Generic Constructions for Verifiably Encrypted Signatures without Random Oracles or NIZKs

Markus Rückert⋆, Michael Schneider, and Dominique Schröder⋆

Technische Universität Darmstadt, Germany
markus.rueckert@cased.de,
mischnei@cdc.informatik.tu-darmstadt.de,
schroeder@me.com

Abstract. Verifiably encrypted signature schemes (VES) allow a signer to encrypt his or her signature under the public key of a trusted third party, while maintaining public signature verifiability. With our work, we propose two generic constructions based on Merkle authentication trees that do not require non-interactive zero-knowledge proofs (NIZKs) for maintaining verifiability. Both are stateful and secure in the standard model. Furthermore, we extend the specification for VES, bringing it closer to real-world needs. We also argue that statefulness can be a feature in common business scenarios.

Our constructions rely on the assumption that CPA (even slightly weaker) secure encryption, "maskable" CMA secure signatures, and collision resistant hash functions exist. "Maskable" means that a signature can be hidden in a verifiable way using a secret masking value. Unmasking the signature is hard without knowing the secret masking value. We show that our constructions can be instantiated with a broad range of efficient signature and encryption schemes, including two lattice-based primitives. Thus, VES schemes can be based on the hardness of worst-case lattice problems, making them secure against subexponential and quantum-computer attacks. Among others, we provide the first efficient pairing-free instantiation in the standard model.

Keywords: Generic construction, Merkle tree, post-quantum, standard model.

1 Introduction

Boneh et al. introduced the concept of verifiably encrypted signatures (VES) at Eurocrypt 2003 as a means of covertly exchanging signatures, while maintaining their verifiability [8]. They include a passive, trusted third party, the adjudicator, which makes VES schemes fall into the category of optimistic fair exchange protocols [2,4].

Signer, receiver, and adjudicator, interact as follows. The signer encrypts his or her signature σ for a document M as a verifiably encrypted signature ϖ. Given ϖ, the receiver can verify that it contains a valid signature for M, but is otherwise unable to extract σ. A commercially important application is *online contract signing* under the supervision of an escrow. As opposed to the classic (offline) scenario, where the escrow needs to be involved in every step of the process, verifiably encrypted signatures make signing contracts more cost-efficient due to the passiveness of the escrow and simultaneously allow the parties to negotiate online rather than meet in person. They simply exchange verifiably encrypted signatures on the negotiated contract, verify them, and finally exchange the corresponding regular signatures. Assume Alice acts honestly, while Bob tries to misuse (e.g., for negotiating a better deal elsewhere or simply for blackmail) the partially signed document without providing a signature himself. The adjudicator can step in and disclose Bob's signature. The contract becomes binding despite Bob's attempt to back out.

Boneh et al. propose the first construction [8], which is provably secure in the random oracle model (ROM), followed by a slightly more efficient construction by Zhang et al. [24]. Lu et al. present the first scheme in the standard model in [13]. Furthermore, they sketch a generic construction based on NIZKs. Another NIZK construction has been proposed by Dodis et al. in [11]. Using NIZKs, however, is generally very inefficient with respect to computational cost and signature size.

The previous efficient instantiations typically use pairings in order to achieve verifiability of El Gamal encrypted signatures. With pairings, however, security proofs have to rely on very special versions of the Diffie-Hellman problem, whose actual hardness is yet to be assessed [7]. Note that there is a second line of work on "verifiable encryption" in a more general scenario by Camenisch and Shoup [9] as well as Ateniese [3]. Their objectives differ from the one in [8] as Boneh et al. demand that transmitting and verifying an encrypted signature can be done in a single move. In particular, the verification process does not involve interactive ZK proofs as required in both [9] and [3]. Therefore, we focus on the line of research coming from Boneh et al.

Our Contribution

Generic construcions of cryptographic schemes help understand their complexity. A common approach is: take a message, encrypt it, and append a NIZK, proving that the encrypted value satisfies some property. Removing NIZKs is non-trivial. In the following, we discuss how they can be avoided in the context of VES.

VES with a Setup Phase. We extend the model of Boneh et al. in the sense that the signer's key may depend on the adjudicator's public key. More precisely, we allow signer and adjudicator to interact *once* during key generation. We believe that this is a good model of real-world business scenarios. To illustrate this, let's consider a notary, the adjudicator, that oversees fair online contract signing. In general, the notary wants to remain passive but he or she still wants to bill his

or her services on a per signature-exchange basis. With our extension and the instantiations therein, we show how the (offline) notary can actually control the number of verifiably encrypted signature that his or her customers can securely exchange. The customer pays for a certain number of signatures in advance and the notary completes the customer's key pair accordingly. Interestingly, the underlying signature scheme can still be used in a black-box way. Thus, smart cards or other signing devices can be used and the secret signing key is not revealed. This is important for contract signing as laws and ordinances often require this for the contract to be legally binding.

Generic Construction. So far, there have been two construction principles for VES schemes: use pairings for efficiency or NIZKs for generic constructions from minimal cryptographic assumptions. Our construction fills the gap between those extremes as it can be considered both efficient (compared to NIZKs) and generic. We believe that our construction principles may also be helpful in finding NIZK-free generic constructions for other schemes. In detail, we show three things.

Firstly, generic constructions for VES schemes need not involve inefficient non-interactive zero-knowledge proofs. We propose two generic constructions in the standard model, which is encouraged by the work of Canetti, Goldreich, and Halevi [10]. Both are based on "random plaintext secure" (RPA) encryption[1], "maskable" existentially unforgeable (EU-CMA) signatures, and a collision-resistant hash function in a Merkle authentication tree [18] that can be handled efficiently. This allows us to scale the resulting scheme according to the individual needs of specific application scenarios.

Maskability is a property of the signature scheme that states that we can choose a random masking value α and mask a given signature σ for a message M by calling $(\tau, \beta) \leftarrow \mathsf{Mask}(\sigma, \alpha)$. The resulting masked signature τ is still valid for the same message under a modified verification procedure that uses some additional advice β. Given (τ, β), it is hard to recover σ. However, with the secret masking value α, one can call $\sigma' \leftarrow \mathsf{Unmask}(\tau, \beta, \alpha)$ and recover a valid (ordinary) signature for M.

Our first construction uses regular (many-time) signature schemes, the other solely requires the existence of a suitable one-time signature scheme. Both of our constructions are stateful and the key generation step depends, as always with tree-based constructions [18], on the desired signature capacity ℓ of the resulting scheme. Using Merkle trees for VES was first considered in [20] for the special case of RSA signatures. By formalizing this approach, we develop this technique to its full potential.

Secondly, pairings are not necessary for efficient VES schemes. In particular, we show the first pairing-free VES scheme in the standard model without NIZKs.

Thirdly, we introduce efficient VES schemes to the post-quantum era because we give lattice-based instantiations that withstand quantum computer and subexponential attacks. Previous instantiations will become insecure in the

---

[1] A weaker notion than CPA security, where the adversary has to distinguish the ciphertext for a *random* message from the encryption of the 0-string (see Section 3).

presence of quantum computers due to the work of Shor [23]. The full version contains a second instantiation from lattices and one from RSA.

Organization. We start by recalling the VES security model in Section 2. There, we also propose our extension. Then, we describe the required building blocks, including "maskability", in Section 3, followed by our generic constructions. In order to demonstrate their feasibility, we instantiate both constructions with various efficient primitives in Section 4 and the full version [21].

2   Verifiably Encrypted Signatures

Verifiably encrypted signature schemes support the encryption of signatures under the public key of a trusted third party, while simultaneously proving that the encryption contains a valid signature. They are built upon digital signature schemes $\mathsf{DSig} = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vf})$, defined via *Key Generation:* $\mathsf{Kg}(1^n)$ outputs a key pair $(\mathsf{ssk}, \mathsf{spk})$. ssk is a private signing key and spk the corresponding public verification key $\mathsf{spk} \in \mathcal{K}$ (public key space); *Signing:* $\mathsf{Sign}(\mathsf{ssk}, M)$ outputs a signature $\sigma \in \Sigma$ (signature space) on a message $M \in \mathcal{M}$ (message space) under ssk; *Signature Verification:* $\mathsf{Vf}(\mathsf{spk}, \sigma, M)$ outputs 1 iff σ is a valid signature on M under spk. Now, a verifiably encrypted signature scheme $\mathsf{VES} = (\mathsf{AdjKg}, \mathsf{AdjSetup}, \mathsf{Kg}, \mathsf{Sign}, \mathsf{Vf}, \mathsf{Create}, \mathsf{VesVf}, \mathsf{Adj})$ consists of the following algorithms. Note that we generalize the model slightly by letting the key generation algorithm of the signer depend on the keys of the adjudicator (see $\mathsf{AdjSetup}$ below). In particular, we view the key generation process as an interactive algorithm between the adjudicator and the signer, i.e., the interaction takes place only once.

