_{S} of the sender to make ciphertexts with exactly the same distribution as real ones. This is indeed true for our scheme:

Fake Encryption: Let $r \in_{\mathsf{R}} \mathbb{Z}_n, b, b' \in_{\mathsf{R}} \{0,1\}$ and $v \in \mathbb{Z}_n^*$ be a random square. Then,

$$\mathsf{Enc}_{pp,fake}(k; r, b, b', v) = ((-1)^b \hat{g}^r \bmod n, (n+1)^k ((-1)^{b'} v)^n \bmod n^2), \pi_{valid}$$

where π_{valid} is constructed following the NIZK prover algorithm.

R's Equivalent Encryption:

$$\mathsf{Enc}_{pp,sk_{\mathsf{R}},pk_{\mathsf{S}}}(k; r, b_1, b_2) =$$
$$((-1)^{b_1} \hat{g}^r \bmod n,\ (n+1)^k (-1)^{b_2} (pk_{\mathsf{S}}^{\alpha_1} \hat{g}^{r\alpha_2} \bmod n)^n \bmod n^2),\ \pi_{valid}$$

where $r \in_{\mathsf{R}} \mathbb{Z}_n$, $b_1, b_2 \in_{\mathsf{R}} \{0,1\}$ and π_{valid} is constructed following the NIZK prover algorithm.

In the following, we will sometimes suppress the randomness from the notation and just write, e.g., $\mathsf{Enc}_{pp,sk_{\mathsf{S}},pk_{\mathsf{R}}}(k)$.

By simple inspection of the scheme it can be seen that:

Lemma 1. *For all k, $\mathsf{Dec}_{pp,sk_{\mathsf{R}},pk_{\mathsf{S}}}(\mathsf{Enc}_{pp,sk_{\mathsf{S}},pk_{\mathsf{R}}}(k)) = k$. Furthermore, encryption by S and by R returns the same ciphertexts: for all messages k and randomness r, b_1, b_2, we have $\mathsf{Enc}_{pp,sk_{\mathsf{S}},pk_{\mathsf{R}}}(k; r, b_1, b_2) = \mathsf{Enc}_{pp,sk_{\mathsf{R}},pk_{\mathsf{S}}}(k; r, b_1, b_2)$.*

Lemma 2. *If DDH in $\langle \hat{g} \rangle$ is hard, then $(k, \mathsf{Enc}_{pp,sk_{\mathsf{S}},pk_{\mathsf{R}}}(k; r, b_1, b_2))$ is computationally indistinguishable from $(k, \mathsf{Enc}_{pp,fake}(k; r', b, b', v))$ for any fixed message k and randomness $r, b_1, b_2, r', b, b', v$, as long as the discrete log of β_2 to the base $\hat{g}$ is unknown.*

Proof. This follows immediately from that fact that $(\hat{g}^r, \beta_2^r)$ is indistinguishable from $(\hat{g}^{r'}, v)$ by assumption.

Definition 16. *Consider the following experiment for an AVPKE scheme and a probabilistic polynomial time adversary $\mathcal{A}$: Run the set-up and key generation and run $A^{\mathcal{O}_E, \mathcal{O}_D}(pp, pk_{\mathsf{R}}, pk_{\mathsf{S}})$. Here, $\mathcal{O}_E$ takes a message k as input and returns $\mathsf{Enc}_{pp,sk_{\mathsf{S}},pk_{\mathsf{R}}}(k)$, while $\mathcal{O}_D$ takes a ciphertext and returns the result of decrypting it under $pk_{\mathsf{S}}, sk_{\mathsf{R}}$ (which will be either reject or a message). $\mathcal{A}$ wins if it makes $\mathcal{O}_D$ accept a ciphertext that was not obtained from $\mathcal{O}_E$. The scheme is* authentic *if any PPT $\mathcal{A}$ wins with negligible probability.*

[15] k can be computed using the standard "discrete log" algorithm from Paillier decryption.

Lemma 3. *If the DDH problem in $\langle \hat{g} \rangle$ is hard, the AVPKE scheme defined above is authentic.*

Proof. Assume for contradiction that we have adversary $\mathcal{A}$ who wins the authenticity game with non-negligible probability. We now stepwise transform the game into an algorithm that will solve the computational Diffie-Hellman problem in $\langle \hat{g} \rangle$, which will certainly contradict the assumption. Let Game 0 be the original authenticity game. In Game 1, we replace $\mathcal{O}_E$ by $\mathcal{O}_E^{fake}$ which on input k from $\mathcal{A}$ returns $\mathsf{Enc}_{pp,fake}(k)$. At the same time, we replace $\mathcal{O}_D$ by an alternative version $\mathcal{O}_D^{alt}$ that does not use the α_2 part of sk_{R}, namely instead of computing $u = pk_{\mathsf{S}}^{\alpha_1} c_1^{\alpha_2}$ it extracts r from the proof in the ciphertext and computes instead $u = pk_{\mathsf{S}}^{\alpha_1} \beta_2^r$ which is the same value (up to a sign) and hence leads to an equivalent decryption. In addition, if it gets an encryption $\mathsf{Enc}_{pp,fake}(k)$ produced by the fake encryption oracle, it always returns k. Since now the discrete log of β_2 base $\hat{g}$ is not used, we can use our assumption and Lemma 2 to conclude that $\mathcal{A}$'s winning probability in the new game is essentially the same. In Game 2, we guess the index j of the first call to $\mathcal{O}_D$ where $\mathcal{A}$ gets an accept for a forged ciphertext, which we can do with an inverse polynomial probability, since $\mathcal{A}$ is poly-time. If $\mathcal{A}$ does not win the game at the j'th call, we declare that it loses. Clearly $\mathcal{A}$ still wins Game 2 with non-negligible probability. In Game 3, we replace $\mathcal{O}_D^{alt}$ by $\mathcal{O}_D^{sim}$ which, up to (but not including) call j, responds to a ciphertext that was output by $\mathcal{O}_E^{fake}$ with the message that was encrypted and rejects anything else. Observe that in the event where $\mathcal{A}$ wins Game 2, $\mathcal{O}_D^{sim}$ simulates $\mathcal{O}_D^{alt}$ perfectly, so $\mathcal{A}$ wins Game 3 with the same probability as Game 2.

Now, observe that we can execute Game 3 up to step j and get $\mathcal{A}$'s ciphertext for the j'th call without access to the secret keys of S and R. We can therefore take group elements $\hat{g}, pk_{\mathsf{S}} = \hat{g}^{sk_{\mathsf{S}}}, \beta_1 = \hat{g}^{\alpha_1}$ as input to the CDH problem (where the goal is to find $\hat{g}^{sk_{\mathsf{S}}\alpha_1}$), and execute Game 3 with $\mathcal{A}$ on this input. Assume $\mathcal{A}$ wins, and let c^* be the ciphertext submitted in the j'th call to $\mathcal{O}_D^{sim}$. Recall that c^* is of form (c_1, c_2, π_{valid}). By simulation soundness of the NIZK used, we can extract the witness claimed in the proof, and so we get r and k, v such that $c_1 = \pm\hat{g}^r$ and $c_2 = (n+1)^k v^n \bmod n^2$.

