Deterministic Wallets in a Quantum World

Nabil Alkeilani Alkadri[1], Poulami Das[2], Andreas Erwig[2], Sebastian Faust[2], Juliane Krämer[3], Siavash Riahi[2], and Patrick Struck[3]

[1] CDC, Technische Universität Darmstadt, Germany
`nabil.alkadri@tu-darmstadt.de`
[2] CAC, Technische Universität Darmstadt, Germany
`{poulami.das,andreas.erwig,sebastian.faust,siavash.riahi}@tu-darmstadt.de`
[3] QPC, Technische Universität Darmstadt, Germany
`{juliane,patrick}@qpc.tu-darmstadt.de`

Abstract. Most blockchain solutions are susceptible to quantum attackers as they rely on cryptography that is known to be insecure in the presence of quantum adversaries. In this work we advance the study of quantum-resistant blockchain solutions by giving a quantum-resistant construction of a deterministic wallet scheme. Deterministic wallets are frequently used in practice in order to secure funds by storing the sensitive secret key on a so-called *cold wallet* that is not connected to the Internet. Recently, Das et al. (CCS'19) developed a formal model for the security analysis of deterministic wallets and proposed a generic construction from certain types of signature schemes that exhibit key rerandomization properties. We revisit the proposed classical construction in the presence of quantum adversaries and obtain the following results.

First, we give a generic wallet construction with security in the quantum random oracle model (QROM) if the underlying signature scheme is secure in the QROM. We next design the first post-quantum secure signature scheme with rerandomizable public keys by giving a construction from generic lattice-based Fiat-Shamir signature schemes. Finally, we show and evaluate the practicality by analyzing an instantiation of the wallet scheme based on the signature scheme qTESLA (ACNS'20).

Keywords: blockchain protocols · deterministic wallets · post-quantum · rerandomizable signatures · provable security · lattice-based cryptography

1 Introduction

In the past decade cryptocurrencies such as Ethereum [eth15] and Bitcoin [Nak09] have gained huge popularity introducing a revolutionary payment paradigm. Cryptocurrencies do not rely on any central authority (i.e., banks) for the validation of transactions but instead use a consensus protocol to reach agreement on the validity of transactions in a decentralized network. As the name suggests the security of cryptocurrencies heavily relies on cryptographic building blocks – most importantly, on digital signature schemes. Digital signatures are used to authenticate money transfers between parties, where each party is identified by a public key with respect to the signature scheme. In a nutshell, a transfer of v coins from sender pk_{S} to receiver pk_{R} is represented by a transaction $\mathsf{tx} := (pk_{\mathrm{S}}, pk_{\mathrm{R}}, v)$. The transaction tx is then sent together with a signature of tx with respect to the sender's public key pk_{S} to the network of miners who validate the transaction. Besides digital signatures many other (partially advanced) cryptographic building blocks are used by cryptocurrencies to achieve a variety of goals. This includes, for instance, non-interactive zero-knowledge proofs and ring signatures for privacy preserving transactions [Noe15, EZS+19], threshold signatures and deterministic wallets for securing funds [DFL19], aggregate signatures for scalability [HMW18], and many more [GS15, FTS+19, TVR16].

Unfortunately, most cryptographic primitives used by cryptocurrencies today can be broken by quantum adversaries. Most notably, the ECDSA signature scheme that is implemented by nearly all popular cryptocurrencies relies on the hardness of computing discrete logarithms, and hence can be broken by Shor's algorithm [Sho94]. Since quantum computers can have devastating consequences for the security of cryptocurrencies [ABL+18], several recent works design cryptocurrencies with post-quantum security features, i.e., they resist both classical and quantum attacks, but run on classical machines. Cryptocurrency projects such as "Bitcoin Post Quantum" [Bit] or QRL [QRL] replace ECDSA with hash-based post-quantum signatures. Other examples include a Monero-based cryptocurrency with privacy guarantees that hold against quantum adversaries [EZS+19], or a security analysis of the proof of work consensus protocol in the quantum random oracle model [CGK+19]. In this work, we follow this line of work and investigate the post-quantum security of deterministic wallet schemes, and propose the first construction that provably resists quantum adversaries.

Deterministic wallets. In cryptocurrencies, secret keys are a particular attractive target for attackers. Indeed, the most devastating attacks in the cryptocurrency space have typically targeted secret keys of users resulting in billions of dollars worth of cryptocurrency being stolen [Ske18, Blo18, Bit18]. To protect keys against theft, one of the most prominent solutions is the concept of a *deterministic wallet*. A deterministic wallet scheme consists of two components: a hot wallet that is permanently connected to the Internet, and a cold wallet, which comes online only rarely (e.g., when a large amount of money has to be transferred). Das et al. [DFL19] formalized the concept of deterministic wallets and defined its security goals. The first security goal is *wallet unforgeability* which states that funds sent to the cold wallet must remain secure even if the hot wallet is corrupted. Second, *wallet unlinkability*, which guarantees that individual transactions that sent money to the same wallet are unlinkable despite being publicly available on the blockchain.

At a high-level a hot/cold wallet scheme works as follows. In an initialization phase, it generates a master key pair (msk, mpk), where the master secret key msk is stored on the cold wallet, while the hot wallet keeps the corresponding master public key mpk. The main ingredient of a deterministic wallet scheme is a deterministic key derivation procedure, which allows both, cold and hot wallet, to derive matching secret and public session keys without interacting with each other. To this end, in addition to the master secret/public key, the hot and cold wallet share a state St. From this state each wallet can derive the corresponding session key by combining the master key with a deterministically derived value $\mathsf{H}(St, ID)$, where ID is an arbitrary key identifier and H is a cryptographic hash function. More concretely, consider a simplified version of the BIP32 deterministic wallet scheme [BIP17] used for Bitcoin. The master secret/public key pair consists of a valid ECDSA key pair $(msk, mpk) := (x, x \cdot G)$, where G is a generator of the ECDSA elliptic curve. The session key pair for identity ID is computed as $pk_{ID} := mpk + w \cdot G$ and $sk := msk + w$ with $w := \mathsf{H}(St, ID)$.

Post-quantum security of deterministic wallets. While deterministic wallet schemes offer an elegant solution to increase the security of users' funds, they are particularly susceptible to attacks by quantum adversaries. To illustrate this, let us first consider how quantum attacks against the underlying signature scheme of a cryptocurrency such as Bitcoin work. Recall that in Bitcoin (as in most other cryptocurrencies) an address for transferring funds to is not represented by the public key itself but by its hash value. More concretely, when a party transfers v coins to some receiver R with public key pk_{R}, then the transaction will store $h = \mathsf{H}(pk_{\mathrm{R}})$. Only when R wants to spend these coins he reveals pk_{R} together with a signature with respect to pk_{R}. This leaves a quantum adversary that wants to steal v coins from R, with two options: either he tries to find pk' such that $\mathsf{H}(pk') = h$, or he waits until pk_{R} is revealed by R and computes the corresponding secret key sk_{R}. The first type of attack is believed to be hard because common cryptographic hash functions such as SHA3-512 are known to be preimage resistant even under quantum attacks when appropriately choosing their parameters. While in the second case a quantum adversary can indeed efficiently attack the signature scheme, he has only a very small window of time to carry out this attack[4]. In particular, he has to frontrun the transaction published by R, which is unlikely, assuming that the majority of miners is following the protocol.

A quantum attacker can have more devastating consequences against a deterministic wallet scheme. More concretely, unlike for normal addresses (hashes of public keys), in a deterministic wallet all session keys are related, and in particular efficiently computable from (msk, mpk). Hence, if the adversary manages to learn mpk then he can recover the corresponding master secret key msk and from that recover *all* session secret keys. Hence, all the money that was ever transferred to the cold wallet is at stake.