Adjudicator Key Generation: $\mathsf{AdjKg}(1^n)$ outputs a key pair $(\mathsf{ask}, \mathsf{apk})$, where ask is the private key and apk the corresponding public key.

Adjudication Setup: The adjudicator provides an algorithm $\mathsf{AdjSetup}(\mathsf{ask}, \mathsf{pk})$ whose input is the private key of the adjudicator ask and a public key pk of the signer. It returns a key pk'.

Key Generation: The key generation algorithm $\mathsf{Kg}(1^n)$ may interact with the adjudicator via the oracle $\mathsf{AdjSetup}(\mathsf{ask}, \cdot)$ to produce the key pair $(\mathsf{sk}, \mathsf{pk})$.

Signing and Verification: Same as in a digital signature scheme

VES Creation: $\mathsf{Create}(\mathsf{sk}, \mathsf{apk}, M)$ takes as input a secret key sk, the adjudicator's public key apk, and a message $M \in \mathcal{M}$. It returns a verifiably encrypted signature ϖ for M.

VES Verification: $\mathsf{VesVf}(\mathsf{apk}, \mathsf{pk}, \varpi, M)$ takes as input the adjudicator's public key apk, a public key pk, a verifiably encrypted signature ϖ, and a message M. It returns a bit.

Adjudication: $\mathsf{Adj}(\mathsf{ask}, \mathsf{apk}, \mathsf{pk}, \varpi, M)$ takes as input the key pair $(\mathsf{ask}, \mathsf{apk})$ of the adjudicator, the public key of the signer pk, a verifiably encrypted signature ϖ, and a message M. It extracts an ordinary signature σ for M.

A VES scheme is *complete* if for all adjudication key pairs $(\mathsf{ask}, \mathsf{apk}) \leftarrow \mathsf{AdjKg}(1^n)$ and for all signature key pairs $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Kg}^{\mathsf{AdjSetup}(\mathsf{ask}, \cdot)}(1^n)$ the following holds: $\mathsf{VesVf}(\mathsf{apk}, \mathsf{pk}, \mathsf{Create}(\mathsf{sk}, \mathsf{apk}, M), M) = 1$ and $\mathsf{Vf}(\mathsf{pk}, \mathsf{Adj}(\mathsf{ask}, \mathsf{apk}, \mathsf{pk}, \mathsf{Create}(\mathsf{sk}, \mathsf{apk}, M)), M) = 1$ for all $M \in \mathcal{M}$.

Experiment $\mathsf{VesForge}_{\mathcal{A}}^{\mathsf{VES}}(n)$

    $(\mathsf{ask}, \mathsf{apk}) \leftarrow \mathsf{AdjKg}(1^n)$

    $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Kg}^{\mathsf{AdjSetup}(\mathsf{ask}, \cdot)}(1^n)$

    $(M^*, \varpi^*) \leftarrow \mathcal{A}^{\mathfrak{C}(\mathsf{sk}, \mathsf{apk}, \cdot), \mathfrak{A}(\mathsf{ask}, \mathsf{apk}, \mathsf{pk}, \cdot, \cdot), \mathfrak{S}(\mathsf{ask}, \cdot)}(\mathsf{pk}, \mathsf{apk})$

    Return 1 iff $\mathsf{VesVf}(\mathsf{apk}, \mathsf{pk}, \varpi^*, M^*) = 1$ and

        $\mathcal{A}$ has never queried M^* to $\mathfrak{C}(\mathsf{sk}, \mathsf{apk}, \cdot)$

        or $\mathfrak{A}(\mathsf{ask}, \mathsf{apk}, \mathsf{pk}, \cdot, \cdot)$.

Experiment $\mathsf{Opac}_{\mathcal{A}}^{\mathsf{VES}}(n)$

    $(\mathsf{ask}, \mathsf{apk}) \leftarrow \mathsf{AdjKg}(1^n)$

    $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Kg}^{\mathsf{AdjSetup}(\mathsf{ask}, \cdot)}(1^n)$

    $(M^*, \sigma^*) \leftarrow \mathcal{A}^{\mathfrak{C}(\mathsf{sk}, \mathsf{apk}, \cdot), \mathfrak{A}(\mathsf{ask}, \mathsf{apk}, \mathsf{pk}, \cdot, \cdot), \mathfrak{S}(\mathsf{ask}, \cdot)}(\mathsf{pk}, \mathsf{apk})$

    Return 1 iff $\mathsf{Vf}(\mathsf{pk}, \sigma^*, M^*) = 1$ and

        $\mathcal{A}$ has never queried M^* to $\mathfrak{A}(\mathsf{ask}, \mathsf{apk}, \mathsf{pk}, \cdot, \cdot)$.

Experiment $\mathsf{Extract}_{\mathcal{A}}^{\mathsf{VES}}(n)$

    $(\mathsf{ask}, \mathsf{apk}) \leftarrow \mathsf{AdjKg}(1^n)$

    $(M^*, \varpi^*, \mathsf{pk}^*) \leftarrow \mathcal{A}^{\mathfrak{A}(\mathsf{ask}, \mathsf{apk}, \cdot, \cdot, \cdot), \mathfrak{S}(\mathsf{ask}, \cdot)}(\mathsf{apk})$

    Let $\sigma^* \leftarrow \mathsf{Adj}(\mathsf{ask}, \mathsf{apk}, \mathsf{pk}^*, \varpi^*, M^*)$

    Return 1 iff $\mathsf{VesVf}(\mathsf{apk}, \mathsf{pk}^*, \varpi^*, M^*) = 1$

        and $\mathsf{Vf}(\mathsf{pk}^*, \sigma^*, M^*) = 0$.

Experiment $\mathsf{Collusion}_{\mathcal{A}}^{\mathsf{VES}}(n)$

    $(\mathsf{apk}, \mathsf{ask}) \leftarrow \mathsf{AdjKg}(1^n)$

    $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Kg}^{\mathsf{AdjSetup}(\mathsf{ask}, \cdot)}(1^n)$

    $\mathsf{state} \leftarrow (\mathsf{apk}, \mathsf{ask}, \mathsf{pk})$

    $(M^*, \varpi^*) \leftarrow \mathcal{A}^{\mathfrak{C}(\mathsf{sk}, \mathsf{apk}, \cdot)}(\mathsf{state})$

    Return 1 iff $\mathsf{VesVf}(\mathsf{apk}, \mathsf{pk}, \varpi^*, M^*) = 1$ and

        $\mathcal{A}$ has never queried $\mathfrak{C}(\mathsf{pk}, \mathsf{apk}, \cdot)$ about M^*.

Fig. 1. Overview over the different security experiments

2.1 Security Model

Security of verifiably encrypted signatures is defined by unforgeability, opacity [8], extractability, and collusion-resistance [22].[2] *Unforgeability* requires that it is hard to forge a verifiably encrypted signature, *opacity* implies that it is difficult to extract an ordinary signature from an encrypted signature, *extractability* guarantees that the adjudicator can always extract a regular signature from a valid verifiably encrypted signature, and *collusion-resistance* prevents signer and adjudicator from successfully colluding in order to produce a verifiably encrypted signature on behalf of another party, provided that the collusion happens in the online phase and *not* during key registration. The security requirement can be interpreted as a stronger form of unforgeability.

Since we allow the key generation algorithm to depend on the interaction with the adjudicator, we also give the adversary access to the corresponding oracle. Unforgeability and opacity are formalized in experiments, where the adversary is given the public keys of the signer and of the adjudicator. Moreover, the adversary has access to three oracles: $\mathfrak{C}$ returns verifiably encrypted signatures for a given message, $\mathfrak{A}$ extracts a regular signature from a verifiably encrypted signature, and $\mathfrak{S}$ allows the adversary to perform setup queries. In the extractability experiment, the adversarial signer is given access to an adjudication oracle and wins if he or she can output an encrypted signature that is hidden irrecoverably. Finally, the collusion-resistance experiment gives the adversary direct access to the adjudicator's private key. The goal is to forge a signature for another party. All experiments are defined in Figure 1.

As usual, n is the security parameter. VES is *secure* if the following holds for any efficient adversary $\mathcal{A}$:

Unforgeability: VES is unforgeable if $\mathsf{VesForge}_{\mathcal{A}}^{\mathsf{VES}}(n)$ outputs 1 with negligible probability.