We can assume that R's decryption algorithm would accept c^*, this implies that v must have Jacobi symbol 1. Also, since the algorithm computes $u = pk_{\mathsf{S}}^{\alpha_1} c_1^{\alpha_2} \bmod n$ and checks that $(c_2 u^{-n})^n \bmod n^2 = \pm 1$, we can assume this check is satisfied. Inserting the expression we have for c_2 we get

$$\pm 1 = (c_2 u^{-n})^n \bmod n^2 = ((n+1)^k v^n u^{-n})^n \bmod n^2 = ((vu^{-1} \bmod n)^n)^n \bmod n^2$$

Here, we have used the fact that $\mathbb{Z}_{n^2}^*$ is the direct product of a group G of order n and a group H of order $\phi(n)$ isomorphic to $\mathbb{Z}_n^*$ under the isomorphism $x \mapsto x^n \bmod n^2$. Since $(vu^{-1} \bmod n)^n \bmod n^2 \in H$ and raising to power n is a 1-1 mapping on H, it follows that $vu^{-1} \bmod n = \pm 1$. Inserting the expression for u, we get

$$\pm v = u = pk_{\mathsf{S}}^{\alpha_1} c_1^{\alpha_2} = pk_{\mathsf{S}}^{\alpha_1} \hat{g}^{r\alpha_2} = \hat{g}^{sk_{\mathsf{S}}\alpha_1} \beta_2^r$$

In conclusion, we can flip a coin and submit plus or minus $v\beta_2^{-r} \bmod n$ as a solution to the CDH problem and this will be correct with half the probability with which $\mathcal{A}$ wins Game 3.

We proceed to show that the AVPKE scheme hides the message encrypted even if adversary knows the secret key of the sender, and even if a decryption oracle is given. This is essentially standard CCA security.

Definition 17. *Consider the following experiment for an AVPKE scheme and a probabilistic polynomial time adversary $\mathcal{A}$: Run the set-up and key generation and run $\mathcal{A}^{\mathcal{O}_E}(pp, pk_{\mathsf{R}}, sk_{\mathsf{S}})$. Here, $\mathcal{O}_E$ takes two messages k_0, k_1 as input, selects a bit η at random and returns $c^* = \mathsf{Enc}_{pp,sk_{\mathsf{S}},pk_{\mathsf{R}}}(k_\eta)$. $\mathcal{O}_D$ takes a ciphertext and returns the result of decrypting it under $pk_{\mathsf{S}}, sk_{\mathsf{R}}$ (which will be either reject or a message). $\mathcal{A}$ may submit anything other than c^* to $\mathcal{O}_D$, and must output a bit η' at the end. It wins if $\eta' = \eta$. The scheme is* private *if any PPT $\mathcal{A}$ wins with negligible advantage over $\frac{1}{2}$.*

In the following we will use the assumption underlying the Paillier encryption scheme, sometimes known as the *composite degree residuosity assumption* (CDRA): a random element x in $\mathbb{Z}_{n^2}^*$ where $x \bmod n$ has Jacobi symbol 1 is computationally indistinguishable from $y^n \bmod n^2$ where $y \in \mathbb{Z}_n^*$ is random of Jacobi symbol 1[16].)

Lemma 4. *Assume that DDH in $\langle \hat{g} \rangle$ is hard and that CDRA holds. Then the AVPKE scheme satisfies Definition 17.*

Proof. Assume for contradiction that adversary $\mathcal{A}$ wins the game with probability non-negligibly larger than 1/2. Let Game 0 be the original game. We change the game stepwise to get new games that can be used to either solve DDH or break CDRA. In Game 1, we replace $\mathcal{O}_D$ by an alternative version $\mathcal{O}_D^{alt}$ that does not use the α_2 part of sk_{R}, namely instead of computing $u = pk_{\mathsf{S}}^{\alpha_1} c_1^{\alpha_2}$ it extracts r from the proof in the ciphertext and computes instead $u = pk_{\mathsf{S}}^{\alpha_1} \beta_2^r$ which is the same value (up to a sign) and hence leads to an equivalent decryption. $\mathcal{A}$ wins Game 1 with exactly the same probability. In Game 2, we replace $\mathcal{O}_E$ by $\mathcal{O}_E^{sim}$, defined as follows: take $a, b \in \langle \hat{g} \rangle$ as input, where $a = \hat{g}^r, b = \beta_2^r$. let $\eta \in_{\mathsf{R}} \{0,1\}, r \in_{\mathsf{R}} \mathbb{Z}_n$ and return $(\pm a, (n+1)^{k_\eta} (\pm \beta_1^{sk_{\mathsf{S}}} b)^n \bmod n^2, \pi_{valid})$, where π_{valid} is a simulated proof. Except for the simulation, this is exactly Game 1, so $\mathcal{A}$ wins with essentially the same probability – note that by simulation soundness, the witness extraction used by $\mathcal{O}_D^{alt}$ still works. In Game 3, we make a, b be random group elements in $\langle \hat{g} \rangle$. Now, $\mathcal{A}$'s winning probability remains essentially the same since otherwise, we could use the difference between Game 2 and 3 to solve DDH. Now note that in Game 3 $(\pm \beta_1^{sk_{\mathsf{S}}} b)^n \bmod n^2$ is in fact a uniformly random element of form $y^n \bmod n^2$, where the Jacobi symbol of $y \bmod n$ is 1. In Game 4, we replace this by x chosen uniformly in $\mathbb{Z}_{n^2}^*$ subject to $x \bmod n$

[16] The original CDRA assumption does not have the restriction to Jacobi symbol 1, but since the Jacobi symbol is easy to compute without the factors of n, the two versions are equivalent.

having Jacobi symbol 1. In Game 4, c^* has no information on η, so here $\mathcal{A}$ wins with probability 1/2. This means that we can use the difference between Game 3 and 4 to break CDRA, and we have a contradiction.

We say that an AVPKE scheme is *secure* if it is authentic, private, supports equivalent encryption by R and indistinguishable fake encryption.

The PSDVS Scheme We now return to the promised PSDVS scheme.

Construction 4 *Let:*

- Ggen *be a* Group Generator*, a probabilistic polynomial time algorithm which on input* 1^κ *outputs* $G, g, n, \hat{g}$ *exactly as in the previous AVPKE construction.*
- H *be a hash function which we model as a random oracle. We assume it maps onto the group* G*.*
- $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *be a simulation-sound non-interactive zero knowledge proof system. In this section, we will use* Σ*-protocols made non-interactive using the Fiat-Shamir heuristic, so in this case* Setup *is empty and there is no common reference string.*

$\mathsf{Setup}(1^\kappa)$**:** Let $(G, g, \hat{g}, n) \leftarrow \mathsf{Ggen}(1^\kappa)$ and let $h \in_{\mathsf{R}} G$. Set $pp = (G, g, \hat{g}, n, h)$. Return pp as the public parameters.

$\mathsf{SignKeyGen}(pp)$**:** Run key generation for the AVPKE scheme as defined above to get keys $ssk = sk_{\mathsf{S}}, spk = pk_{\mathsf{S}}$ for the signer S. Output ssk as the signer's secret key and spk as the signer's public key.

$\mathsf{VerKeyGen}(pp)$**:**

1. Run key generation for the AVPKE scheme as defined above to get keys $sk_{\mathsf{R}}, pk_{\mathsf{R}}$ for the verifier R. (These keys will be used to sign messages and verify signatures.)
2. Choose $k_{\mathsf{R}} \in_{\mathsf{R}} \mathbb{Z}_n$. (This key will be used by the verifier to simulate signatures using $\mathsf{VerSigSim}$.)
3. Choose $r_{\mathsf{R}} \in_{\mathsf{R}} \mathbb{Z}_n$ and let $c_{\mathsf{R}} = g^{k_{\mathsf{R}}} h^{r_{\mathsf{R}}}$. (This commitment will be used by the verifier to support its proofs of fake-ness.)
4. $vsk = (sk_{\mathsf{R}}, k_{\mathsf{R}}, r_{\mathsf{R}})$, $vpk = (pk_{\mathsf{R}}, c_{\mathsf{R}})$.

Output vsk as the verifier's secret key and vpk as the verifier's public key.