1.1 Our Contributions

Our main contribution in this work is to give the first construction of a post-quantum secure deterministic wallet. Our scheme is intended to be used on classical computers, and to remain secure even in the presence of quantum adversaries. To achieve our goal we extend the security model of Das et al. [DFL19] to the quantum setting and prove that certain standard post-quantum secure signature schemes can be used to construct post-quantum secure wallets. Concretely, our contributions are as follows:

– We extend the security model for deterministic wallets introduced by Das et al. [DFL19] to the quantum world. In particular, we show that if the underlying signature scheme satisfying the property of honestly rerandomizable keys is post-quantum secure, then it can be used to build post-quantum secure deterministic wallets. We relax the notion of rerandomizable keys as given by [DFL19] to consider only rerandomization of public keys. Subsequently, we show that this relaxed notion is sufficient for the security of wallets, and hence we are able to prove post-quantum security based on this relaxed notion.
– We design the first post-quantum secure signature scheme with rerandomizable public keys. This is achieved by giving a generic construction from a Fiat-Shamir signature scheme based on lattice assumptions.
– We discuss optimizations of our post-quantum secure signature scheme with rerandomizable public keys and evaluate its feasibility for blockchains.

1.2 Our Techniques

Signature schemes with rerandomizable keys [FKM+16] are the main building block of the wallet scheme presented in [DFL19]. At a high-level besides the standard algorithms of a digital signature scheme for key generation, signing, and verification, a signature scheme with rerandomizable keys has two additional algorithms, namely `RSig.RandSK` and `RSig.RandPK`. These algorithms take as input the secret key sk, respectively public key pk and randomness ρ, and output fresh keys sk', respectively pk'. Moreover, the unforgeability property of the signature scheme must hold even if the adversary sees signatures that are generated using rerandomized secret keys.

We show that certain post-quantum secure signature schemes support rerandomization of keys and satisfy the security notion of unforgeability under honestly rerandomized keys in the quantum world. In [FKM+16] it was shown that Schnorr's signature scheme [Sch91] has rerandomizable keys with unforgeability in the random oracle model (ROM). This motivates to study post-quantum secure Schnorr-like signature schemes. More concretely, we investigate if lattice-based, Schnorr-like signature schemes can have rerandomizable keys with unforgeability in the quantum random oracle model

[4] In Bitcoin in most cases transactions are considered to be final after 60 minutes.

(QROM). Lattice-based schemes are particularly suitable for constructing post-quantum secure rerandomizable signature schemes because (a) lattice-based assumptions are conjectured to be secure under quantum computer attacks; and (b) unlike hash-based signature schemes, lattice-based schemes exhibit an algebraic structure, which enables rerandomization of keys.

The key pair (pk, sk) of such Schnorr-based lattice schemes consists of an instance of a hard lattice problem, where the secret key sk typically follows either the discrete Gaussian distribution or the uniform distribution over a small set. The first idea that comes to mind when rerandomizing keys in the lattice setting is the following: Given (pk, sk) and randomness ρ, sk is rerandomized additively by computing $sk' = sk + \rho$ (as carried out in [FKM+16] for Schnorr's scheme). In the lattice setting however, we must ensure that the sum sk' follows the correct distribution, e.g., the Gaussian or uniform distribution. If this is not the case, one can sample a new randomness from ρ in a deterministic way until a valid sk' is generated. Naturally, the same (correct) ρ must be used when rerandomizing pk. This approach satisfies (under a specified distribution of sk) the original definition of signature schemes with rerandomizable keys (see Definition 3), as the initially generated key pair and any rerandomization of it are identically distributed. However, this approach cannot be used for building hot/cold wallets, because the hot and cold wallet must agree on the correct ρ for each session key generation. This contradicts the main goal of using hot/cold wallets, which requires that the cold wallet stays off-line, and hence cannot frequently communicate with the hot wallet to synchronize on ρ. In Section 4.3 we give more details on this approach as well as others, and argue why they are not suitable in the wallet setting.

In this work we show that the key pair (pk, sk) can still be rerandomized additively in a way that fits to the setting of hot/cold wallets. The main observation that we exploit is that the sum of two Gaussians is also Gaussian distributed (see Lemma 3). Based on this observation, our approach works as follows. Let sk be Gaussian distributed. Given randomness ρ, sk is rerandomized additively by adding to sk a freshly Gaussian distributed secret key sk^*. The key sk^* is deterministically sampled using the randomness ρ, i.e., we use ρ as the randomness required in the Gaussian sampler algorithm. We obtain a rerandomized secret key that is Gaussian distributed, but with a slightly larger standard deviation than the one of the original secret key (cf. Lemma 3). Consequently, we can construct a signature scheme with rerandomizable keys, in which the distribution of rerandomized public keys is identical to the distribution of the original public key, while rerandomized secret keys follow a different distribution than a freshly sampled secret key. We formally define such relaxed notion in Section 2 and call it a *signature scheme with rerandomizable public keys*. We then show in Section 3 that this notion is sufficient for post-quantum secure wallets and present a lattice-based construction of such a scheme in Section 4 with a security proof in the QROM. Finally, we show in Section 5 that our construction can be instantiated with state-of-the-art lattice-based signature schemes such as qTESLA [ABB+20]. Hence, it can use their proposed parameters and enjoy their performance and efficiency.

We emphasize that the post-quantum security model considers the adversary to be quantum while the challenger - representing the honest user in a real-world application - remains classical. As a result, every oracle that is provided by the challenger can be accessed only classically, while oracles that can be accessed by the adversary directly can be accessed using quantum computing power, i.e., in superposition. This describes a threat model where an adversary can use its quantum power to locally access the random oracle, while he observes signatures created by a user on a classical machine. In our work, we consider this standard post-quantum security model since it entails that the cryptographic scheme is still used on classical computers. This is, in contrast to the (fully-) quantum setting, where the scheme itself is implemented on quantum computers as well. While this is a stronger security model, it is more of theoretical interest as it requires users to have access to quantum computers as well.

1.3 Related Work

The concept of hot/cold wallets is used in many cryptocurrencies in order to provide stronger security guarantees to its users. Various works have proposed formal security models for analyzing the security of wallet schemes. Gutoski and Stebila [GS15] have discussed flaws in the BIP32 construction [BIP17] and possible countermeasures. However, they do not consider the standard notion of unforgeability but rather a restricted model where the adversary can corrupt the cold wallet and recover secret keys. Other works worth mentioning are “privilege escalation attacks” by Fan et al. [FTS+19] which unfortunately lacks any formal security analysis, and the analysis of the Bitcoin Electrum wallet in the Dolev-Yao model by Turuani et al. [TVR16]. The latter considers cryptographic primitives (e.g., signature schemes and encryption schemes) as idealized objects, hence fails to capture potential vulnerabilities such as related key attacks which are relevant in case of hot/cold wallets.

As mentioned earlier, we closely follow the model introduced by Das et al. [DFL19], where the notion of a *stateful deterministic wallet* is introduced and two desirable security properties called *wallet unlinkability* and *wallet unforgeability* are considered. The first property ensures that the session public keys generated by $\texttt{SW.RandPK}$ are unlinkable to the master public key. This property is guaranteed as long as the hot wallet has not been corrupted. The second property ensures unforgeability of signatures signed by the secret keys of the cold wallet even when the hot wallet is corrupted.

According to [DFL19] a *stateful deterministic wallet* $\texttt{SW}$ consists of two components – a hot wallet and a cold wallet which share a common state St. $\texttt{SW}$ is given by a tuple of algorithms ($\texttt{SW.KGen}$, $\texttt{SW.RandSK}$, $\texttt{SW.RandPK}$, $\texttt{SW.Sign}$, $\texttt{SW.Verify}$), where the session public key and secret key derivation algorithms $\texttt{SW.RandPK}$ and $\texttt{SW.RandSK}$ are run respectively within the hot and cold wallet to deterministically derive matching session (public/secret) keys from the (public/secret) master keys. Unlike deterministic wallets in use (e.g., the BIP32 construction), the state St of the wallet scheme of [DFL19] is refreshed within the (hot/cold) wallets with each key derivation. This approach allows to show forward unlinkability, which intuitively means that even upon leakage of the state, all session keys derived *before* the state leakage remain unlinkable. The second security property – *wallet unforgeability* – is achieved by a reduction to standard $\mathsf{EUF\text{-}CMA}$ security of a concrete signature scheme (such as ECDSA and Schnorr) in a modularized fashion. As the intermediary step the authors show that these signature schemes satisfy the properties of signature schemes with rerandomizable keys.