---

[2] Note that in [22] this was called "abuse-freeness". Here, however, we prefer "collusion-resistance" because abuse-freeness already has a slightly different meaning in the context of fair-exchange, which creates confusion.

Opacity: VES is opaque if $\mathsf{Opac}_{\mathcal{A}}^{\mathsf{VES}}(n)$ outputs 1 with negligible probability.

Extractability: VES is extractable if $\mathsf{Extract}_{\mathcal{A}}^{\mathsf{VES}}(n)$ outputs 1 with negligible probability.

Collusion-Resistance: VES is *collusion-resistant* if $\mathsf{Collusion}_{\mathcal{A}}^{\mathsf{VES}}(n)$ outputs 1 with negligible probability. In this experiment the adversary gets as input the adjudication key pair and a signer public key pk. The adversary gets access to a VES creation oracle $\mathfrak{C}(\mathsf{sk}, \mathsf{apk}, \cdot)$. It does not need the other oracles because it has ask.

A scheme is $(t, q_{\mathfrak{C}}, q_{\mathfrak{A}}, q_{\mathfrak{S}}, \epsilon)$-secure, if no adversary, running in time at most t, making at most $q_{\mathfrak{C}}$ verifiably encrypted signature oracle queries, at most $q_{\mathfrak{A}}$ adjudication oracle queries, and at most $q_{\mathfrak{S}}$ key registration queries, can succeed with probability at least ϵ in the VesForge, Opac, Extract (with $q_{\mathfrak{C}} = 0$), or Collusion (with $q_{\mathfrak{A}} = q_{\mathfrak{S}} = 0$) experiment.

Since we have extended the model slightly, we have to re-prove the relations among the different properties. The relations in [22] still hold in our setting as phrased in the following propositions. We defer the details to the full version.

Proposition 1. *If unforgeable VES schemes exist, there is also an unforgeable VES scheme that is not extractable.*

Proposition 2. *If unforgeable and extractable VES schemes exist, there is also an unforgeable and extractable VES scheme that is not collusion-resistant.*

In addition, there is a new relation that can be quite useful when proving a VES secure. It states that a VES is unforgeable if it is collusion-resistant. To see this, observe that giving ask to $\mathcal{A}$ in the collusion-resistance experiment enables the adversary to simulate the oracles $\mathfrak{S}$ and $\mathfrak{A}$ itself. Thus, if $\mathcal{A}$ is successful in the unforgeability experiment it can also break collusion-resistance.

Proposition 3. *If VES is collusion-resistant, it is also unforgeable.*

2.2   Discussion

As already discussed in the introduction, giving the key generation algorithm access to the adjudicator corresponds to the natural case where we have an initial setup phase. Note that the oracle $\mathfrak{S}$ only takes as input the public key and *not* the private key. Thus, this phase cannot be compared with the models that require the signer to prove knowledge of the secret key (e.g., KOSK). Moreover, this phase only takes place once, during key generation and *not* during each signature creation process. The adjudicator remains offline, i.e., our modification is suitable for fair exchange protocols with a *passive* adjudicator. Via AdjSetup, the adjudicator may define parts of signer keys. Giving the adjudicator too much control over, however, is discouraged as it affects collusion-resistance.

In [22], the authors show that "abuse-freeness" is already implicit as long as the underlying signature scheme is unforgeable and treated as a black box (key independence, Definition 2 below). Since we consider a slightly stronger definition, we have to re-prove this result for collusion-resistance (Lemma 2).

Note that our constructions satisfy these properties. Thus, it is sufficient to prove opacity and extractability because key-independent and extractable schemes are automatically unforgeable (Lemma 1).

3   Generic Construction

We propose two efficient generic constructions based on CPA-secure (even slightly weaker) encryption, "maskable" digital signature schemes, and collision resistant hash functions. As our instantiations in Section 4 and in the full version demonstrate, this does not overly restrict the possible choices. In order to ensure that verifiably encrypted signatures can always be decrypted (cf. extractability), we build a Merkle authentication tree from a collision-resistant hash function. While our first construction uses regular signature schemes, our second construction reduces the assumptions even further by merely relying on one-time signatures.

3.1   Building Blocks

Let $G : \{0,1\}^* \rightarrow \{0,1\}^n$ be a collision resistant hash function and let $\{x_i\}_1^\ell$ be the ordered set of values $x_1, \ldots, x_\ell$. Furthermore, we use $x \leftarrow_\$ S$, when choosing an $x \in S$ uniformly at random and $x \overset{\Delta}{\leftarrow} S$ if x is chosen according to a distribution Δ over S. For our generic constructions, we need to define "maskable" signature schemes, secure encryption schemes, and Merkle authentication trees.

Digital Signature Scheme. The definition of digital signature schemes $\mathsf{DSig} = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vf})$ follows the well-known and established definition due to [12].

For our generic construction, we need a signature scheme that is "maskable". Generally speaking, signatures can be hidden by a masking value, such that we can still verify them. Furthermore, we have to be able to recover a valid signature from a valid masked one. We formalize this in the following definition.

Definition 1 (Maskability). *Let DSig be a signature scheme with public-key space $\mathcal{K}$, signature space Σ, and message space $\mathcal{M}$. It is* maskable *if there is a corresponding masking scheme $\mathsf{MS_{DSig}} = (\mathsf{Advice}, \mathsf{Mask}, \mathsf{Unmask}, \mathsf{Vf})$ with the following specification:*

Sets: Let $\mathcal{S}$ be a set of masking values and let Δ be a distribution over $\mathcal{S}$. Furthermore, let $\mathcal{V}$ be the set of advice strings for verifying masked signatures and $\mathcal{T}$ be the space of masked signatures.
Advice: On input spk, α, Advice outputs an advice $\beta \in \mathcal{V}$.
Mask: On input $\mathsf{spk}, \sigma, \alpha, M$, the algorithm Mask outputs a masked signature $\tau \in \mathcal{T}$. Notice that we do not require a perfect masking scheme but allow the scheme to output the special symbol $\tau = \bot$.
Unmask: On input τ, β, α, M, the algorithm Unmask outputs a signature $\sigma \in \Sigma$.
Verification: On input $\mathsf{spk}, \tau, \beta$, and M the algorithm $\mathsf{MS_{DSig}}.\mathsf{Vf}$ outputs a bit, indicating the validity of the masked signature. If $\tau = \bot$, it returns 0.

Experiment $\mathsf{Recover}_{\mathcal{A}}^{\mathsf{MS_{DSig}}}(n)$
> $(\mathsf{ssk}, \mathsf{spk}) \leftarrow \mathsf{DSig.Kg}(1^n)$
> $\alpha \overset{\Delta}{\leftarrow} \mathcal{S}$
> $\beta \leftarrow \mathsf{Advice}(\mathsf{spk}, \alpha)$
> $\sigma^* \leftarrow \mathcal{A}^{\mathfrak{M}(\mathsf{ssk}, \mathsf{spk}, \alpha, \cdot), \mathsf{DSig.Sign}(\mathsf{ssk}, \cdot)}(\mathsf{spk}, \beta)$
>   Let $\{M_i\}_1^\ell$ be the queries to $\mathsf{DSig.Sign}$ and let M^* be the query to $\mathfrak{M}$.
>   Return 1 iff $M^* \notin \{M_i\}_1^\ell$ and $\mathsf{DSig.Vf}(\mathsf{spk}, \sigma^*, M^*) = 1$.

Fig. 2. Experiment for the hiding property of a masking scheme

Validity: We require $\mathsf{MS_{DSig}.Vf}(\mathsf{spk}, \tau, \beta, M) = 1 \implies \mathsf{DSig.Vf}(\mathsf{spk}, \sigma', M) = 1$, where $\sigma' = \mathsf{Unmask}(\tau, \beta, \alpha, M)$ and $\beta = \mathsf{Advice}(\mathsf{spk}, \alpha)$, for all keys, masked signatures, messages, and masking values. Note that β is honestly created.

Hiding: The masking scheme needs to hide the signature σ (for M) in (τ, β) such that no adversary can recover a valid signature for M without knowing the masking value α. This must even hold if the adversary can query an oracle $\mathfrak{M}$ *once* that returns a masked signature for an adversely chosen message and a randomly chosen α: $\mathfrak{M}(\mathsf{ssk}, \mathsf{spk}, \alpha, M) = [\sigma \leftarrow \mathsf{DSig.Sign}(\mathsf{ssk}, M); \tau \leftarrow \mathsf{Mask}(\mathsf{spk}, \sigma, \alpha, M); \mathrm{Return}\ \tau;]$. Furthermore, the adversary can make arbitrary queries to an ordinary signature oracle. MS is (t, ϵ)-*hiding* if there is no adversary, running in time t, that wins the Experiment in Figure 2 with probability at least ϵ.