$\mathsf{Sign}(pp, ssk = sk_{\mathsf{S}}, pk_{\mathsf{R}}, m)$**:**

1. Choose $t \in_{\mathsf{R}} G, r \in_{\mathsf{R}} \mathbb{Z}_n, b_1, b_2 \in_{\mathsf{R}} \{0,1\}, s \in_{\mathsf{R}} \mathbb{Z}_n^*, k_s \in_{\mathsf{R}} \mathbb{Z}_n$.
2. Let $\sigma \leftarrow (t, \mathsf{H}(m,t)^{k_s}, \mathsf{Enc}_{pp, sk_{\mathsf{S}}, pk_{\mathsf{R}}}(k_s; r, b_1, b_2))$.
3. $\pi \leftarrow \mathsf{NIZK.Prove}(u = (\sigma = (\sigma_1, \sigma_2, \sigma_3), pk_V, pk_S, m), w = (sk_S, k_s, r, b_1, b_2))$ be a zero knowledge proof of knowledge of witness w such that:

$$\sigma_2 = \mathsf{H}(m, \sigma_1)^{k_s} \wedge \sigma_3 = \mathsf{Enc}_{pp, sk_{\mathsf{S}}, pk_{\mathsf{R}}}(k_s; r, b_1, b_2)$$

Output σ as the signature, and π as the proof of real-ness. Recall that the ciphertext by construction already contains a proof implying that the ciphertext contains a well defined plaintext. The role of the tag t in the signature is to let the verifier give a proof for his way to simulate a signature. This will become clear below.

$\mathsf{Verify}(pp, spk = pk_{\mathsf{S}}, vsk = (sk_{\mathsf{R}}, k_{\mathsf{R}}, r_{\mathsf{R}}), m, \sigma = (\sigma_1, \sigma_2, \sigma_3))$**:**

1. Decrypt σ_3 as $k_s = \mathsf{Dec}_{pp,sk_{\mathsf{R}},pk_{\mathsf{S}}}(\sigma_3)$. If this fails, set $d = 0$ and abort.
2. If $\sigma_2 = \mathsf{H}(m, \sigma_1)^{k_s}$, set $d = 1$. Otherwise, set $d = 0$.

Output d as the verification decision.

Informally, forging a signature is hard since the verifier rejects forged ciphertexts and valid ones hide the k_s value inside, by properties of the AVPKE scheme. The only other option is to reuse an existing k_s in a new signature, which is hard if CDH is hard in G. Namely, you have to raise a random group element (output by the random oracle) to the secret exponent k_s.

$\mathsf{PubSigSim}(pp, m)$**:**

1. Choose $k, k' \in_{\mathsf{R}} \mathbb{Z}_n$, such that $k \neq k'$.
2. Choose $t \in_{\mathsf{R}} G$, $r \in_{\mathsf{R}} \mathbb{Z}_n, b, b' \in_{\mathsf{R}} \{0, 1\}$, $v \in_{\mathsf{R}} \mathbb{Z}_n$, such that v has Jacobi symbol 1.
3. $\sigma \leftarrow (t, \mathsf{H}(m, t)^k, \mathsf{Enc}_{pp,fake}(k'; r, b, b', v))$.
4. Let $\pi \leftarrow \mathsf{NIZK.Prove}(u = \sigma = (\sigma_1, \sigma_2, \sigma_3), w = (k, k'))$ be a zero-knowledge proof of knowledge such that:

$$\sigma_2 = \mathsf{H}(m, \sigma_1)^k \wedge \sigma_3 = \mathsf{Enc}_{pp,fake}(k'; \cdot, \cdot, \cdot, \cdot) \wedge k \neq k'.$$

Output σ as the simulated signature, and π as the proof of fake-ness. The notation $\mathsf{Enc}_{pp,fake}(k'; \cdot, \cdot, \cdot, \cdot)$ means that the proof only has to establish that the plaintext inside the encryption is some value k' different from k.

Informally, a simulated signature looks like a real one since fake encryptions are indistinguishable from real ones and by privacy of the encryption scheme, one cannot decide efficiently if $k = k'$ or not. Clearly, a fake signature as defined is always rejected by the verifier.

$\mathsf{VerSigSim}(pp, spk = pk_{\mathsf{S}}, vpk = (pk_{\mathsf{R}}, c_{\mathsf{R}}), vsk = (sk_{\mathsf{R}}, k_{\mathsf{R}}, r_{\mathsf{R}}), m)$**:**

1. Choose $r_t \in_{\mathsf{R}} \mathbb{Z}_n$, $t = g^{k_{\mathsf{R}}} h^{r_t}$, $k_s \in_{\mathsf{R}} \mathbb{Z}_n$, $b_1, b_2 \in_{\mathsf{R}} \{0, 1\}$ and $v \in_{\mathsf{R}} \mathbb{Z}_n^*$ of Jacobi symbol 1.
2. $\sigma \leftarrow (t, \mathsf{H}(m, t)^{k_s}, \mathsf{Enc}_{pp,sk_{\mathsf{R}},pk_{\mathsf{S}}}(k_s; r, b_1, b_2))$.
3. Let $\pi \leftarrow \mathsf{NIZK.Prove}(u = ((\sigma_1, \sigma_2, \sigma_3), c_{\mathsf{R}}, m), w = (k_{\mathsf{R}}, r_{\mathsf{R}}, r_t))$ be a zero-knowledge proof of knowledge of witness $w = (k_{\mathsf{R}}, r_{\mathsf{R}}, r_t)$ such that:

$$\sigma_1 = g^{k_{\mathsf{R}}} h^{r_t} \wedge c_{\mathsf{R}} = g^{k_{\mathsf{R}}} h^{r_{\mathsf{R}}}.$$

Output σ as the simulated signature and π as the proof of fake-ness.

Informally, the simulated signature has exactly the same distribution as a real signature, and cannot be distinguished even given the verifier's key: for every t there exists a r_t that the verifier could have used to generate it. Only the verifier can prove fake-ness since no one else knows k_{R}, and so giving a proof would, with overwhelming probability, require opening c_{R} to a value different from k_{R}, which is infeasible if discrete log is hard in G.

Theorem 4. *If the AVPKE scheme is secure, and under the DDH assumption, Construction 4 is a secure PSDVS scheme.*

Remark 4. To get a concrete instantiation of the PSDVS scheme, we need to instantiate the AVPKE scheme (as explained above) and also the NIZKs assumed in Construction 4 (as explained in the Appendix A). This way, we get an instantiation in the random oracle model, secure under the strong RSA, the DDH and the CDRA assumptions.

Proof. In this proof we refer to the definitions in Section 3.1.

Unforgeability. Suppose adversary $\mathcal{A}$ wins the unforgeability game (in presence of a signing and verification oracle), and let $\sigma^* = (\sigma_1^*, \sigma_2^*, \sigma_3^*)$ be the forged signature, and m^* the message in question. If σ_3^* is a ciphertext that was not output by the signing oracle, then $\mathcal{A}$ can be used to break authenticity of the AVPKE scheme, since R only accepts a signature if decryption of σ_3^* does not reject. So we may assume that σ_3^* is a valid ciphertext $\mathsf{Enc}_{pp,sk_\mathsf{S},pk_\mathsf{R}}(k)$ that was used in a genuine signature $\sigma = (t, \mathsf{H}(m,t)^k, \mathsf{Enc}_{pp,sk_\mathsf{S},pk_\mathsf{R}}(k))$. Since $\mathcal{A}$ wins, $m \neq m^*$, so we can assume that $\mathsf{H}(m,t)$ and $\mathsf{H}(m^*, \sigma_1^*)$ are independent random variables output by the random oracle, and furthermore since R accepts the forged signature, we have $\sigma_2^* = \mathsf{H}(m, t^*)^k$.

This means we can use $\mathcal{A}$ to solve CDH in $\langle \hat{g} \rangle$ which contradicts the assumption that DDH (and hence CDH) is hard: Given a random CDH challenge $\hat{h}, \hat{h}^a, \hat{h}^b$, we will guess which genuine signature and which calls to the random oracle will be involved in the forgery. We will program the random oracle so that $\mathsf{H}(t, m) = \hat{h}$ and set $\sigma_2 = \hat{h}^a$. This means that we are implicitly claiming that the exponent used in the signature is a. This does not match the encryption $\mathsf{Enc}_{pp,sk_\mathsf{S},pk_\mathsf{R}}(k)$, but we program the verification oracle to accept the signature anyway, and by privacy of the encryption scheme, the inconsistency does not affect $\mathcal{A}$'s behavior significantly. Finally, we set $\mathsf{H}(t^*, m^*) = \hat{h}^b$, and it is now clear that if $\mathcal{A}$ wins, we have $\sigma_2^* = (\hat{h}^b)^a = \hat{h}^{ab}$.