We note that the model by Das et al. [DFL19] only considers adversaries in the classical setting and does not protect against quantum adversaries. Our work fills this gap by designing the first *post-quantum secure deterministic wallet.*

Many prior works have investigated lattice-based Fiat-Shamir signatures, e.g., [Lyu09, DDLL13, BG14, DKL+18, ABB+20], and in particular, their security was analyzed in the QROM, e.g., by [KLS18, Unr17, LZ19, DFMS19]. To the best of our knowledge we propose the first work on lattice-based signature schemes with honestly rerandomizable keys and prove its security in the QROM. Inspired by Das et al. [DFL19] and Fleischhacker et al. [FKM+16], we use the abstraction of signature schemes with rerandomizable keys but transfer this concept to the post-quantum setting.

As mentioned in the introduction, various works build blockchains with security features against quantum adversaries. Most recently, Esgin et al. [EZS+19] have proposed a new ring signature scheme based on lattice assumptions for the blochchain setting which focus on similar anonymity guarantees to Monero [Noe15]. In Monero-like cryptocurrencies the sender of a transaction can hide her identity in a set of transactions using ring signatures. In particular, the public key related to the sender's signature is never revealed explicitly in the blockchain network, hence remains unlinkable to the sender. We note that this notion of unlinkability is different from our notion of session key unlinkability.

Blockchain initiatives such as the "Bitcoin Post-Quantum" [Bit] and QRL [QRL] replace ECDSA with hash-based signature schemes which are post-quantum secure. Despite the hash-based schemes being quite efficient, the underlying hash function does not permit to construct a signature scheme with rerandomizable keys which plays a key role in our wallet scheme.

2 Preliminaries

2.1 Basic Notations

We let $\mathbb{N}, \mathbb{Z}, \mathbb{R}$ denote the set of natural numbers, integers, and real numbers, respectively. For any positive integer k we write $[k]$ to denote the set of integers $\{1, \ldots, k\}$. For a positive integer q we let $\mathbb{Z}_q$ denote the set of integers in the range $[-\frac{q}{2}, \frac{q}{2}) \cap \mathbb{Z}$. We define the ring $R = \mathbb{Z}[x]/\langle x^n + 1 \rangle$ and its quotient $R_q = R/qR$, where n is a power of 2. Elements in R and R_q (including $\mathbb{Z}$ and $\mathbb{Z}_q$) are denoted by regular font letters. Column vectors and matrices with entries from R or R_q are denoted by bold lower-case letters and bold upper-case letters, respectively. We define the ℓ_2 and ℓ_∞ norms of $v = \sum_{i=0}^{n-1} v_i x^i \in R$ by $\|v\| = (\sum_{i=0}^{n-1} |v_i|^2)^{1/2}$ and $\|v\|_\infty = \max_i |v_i|$, respectively. For $\mathbf{w} = (w_1, \ldots, w_k) \in R^k$ we define $\|\mathbf{w}\| = (\sum_{i=1}^{k} \|w_i\|^2)^{1/2}$ and $\|\mathbf{w}\|_\infty = \max_i \|w_i\|_\infty$. We let $\mathbb{T}_\kappa^n$ denote the set of all $(n-1)$-degree polynomials with coefficients from $\{-1, 0, 1\}$ and Hamming weight κ. We always denote the security parameter by $\lambda \in \mathbb{N}$, and $o(\lambda)$ denotes a linear function in λ. A function $f : \mathbb{N} \longrightarrow \mathbb{R}$ is called *negligible* if there exists an $n_0 \in \mathbb{N}$ such that for all $n > n_0$, it holds $f(n) < \frac{1}{p(n)}$ for any polynomial p. With $\mathrm{negl}(\lambda)$ we denote a negligible function in λ. A probability is called overwhelming if it is at least $1 - \mathrm{negl}(\lambda)$. The *statistical distance* between two distributions X, Y over a countable domain D is defined by $\frac{1}{2} \sum_{n \in D} |X(n) - Y(n)|$. We write $x \leftarrow D$ to denote that x is sampled according to a distribution D. We let $x \leftarrow_\$ S$ denote choosing x uniformly random from a finite set S. Unless specified otherwise, every adversary is considered to be an efficient quantum polynomial time algorithm.

2.2 Quantum Random Oracle Model

In this section, we recall the quantum random oracle model and existing results that we will use. Since quantum computation is only necessary in the proofs of these results, we do not provide information on quantum computation here, but refer to [NC11] for a detailed discussion on the topic.

In [BR93], Bellare and Rogaway introduced the *random oracle model* (ROM). In this model every party has access to an oracle implementing a random function. Upon being queried on some input x, the oracle answers with a random output y. Every further invocation on input x, even by other parties, results in the same y. In security proofs, one often models a hash function as a random oracle. Since hash functions are public, Boneh et al. [BDF+11] observed that the ROM is not appropriate in the post-quantum setting. In the real world an adversary equipped with a quantum computer is able to implement the hash function and evaluate it in superposition. Thus, Boneh et al. introduced the *quantum random oracle model* (QROM). In this model, parties with quantum computing power get access to the oracle $|\mathsf{H}\rangle$, where $|\mathsf{H}\rangle : |x, y\rangle \mapsto |x, y \oplus \mathsf{H}(x)\rangle$. In our proofs we will also consider reprogrammed random oracles. For a random oracle H we write $\mathsf{H}_{x \to y}$ for the random oracle that is reprogrammed on input x to y. Further on, we denote the classical random oracle by the symbol H and the quantum random oracle by the notation $|\mathsf{H}\rangle$.

Nowadays, the QROM is considered the de facto standard for post-quantum security proofs of cryptographic primitives which rely on random oracles. Below we describe some results for quantum random oracles that are required for our proofs.

The one-way to hiding (O2H) lemma [Unr15] is an important tool for security proofs in the quantum random oracle model. It gives bounds on the advantage of an adversary in distinguishing between different random oracles when the adversary is allowed to query them in superposition. Below we state the lemma using the reformulation by Ambainis et al. [AHU19].

Lemma 1 (One-way to hiding (O2H) [AHU19]). *Let* $\mathsf{G}, \mathsf{H}\colon \mathcal{X} \to \mathcal{Y}$ *be random functions, let* z *be a random value, and let* $\mathcal{S} \subset \mathcal{X}$ *be a random set such that* $\forall x \notin \mathcal{S}$, $\mathsf{G}(x) = \mathsf{H}(x)$. $(\mathsf{G}, \mathsf{H}, \mathcal{S}, z)$ *may have arbitrary joint distribution. Furthermore, let* $\mathcal{A}^{|\mathsf{H}\rangle}$ *be a quantum oracle algorithm which queries* $|\mathsf{H}\rangle$ *at most* q *times. Let* Ev *be an arbitrary classical event. Define an oracle algorithm* $\mathcal{B}^{|\mathsf{H}\rangle}$ *as follows: Pick* $i \leftarrow_{\$} [q]$. *Run* $\mathcal{A}^{|\mathsf{H}\rangle}(z)$ *until just before its* i*-th round of queries to* $|\mathsf{H}\rangle$. *Measure the query in the computational basis, and output the measurement outcome. It holds that*

$$\left|\Pr[\mathsf{Ev}\colon \mathcal{A}^{|\mathsf{H}\rangle}(z)] - \Pr[\mathsf{Ev}\colon \mathcal{A}^{|\mathsf{G}\rangle}(z)]\right| \leq 2q\sqrt{\Pr[x \in \mathcal{S}\colon \mathcal{B}^{|\mathsf{H}\rangle}(z) \Rightarrow x]}\,.$$

Another tool that we will use are Zhandry's small range distributions, defined below. These are distributions where the set of possible outputs is limited.