Notice that the above definition can be trivially satisfied by an encryption scheme. Then, the output of Advice is a zero-knowledge proof, demonstrating that it was honestly encrypted. Here, the masking value α would comprise the public and secret keys for the encryption scheme.

As we are interested in efficient instantiations, we propose that somewhat homomorphic signature schemes can provide the same functionality. We demonstrate this with the following example.

Example 1. Take the RSA signature scheme with full-domain hash function H and public key (N, v). The verification function for a signature σ on a message M checks whether $0 \leq \sigma < N$ and $\sigma^v = \mathsf{H}(M)$ over $\mathbb{Z}_N$. We let $\Sigma = \mathcal{V} = \mathbb{Z}_N$ and $\mathcal{S} = \mathbb{Z}_N^*$. Δ is the uniform distribution. $\mathsf{Mask}((N, v), \sigma, \alpha, M)$ outputs $\sigma\alpha \bmod N$ and $\mathsf{Advice}((N, v), \alpha)$ returns $\alpha^v \bmod N$. Thus, $\mathsf{Unmask}(\tau, \beta, \alpha, M)$ has to compute $\sigma \leftarrow \tau\alpha^{-1} \bmod N$. The modified verification algorithm $\mathsf{MS.Vf}(\mathsf{spk}, \tau, \beta, M)$ checks whether $0 \leq \tau < N$ and $\tau^v = \mathsf{H}(M)\beta$. Observe that validity and the hiding property are satisfied in the random oracle model.

Remark 1. Given the above example, it is easy to see that one can forge a masked signature (τ, β) that passes $\mathsf{MS.Vf}$, unless α and $\beta = \mathsf{Advice}(\mathsf{pk}, \alpha)$ are well-formed. One could simply compute $\beta \leftarrow \tau^v/\mathsf{H}(M)$ over $\mathbb{Z}_N$ for arbitrary M and τ. The result (τ, β, M) would be valid because $\tau^v \equiv \mathsf{H}(M)\beta$. However, in our constructions, the attacker is not able to choose β freely. It is chosen

during a trusted setup procedure and then authenticated with a hash tree. This authentication mechanism yields an implicit rejection of adversely chosen β.

Notice that AdjSetup and the setup oracle $\mathfrak{S}$ are always controlled by the experiments in Figure 1 to make the setup procedure trusted. A straightforward extension of our security model would be to remove this trusted setup procedure.

Random Plaintext Attacks (RPA). Let PKE $=$ (Kg, Enc, Dec) be a public key encryption scheme, defined via *Key Generation:* Kg(1^n) outputs a key pair (esk, epk). epk is the public encryption key and esk is the secret decryption key; *Encryption:* Enc(epk, M) outputs a ciphertext c for M; *Decryption:* Dec(esk, c) attempts to decrypt c and returns the enclosed message, or $\perp$ upon failure. We define a weaker notion of security for encryption schemes that we call *random plaintext attacks* that is similar to a key encapsulation mechanism (KEM). The idea is that the adversary obtains a randomly chosen string s and a ciphertext c. The task is to determine whether c encrypts s or the 0-string.

We say that PKE is indistinguishable under random plaintext attacks (IND-RPA) if no efficient algorithm $\mathcal{A}$ can associate a randomly generated plaintext with its ciphertext. The adversary wins if it is able to guess b with probability $> 1/2$. PKE is *RPA secure* if for any efficient $\mathcal{A}$ and a negligible $\epsilon = \epsilon(n)$

$$\left| \mathrm{Prob}\left[b = \mathcal{A}(\mathsf{epk}, C) : (\mathsf{esk}, \mathsf{epk}) \leftarrow \mathsf{Kg}(1^n); b \leftarrow_\$ \{0,1\}; M_0 \leftarrow_\$ \mathcal{M}; M_1 \leftarrow 0^{|M_0|}; C \leftarrow_\$ \mathsf{Enc}(\mathsf{epk}, M_b) \right] - \tfrac{1}{2} \right| < \epsilon.$$

CPA $\implies$ *RPA.* We claim that the notion of random plaintext attacks is strictly weaker than chosen plaintext attacks, in the sense that any CPA scheme is also RPA, but not vice-versa.

Proposition 4. *A CPA secure scheme is also RPA secure. If an RPA secure scheme exists then there is also an RPA secure scheme that is not CPA secure.*

The first part is obvious. As for the second part, the basic idea is letting Enc$(1\|0^{n-1})$ in the modified RPA scheme output esk. Clearly, the scheme is not CPA secure but it is still RPA secure.

Merkle Authentication Trees. Merkle presented a tree structure that can be used to authenticate big amounts of data using only a single hash value [18]. Originally his idea was to create digital signatures out of one-time signature schemes, but many other applications of Merkle trees appeared in the past. With our constructions, we add verifiable encryption to this list of applications.

A Merkle tree is a complete binary tree of height h that is built from the bottom up to the root such that the leaves define the whole tree. The leaves are numbered consecutively from left to right. Inner nodes are constructed using the following rule: a node's value is the hash value of the concatenation of its children left and right: node $=$ G(left$\|$right), where G : $\{0,1\}^* \rightarrow \{0,1\}^n$ is a collision resistant hash function. The root of the tree is used to authenticate the leaf values. For the authentication process, additional tree nodes are required.

These nodes form the *authentication path* of a leaf. Consider the path from the leaf with index $\varphi \in [1, 2^h]$ to the root. The siblings of the nodes on this path form the authentication path of this leaf. Using this path and the construction rule $\mathsf{G}(\mathsf{left}\|\mathsf{right})$, the root of the tree can be reconstructed. If the calculated root matches the original one, the leaf data is correctly authenticated.

Using an adversary that is able to replace a leaf value, such that the replaced leaf is still correctly authenticated by the tree, one can find collisions in the underlying hash function G. For an overview of techniques, results, and references, we refer the reader to [5, Chapter 3].

3.2 Generic Construction

The general idea is to use a maskable signature scheme with one-time masking values and encrypt these masking values under the adjudicator's public key. The *validity* property of the masking scheme ensures completeness, and opacity will be guaranteed by the *hiding* property. In general, we take an ordinary σ and hide it by applying Mask, using one of ℓ predefined one-time masking values α. If, for any reason, the masking scheme returns an invalid masked signature, the process is repeated with the next α. This allows for a broader range of (imperfect) masking schemes. The corresponding advice β for verification is also precomputed. Then, β and an encryption γ of α are used to build a Merkle authentication tree that allows a verifier to efficiently check whether β and γ correspond. The adjudicator forms the tree during the initial registration phase and signs its root under a certification key pair $(\mathsf{csk}, \mathsf{cpk})$ in order to prevent malicious signers from cheating in the extractablity experiment.

Construction 1. *Let* DSig *be a maskable signature scheme with masking scheme* $\mathsf{MS}_{\mathsf{DSig}}$, PKE *be a public key encryption scheme and* $\mathsf{G} : \{0,1\}^* \mapsto \{0,1\}^n$ *be a collision resistant hash function. Choose an adequate* $h \in \mathbb{N}$, *such that the resulting scheme admits* $\ell = 2^h$ *signatures.* $\mathsf{VES}_1 = (\mathsf{AdjKg}, \mathsf{AdjSetup}, \mathsf{Kg}, \mathsf{Sign}, \mathsf{Vf}, \mathsf{Create}, \mathsf{VesVf}, \mathsf{Adj})$ *is defined as follows.*

Adjudicator Key Generation: *Call* $(\mathsf{ask}, \mathsf{apk}) \leftarrow \mathsf{PKE.Kg}(1^n)$, $(\mathsf{csk}, \mathsf{cpk}) \leftarrow \mathsf{DSig.Kg}(1^n)$ *and output* $((\mathsf{ask}, \mathsf{csk}), (\mathsf{apk}, \mathsf{cpk}))$.