Provable Public Simulation. $\mathsf{PubSigSim}$ indistinguishability follows from privacy of the AVPKE scheme: if an adversary $\mathcal{A}$ wins the PubSigSim-Ind game, we can use $\mathcal{A}$ to construct an adversary that wins the privacy game. The adversary can use sk_S to emulate the signing oracle and the use the decryption oracle to emulate the (less powerful) verification oracle. $\mathsf{PubSigSim}$ correctness and $\mathsf{PubSigSim}$ soundness follow immediately from completeness and soundness of the NIZK used.

Provable Verifier Simulation. $\mathsf{VerSigSim}$ indistinguishability is clear, since the signature produced by $\mathsf{VerSigSim}$ has exactly the same distribution as regular signatures. $\mathsf{VerSigSim}$ correctness follows from completeness of the NIZK used. Finally, for $\mathsf{VerSigSim}$ soundness, assume that a corrupt S could produce a proof that $\mathsf{VerSigVal}$ would accept. By soundness of the NIZK used, this would mean that we could extract from the proof data such that both the tag t in the signature and c_R could be opened as commitments to the same value, say x. However, R already knows from the key generation how to open c_R to the value k_R. The adversary gets no information on k_R during the game because commitments are perfectly hiding, and so $x \neq k_\mathsf{R}$ with overwhelming probability. This means we can use the adversary to break the binding property of the commitment scheme.

Sign. It is easy to see that Provable Signing Correctness and Soundness follow immediately from completeness and soundness of the NZIK used in Sign.

In Appendix A we list, for completeness, the standard Σ-protocols we need to instantiate our construction.

3.5 Sketch of a PSDVS Scheme Based on Prime Order Groups

In this subsection we sketch a variant of the previous scheme where we need only prime order groups and no trusted set-up. So we let G be a group of prime order p with generator g, where $p = 2q + 1$ and q is also prime. We let H denote the subgroup of Z_p^* of order q. H is also the group of squares modulo p. We let h be a generator of H. These parameters can be generated in public and can be verified by anyone, so no trusted set-up with secret trapdoor is required.

We make public keys for S and R as follows: $pk_{\mathsf{S}} = h^{sk_{\mathsf{S}}}$, $pk_{\mathsf{R}} = h^{sk_{\mathsf{R}}}$, $sk_{\mathsf{S}}, sk_{\mathsf{R}} \in_{\mathsf{R}} \in Z_q$.

The parties can compute a shared key $k = h^{sk_{\mathsf{S}} sk_{\mathsf{R}}}$ from only the public keys, which is pseudorandom for anyone else if DDH in H is hard.

We let the signature on m be $\mathsf{H}(m,t)^k$, which can be verified by R in the obvious way.

To prove in ZK that a signature $\sigma = (\sigma_1, \sigma_2)$ is valid, one proves knowledge of sk_{S}, such that $g^{pk_{\mathsf{S}}} = g^{h^{sk_{\mathsf{S}}}}$ and $\sigma_2 = \mathsf{H}(m, \sigma_1)^k = \mathsf{H}(m, \sigma_1)^{pk_{\mathsf{R}}^{sk_{\mathsf{S}}}}$ share the same "level-2 " discrete log, which can be done using a standard protocol which, however, requires a number of exponentiations linear in the security parameter.

For $\mathsf{PubSigSim}$, one generates a fake signature by choosing a random element $e \in \mathbb{Z}_p^*$ and letting the simulated signature be $\mathsf{H}(m,t)^{-e^2 \bmod p}$. $-e^2 \bmod p$ is not a square modulo p and hence is not in H. It will therefore not be accepted, since the correct k is in H by construction. This is indistinguishable from a genuine signature if we make a nonstandard variant of the DDH assumption saying that g raised to a random square is indistinguishable from g raised to a random non-square. One can show in ZK that $\mathsf{H}(m,t)^{-e^2 \bmod p}$ has the right form by showing that the discrete log of its inverse is a square, which can be done with standard Σ protocols.

The case of $\mathsf{VerSigSim}$ is handled in exactly the same way as the previous construction.

4 FE-based Construction

In this section, we present an MDVS scheme based on functional encryption. One disadvantage of this scheme is that it requires a trusted setup; secret verification keys must be derived from a master secret key. However, the accompanying advantage is that this scheme has verifier-identity-based signing; verifiers' public keys consist simply of their identity, allowing any signer to encrypt to any set of verifiers without needing to retrieve their keys from some PKI first.

At a high level, we are first given a digital signature scheme (DS) and a functional encryption scheme (FE). The keys of the signer with identity i are a secret DS signing key sk_i and corresponding public DS verification key vk_i. An MDVS signature c is a FE ciphertext obtained by encrypting the plaintext that consists of the message m, the signer's DS verification key vk_i, a set of designated verifier identities $\mathcal{D}$, and the signer's DS signature σ on the message using the secret DS signing key sk_i. That is, $c = \mathsf{FE.Enc}(pp, (m, vk_i, \mathcal{D}, \sigma))$. Verifier j's public key is simply their identity j (that is, $vpk_j = j$). Their secret key consists of a DS key pair (sk_j, vk_j), and an FE secret key dk_j. dk_j is the secret key for a function that checks whether j is among the specified designated verifiers, and then checks whether the DS signature σ inside the ciphertext c is either a valid signature under the signer's verification key vk_i, or under the verifier's verification key vk_j. However, this basic scheme does not give us the off-the-record property; we therefore tweak it slightly, as we describe below.

From One to Many DS Signatures In order to ensure that any subset of valid verifiers cannot convince an outsider of the origin of the MDVS signature, we need to replace the one DS signature in the ciphertext with a set of DS signatures. The reason is that, if only one signature is contained in the ciphertext, any designated verifier can prove to an outsider that "it was either me or the signer that constructed the signature". If more than one verifier proves this about the same MDVS signature, then the signature must have come from the signer.

To prevent this kind of "intersection attack", we allow the ciphertext to contain a set Σ of DS signatures, and change the corresponding FE secret keys to check if there exists a DS signature in the set that either verifies under the signer's or the verifier's DS verification key. Now, an outsider will no longer be convinced that it was the signer who constructed the MDVS signature, since each of the colluding verifiers could have constructed a DS signature that verifies under their own verification key, and then encrypted this set together with the public verification key of the signer.

Achieving Consistency In order to achieve consistency, we need security against malicious encryption in the underlying FE scheme. We need to ensure that any (possibly maliciously generated) ciphertext is consistent with one specific message across decryption with different functions. Otherwise, a malicious MDVS signer may be able to construct a ciphertext (i.e. a signature) that will be valid for one designated verifier but not valid for another, thereby breaking the consistency property. Security against a malicious encryption is a property of verifiable functional encryption (VFE), which was introduced by Badrinarayanan et. al [BGJS16]. However, it turns out that we do not need the full power of VFE, which also includes precautions against a malicious setup. Thus, we define a weaker notion of VFE, and substitute the standard FE scheme with this new scheme allowing us to achieve the MDVS consistency property.

In Section 4.1 we introduce ciphertext verifiable functional encryption, followed by our MDVS construction based on functional encryption which is presented in Section 4.2.

4.1 Verifiable Functional Encryption

An FE scheme starts with an authority that generates the public parameters pp and a master secret key msk. Then the holder of the master secret key can generate a decryption key dk_f associated with some function f that belongs to some predefined function family. Anyone can generate an encryption c of some value x, using only the public parameters, and the party that has been given the decryption key dk_f can decrypt the ciphertext c to obtain $f(x)$.