Definition 1 (Small-range distributions [Zha12a]). *Let* $\mathcal{X}$, $\mathcal{Y}$ *be sets,* r *be an integer,* D *be a distribution on* $\mathcal{Y}$, P *be a random function from* $\mathcal{X}$ *to* $[r]$, *and* $\boldsymbol{y} = (y_1, \ldots, y_r)$ *be* r *samples of* D. *Define a function* $\mathsf{H}\colon \mathcal{X} \to \mathcal{Y}$ *by* $\mathsf{H}(x) \mapsto y_{P(x)}$. *The distribution of* H, *induced by* P *and* $\boldsymbol{y}$, *is called a* small-range distribution *with* r *samples of* D.

The following lemma provides a bound on the distinguishing advantage between a random oracle and an oracle drawn from a small-range distribution when superposition access is granted.

Lemma 2 ([Zha12a]). *There is a universal constant* C *such that, for any set* $\mathcal{X}$ *and* $\mathcal{Y}$, *distribution* D *on* $\mathcal{Y}$, *integer* l, *and any quantum algorithm* $\mathcal{A}$ *making* q *queries to an oracle* $\mathsf{H}\colon \mathcal{X} \to \mathcal{Y}$, *the following two cases are indistinguishable, except with probability less than* $\frac{Cq^3}{l}$:

– $\mathsf{H}(x) = y_x$ *where* $\boldsymbol{y}$ *is a list of samples of* D *of size* $|\mathcal{X}|$.
– H *is drawn from the small-range distribution with* l *samples of* D.

2.3 Cryptographic Primitives

Definition 2 (Signature Scheme). *Let* λ *be a security parameter. A signature scheme* Sig *with key space* $\mathcal{K}$, *message space* $\mathcal{M}$, *and signature space* $\mathcal{S}$ *is a tuple of polynomial-time algorithms* $(\texttt{KGen}, \texttt{Sign}, \texttt{Verify})$ *such that*

$\texttt{KGen}(1^\lambda)$ *is the key generation algorithm that outputs a key pair* $(pk, sk) \in \mathcal{K}$, *where* pk *is a public key and* sk *is a secret key.*
$\texttt{Sign}(sk, m)$ *is the signing algorithm that takes as input a secret key* sk *and a message* $m \in \mathcal{M}$. *It outputs a signature* $\sigma \in \mathcal{S}$.
$\texttt{Verify}(pk, m, \sigma)$ *is the verification algorithm that takes as input a public key* pk, *a message* m *with a signature* σ. *It outputs* 1 *if* σ *is valid and* 0 *otherwise.*
Correctness: *A signature scheme is correct if for all* $\lambda \in \mathbb{N}$, $m \in \mathcal{M}$, $(pk, sk) \leftarrow \texttt{KGen}(1^\lambda)$, *and all* $\sigma \leftarrow \texttt{Sign}(sk, m)$, *it holds that* $\Pr[\texttt{Verify}(pk, m, \texttt{Sign}(sk, m)) = 1] \geq 1 - \mathrm{negl}(\lambda)$.

We will use the notion of signature schemes with rerandomizable keys [FKM+16]. In the following we recall its definition.

Definition 3 (Signature Scheme with Rerandomizable Keys). *A signature scheme with perfectly rerandomizable keys* $\mathtt{RSig}$ *is given by a tuple of algorithms:*

$$(\mathtt{RSig.KGen}, \mathtt{RSig.RandSK}, \mathtt{RSig.RandPK}, \mathtt{RSig.Sign}, \mathtt{RSig.Verify}),$$

where $\mathtt{RSig.KGen}$*,* $\mathtt{RSig.Sign}$*, and* $\mathtt{RSig.Verify}$ *satisfy the definition of a standard signature scheme (cf. Definition 2). For randomness space* $\mathcal{R}$*,* $(\mathtt{RSig.RandSK}, \mathtt{RSig.RandPK})$ *are two polynomial-time algorithms such that*

$\mathtt{RSig.RandSK}(sk, \rho)$ *is a secret key rerandomization algorithm that takes as input the secret key sk and a randomness* $\rho \in \mathcal{R}$ *and outputs a randomized secret key* sk'*.*

$\mathtt{RSig.RandPK}(pk, \rho)$ *is a public key rerandomization algorithm that takes as input the public key pk and a randomness* $\rho \in \mathcal{R}$ *and outputs a randomized public key* pk'*.*

$\mathtt{RSig}$ *satisfies the following properties:*

Rerandomizability of keys: *For all* $(sk, pk) \in \mathtt{RSig.KGen}(1^\lambda)$ *and all* $\rho \in \mathcal{R}$*, the distributions of* (sk', pk') *and* (sk'', pk'') *are identical, where* $(sk'', pk'') \leftarrow \mathtt{RSig.KGen}(1^\lambda)$ *and* $sk' \leftarrow \mathtt{RSig.RandSK}(sk, \rho)$*,* $pk' \leftarrow \mathtt{RSig.RandPK}(pk, \rho)$*.*

Correctness under rerandomizable keys:

1. *For all* $\lambda \in \mathbb{N}$*,* $(pk, sk) \leftarrow \mathtt{RSig.KGen}(1^\lambda)$*,* $m \in \mathcal{M}$*, and all* $\sigma \leftarrow \mathtt{RSig.Sign}(sk, m)$*, it holds that*

$$\Pr[\mathtt{RSig.Verify}(pk, m, \mathtt{RSig.Sign}(sk, m)) = 1] \geq 1 - \mathrm{negl}(\lambda)\,.$$

2. *For all* $(pk, sk) \leftarrow \mathtt{RSig.KGen}(1^\lambda)$*, all* $\rho \in \mathcal{R}$*,* $m \in \mathcal{M}$*, and for a pair of rerandomized keys* $sk' \leftarrow \mathtt{RSig.RandSK}(sk, \rho)$ *and* $pk' \leftarrow \mathtt{RSig.RandPK}(pk, \rho)$*, it holds*

$$\Pr[\mathtt{RSig.Verify}(pk', m, \mathtt{RSig.Sign}(sk', m)) = 1] \geq 1 - \mathrm{negl}(\lambda)\,.$$

We also consider a relaxed version of Definition 3. In the following definition we introduce the notion of *signature schemes under rerandomizable public keys*, where the property of rerandomizability of keys holds only for the generated public keys, but not in case of secret keys. We present a concrete instantiation of such a scheme in Section 4.

Definition 4 (Signature Scheme with Rerandomizable Public Keys). *A signature scheme with perfectly rerandomizable public keys* $\mathtt{RSig}'$ *is given by a tuple of algorithms* $(\mathtt{RSig}'.\mathtt{KGen}, \mathtt{RSig}'.\mathtt{RandSK}, \mathtt{RSig}'.\mathtt{RandPK}, \mathtt{RSig}'.\mathtt{Sign}, \mathtt{RSig}'.\mathtt{Verify})$*, which are defined as in Definition 3.* $\mathtt{RSig}'$ *satisfies the following properties:*

Rerandomizability of public keys: *For all public keys* $(\cdot, pk) \leftarrow \mathtt{RSig}'.\mathtt{KGen}(1^\lambda)$ *and* $\rho \in \mathcal{R}$*, the distributions of* pk' *and* pk'' *are identical, where* $pk' \leftarrow \mathtt{RSig}'.\mathtt{RandPK}(pk, \rho)$*, and* $pk'' \leftarrow \mathtt{RSig}'.\mathtt{KGen}(1^\lambda)$*.*

Correctness under rerandomizable keys: *This property is defined as the property of correctness for signature schemes under rerandomizable keys in Definition 3.*