Adjudication Setup: *On input* $((\mathsf{ask}, \mathsf{csk}), \mathsf{spk})$, *perform the following steps:*
  1. Choose $\alpha_i \stackrel{\triangle}{\leftarrow} \mathcal{S}$ *and set* $\beta_i \leftarrow \mathsf{Advice}(\mathsf{spk}, \alpha_i)$, $\gamma_i \leftarrow \mathsf{Enc}(\mathsf{apk}, \alpha_i)$ *for* $i = 1, \ldots, \ell$; *2. Construct a Merkle tree* T *using* G, *i.e., with leaves* $\mathsf{G}\Big(\mathsf{G}(\beta_i)\|\mathsf{G}(\gamma_i)\Big)$ *that fully define the root* ρ; *3. Compute the signature* $\sigma_\rho \leftarrow \mathsf{DSig.Sign}(\mathsf{csk}, \rho)$; *4. Output* $(\{\alpha_i\}_1^\ell, \{\gamma_i\}_1^\ell, \rho, \sigma_\rho)$.

Key Generation: *Perform the following steps: 1. Call* $(\mathsf{ssk}, \mathsf{spk}) \leftarrow \mathsf{DSig.Kg}(1^n)$; *2. Call* $(\{\alpha_i\}_1^\ell, \{\gamma_i\}_1^\ell, \rho, \sigma_\rho) \leftarrow \mathsf{AdjSetup}((\mathsf{ask}, \mathsf{csk}), \mathsf{spk})$; *3. Initialize a signature counter* $c \leftarrow 0$; *4. Output* $\mathsf{pk} = (\mathsf{spk}, \rho, \sigma_\rho)$ *and* $\mathsf{sk} = (\mathsf{ssk}, c, \{\alpha_i\}_1^\ell, \{\gamma_i\}_1^\ell)$.

Sign, Verify: *As defined in the underlying signature scheme* DSig.

Create: *On input* $\mathsf{sk}, \mathsf{pk}, M$, *the algorithm* Create *works in three steps: 1. Increment the counter* c: $c \leftarrow c + 1$; *2. Sign the message* M *using the underlying signature scheme:* $\sigma \leftarrow \mathsf{DSig.Sign}(\mathsf{ssk}, M)$; *3. Mask the signature* σ *with the secret*

value α_c: $\tau \leftarrow \mathsf{Mask}(\mathsf{spk}, \sigma, \alpha_c, M)$; *4. If* $\mathsf{MS_{DSig}.Vf}(\mathsf{spk}, \tau, \beta_c, M) = 0$ *increase c and go to 3. The output is* $\varpi = (\tau, \beta_c, \gamma_c, \pi_c)$, *where* $\beta_c \leftarrow \mathsf{Advice}(\mathsf{spk}, \alpha_c)$ *and* π_c *is the authentication path for leaf c.*

VES Verification: *On input* $((\mathsf{apk}, \mathsf{cpk}), (\mathsf{spk}, \rho, \sigma_\rho), (\tau, \beta, \gamma, \pi), M)$, VesVf *outputs 1 iff 1.* $\mathsf{DSig.Vf}(\mathsf{cpk}, \sigma_\rho, \rho) = 1$; *2.* π *is correct for* β *and* γ *with respect to* ρ; *3.* $\mathsf{MS_{DSig}.Vf}(\mathsf{spk}, \tau, \beta, M) = 1$.

Adjudication: *On input* $((\mathsf{ask}, \mathsf{csk}), (\mathsf{apk}, \mathsf{cpk}), (\mathsf{spk}, \rho, \sigma_\rho), (\tau, \beta, \gamma, \pi), M)$, Adj *verifies the input using* VesVf. *If it is correct, it decrypts* $\alpha' \leftarrow \mathsf{Dec}(\mathsf{ask}, \gamma)$, *calls* $\sigma' \leftarrow \mathsf{Unmask}(\tau, \beta, \alpha', M)$, *and outputs* σ'.

3.3 Generic Construction Using One-Time Signatures

Since we already need a Merkle authentication tree for Construction 1, we can as well use a suitable one-time signature instead of a regular one. These signatures are potentially easier to achieve, i.e., they may be secure under milder assumptions. The following construction demonstrates that the second tree that is needed to turn a one-time signature scheme into a "many-time" signature scheme via the Merkle transformation can be easily merged with the first one.

Construction 2. *With* OTS *we denote a maskable one-time signature scheme with masking scheme* $\mathsf{MS_{OTS}}$. *We define* $\mathsf{VES_2}$ *as follows:*

Adjudicator Key Generation: *Call* $(\mathsf{ask}, \mathsf{apk}) \leftarrow \mathsf{PKE.Kg}(1^n)$, $(\mathsf{csk}, \mathsf{cpk}) \leftarrow \mathsf{DSig.Kg}(1^n)$ *and output* $((\mathsf{ask}, \mathsf{csk}), (\mathsf{apk}, \mathsf{cpk}))$.

Adjudication Setup: *On input* $((\mathsf{ask}, \mathsf{csk}), \{\mathsf{spk}_i\}_1^\ell)$, *perform the following steps: 1. Choose* $\alpha_i \stackrel{\Delta}{\leftarrow} \mathcal{S}$ *and set* $\beta_i \leftarrow \mathsf{Advice}(\mathsf{spk}, \alpha_i)$, $\gamma_i \leftarrow \mathsf{Enc}(\mathsf{apk}, \alpha_i)$ *for* $i = 1, \ldots, \ell$; *2. Construct a Merkle authentication tree* T *using the hash function* G, *where the leaves are of the form* $\mathsf{G}\big(\mathsf{G}(\beta_i)||\mathsf{G}(\gamma_i)||\mathsf{G}(\mathsf{spk}_i)\big)$. *Denote the root node with* ρ; *3. Compute the signature* $\sigma_\rho \leftarrow \mathsf{DSig.Sign}(\mathsf{csk}, \rho)$; *4. Output* $(\{\alpha_i\}_1^\ell, \{\gamma_i\}_1^\ell, \rho, \sigma_\rho)$.

Key Generation: *Run the following steps: 1. Call* $(\mathsf{ssk}_i, \mathsf{spk}_i) \leftarrow \mathsf{OTS.Kg}(1^n)$ *for* $i = 1, \ldots \ell$; *2. Call* $(\{\alpha_i\}_1^\ell, \{\gamma_i\}_1^\ell, \rho, \sigma_\rho) \leftarrow \mathsf{AdjSetup}((\mathsf{ask}, \mathsf{csk}), \{\mathsf{spk}_i\}_1^\ell)$; *3. Initialize a signature counter* $c \leftarrow 0$; *4. Output* $\mathsf{pk} = (\rho, \sigma_\rho)$ *and* $\mathsf{sk} = (\{\mathsf{ssk}_i\}_1^\ell, \{\mathsf{spk}_i\}_1^\ell, c, \{\alpha_i\}_1^\ell, \{\gamma_i\}_1^\ell)$.

Sign, Verify: *As defined in* OTS.

Create: *On input* $\{\mathsf{ssk}_i\}_1^\ell, \{\mathsf{spk}_i\}_1^\ell, c, \{\alpha_i\}_1^\ell, \{\gamma_i\}_1^\ell, M$, Create *works in four steps: 1. Increment c:* $c \leftarrow c + 1$; *2. Sign M:* $\sigma \leftarrow \mathsf{OTS.Sign}(\mathsf{ssk}_c, M)$; *3. Mask* σ: $\tau \leftarrow \mathsf{Mask}(\mathsf{spk}_c, \sigma, \alpha_c, M)$; *4. If* $\mathsf{MS_{DSig}.Vf}(\mathsf{spk}_c, \tau, \beta_c, M) = 0$ *go to 1. The output is* $\varpi = (\tau, \beta_c, \gamma_c, \pi_c, \mathsf{spk}_c)$, *where* $\beta_c \leftarrow \mathsf{Advice}(\mathsf{spk}, \alpha_c)$ *and* π_c *is the authentication path for leaf c.*

VES Verification: *On input* $((\mathsf{apk}, \mathsf{cpk}), (\rho, \sigma_\rho), (\tau, \beta, \gamma, \pi, \mathsf{spk}), M)$, VesVf *outputs 1 iff 1.* $\mathsf{DSig.Vf}(\mathsf{cpk}, \sigma_\rho, \rho) = 1$; *2.* ρ *can be reconstructed using* π, β, γ, *and* spk; *3.* $\mathsf{MS_{DSig}.Vf}(\mathsf{spk}, \tau, \beta, M) = 1$.

Adjudication: *On input* $((\mathsf{ask}, \mathsf{csk}), (\mathsf{apk}, \mathsf{cpk}), (\rho, \sigma_\rho), (\tau, \beta, \gamma, \pi, \mathsf{spk}), M)$, Adj *verifies the input using* VesVf. *If it is correct, it decrypts* $\alpha' \leftarrow \mathsf{Dec}(\mathsf{ask}, \gamma)$, *calls* $\sigma' \leftarrow \mathsf{Unmask}(\tau, \beta, \alpha', M)$, *and outputs* σ'.