The standard security properties of functional encryption consider only the case where an adversary holds a set of decryption keys $dk_{f_1}, \ldots, dk_{f_q}$, and wants to learn more than it is allowed to about some encrypted message. The security property says that given an encryption of x, the adversary should only learn $f_1(x), \ldots, f_q(x)$.

However, in some settings, we additionally need security against malicious encryption, and possibly a malicious key generation authority. To achieve this, Badrinarayanan et. al [BGJS16] introduced verifiable functional encryption (VFE), which is an FE scheme extended with verification algorithms that check the validity of the ciphertexts and decryption keys.

We require security only against malicious encryption, allowing us to define a weaker notion of verifiability for functional encryption that handles malicious encryptors and decryptors, while assuming an honest authority.

Verifiable Functional Encryption. Let $\mathcal{F} = \mathcal{F}_\kappa$ be a function family, $\mathcal{M} = \mathcal{M}_\kappa$ the message space, and $\mathcal{Y} = \mathcal{Y}_\kappa$ the output space such that $\mathcal{F} : \mathcal{M} \to \mathcal{Y}$. Let $\mathcal{C} = \mathcal{C}_\kappa$ be the ciphertext space. Then, we define a *ciphertext verifiable functional encryption* (VFE) scheme for function family $\mathcal{F}$ by the following algorithms:

$\mathsf{Setup}(1^\kappa) \to (pp, msk)$**:** The PPT algorithm Setup, on input the security parameter κ, outputs the public parameters pp and the master secret key msk.

$\mathsf{KeyGen}(msk, f) \to dk_f$**:** The PPT key generation algorithm KeyGen, on input the master secret key msk and a function $f \in \mathcal{F}$, outputs a secret key dk_f.

$\mathsf{Enc}(pp, m) \to c$**:** The PPT encryption algorithm Enc, on input the public parameters pp and a message m, outputs a ciphertext c.

$\mathsf{Dec}(dk_f, c) \to y'$**:** The decryption algorithm Dec, on the decryption key dk_f and the ciphertext c, outputs $y' \in \mathcal{Y} \cup \{\perp\}$.

$\mathsf{Verify}(pp, c) \to d$**:** The public verification algorithm Verify, on input the public parameters pp and the ciphertext c, outputs a boolean decision $d = 0$ (reject) or $d = 1$ (accept).

Properties The scheme will have the standard correctness and IND-CPA security of a FE scheme, as described in the following definitions.

Definition 18 (Correctness). *Let $\kappa \in \mathbb{N}$ be the security parameter, and let $\mathcal{F} : \mathcal{M} \to \mathcal{Y}$ be a function family. Let* $\mathsf{VFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a FE scheme for function family $\mathcal{F}$. For all messages $m \in \mathcal{M}$, all functions $f \in \mathcal{F}$ we have*

$$Pr\left[\mathsf{Dec}(\mathsf{KeyGen}(msk, f), \mathsf{Enc}(pp, m)) \neq f(m)\right] \leq \mathsf{negl}(\kappa),$$

where $(pp, msk) \leftarrow \mathsf{Setup}(1^\kappa)$, and the probability is taken over the random choices of all algorithms.

Definition 19 (IND-CPA Security). *Let $\kappa \in \mathbb{N}$ be the security parameter, and let $\mathcal{F} : \mathcal{M} \to \mathcal{Y}$ be a function family. Let* $\mathsf{VFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a FE scheme for function family $\mathcal{F}$. Consider the following game between a challenger and an adversary $\mathcal{A}$:*

$$\mathsf{Game}^{\mathsf{IND-CPA}}_{\mathsf{VFE},\mathcal{F},\mathcal{A}}(\kappa)$$

```
1. $(pp, msk) \leftarrow \mathsf{Setup}(1^\kappa)$
2. $(m_0, m_1) \leftarrow \mathcal{A}^{\mathcal{O}_G}(pp)$
3. $b \leftarrow_\$ \{0,1\}$
4. $c \leftarrow \mathsf{Enc}(pp, m_b)$
5. $b' \leftarrow \mathcal{A}^{\mathcal{O}_G}(c)$
```

The key generation oracle is defined $\mathcal{O}_G(f_i) := \mathsf{KeyGen}(msk, f_i)$*.*

We say that $\mathcal{A}$ wins the IND-CPA game *if* $b = b'$, $|m_0| = |m_1|$, *and* $f_i(m_0) = f_i(m_1)$ *for all queries* $f_i \in \mathcal{F}$ *to oracle* $\mathcal{O}_G$*. We say a FE scheme satisfies the IND-CPA security property if, for all PPT* $\mathcal{A}$*,*

$$\mathsf{adv}^{\mathsf{IND-CPA}}_{\mathsf{VFE},\mathcal{F},\mathcal{A}}(\kappa) = Pr[\mathcal{A} \textit{ wins the game}] - \frac{1}{2} \leq \mathsf{negl}(\kappa).$$

We also need security against a malicious encryptor. Here we simplify the verifiability of [BGJS16] to verifiability of ciphertexts:

Definition 20 (Ciphertext Verifiability). *A scheme* VFE *for function family* $\mathcal{F}$ *is ciphertext verifiable, if, for all* $c \in \{0,1\}^*$*, there exists* $x \in \mathcal{M}$ *such that for all* $f \in \mathcal{F}$ *and* $dk_f \leftarrow \mathsf{KeyGen}(msk, f)$*, if* $\mathsf{Verify}(pp, c) = 1$*, then*

$$Pr\left[\mathsf{Dec}(dk_f, c) = f(x)\right] = 1,$$

where $(pp, msk) \leftarrow \mathsf{Setup}(1^\kappa)$*.*

Definition 20 states that for all ciphertexts constructed by a malicious encryptor, it must hold that if the ciphertext c passes the verification algorithm, then there exists a unique input x that can be associated with ciphertext c, meaning that for all functions $f \in \mathcal{F}$ the decryption of c will yield $f(x)$.

Ciphertext Verifiable Functional Encryption We consider a *ciphertext verifiable functional encryption* scheme $\mathsf{VFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Verify})$ for function family $\mathcal{F}$. The formal definition and the security notions can be found in Appendix **??**.

Ciphertext Verifiability This property states that for all ciphertexts c, it must hold that if c passes the verification algorithm, then there exists a unique plaintext x asociated with c, meaning that for all functions $f \in \mathcal{F}$ the decryption of c will yield $f(x)$.

As an example, consider the functions f_{even} and f_{odd} that determine whether an input natural number is even or odd. The adversary cannot create a ciphertext c that passes the verification algorithm, and on decryption using these functions will yield a result that claims that the element encrypted in c is both even and odd.

As discussed in [GJKS15], an FE scheme satisfying these properties can be achieved by combining a standard FE scheme with a simulation-sound NIZK proof of knowledge to achieve security against malicious encryption.

4.2 The MDVS Construction

Construction 5 *Let* $\mathsf{SIGN} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *be a standard digital signature scheme and let* $\mathsf{VFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Verify})$ *be a functional encryption scheme secure with ciphertext verifiability. Then we define a MDVS scheme* $\mathsf{FEMDVS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Sim})$ *as follows:*

$\mathsf{Setup}(1^\kappa)$**:** $(pp^{\mathsf{FE}}, msk^{\mathsf{FE}}) \leftarrow \mathsf{VFE.Setup}(1^\kappa)$.

Output public parameter $pp = pp^{\mathsf{FE}}$ and master secret key $msk = msk^{\mathsf{FE}}$.

$\mathsf{SignKeyGen}(i)$**:** $(sk_i, vk_i) \leftarrow \mathsf{SIGN.KeyGen}(1^\kappa)$.