2.4 Security Notions

Security of signature schemes is captured by the standard security notion of *existential unforgeability under adaptive chosen-message attacks* (EUF-CMA), presented below.

```
Game EUF-CMA$_\Sigma^\mathcal{A}$
1: $\mathcal{Q} := \varnothing$
2: $H := \varnothing$
3: $(pk, sk) \leftarrow \texttt{KGen}(1^\lambda)$
4: $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{H},\mathsf{O}}(pk)$
5: if $(m^* \in \mathcal{Q})$ then
6:     return 0
7: return $\texttt{Verify}(pk, m^*, \sigma^*)$
```

```
$\mathsf{H}(m')$
1: if $(m' \in H)$ then
2:     return $\mathsf{H}(m') \in H$
3: $\mathsf{H}(m') \leftarrow_\$ \{0,1\}^{o(\lambda)}$
4: $H := H \cup \{(m', \mathsf{H}(m'))\}$
5: return $\mathsf{H}(m')$
$\mathsf{O}(m)$
1: $\mathcal{Q} := \mathcal{Q} \cup \{m\}$
2: $\sigma \leftarrow \texttt{Sign}(sk, m)$
3: return $\sigma$
```

Fig. 1. The security game EUF-CMA of signature schemes.

Definition 5 (EUF-CMA Security). *Let $H : \{0,1\}^* \to \{0,1\}^{o(\lambda)}$ be a hash function modeled as (quantum) random oracle. A signature scheme Σ is called $(t, q_{\texttt{Sign}}, q_H, \varepsilon)$-EUF-CMA in the (quantum) random oracle model if for any adversary $\mathcal{A}$ running in time at most t and making at most $q_{\texttt{Sign}}$ signature queries and at most q_H (superposition) queries to H, the game EUF-CMA$_\Sigma^\mathcal{A}$ depicted in Figure 1 outputs 1 with probability at most ε, i.e.,* $\Pr[\textit{EUF-CMA}_\Sigma^\mathcal{A} = 1] \leq \varepsilon$.

In the following we present the notion of *EUF-CMA-HRK security under honestly rerandomizable keys* due to [DFL19]. This notion is similar to *EUF-CMA-RK security under rerandomizable keys* due to [FKM$^+$16], however with certain differences which makes it a weaker notion. In the EUF-CMA-RK game, an adversary $\mathcal{A}$ gets access to a signing oracle. The signing oracle takes a message and a randomness as input and provides a signature on this message under the rerandomized key as an answer. Note that the rerandomized key was derived from the randomness input by $\mathcal{A}$. This means that $\mathcal{A}$ can obtain signatures under keys with randomness of $\mathcal{A}$'s choice. $\mathcal{A}$ can win the EUF-CMA-RK game if it can produce a valid forgery under a rerandomized key of its choice (note that the randomness can also be null).

In the EUF-CMA-HRK game, we restrict $\mathcal{A}$'s capabilities in the following way. In addition to the signing oracle, in the EUF-CMA-HRK game $\mathcal{A}$ is given access to a Rand oracle to derive a fresh randomness. This randomness can be later used to get a signature under the rerandomized key by querying the signing oracle. Here, $\mathcal{A}$ can only win the EUF-CMA-HRK game if it produces a valid forgery under a rerandomized key, where the underlying randomness was obtained honestly by querying the Rand oracle. We formally present *EUF-CMA-HRK security under honestly rerandomizable (public) keys* below.

Definition 6 (EUF-CMA-HRK Security under Honestly Rerandomized (Public) Keys). *Let $H : \{0,1\}^* \to \{0,1\}^{o(\lambda)}$ be a hash function modeled as (quantum) random oracle. A signature scheme with honestly rerandomizable (public) keys* $\texttt{RSig}$ *is called $(t, q_{\texttt{Sign}}, q_H, \varepsilon)$-EUF-CMA-HRK in the (quantum) random oracle model if for any adversary $\mathcal{A}$ running in time at most t and making at most $q_{\texttt{Sign}}$ signature queries and at most q_H (quantum) random oracle queries to H, the game EUF-CMA-HRK$_{\texttt{RSig}}^\mathcal{A}$ depicted in Figure 2 outputs 1 with probability at most ε, i.e.,* $\Pr[\textit{EUF-CMA-HRK}_{\texttt{RSig}}^\mathcal{A} = 1] \leq \varepsilon$.

2.5 Lattice-Based Fiat-Shamir Signatures

In this section we review a generic construction of lattice-based Fiat-Shamir signatures. We first define the discrete Gaussian distribution and recall a lemma, which shows that the sum of Gaussian distributed random variables is also Gaussian distributed. This property is crucial for our analysis.

```
Game EUF-CMA-HRK$^{\mathcal{A}}_{\texttt{RSig}}$
 1: RList := ∅
 2: $\mathcal{Q}$ := ∅
 3: $H$ := ∅
 4: $(pk, sk) \leftarrow$ RSig.KGen$(1^\lambda)$
 5: $(m^*, \sigma^*, \rho^*) \leftarrow \mathcal{A}^{\mathsf{H},\mathsf{Rand},\mathsf{OHR}}(pk)$
 6: if $(m^* \in \mathcal{Q})$ then
 7:     return 0
 8: if $(\rho^* \neq$ NULL$)$ then
 9:     if $(\rho^* \notin$ RList$)$ then
10:         return 0
11:     $pk \leftarrow$ RSig.RandPK$(pk, \rho^*)$
12: return RSig.Verify$(pk, m^*, \sigma^*)$

H$(m')$ (see Figure 1)

Rand
 1: $\rho \leftarrow_\$ \mathcal{R}$
 2: RList $\leftarrow$ RList $\cup \{\rho\}$
 3: return $\rho$

OHR$(m, \rho)$
 1: $\mathcal{Q} := \mathcal{Q} \cup \{m\}$
 2: if $(\rho \neq$ NULL$)$ then
 3:     if $(\rho \notin$ RList$)$ then
 4:         return $\perp$
 5:     $sk \leftarrow$ RSig.RandSK$(sk, \rho)$
 6: $\sigma \leftarrow$ RSig.Sign$(sk, m)$
 7: return $\sigma$
```

Fig. 2. The security game EUF-CMA-HRK of signature schemes with rerandomizable (public) keys.

Definition 7 (Discrete Gaussian Distribution). *The discrete Gaussian distribution $D_{\mathbb{Z}^n,\sigma,\mathbf{c}}$ over $\mathbb{Z}^n$ with standard deviation $\sigma > 0$ and center $\mathbf{c} \in \mathbb{R}^n$ is defined as follows: For every $\mathbf{x} \in \mathbb{Z}^n$ the probability of $\mathbf{x}$ is given by $D_{\mathbb{Z}^n,\sigma,\mathbf{c}}(\mathbf{x}) = \rho_{\sigma,\mathbf{c}}(\mathbf{x})/\rho_{\sigma,\mathbf{c}}(\mathbb{Z}^n)$, where $\rho_{\sigma,\mathbf{c}}(\mathbf{x}) = \exp(\frac{-\|\mathbf{x}-\mathbf{c}\|^2}{2\sigma^2})$ and $\rho_{\sigma,\mathbf{c}}(\mathbb{Z}^n) = \sum_{\mathbf{x}\in\mathbb{Z}^n} \rho_{\sigma,\mathbf{c}}(\mathbf{x})$. The subscript $\mathbf{c}$ is taken to be $\mathbf{0}$ when omitted.*

Lemma 3 ([BF11, Theorem 9]). *Let $\mathcal{L} \subseteq \mathbb{Z}^m$ be a lattice and $\sigma \in \mathbb{R}$. For $i = 1, \ldots, n$ let $\mathbf{t}_i \in \mathbb{Z}^m$ and let X_i be mutually independent random variables sampled from $D_{\mathcal{L}+\mathbf{t}_i,\sigma}$. Let $\mathbf{c} = (c_1, \ldots, c_n) \in \mathbb{Z}^n$ and define $d = \gcd(c_1, \ldots, c_n)$, $\mathbf{t} = \sum_1^n c_i \mathbf{t}_i$. Suppose that $\sigma > \|\mathbf{c}\| \cdot \eta_\varepsilon(\mathcal{L})$, where $\eta_\varepsilon(\mathcal{L})$ is the smoothing parameter [MR04] for some negligible ε. Then $Z = \sum_1^n c_i X_i$ is statistically close to $D_{d\mathcal{L}+\mathbf{t},\|\mathbf{c}\|\sigma}$.*

Next, we describe two functions used in the signature scheme:

- $\mathsf{E} : \{0,1\}^* \longrightarrow \{0,1\}^*$ is a function that expands given strings to any desired length. It is used to extract the randomness used for signing.
- $\mathsf{H} : \{0,1\}^* \longrightarrow \mathbb{T}^n_\kappa$ is a hash function modeled as a (quantum) random oracle and used for signing and verification.