3.4   Proof of Security

We show in this section that VES_1 satisfies the desired security requirements. Security of Construction 2 is proven analogously, the assumptions are just weaker, i.e., we only need a maskable one-time signature scheme instead of a regular one. We show extractability, unforgeability, collusion-resistance, and opacity. The proofs for VES_1 and VES_2 are essentially the same, we focus on VES_1.

Theorem 1 (Extractability). *If* G *is collision resistant,* DSig *is unforgeable, and* $\mathsf{MS}_{\mathsf{DSig}}$ *satisfies validity then* VES_1 *(*VES_2*) is extractable.*

Proof. The reduction plays against unforgeability of DSig and uses the validity of $\mathsf{MS}_{\mathsf{DSig}}$ and the collision-resistance of G and in the analysis. The unforgeability ensures that the adversary has to call $\mathsf{AdjSetup}$ to create its public key and validity guarantees that an extracted signatures is valid if computed from an honestly masked signature. Most importantly, the collision-resistance of G prevents the adversary from altering the leaves of the authentication tree, i.e., from being able to dishonestly mask a signature.

The reduction chooses the adjudication key honestly during the simulation and has access to a signature oracle for DSig and to the verification key spk. Thus, the adversary's environment can be perfectly simulated. The adversarial user $\mathcal{A}$ outputs a public key $(\mathsf{pk}^*, \rho^*, \sigma_\rho^*)$ and a pair $(M^*, (\tau^*, \alpha^*, \gamma^*, \pi^*))$ for which VesVf outputs 1. Furthermore, we let σ' be the result of the adjudication algorithm for $(\tau^*, \beta^*, \gamma^*, \pi^*)$.

Towards contradiction, assume that extraction fails, i.e., $\mathsf{DSig.Vf}(\mathsf{spk}, \sigma', M^*) = 0$. From VesVf, we know that ρ^* was previously created by the simulator together with a signature σ_ρ^*, using the external signature oracle. Otherwise, we would have an existential forgery that refutes unforgeability of DSig. Assume that ρ^* was formed using $\{\alpha_i\}_1^\ell, \{\beta_i\}_1^\ell, \{\gamma_i\}_1^\ell$.

VesVf guarantees that π^* is an authentication path for the leaf $\mathsf{G}(\mathsf{G}(\beta^*)||\mathsf{G}(\gamma^*))$ w.r.t. ρ. Thus, there is an index $i \in \{1, \ldots, \ell\}$ such that $\beta^* = \beta_i = \mathsf{Advice}(\mathsf{spk}, \alpha_i)$ and $\gamma^* = \gamma_i$. Otherwise, we would have at least one collision in the hash tree, which refutes collision resistance of G.

Finally, VesVf ensures that $\mathsf{MS}_{\mathsf{DSig}}.\mathsf{Vf}(\mathsf{spk}, \tau^*, \beta^*, M^*) = 1$, which implies the contradiction $\mathsf{DSig.Vf}(\mathsf{spk}, \sigma', M^*) = 1$ because of the *validity* of $\mathsf{MS}_{\mathsf{DSig}}$.   □

In order to prove unforgeability, we need to observe that both constructions apply signature and encryption keys separately because $\mathsf{DSig.Sign}$ is called as a black box and the result is encrypted, or masked in our context. More precisely, they satisfy the following definition of key-independence.

Definition 2 (Key-Independence [22]). *Let the signer's private key* sk *consist of two independent elements* $(\mathsf{kisk}, \mathsf{ssk})$ *and let* $\mathsf{pk} = (\mathsf{kipk}, \mathsf{spk})$ *be the corresponding public key.* VES *is key-independent if there is an efficient encryption algorithm* $\mathsf{KI\text{-}Enc}$ *such that* $\mathsf{KI\text{-}Enc}(\mathsf{apk}, \mathsf{kipk}, \mathsf{kisk}, \mathsf{DSig.Sign}(\mathsf{ssk}, M), M) \equiv \mathsf{VES.Create}(\mathsf{sk}, \mathsf{apk}, M)$ *for all* $M \in \mathcal{M}$, *where* DSig *is employed signature.*

Lemma 1 ([22]). *Let* VES *be extractable and key-independent.* VES *is unforgeable if the underlying signature scheme* DSig *is unforgeable.*

Regarding our novel security requirement against collusion of the adjudicator and a user, we prove the following useful lemma.

Lemma 2. *Let* VES *be extractable and key-independent.* VES *is collusion-resistant if the underlying signature scheme* DSig *is unforgeable.*

Proof. Suppose that there exists an adversary $\mathcal{A}$ that successfully breaks collusion-resistance with non-negligible probability $\epsilon(n)$ after at most q queries to the oracle $\mathfrak{C}$. We show how to forge ordinary signatures in DSig, running $\mathcal{A}$ as a black-box, with q queries to the signature oracle. The reduction $\mathcal{B}$, playing against unforgeability of DSig, receives a public verification key spk and has access to a signing oracle DSig.Sign(ssk, $\cdot$). It generates an adjudication key pair (ask, apk) $\leftarrow$ VES.AdjKg(1^n) and runs the remaining part of VES.Kg, including AdjSetup, to obtain a VES key pair (sk, pk). This is possible because VES is key-independent. Afterwards, $\mathcal{B}$ sets state $\leftarrow$ (ask, apk, pk) and runs $\mathcal{A}$(state) as a black-box. Whenever $\mathcal{A}$ queries M to $\mathfrak{C}$, $\mathcal{B}$ calls its external signing oracle $\sigma \leftarrow$ Sign(ssk, M) and computes $\varpi \leftarrow$ KI-Enc(apk, kipk, kisk, σ, M). Finally, $\mathcal{A}$ stops and outputs (M^*, ϖ^*). $\mathcal{B}$ extracts the corresponding signature $\sigma^* \leftarrow$ VES.Adj(ask, apk, pk, ϖ^*, M^*) and returns (M^*, σ^*). Observe that the environment of $\mathcal{A}$ is perfectly simulated and all oracle queries are simulated efficiently. By definition, $\mathcal{A}$ has not queried M^* to $\mathfrak{C}$. Thus, $\mathcal{B}$ has not queried M^* to its signature oracle. Moreover, the resulting (M^*, ϖ^*) yields an ordinary message-signature pair (M^*, σ^*) because VES is extractable. As a consequence, $\mathcal{B}$'s attack is legitimate and it succeeds with probability $\epsilon(n)$ after q queries to the signature oracle.   □

Since VES_1 and VES_2 are extractable and key-independent, unforgeability follows from Lemma 1, and Lemma 2 guarantees collusion-resistance.

Corollary 1 (Unforgeability). *If* DSig *is unforgeable and* VES_1 *(*VES_2*) is extractable and key-independent, then* VES_1 *(*VES_2*) is unforgeable.*

Corollary 2 (Collusion-resistance). *If* DSig *is unforgeable and* VES_1 *(*VES_2*) is extractable and key-independent, then* VES_1 *(*VES_2*) is collusion-resistant.*

Concerning Opacity, we show the following:

Theorem 2 (Opacity). *If* DSig *is unforgeable,* PKE *is RPA secure,* $\mathsf{MS}_{\mathsf{DSig}}$ *is hiding, and* G *is collision resistant then* VES_1 *(*VES_2*) is opaque.*

Proof. An adversary breaking opacity can succeed in two different ways. First, by forging the underlying signature scheme, and second, by decrypting a given verifiably encrypted signature. We say that an algorithm $\mathcal{A}$ is a

1. type-1 adversary ($\mathcal{A}_1$), if it outputs a message-signature pair (M^*, σ^*) such that it *has never queried* M^* to $\mathfrak{C}$, or if it *invokes* $\mathfrak{A}$ on M' without having queried M' to $\mathfrak{C}$ before.
2. type-2 adversary ($\mathcal{A}_2$), if it outputs a message-signature pair (M^*, σ^*) such that it *has queried* M^* to $\mathfrak{C}$ and it *has never invoked* $\mathfrak{A}$ on M' without having queried M' to $\mathfrak{C}$ before.

$\mathcal{A}_1$ can be directly used to forge signatures in DSig. The reduction has control over the adjudicator's private key and can therefore extract ordinary signatures (forgeries) from $\mathcal{A}_1$'s output. We refer the reader to the full version.

Type-2 Attacker. We perform a change to the simulation of $\mathcal{A}_2$'s environment and argue that each does not change $\mathcal{A}_2$'s success probability but for a negligible amount. Let ϵ be $\mathcal{A}_2$'s success probability in the (unmodified) opacity experiment. We change the algorithm AdjSetup.