Output the signer's secret key $ssk_i = sk_i$ and public key $spk_i = vk_i$.[17]

$\mathsf{VerKeyGen}(msk, j)$**:**

1. $vpk_j = j$,
2. $(sk_j, vk_j) \leftarrow \mathsf{SIGN.KeyGen}(1^\kappa)$,
3. $dk_j \leftarrow \mathsf{VFE.KeyGen}(msk^{\mathsf{FE}}, f_j)$, where f_j is defined as follows.

```
Function $f_j$
Input: $m, vk_i, \{vpk_{j'}\}_{j' \in \mathcal{D}}, \Sigma$;
Const: $vpk_j, vk_j$;
1. If $vpk_j \notin \{vpk_{j'}\}_{j' \in \mathcal{D}}$: output $\perp$;
2. If $\exists \sigma \in \Sigma$ : $\mathsf{SIGN.Verify}(vk_i, m, \sigma) = 1$ OR $\mathsf{SIGN.Verify}(vk_j, m, \sigma) = 1$:
     output $(m, vk_i, \{vpk_{j'}\}_{j' \in \mathcal{D}})$;
3. Else: output $\perp$
```

Output the verifiers secret key $vsk_j = (sk_j, dk_j)$ and public key $vpk_j = j$.[18]

$\mathsf{Sign}(pp, ssk_i, \{vpk_j\}_{j \in \mathcal{D}}, m)$**:**

1. $\sigma \leftarrow \mathsf{SIGN.Sign}(sk_i, m)$.
2. Output $c = \mathsf{VFE.Enc}(pp^{\mathsf{FE}}, (m, vk_i, \{vpk_j\}_{j \in \mathcal{D}}, \{\sigma, \perp, \cdots, \perp\}))$.

$\mathsf{Verify}(pp, spk_i, vsk_j, \{vpk_j\}_{j \in \mathcal{D}}, m, c)$**:**

1. Check whether $\mathsf{VFE.Verify}(pp^{\mathsf{FE}}, c) = 1$. If not, output 0.
2. Compute $(m', vk'_i, \{vpk_j\}_{j \in \mathcal{D}'}) \backslash \perp \leftarrow \mathsf{VFE.Dec}(dk_j, c)$. If the output is $\perp$, output 0.
3. Check $m' = m$, $vk'_i = vk_i$ (with $spk_i = vk_i$), and $\mathcal{D}' = \mathcal{D}$. If all hold, output 1. Otherwise output 0.

$\mathsf{Sim}(pp, spk_i, \{vpk_j\}_{j \in \mathcal{D}}, \{vsk_j\}_{j \in \mathcal{C}}, m)$**:**

1. For each $j \in \mathcal{C}$, $vsk_j = (sk_j, dk_j)$.
2. Compute $\sigma_j \leftarrow \mathsf{SIGN.Sign}(sk_j, m^*)$.
3. Let $\Sigma = \{\sigma_j\}_{j \in \mathcal{C}^*}$, add default values to get the required size.
4. Output $c = \mathsf{VFE.Enc}(pp^{\mathsf{FE}}, (m^*, spk_i, \{vpk_j\}_{j \in \mathcal{D}}, \Sigma))$.

[17] We assume that the mapping $i \to (ssk_i, spk_i)$ is unique in the system. This can be achieved without loss of generality by pseudorandomly generating the randomness required in the key generation process from the identity i and the master secret key.

[18] We assume that the mapping $j \to (vsk_j, vpk_j)$ is unique in the system. This can be achieved wlog by pseudorandomly generating the randomness required in the key generation process from the identity j and the master secret key.

Theorem 5. *Assume that* VFE *is an IND-CPA secure functional encryption scheme with ciphertext verifiability, and* SIGN *is an existential unforgeable digital signature scheme. Then Construction 5 is a correct and secure MDVS scheme with privacy of identities and verifier-identity-based signing.*

Proof. Correctness: Follows directly from an inspection of the algorithms and the correctness of the functional encryption scheme.

Consistency: Assume that there exist an adversary that produces an inconsistent signature: $(i^*, \{vpk_j\}_{j\in\mathcal{D}^*}, m^*, c^*)$, such that there exists $j_1, j_2 \in \mathcal{D}^*$ (for which the adversary does not have the corresponding secret keys) such that:

$$\mathsf{Verify}(spk_{i^*}, vsk_{j_1}, \{vpk_j\}_{j\in\mathcal{D}^*}, m^*, c^*) = 1,$$
$$\mathsf{Verify}(spk_{i^*}, vsk_{j_2}, \{vpk_j\}_{j\in\mathcal{D}^*}, m^*, c^*) = 0,$$

where $spk_{i^*} = vk_{i^*}$, $vsk_{j_1} = (sk_{j_1}, dk_{j_1})$, and $vsk_{j_2} = (sk_{j_2}, dk_{j_2})$.

Since the verification for j_1 yields 1, then both j_1 and j_2 will verify the ciphertext: $\mathsf{VFE.Verify}(pp^{\mathsf{FE}}, c) = 1$, meaning that (thanks to ciphertext verifiability) there exists a unique encrypted message $(m, vk_i, \{vpk_j\}_{j\in\mathcal{D}}, \sigma)$ in c that is consistent across decryption with different functions.

Then j_1 will decrypt: $(m, vk_i, \{vpk_j\}_{j\in\mathcal{D}}) \leftarrow \mathsf{VFE.Dec}(dk_j, c)$, which is equal to $(m^*, vk_{i^*}, \{vpk_j\}_{j\in\mathcal{D}^*})$. On the other hand j_2 will decrypt

$$(m', vk'_i, \{vpk_j\}_{j\in\mathcal{D}'}) \text{ or } \perp \leftarrow \mathsf{VFE.Dec}(dk_j, c)$$

If the output was $\perp$, then either $j_2 \notin \mathcal{D}$ or there does not exists a valid signature in σ. Otherwise the output was $(m', vk'_i, \{vpk_j\}_{j\in\mathcal{D}'})$ which is different from $(m, vk_i, \{vpk_j\}_{j\in\mathcal{D}})$ in at least one component. Thus, the output of the decrypt of j_1 and j_2 is not consistent with a unique message, which contradicts the fact that $\mathsf{VFE.Verify}(pp^{\mathsf{FE}}, c) = 1$ (i.e. there exists a unique encrypted message).

Thus, an adversary that violated the consistency of the MDVS scheme, violates the ciphertext verifiability of the FE scheme.

Existential Unforgeability: Assume that the adversary produces a valid forgery: $(i^*, \{vpk_j\}_{j\in\mathcal{D}^*}, m^*, c^*)$, where c^* is the signature (a FE ciphertext) on message m^*, from signer i^* designated to the verifiers in the set $\mathcal{D}^*$.

Then there exists a designated verifier $j \in \mathcal{D}^*$, who has not been corrupted by the adversary, such that the FE decryption will yield m', vk'_i and $\mathcal{D}'$, where $m' = m^*$, $vk'_i = vk_{i^*}$, and $\mathcal{D}' = \mathcal{D}^*$ (otherwise it would not be a valid forgery).

The only thing left in the FE ciphertext is the set of signatures σ. In order for c^* to be a forgery, then there must exist a digital signature $\sigma \in \sigma$ such that

$$\mathsf{SIGN.Verify}(vk'_i, m', \sigma) = 1 \text{ OR } \mathsf{SIGN.Verify}(vk_j, m', \sigma) = 1.$$

This means that the adversary must create a digital signature forgery for either signer i or "signer" j, without knowing the corresponding secret signing keys. This contradicts the assumption that the digital signature satisfies existential unforgeability.

Privacy of Identities: The adversary receives a signature c^*, which is an FE encryption of one of the two following messages:

1. $(m^*, vk_{i_0}, \{vpk_j\}_{j\in\mathcal{D}_0}, \{\mathsf{SIGN.Sign}(sk_{i_0}, m^*), \bot, \cdots, \bot\})$,
2. $(m^*, vk_{i_1}, \{vpk_j\}_{j\in\mathcal{D}_1}, \{\mathsf{SIGN.Sign}(sk_{i_1}, m^*), \bot, \cdots, \bot\})$.