The signature scheme is formally described in Figure 3. It makes use of a uniformly random matrix $\mathbf{A} \in R_q^{k_1 \times k_2}$, which is publicly known and shared among all users in a multi-user setting. We assume that $\mathbf{A}$ is an implicit input to all algorithms of the scheme in addition to all algorithms in Section 4. In order to save bandwidth it can also be generated by expanding a uniformly random seed using the function E, and including the seed in the secret and public key rather than storing the whole matrix $\mathbf{A}$. In this case, E is modelled as a random oracle. We note that this setting makes sense in the context of blockchains, since the randomly chosen seed can be included in the first block known as the genesis block, which is assumed to be honestly generated. Furthermore, since $\mathbf{A}$ is computed as the output of the random oracle on input the seed, $\mathbf{A}$ is truly random and cannot have a trapdoor embedded as shown in [MP12].

Basically, the key generation algorithm generates an instance of a computationally hard lattice problem called *Module Learning with Errors* (MLWE) [LS15] (or a special variant of it such as Ring Learning with Errors (RLWE) [LPR10]). The secret of this instance is chosen from some distribution χ. In the state-of-the-art lattice-based signature schemes, e.g., Dilithium [DKL+18] and qTESLA [ABB+20], the distribution of the secrets is either the discrete Gaussian distribution $D_{\mathbb{Z}^n,\sigma}$ or the distribution R_d that outputs uniformly random polynomials from R whose ℓ_∞ norm is bounded by some integer $d \geq 1$.

```
$\boxed{\mathsf{LB}.\mathtt{KGen}(1^\lambda)}$
1: $\mathbf{s} \leftarrow \chi^{k_2}$, $\mathbf{e} \leftarrow \chi^{k_1}$
2: $\mathbf{b} \leftarrow \mathbf{A}\mathbf{s} + \mathbf{e} \pmod q$
3: $sk := (\mathbf{s}, \mathbf{e})$, $pk := \mathbf{b}$
4: return $(sk, pk)$

$\boxed{\mathsf{LB}.\mathtt{Verify}(pk, m, (\mathbf{z}_1, \mathbf{z}_2, c))}$
1: $\mathbf{w} \leftarrow \mathbf{A}\mathbf{z}_1 + \mathbf{z}_2 - \mathbf{b}c \pmod q$
2: if $\big((\mathbf{z}_1, \mathbf{z}_2) \in R_{B_1}^{k_2} \times R_{B_2}^{k_1} \wedge$
       $\mathsf{H}(\mathbf{w}, m) = c\big)$ then
3:     return 1
4: return 0

$\boxed{\mathsf{LB}.\mathtt{Sign}(sk, m)}$
1: $\mathbf{r} \leftarrow_{\$} \{0,1\}^{o(\lambda)}$
2: $\mathsf{ctr} \leftarrow 1$
3: $(\mathbf{y}_1, \mathbf{y}_2) \in R_Y^{k_2} \times R_Y^{k_1} \leftarrow \mathsf{E}(\mathbf{r}, \mathsf{ctr})$
4: $\mathbf{v} \leftarrow \mathbf{A}\mathbf{y}_1 + \mathbf{y}_2 \pmod q$
5: $c \leftarrow \mathsf{H}(\mathbf{v}, m)$
6: $\mathbf{z}_1 \leftarrow \mathbf{y}_1 + \mathbf{s}c$
7: $\mathbf{z}_2 \leftarrow \mathbf{y}_2 + \mathbf{e}c$
8: if $\big((\mathbf{z}_1, \mathbf{z}_2) \notin R_{B_1}^{k_2} \times R_{B_2}^{k_1}\big)$ then
9:     $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$
10:    goto 3
11: return $(\mathbf{z}_1, \mathbf{z}_2, c)$
```

Fig. 3. A formal description of a generic (non-optimized) Fiat-Shamir signature scheme from lattice assumptions.

A signature consists of a tuple $(\mathbf{z}_1, \mathbf{z}_2, c)$, where the pair $(\mathbf{z}_1, \mathbf{z}_2)$ is uniformly random over a subset of $R^{k_2} \times R^{k_1}$ and $c \in \mathbb{T}_\kappa^n$ is output from the random oracle H. The vectors $\mathbf{z}_1, \mathbf{z}_2$ are each generated by adding a masking term to a term related to the secret key and c. More precisely, we have $\mathbf{z}_1 = \mathbf{y}_1 + \mathbf{s}c$ and $\mathbf{z}_2 = \mathbf{y}_2 + \mathbf{e}c$, where the secret masking pair $(\mathbf{y}_1, \mathbf{y}_2)$ is uniformly random over $R_Y^{k_2} \times R_Y^{k_1}$ and $R_Y \subset R$ for some predefined positive integer Y. The signature is only output after verifying that the pair $(\mathbf{z}_1, \mathbf{z}_2)$ lies in $R_{B_1}^{k_2} \times R_{B_2}^{k_1}$, i.e., $\|\mathbf{z}_1\|_\infty \leq B_1$ and $\|\mathbf{z}_2\|_\infty \leq B_2$, where the bounds B_1, B_2 are defined depending on the distribution of the secret key. This ensures that signatures are uniformly distributed over $R_{B_1}^{k_2} \times R_{B_2}^{k_1} \times \mathbb{T}_\kappa^n$ and do not leak information about the secret key. If this is not the case, the algorithm restarts with a fresh masking pair $(\mathbf{y}_1, \mathbf{y}_2)$. The average number of repetitions is denoted by $M = O(1)$. Valid signatures are generated with probability $\left(\frac{2B_1+1}{2Y+1}\right)^{k_2 n} \cdot \left(\frac{2B_2+1}{2Y+1}\right)^{k_1 n}$, which is usually chosen such that it is at least $1/M$. We note that this generic construction can be optimized by either following the technique due to Bai and Galbraith [BG14] (adopted in qTESLA) or the approach used in Dilithium. The first one optimizes the signature size, while the second one optimizes the total size of public key and signature.

Finally, the EUF-CMA security of lattice-based Fiat-Shamir signatures in the quantum random oracle model was analyzed in several works, e.g., in [KLS18, DKL⁺18, ABB⁺20, Unr17, LZ19, DFMS19].