Adjudication Setup: The algorithm AdjSetup$'$ selects the elements α_i, β_i as before and chooses a random index $c^* \leftarrow_\$ \{0, \dots, \ell\}$. It computes all $\gamma_{i \neq c^*}$ as before but $\gamma_{c^*} \leftarrow \mathsf{Enc}(\mathsf{apk}, 0^n)$. It outputs the corresponding tree, root ρ, and signature σ_ρ as before.

Due to the RPA security of the encryption scheme, this only changes $\mathcal{A}_2$'s success probability by a negligible ϵ'. The next change to AdjSetup allows us the reduction to use $\mathcal{A}_2$ to refute the hiding property of $\mathsf{MS_{DSig}}$.

Adjudication Setup: The algorithm AdjSetup$''$ works like AdjSetup$'$, but receives β_{c^*} from Recover and embeds it into the the leaf at index c^*.

The success probability of $\mathcal{A}_2$ does not change because β_{c^*} is distributed as before. Also, knowledge of α_{c^*} is not necessary to build the modified public key.

The remaining oracles, $\mathfrak{C}$ and $\mathfrak{A}$, are perfectly simulated for all indices $\neq c^*$ because the reduction has access to all masking values (except α_{c^*}) and can therefore answer all adjudication queries. In particular, this is the reason why we do not require some form of CCA secure encryption: all plaintexts are known and authenticated. Also, using these masking values together with the signature oracle in the Recover experiment, enables the reduction to answer queries to $\mathfrak{C}$.

Eventually, $\mathcal{A}_2$ outputs a message-signature pair (M^*, σ^*). If it is valid for the index c^*, the reduction outputs σ^* to refute the hiding property. Otherwise, it aborts. The reduction's success probability is noticeable if ϵ is noticeable.   □

4   Efficient Instantiations

We show that the assumptions in our generic constructions are sound and that *maskability* does not overly restrict the choice of signature schemes. We aim at providing VES schemes based on a broad range of cryptographic assumptions, including post-quantum ones. Here, we present the first efficient *pairing-free* VES in the *standard model*. The full version contains a instantiations of Construction 1 from lattices and RSA in the random oracle model.

4.1   An Instantiation Based on Worst-Case Lattice Problems in Ideal Lattices (Construction 2)

We propose an instantiation based on the hardness of lattice problems for Construction 2 in the *standard model*. The impact of this instantiation is significant. Not only are lattice-based constructions immune to quantum computer

attacks, but they are also desirable in the classic scenario. Computations in lattices are efficient (mostly basic linear algebra) and cryptographic hardness can be based on worst-case assumptions by Ajtai's worst-case to average-case reduction [1].

Lattices. A full-rank lattice in $\mathbb{R}^m$ is a set $\Lambda = \{\sum_{i=1}^m x_i \mathbf{b}_i \mid x_i \in \mathbb{Z}\}$, where $\mathbf{b}_1, \ldots, \mathbf{b}_m$ are linearly independent over $\mathbb{R}$. The matrix $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_m]$ is a *basis* of the lattice Λ and we write $\Lambda = \Lambda(\mathbf{B})$. The number of linearly independent vectors in the basis is the dimension of the lattice. *Ideal lattices* are a special class of lattices. Given a monic, irreducible (over $\mathbb{Z}$) polynomial f of degree n, the ring $R = \mathbb{Z}_p[x]/\langle f \rangle \cong \mathbb{Z}_p^n$, and an ideal I of R, the ideal lattice Λ is the set of coefficient vectors of the polynomials in I. In other words, $\Lambda = \{\mathbf{a} \in \mathbb{Z}^n : \sum_{i=0}^{n-1} a_i x^i \in I\}$. The main computational problem in lattices is the approximate shortest vector problem (SVP^∞), where an algorithm is given a basis of a lattice Λ and is supposed to find a sufficiently short vector $\mathbf{v} \in \Lambda \setminus \{\mathbf{0}\}$ with respect to the ℓ_∞ norm. The SVP^∞ in ideal lattices is called ISVP^∞. We write $\mathsf{ISVP}^\infty(f, \nu)$ for the problem of finding a vector $\mathbf{v}$ with $\|\mathbf{v}\|_\infty \leq \nu\lambda_1$ (λ_1 is the minimum distance in the lattice) in *all* lattices corresponding to ideals in the ring R.

$\mathcal{H}_{R,m}$ is the family of compression functions that map elements from R^m to R. Functions $\mathsf{h} \in \mathcal{H}_{R,m}$ are module homomorphisms, especially $\mathsf{h}(a + b) = \mathsf{h}(a) + \mathsf{h}(b)$. The problem $Col(\mathsf{h}, d)$ asks to find two distinct $\mathbf{x}$ and $\mathbf{x}'$ with $\max\{\|\mathbf{x}\|_\infty, \|\mathbf{x}'\|_\infty\} \leq d$ such that $\mathsf{h}(\mathbf{x}) = \mathsf{h}(\mathbf{x}')$. A polynomial time algorithm that solves $Col(\mathsf{h}, d)$ for $d = 10\phi p^{1/m} n \log^2(n)$ can be used to solve $\mathsf{ISVP}^\infty(f, \nu)$, where $\nu = \tilde{\mathcal{O}}(\phi^5 n^2)$. Here, ϕ is a small ring constant defined in [15].

LM-OTS signatures. We use the one-time signature scheme of Lyubashevsky and Micciancio [15]. Its security is based on the problem of finding short vectors in ideal lattices, which is conjectured to be intractable by quantum computers.

Let $p = \tilde{\Theta}((\phi n)^4)$, $m = \lceil \log n \rceil$. Define the constant $D = 10\phi p^{1/m} n \log^2(n)$. The LM-OTS scheme is a tuple $(\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vf})$, defined via *Key Generation:* $\mathsf{Kg}(1^n)$ outputs a signing key $(\mathbf{k}, \mathbf{l}) \in R^m \times R^m$ with $\|\mathbf{k}\|_\infty$ and $\|\mathbf{l}\|_\infty$ bounded. The verification key is $(\mathsf{h}, \mathsf{h}(\mathbf{k}), \mathsf{h}(\mathbf{l}))$, where $\mathsf{h} \leftarrow_\$ \mathcal{H}_{R,m}$; *Signing:* $\mathsf{Sign}((\mathbf{k}, \mathbf{l}), M)$, $M \in R$, $\|M\|_\infty \leq 1$, returns $\sigma = \mathbf{k}M + \mathbf{l} \in R_D^m$, where R_D^m restricts the coefficients in R to the range $[-D, D]$; *Verification:* $\mathsf{Vf}((\mathsf{h}, \mathsf{h}(\mathbf{k}), \mathsf{h}(\mathbf{l})), \sigma, M)$ returns 1 iff $\sigma \in R_{2\psi mnD-D}^m$ and $\mathsf{h}(\sigma) = \mathsf{h}(\mathbf{k})M + \mathsf{h}(\mathbf{l})$, for some constant ψ. The above scheme is a slight modification compared to [15] with regard to the admissible signature length. Observe that honestly generated signatures are in R_D^m, whereas signatures in $R_{2\psi mnD-D}^m$ may still be valid. The scheme remains secure under the stronger assumption that $Col(\mathsf{h}, 2\psi mnD - D)$, instead of $Col(\mathsf{h}, D)$, is hard. This change is required for the masking scheme to be hiding. If there is a successful adversary against unforgeability of this modified signature scheme, then one can find a collision $(\mathbf{x}, \mathbf{x}')$ under h such that $\max\{\|\mathbf{x}\|_\infty, \|\mathbf{x}'\|_\infty\} \leq D$. This can be used to solve ISVP^∞ in the worst case.