In the PSI game, the adversary is not allowed to ask for verification keys for any of the designated verifiers in $\mathcal{D}_0$ or $\mathcal{D}_1$. This means that for all verification keys (i.e. FE decryption keys) the adversary can ask for, we have that the underlying function f_j evaluated on the two messages will yield $\bot$, since j is not in any of the two sets of designated verifiers.

Thus, privacy of the identities (PSI and PVI) follows directly from the IND-CPA security of the functional encryption scheme.

Off-The-Record: The off-the-record property follows directly from the IND-CPA security of the functional encryption scheme. The messages in the real and simulated version have the same length, since we add default values to σ to ensure we have the same number of elements in both cases.

For all functions f_j that the adversary gets an FE decryption key for, we argue that evaluation on each of the two messages results in the same output. First we look at the decryption keys dk_j for $j \notin \mathcal{D}^*$ (i.e. not a designated verifier). In both the real and the simulated case the decryption will yield $\bot$, since j is not a designated verifier.

Next, we look at the decryption keys dk_j for $j \in \mathcal{D}^*$. In this case j must also be in the corruption set $\mathcal{C}^*$. Thus, in the real case the set σ contains σ_{i^*} a digital signature of message m^* under the signing key of signer i^*, and the decryption will yield $(m^*, vk_{i^*}, \{vpk_j\}_{j\in\mathcal{D}^*})$. In the simulated case the set σ contains σ_j a digital signature of message m^* under the signing key of party j. Thus, the decryption will again yield $(m^*, vk_{i^*}, \{vpk_j\}_{j\in\mathcal{D}^*})$, since function f_j does not differentiate whether it was the verification vk_{i^*} or vk_j that was used to verify the digital signature. □

References

AV19. Prabhanjan Ananth and Vinod Vaikuntanathan. Optimal bounded-collusion secure functional encryption. *IACR Cryptology ePrint Archive*, 2019:314, 2019.

BGB04. Nikita Borisov, Ian Goldberg, and Eric Brewer. Off-the-record communication, or, why not to use PGP. In *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, pages 77–84. ACM, 2004.

BGJS16. Saikrishna Badrinarayanan, Vipul Goyal, Aayush Jain, and Amit Sahai. Verifiable functional encryption. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016*, pages 557–587, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

Cha96. David Chaum. Private signature and proof systems, 1996. US Patent 5,493,614.

Cha11. Ting Yi Chang. An id-based multi-signer universal designated multi-verifier signature scheme. *Inf. Comput.*, 209(7):1007–1015, 2011.

Cho06. Sherman S. M. Chow. Identity-based strong multi-designated verifiers signatures. In *Public Key Infrastructure, Third European PKI Workshop: Theory and Practice, EuroPKI 2006, Turin, Italy, June 19-20, 2006, Proceedings*, pages 257–259, 2006.

Cho08. Sherman S. M. Chow. Multi-designated verifiers signatures revisited. *I. J. Network Security*, 7(3):348–357, 2008.

DJ03. Ivan Damgård and Mads Jurik. A length-flexible threshold cryptosystem with applications. In *ACISP*, volume 2727 of *Lecture Notes in Computer Science*, pages 350–364. Springer, 2003.

FO97. Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 1997.

GJKS15. Vipul Goyal, Abhishek Jain, Venkata Koppula, and Amit Sahai. Functional encryption for randomized functionalities. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *Theory of Cryptography*, pages 325–351, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

GVW12. Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 162–179, 2012.

JSI96. Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. Designated verifier proofs and their applications. In Ueli Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, pages 143–154, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

LSMP07. Yong Li, Willy Susilo, Yi Mu, and Dingyi Pei. Designated verifier signature: Definition, framework and new constructions. In *Ubiquitous Intelligence and Computing, 4th International Conference, UIC 2007, Hong Kong, China, July 11-13, 2007, Proceedings*, pages 1191–1200, 2007.

LV04. Fabien Laguillaumie and Damien Vergnaud. Multi-designated verifiers signatures. In *Information and Communications Security, 6th International Conference, ICICS 2004, Malaga, Spain, October 27-29, 2004, Proceedings*, pages 495–507, 2004.

LV07. Fabien Laguillaumie and Damien Vergnaud. Multi-designated verifiers signatures: anonymity without encryption. *Inf. Process. Lett.*, 102(2-3):127–132, 2007.

Mar13. Moxie Marlinspike. Advanced cryptographic ratcheting. 2013.

MW08. Yang Ming and Yumin Wang. Universal designated multi verifier signature scheme without random oracles. *Wuhan University Journal of Natural Sciences*, 13(6):685–691, Dec 2008.

NSM05. Ching Yu Ng, Willy Susilo, and Yi Mu. Universal designated multi verifier signature schemes. In *11th International Conference on Parallel and Distributed Systems, ICPADS 2005, Fuduoka, Japan, July 20-22, 2005*, pages 305–309, 2005.

SHCL08. Seung-Hyun Seo, Jung Yeon Hwang, Kyu Young Choi, and Dong Hoon Lee. Identity-based universal designated multi-verifiers signature schemes. *Computer Standards & Interfaces*, 30(5):288–295, 2008.

SKS06. G. Shailaja, K. Phani Kumar, and Ashutosh Saxena. Universal designated multi verifier signature without random oracles. In *9th International Con-*

ference in Information Technology, ICIT 2006, Bhubaneswar, Orissa, India, 18-21 December 2006, pages 168–171, 2006.

Tia12. Haibo Tian. A new strong multiple designated verifiers signature. *IJGUC*, 3(1):1–11, 2012.

Ver08. Damien Vergnaud. New extensions of pairing-based signatures into universal (multi) designated verifier signatures. *CoRR*, abs/0802.1076, 2008.

ZAYS12. Yunmei Zhang, Man Ho Au, Guomin Yang, and Willy Susilo. (strong) multi-designated verifiers signatures secure against rogue key attack. In *Network and System Security - 6th International Conference, NSS 2012, Wuyishan, Fujian, China, November 21-23, 2012. Proceedings*, pages 334–347, 2012.

Zhe97. Yuliang Zheng. Digital signcryption or how to achieve cost (signature and encryption) << cost (signature) + cost (encryption). In *Annual International Cryptology Conference*, pages 165–179. Springer, 1997.

A Instantiation of Non-Interactive ZK Proofs

We list here, for completeness, the standard Σ-protocols we need. We assume the same set-up as above, that is, a group G of order an RSA modulus n is given (a product of safe primes), as well as a generator g of G, and a generator $\hat{g}$ of the subgroup of squares mod n.

The protocols we list here are well-known or simple variations of know protocols. It is easy to show the standard completeness, soundness and honest verifier ZK properties, so we will not present these proofs, but recall some ideas where these may be less well known. The protocols can be turned into non-interactive ZK proofs of knowledge in the standard way in the random oracle model using the Fiat-Shamir heuristic.

All proofs have negligible soundness error, so we always need only 1 iteration of each protocol.

A general technical remark: some of the protocols are usually designed for use in a group of prime order, while here we use them in a group of order n. The only difficulty this could lead to is that the proofs of soundness requires us to invert various non-zero numbers modulo the group order. This could in principle fail modulo n, but this would lead to finding a non-trivial factor of n (which is generated in a trusted manner as part of the setup) and so can only happen with negligible probability if factoring is hard, which we have to assume throughout anyway.

Protocols for the AVPKE scheme. Some number theoretic background first: because $n = pq = (2p'+1)(2q'+1)$ is a product of safe primes, the subgroup of squares in $\mathbb{Z}_n^*$ has order $p'q'$ and $\hat{g}$ is chosen to be a generator. It lies inside the subgroup of numbers of Jacobi symbol 1 which has order $2p'q'$ and is generated by $\hat{g}$ and -1.

The first protocol has as public input $\hat{g}, \hat{h} \in \mathbb{Z}_n^*$ and we assume $\hat{h}$ Jacobi symbol 1, this can be easily checked by the verifier. Recall that an honest prover knows r such that $\hat{h} = \hat{g}^r$. The protocol goes as follows:

Protocol $\mathsf{Composite\ order\ discrete\ log}$.