3 The Stateful Model for Wallets

Our formal security model for post-quantum secure stateful deterministic wallets is based on the model of [DFL19]. In this section, we recall the formal definition of a stateful wallet and the security properties that we want to guarantee for such a wallet. A stateful deterministic wallet scheme consists of two entities, a cold wallet and a hot wallet, that can respectively derive a valid pair of secret and public keys without the need for any interaction among each other. In more detail, upon initialization of the scheme, the cold wallet generates a master key pair (msk, mpk) and some initial state information St_0 and forwards (mpk, St_0) to the hot wallet. After this initial setup, the idea is that an arbitrary number of valid session key pairs can be generated by using the session secret/public key derivation algorithms within the respective wallets without further interaction. More precisely the public key derivation algorithm takes as input the current state and the master public key to generate a session public key. While the secret key derivation takes as inputs the current state and the master secret key and generates a session secret key. Since both public key and secret key derivation algorithms are

```
SW.KGen(1^λ)
  1: $St \leftarrow_\$ \{0,1\}^\lambda$
  2: $(mpk, msk) \leftarrow$ RSig.KGen($1^\lambda$)
  3: return $(St, mpk, msk)$

SW.Sign($sk, pk, m$)
  1: $m' \leftarrow (m, pk)$
  2: $\sigma \leftarrow$ RSig.Sign($sk, m'$)
  3: return $\sigma$

SW.Verify($pk, m, \sigma$)
  1: $m' \leftarrow (m, pk)$
  2: return RSig.Verify($pk, m', \sigma$)

SW.RandSK($msk, ID, St$)
  1: $(\rho, St) \leftarrow \mathsf{H}(St, ID)$
  2: $sk_{ID} \leftarrow$ RSig.RandSK($msk, \rho$)
  3: return $(sk_{ID}, St)$

SW.RandPK($mpk, ID, St$)
  1: $(\rho, St) \leftarrow \mathsf{H}(St, ID)$
  2: $pk_{ID} \leftarrow$ RSig.RandPK($mpk, \rho$)
  3: return $(pk_{ID}, St)$
```

Fig. 4. Generic Construction of a stateful deterministic wallet SW from a signature scheme with honestly rerandomizable keys RSig and a random oracle H.

deterministic, and the two wallets share the same current state, the key derivation algorithms output a valid session key pair. In order to keep track of which key has been derived with which state, each session key is indexed by a parameter *ID*, which is given as input into the key derivation procedures. In the following we recall the definition of a deterministic stateful wallet scheme and its correctness.

Definition 8 (Stateful Wallet). *A* stateful wallet scheme *is a tuple of algorithms* SW := (SW.KGen , SW.RandSK, SW.RandPK, SW.Sign, SW.Verify), *which are defined as follows:*

SW.KGen**:** *The master key generation algorithm takes as input public parameters param and outputs a master key pair (msk, mpk) as well as an initial state St_0.*

SW.RandSK**:** *The secret key derivation algorithm takes as input a master secret key msk, a state St and an identity ID and outputs a session secret key sk_{ID} and the state St.*

SW.RandPK**:** *The public key derivation algorithm takes as input a master public key mpk, a state St and an identity ID and outputs a session secret key pk_{ID} and the state St.*

SW.Sign**:** *The probabilistic signing algorithm takes as input a session secret key sk_{ID} for some ID and a message m and outputs a signature σ.*

SW.Verify**:** *The verification algorithm takes as input a session public key pk_{ID} for some ID, a message m, and a signature σ and outputs 1 if σ is a valid signature for m under public key pk_{ID}. It outputs 0 otherwise.*

Definition 9 (Correctness of Stateful Wallets). *For $n \in \mathbb{N}$, any $(St_0, msk, mpk) \in$ SW.KGen$(param)$, and any $\boldsymbol{ID} := (ID_1, ..., ID_n) \in \{0,1\}^*$, we define the sequence (sk_i, St_i) and (pk_i, St_i) for $1 \leq i \leq n$ recursively as*

$$(sk_i, St_i) := \mathtt{SW.RandSK}(msk, ID_i, St_{i-1}),$$
$$(pk_i, St_i) := \mathtt{SW.RandPK}(mpk, ID_i, St_{i-1}).$$

SW *is* correct *if for all $m \in \{0,1\}^*$ and i with $1 \leq i \leq n$ it holds that*

$$\Pr_{\sigma \leftarrow\$ \mathtt{SW.Sign}(sk_i, m)}[\mathtt{SW.Verify}(pk_i, \sigma, m) = 1] \geq 1 - \text{negl}(\lambda)\,.$$

A generic construction of a stateful deterministic wallet scheme SW := (SW.KGen, SW.RandSK, SW.RandPK, SW.Sign, SW.Verify) from a signature scheme with honestly rerandomizable keys RSig following Definition 8 is presented in Figure 4. Such a scheme should satisfy the following two security properties - *wallet unlinkability* and *wallet unforgeability* - which are described below.

```
Game WUNL
1: $(mpk, msk, St) \leftarrow \texttt{SW.KGen}()$
2: $b \leftarrow_\$ \{0,1\}$
3: $ID^* \leftarrow \mathcal{A}_1^{\mathsf{WalSign},\mathsf{PK}}(mpk)$
4: if $(Keys[ID^*] \neq \perp)$ then
5:     return 0
6: $(pk^0_{ID^*}, St^*) \leftarrow \texttt{SW.RandPK}(mpk, ID^*, St)$
7: $(sk^0_{ID^*}, St^*) \leftarrow \texttt{SW.RandSK}(msk, ID^*, St)$
8: $St \leftarrow St^*$
9: $(\overline{mpk}, \overline{msk}, \overline{St}) \leftarrow \texttt{SW.KGen}()$
10: $(pk^1_{ID^*}, St^*) \leftarrow \texttt{SW.RandPK}(\overline{mpk}, ID^*, \overline{St})$
11: $(sk^1_{ID^*}, St^*) \leftarrow \texttt{SW.RandSK}(\overline{msk}, ID^*, \overline{St})$
12: $Keys[ID^*] \leftarrow (pk^b_{ID^*}, sk^b_{ID^*})$
13: $b' \leftarrow \mathcal{A}_2^{\mathsf{WalSign},\mathsf{PK},\mathsf{getState}}(mpk, pk^b_{ID^*})$
14: return $b' = b$
```

```
Oracle PK($ID$)
1: $(pk_{ID}, St^*) \leftarrow \texttt{SW.RandPK}(mpk, ID, St)$
2: $(sk_{ID}, St^*) \leftarrow \texttt{SW.RandSK}(msk, ID, St)$
3: $Keys[ID] \leftarrow (pk_{ID}, sk_{ID})$
4: $St \leftarrow St^*$
5: return $pk_{ID}$
```

```
Oracle getState()
1: return $St$
```

```
Oracle WalSign($m$, $ID$)
1: if $(Keys[ID] = \perp)$ then
2:     return $\perp$
3: $(pk_{ID}, sk_{ID}) \leftarrow Keys[ID]$
4: $\sigma \leftarrow \texttt{SW.Sign}(sk_{ID}, m)$
5: return $\sigma$
```

Fig. 5. Unlinkability game WUNL for stateful wallets.

3.1 Wallet Unlinkability

Intuitively, the unlinkability property guarantees that session public keys that have been derived from the same master public key are computationally indistinguishable from the distribution of session public keys that have been derived from a different, independently chosen master public key. However, considering that hot wallet corruptions reveal the state and hence trivially break the unlinkability property, [DFL19] introduces the notion of forward unlinkability. This notion guarantees unlinkability prior to any hot wallet corruption.

The formal security game for unlinkability is defined in Figure 5 and proceeds as follows: Upon the initialization of the wallet scheme by executing $(mpk, msk, St) \leftarrow_\$ \texttt{SW.KGen}()$, the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ obtains mpk and runs its subprocedure $\mathcal{A}_1$ on input mpk, where $\mathcal{A}_1$ has access to oracles WalSign and PK. These oracles represent the adversary's capability to observe signatures with corresponding session public keys of the wallet on the ledger. More concretely, $\mathcal{A}_1$ can call WalSign on an arbitrary message m and any ID and receives a valid signature for m under public key pk_{ID}. Further, $\mathcal{A}_1$ can query the PK oracle on any ID and receives the session public key pk_{ID}.