Instantiation. Let $\psi \in \mathbb{N}_{>0}$ be a small constant. The following table summarizes the instantiation using Construction 2 and the masking scheme $\mathsf{MS}_{LM\text{-}OTS}$.

pk	Σ	$\mathcal{T}$	$\mathcal{S}$	$\mathcal{V}$	Advice	Mask	Mask.Vf	Unmask
$(h, h(\mathbf{k}), h(\mathbf{l}))$	$R_{2\psi mnD-D}^m$	$R_{\psi mnD-D}^m$	$R_{\psi mnD}^m$	R	$h(\alpha)$	$\sigma + \alpha$	$h(\tau) \equiv h(\mathbf{k})M + h(\mathbf{l}) + \beta,\ \tau \in \mathcal{T}$	$\tau - \alpha$

The masking distribution Δ is component-wise uniform. Signatures σ are masked via $\tau \leftarrow \sigma + \alpha$. Therefore, the verification function is easy to adapt because h is a module homomorphism. Notice that $\tau \leftarrow \sigma + \alpha$ may lie outside $R_{2\psi mnD-D}^m$. In this case, Mask returns the special symbol $\perp$ and Mask.Vf fails. Fortunately, we have $\sigma + \alpha \in R_{2\psi mnD-D}^m$ with probability $\approx e^{-1/\psi}$ (a generalization of Lemma 5.1 in [14]). There are two ways to deal with this completeness error. The first is to prepare $\omega(\log(n))$ many masking values for each leaf of the tree for a negligible error. However, this would waste time and space as a negligible completeness error is not necessary. By a simple trial-and-error approach, we may to discard some of the α's in VES.Create and move on to the next leaf. In practice, the number of failures is small. Choosing $\psi = 2$, for example, yields a success probability > 0.6 and it can be brought arbitrarily close to 1 by allowing larger ψ. Thus, ψ allows a tradeoff between completeness and size/security.

The following propositions show that $\mathsf{MS}_{LM\text{-}OTS}$ is applicable.

Proposition 5. $\mathsf{MS}_{LM\text{-}OTS}$ *supports validity.*

Proof. Let τ be a masked signature for M with advice $\beta = h(\alpha)$. If τ is valid under Mask.Vf, then $h(\tau) = h(\mathbf{k})M + h(\mathbf{l}) + \beta$. But if this equation holds, then it implies that $h(\tau - \alpha) = h(\mathbf{k})M + h(\mathbf{l})$. Observe that $\tau - \alpha \in R_{2\psi mnD-D}^m$ because $\|\tau - \alpha\|_\infty \le \|\tau\|_\infty + \|\alpha\|_\infty \le 2\psi mnD - D$. Thus, we obtain a signature $\sigma = \tau - \alpha$ that passes $LM\text{-}OTS$.Vf for M as required. $\square$

Proposition 6. $\mathsf{MS}_{LM\text{-}OTS}$ *is hiding if* $Col(h, 2\psi mnD - D)$ *is hard.*

Proof. The proof is done in two steps. First, we show that two alternative LM-OTS signatures $\sigma \ne \sigma'$ for a given message M, which always exist because $|R_D^m| \gg |R|$ makes h compressing, are indistinguishable when masked according to the above rejection procedure. Second, we show that a successful attacker that wins in the Recover experiment can be used to find a collision under h.

The first part is showing that the statistical distance $\mathsf{SD}(\sigma + \alpha, \sigma' + \alpha)$, conditioned on $\sigma + \alpha, \sigma' + \alpha \in \mathcal{T}$, is zero over the choice of α. Notice that $\mathsf{SD}(\sigma + \alpha, \sigma' + \alpha) = 1/2 \sum_{t \in \mathcal{T}} |\operatorname{Prob}[\alpha = t - \sigma] - \operatorname{Prob}[\alpha = t - \sigma']| = 1/2 \sum_{t \in \mathcal{T}} |\prod_{i=1}^{mn} \operatorname{Prob}[\alpha_i = t_i - \sigma_i] - \prod_{i=1}^{mn} \operatorname{Prob}[\alpha_i = t_i - \sigma_i']|$. The index i specifies a coefficient of the vectors in $R^m \cong \mathbb{Z}_q^{mn}$. Now, $\operatorname{Prob}[\alpha_i = t_i - \sigma_i] = 1/|R_{\psi mnD}^m|$ because $t_i - \sigma_i \in \mathcal{S}$ and $\alpha \leftarrow_\$ \mathcal{S}$. Thus, the distance is zero. The second part is a reduction from the collision problem. It chooses its own signature key with public key $(h(\mathbf{k}), h(\mathbf{l}))$ to simulate the oracle $\mathfrak{M}$, not the additional signature oracle because LM-OTS is one-time. On input M^*, it masks a signature $\sigma \in R_D^m$. The adversary's output will be $\sigma^* \in R_{2\psi mnD-D}^m$ with $h(\sigma^*) = h(\mathbf{k})M^* + h(\mathbf{l}) = h(\sigma)$. Since two alternative signatures are indistinguishable when masked, we have $\sigma^* \ne \sigma$ with probability $1/2$ and we obtain the desired collision. $\square$

Encryption. One could use ring-LWE [17] for encryption and furthermore use SWIFFT [16] for collision resistant hashing. This would base security entirely on worst-case ideal lattice problems.

5   Conclusions

With our work, we have extended the model of Boneh et al. by allowing an initial setup phase that is common in real-world scenarios. Moreover, we have proposed two novel generic constructions for verifiably encrypted signatures. Both rely on a certain class of signature schemes, a weaker-than-CPA secure encryption scheme, and a collision-resistant hash function. Both work without NIZKs or random oracles. To demonstrate their feasibility, we have instantiated them with a range of primitives, including post-quantum ones.

Acknowledgments

References

1. Ajtai, M.: Generating hard instances of lattice problems (extended abstract). In: STOC, pp. 99–108 (1996)
2. Asokan, N., Shoup, V., Waidner, M.: Optimistic fair exchange of digital signatures. IEEE Journal on Selected Areas in Communications 18(4), 593–610 (2000)
3. Ateniese, G.: Verifiable encryption of digital signatures and applications. ACM Trans. Inf. Syst. Secur. 7(1), 1–20 (2004)
4. Bao, F., Deng, R.H., Mao, W.: Efficient and practical fair exchange protocols with off-line ttp. In: IEEE Symposium on Security and Privacy, pp. 77–85. IEEE Computer Society, Los Alamitos (1998)
5. Bernstein, D.J., Buchmann, J.A., Dahmen, E. (eds.): Post-Quantum Cryptography. Springer, Heidelberg (2008)
6. Biham, E. (ed.): EUROCRYPT 2003. LNCS, vol. 2656. Springer, Heidelberg (2003)
7. Boneh, D.: A brief look at pairings based cryptography. In: FOCS, pp. 19–26. IEEE Computer Society, Los Alamitos (2007)
8. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In: Biham [6], pp. 416–432
9. Camenisch, J., Shoup, V.: Practical verifiable encryption and decryption of discrete logarithms. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 126–144. Springer, Heidelberg (2003)
10. Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited. J. ACM 51(4), 557–594 (2004)
11. Dodis, Y., Lee, P.J., Yum, D.H.: Optimistic fair exchange in a multi-user setting. In: Okamoto, Wang [19], pp. 118–133
12. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM J. Comput. 17(2), 281–308 (1988)
13. Lu, S., Ostrovsky, R., Sahai, A., Shacham, H., Waters, B.: Sequential aggregate signatures and multisignatures without random oracles. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 465–485. Springer, Heidelberg (2006)
14. Lyubashevsky, V.: Towards Practical Lattice-Based Cryptography. PhD thesis (2008)

15. Lyubashevsky, V., Micciancio, D.: Asymptotically efficient lattice-based digital signatures. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 37–54. Springer, Heidelberg (2008)
16. Lyubashevsky, V., Micciancio, D., Peikert, C., Rosen, A.: Swifft: A modest proposal for fft hashing. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 54–72. Springer, Heidelberg (2008)
17. Lyubashevsky, V., Regev, O., Peikert, C.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110. Springer, Heidelberg (2010)
18. Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 218–238. Springer, Heidelberg (1990)
19. Okamoto, T., Wang, X. (eds.): PKC 2007. LNCS, vol. 4450. Springer, Heidelberg (2007)
20. Rückert, M.: Verifiably encrypted signatures from RSA without NIZKs. In: Roy, B., Sendrier, N. (eds.) INDOCRYPT 2009. LNCS, vol. 5922, pp. 363–377. Springer, Heidelberg (2009)
21. Rückert, M., Schneider, M., Schröder, D.: Generic constructions for verifiably encrypted signatures without random oracles or NIZKs. Cryptology ePrint Archive, Report 2010/200 (2010), http://eprint.iacr.org/
22. Rückert, M., Schröder, D.: Security of verifiably encrypted signatures and a construction without random oracles. In: Shacham, H. (ed.) Pairing 2009. LNCS, vol. 5671, pp. 17–34. Springer, Heidelberg (2009)
23. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. 26(5), 1484–1509 (1997)
24. Zhang, F., Safavi-Naini, R., Susilo, W.: Efficient verifiably encrypted signature and partially blind signature from bilinear pairings. In: Johansson, T., Maitra, S. (eds.) INDOCRYPT 2003. LNCS, vol. 2904, pp. 191–204. Springer, Heidelberg (2003)