1. P chooses $s \in_{\mathsf{R}} \{0,1\}^{3\kappa}$ and sends $a = \hat{g}^s$ to V (interpreting s as a binary number).

2. V sets $e \in_{\mathsf{R}} \mathbb{Z}_n$ and sends it to P.
3. P returns $z = s + er$ to V, and V checks that $\hat{g}^z = a\hat{h}^e \bmod n$.

This protocol is easily seen to be complete and statistical honest verifier zero-knowledge (note that s is chosen to be exponentially larger than er so z is statistically close to a random 3κ bit number. Soundness is more tricky. *Assuming* that $\hat{h}$ is in the group generated by $\hat{g}$, then the protocol is a proof of knowledge of r, under the strong RSA assumption modulo n, as shown by Fujisaki and Okamoto [FO97]. Now, since $\hat{h}$ has Jacobi symbol 1, either $\hat{h}$ or $-\hat{h}$ is in the subgroup, so the protocol proves that P knows the discrete log of $\hat{h}$ or $-\hat{h}$.

For the second protocol we have public input $c = (n+1)^k v^n \bmod n^2$ and $\beta = \gamma^k$ for some $\gamma \in G$. The prover knows k and v. The protocol goes as follows:

Protocol Plaintext and discrete log knowledge.

1. P chooses $s \in_{\mathsf{R}} \mathbb{Z}_n, u \in_{\mathsf{R}} \mathbb{Z}_n^*$ and sends $a = (n+1)^s u^n \bmod n^2$, $\alpha = \gamma^s$ to V.
2. V sets $e \in_{\mathsf{R}} \mathbb{Z}_n$ and sends it to P.
3. P returns $z = s + er \bmod n, w = uv^e \bmod n$, and V checks that $(n+1)^z w^n = ac^e \bmod n^2$ and that $\gamma^z = \alpha\beta^e$.

It is trivial to prove this protocol complete, sound and honest-verifier zero-knowledge.

To to the proof π_{valid} in the AVPKE scheme, we run the Composite order discrete log and the Plaintext and discrete knowledge protocols where in the latter we omit γ, β, α.

Protocols for the PVDVS scheme. The first protocol works with the well-known Pedersen commitments, where a commitment to x with randomness r is of form $g^x h^r$ where $g, h \in G$. These commitments are perfectly hiding and computationally binding if discrete log in G is hard. The protocol shows that two Pedersen commitments contain the same value. The public input is $c_1 = g_1^x h_1^{r_1}, c_2 = g_2^x h_2^{r_2}$. The protocol works as follows:

Protocol Commitment equality.

1. P sets $y, s_1, s_2 \in_{\mathsf{R}} \mathbb{Z}_n$, and sends $a_1 = g_1^y h_1^{s_1}, a_2 = g_2^y h_2^{s_2}$ to V
2. V sets $e \in_{\mathsf{R}} \mathbb{Z}_n$ and sends to P.
3. P sends $z = y + ex \bmod n$ and $u_1 = s_1 + er_1 \bmod n, u_2 = s_2 + er_2 \bmod n$ to V.
4. V checks that $g_1^z h_1^{u_1} = a_1 c_1^e$ and that $g_2^z h_2^{u_2} = a_2 c_2^e$.

The NIZK in VerSigSim uses this protocol.

For the NIZK in PubSigSim, recall that we have $\mathsf{H}(m,t)^k$, $E_{fake}(k', r, b, b', v)$ as input, and we want to demonstrate that $k \neq k'$. The prover includes $\mathsf{H}(m,t)^{k'}$ in the proof, and runs the Plaintext and discrete log knowledge protocol to demonstrate that the exponent k' is also present in the encryption. For this, we consider only the last part of the ciphertext, which is a Paillier encryption of k', as this part already uniquely determines k'. The prover can use k' and $(-1)^{b'}v$ as witness. Finally, the verifier checks that $\mathsf{H}(m,t)^{k'} \neq \mathsf{H}(m,t)^k$. Soundness and completeness of this should be clear. For zero-knowledge, the simulator would produce $\mathsf{H}(m,t)^{k''}$ for random k'', and simulate the Plaintext and discrete log knowledge protocol. Of course the statement in question is now false, but by privacy of the encryption scheme, the simulated public data is indistinguishable from the case where the statement is true, so the simulator must still produce an indistinguishable transcript.

Finally, for the proof of real signature in Sign, we have to show that the ciphertext encrypting k is completely well formed so that R will accept, and that this exponent is used in the MAC. For this we use the following protocol. The public input is $\delta = \gamma^k$ for a publicly known $\gamma \in G$ (namely a hash-value), $pk_{\mathsf{S}} = \hat{g}^{sk_{\mathsf{S}}} \bmod n$ and

$$C = E_{sk_{\mathsf{S}},pk_{\mathsf{R}}}(k,r,b_1,b_2) = ((-1)^{b_1}\hat{g}^r \bmod n,\ (n{+}1)^k((-1)^{b_2}\beta_1^{sk_{\mathsf{S}}}\beta_2^r \bmod n)^n \bmod n^2),$$

for publicly known $\beta_1, \beta_2 \in Z^*_{n^2}$, which in our context come from R's public key pk_{R}. The prover's secret witness is $sk_{\mathsf{S}}, k, r, b_1, b_2$.

Protocol Valid signature.

1. P chooses $s \in \mathbb{Z}_n$, $x,y \in \{0,1\}^{3\kappa}$, $c_1, c_2 \in \{0,1\}$. He then sends to V: $a = E_{sk_{\mathsf{S}}+x,pk_{\mathsf{R}}}(s,y,c_1,c_2)$, $\xi = \hat{g}^x \bmod n$ and $\omega = \gamma^s$.
2. V sets $e {\in_{\mathsf{R}}} \mathbb{Z}_n$ and sends it to P.
3. P returns $z_s = s{+}er \bmod n$, $z_x = x{+}esk_{\mathsf{S}}$, $z_y = y{+}er$, $d_1 = (c_1{+}eb_1) \bmod 2$, $d_2 = (c_2 + eb_2) \bmod 2$. V checks that $E_{z_x,pk_{\mathsf{R}}}(z_s,z_y,d_1,d_2) = aC^e$, that $\hat{g}^{z_x} = \xi pk_{\mathsf{S}}^e$ and that $\gamma^{z_s} = \omega\delta^e$.

It is tedious but straightforward to check completeness. For (statistical) honest verifier ZK we use that x,y are both exponentially lager than esk_{S} and er, respectively. For soundness, assume we get acceptable answers $(z_s, z_x, z_y, d_1, d_2)$ and $(z'_s, z'_x, z'_y, d'_1, d'_2)$ to challenges e, e'. This will imply that we get equations:

$$E_{z_x - z'_x,pk_{\mathsf{R}}}(z_s - z'_s, z_y - z'_y, (d_1 - d'_1) \bmod 2, (d_2 - d'_2) \bmod 2) = C^{e-e'}$$

$$\hat{g}^{z_x - z'_x} = pk_{\mathsf{S}}^{e-e'},\quad \gamma^{z_s - z'_s} = \delta^{e-e'}$$

Now, note that the protocol is in fact (implicitly) using the Composite order discrete log protocol to prove knowledge of the discrete logs of plus or minus the numbers $pk_{\mathsf{S}} = \hat{g}^{sk_{\mathsf{S}}} \bmod n$ and $(-1)^{b_1}\hat{g}^r \bmod n$, where the latter occurs inside the ciphertext.

The soundness proof for that protocol from [FO97] argues if we get acceptable answers to two challenges e, e', then under the strong RSA assumption, it must be that $e - e'$ divides the difference between the answers. So under this assumption we get that $e - e'$ divides $z_x - z'_x$ and $z_y - z'_y$. It is now straightforward to extract a valid witness, essentially by dividing by $e - e'$ in the exponent on both sides.