Finally, $\mathcal{A}_1$ outputs an ID^*. If neither WalSign nor PK has been queried on ID^* before, the game proceeds to the challenge phase, in which two session key pairs $(pk^0_{ID^*}, sk^0_{ID^*})$ and $(pk^1_{ID^*}, sk^1_{ID^*})$ are generated, where $(pk^0_{ID^*}, sk^0_{ID^*})$ are derived from mpk and msk respectively, while $(pk^1_{ID^*}, sk^1_{ID^*})$ are derived from a freshly generated master key pair. After a uniformly random bit b is chosen, the subprocedure $\mathcal{A}_2$ is executed on input mpk and $pk^b_{ID^*}$. $\mathcal{A}_2$ gets access to oracles WalSign, PK and getState, where getState returns the current state of the wallet scheme. $\mathcal{A}$ wins the game, if its subprocedure $\mathcal{A}_2$ returns a bit b', such that $b' = b$. We define the advantage of an adversary $\mathcal{A}$ as its winning probability in game WUNL over random guessing.

Definition 10 (Unlinkability). *Let* SW *be a stateful wallet scheme (cf. Definition 8). We say that* SW *is pq-unlinkable if for any quantum adversary $\mathcal{A}$, the advantage in game* WUNL *(cf. Figure 5) is negligible.*

```
Game WUF
1: $(mpk, msk, St) \leftarrow \texttt{SW.KGen}()$
2: $b \leftarrow_{\$} \{0,1\}$
3: $(m^*, \sigma^*, ID^*) \leftarrow \mathcal{A}^{\mathsf{WalSign},\mathsf{PK}}(mpk, St)$
4: if $(Keys[ID^*] \neq \bot)$ then
5:     return 0
6: $(pk_{ID^*}, sk_{ID^*}) \leftarrow Keys[ID^*]$
7: if $(m^* \in M[ID^*])$ then
8:     return 0
9: if $(\texttt{SW.Verify}(pk_{ID^*}, m^*, \sigma^*) = 0)$ then
10:     return 0
11: return 1
```

```
Oracle PK(ID)
1: $(pk_{ID}, St^*) \leftarrow \texttt{SW.RandPK}(mpk, ID, St)$
2: $(sk_{ID}, St^*) \leftarrow \texttt{SW.RandSK}(msk, ID, St)$
3: $Keys[ID] \leftarrow (pk_{ID}, sk_{ID})$
4: $St \leftarrow St^*$
5: return $pk_{ID}$
```

```
Oracle WalSign(m, ID)
1: if $(Keys[ID] = \bot)$ then
2:     return $\bot$
3: $(pk_{ID}, sk_{ID}) \leftarrow Keys[ID]$
4: $\sigma \leftarrow \texttt{SW.Sign}(sk_{ID}, m)$
5: $M \leftarrow_{\cup} \{m\}$
6: return $\sigma$
```

Fig. 6. Unforgeability game WUF for stateful wallets.

3.2 Wallet Unforgeability

At a high level, unforgeability for stateful wallets ensures that funds held by the cold wallet remain secure even in case an adversary corrupts the hot wallet and/or observes transactions on the ledger signed by the cold wallet. In order to model this property, we define the game WUF, in which the adversary $\mathcal{A}$ obtains a master public key mpk and the initial state St_0 as input. This models the situation in which an adversary corrupts the hot wallet right after initialization of the wallet scheme. Further, $\mathcal{A}$ gets access to the oracles PK and WalSign, which are defined in the same way as in the game WUL, with the difference that WalSign now additionally keeps track of all queried messages. Eventually, $\mathcal{A}$ outputs a forgery consisting of a message m^*, a signature σ^* and an ID^*. $\mathcal{A}$ wins the game if (1) m^* has not been queried to WalSign before, (2) PK has been previously queried on ID^* and (3) σ^* is a valid signature for m^* under public key pk_{ID^*}.

Note that the adversary knows mpk and St_0 and hence can generate any session public key for any ID itself, which seems to make the PK oracle redundant. However, PK is still needed for bookkeeping purposes, i.e., to ensure that the session key pair for $\mathcal{A}$'s forgery has been created before $\mathcal{A}$ outputs its forgery. We define the advantage of an adversary $\mathcal{A}$ as its probability of winning the game WUF.

As mentioned in [DFL19], the fact that the adversary can derive arbitrary session public keys makes the wallet scheme vulnerable to *related key attacks*, in case the underlying signature scheme is prone to such attack. Intuitively, upon an adversary learning a signature σ_{ID} and a corresponding session public key pk_{ID}, a related key attack allows the adversary to transform σ_{ID} into a valid signature σ_{ID^*} under public key pk_{ID^*}. This attack may have a severe impact on the security guarantees of our wallet scheme, since it may allow an adversary to steal all funds of a cold wallet. One common counter measure against related key attack used in [DFL19,MSM+15] is called *public key prefixing*, i.e., a signature on a message μ is computed as $\texttt{Sign}(sk, (pk, \mu))$. In many signature schemes the signature is computed on the hash of the message and not the message itself. Therefore by prefixing the public key an adversary not only has to transform σ_{ID} into a valid signature σ_{ID^*} under public key pk_{ID^*} but also find a collision for the hash function in order to mount a related key attack.

Definition 11 (Unforgeability). *Let* SW *be a stateful wallet scheme (cf. Definition 8). We say that* SW *is pq-unforgeable if for any quantum adversary $\mathcal{A}$, the advantage in game* WUF *(cf. Figure 6) is negligible.*

3.3 Relevance of Our Relaxed Notions

In Section 2 we defined the notions of *signature schemes with rerandomizable public keys* (cf. Definition 4) and EUF-CMA-HRK (cf. Definition 6). While these notions deviate from the ones used in previous works [FKM+16, DFL19], it turns out that our relaxed notions are sufficient for building deterministic wallets as we discuss below.

Rerandomizable public keys. One of the benefits of using deterministic wallets is that individual payments to the wallet are unlinkable (cf. Figure 5). To satisfy the unlinkability definition, Das et al. [DFL19] require that the underlying signature scheme must have rerandomizable secret and public keys. However, as it can be observed in the unlinkability game, the adversary gets access only to the public key, while the secret key is never revealed to the adversary (as revealing it would trivially break the security of the scheme). Hence, it is sufficient to use our relaxed notion of rerandomizable public keys in order to achieve the unlinkability property. While the post-quantum secure signature scheme that we consider in this work does not offer rerandomizable public and secret keys as required by [DFL19], it fortunately achieves our relaxed notion of rerandomizable public keys. Thus, it is sufficient to instantiate a wallet scheme that achieves the unlinkability property.

EUF-CMA-HRK. As in [DFL19], we use the notion of *EUF-CMA under honestly rerandomizable keys*, where unforgeability holds if the randomness used to derive the keys is *honestly* generated. This is in contrast to the stronger notion of EUF-CMA-RK as defined in [FKM+16], where unforgeability must hold for *adversarial* chosen randomness. Stateful deterministic wallet schemes, however, derive the randomness deterministically from the state (see Figure 4), which is generated initially during a trusted setup when the master keys are created. Hence, the adversary has no influence on the randomness used during the rerandomization procedures. To conclude, the notion of EUF-CMA-HRK is not only suitable but also sufficient in the wallet setting.

3.4 Post-Quantum Security of Wallets

In this section we show that the generic construction achieves both unlinkability and unforgeability against quantum adversaries. Recall that since we are in the post-quantum setting, the oracles provided by the challenger in the unlinkability game (PK, $\mathsf{getState}$, $\mathsf{WalSign}$) and in the unforgeability game (PK, $\mathsf{WalSign}$) are run on a classical computer. Hence, also the (quantum) adversary gets only classical access to these oracles. However, the adversary can use its quantum computing power to access the quantum random oracle $|\mathsf{H}\rangle$, i.e., querying the random oracle in superposition.

The following theorem shows the unlinkability.

Theorem 1. *Let* $\mathtt{RSig}$ *be a signature scheme with rerandomizable public keys (cf. Definition 4) and* H *a random oracle. Then the stateful wallet scheme* $\mathtt{SW}$ *built from* $\mathtt{RSig}$ *and* H *(cf. Figure 4) is pq-unlinkable according to Definition 10, i.e., against quantum adversaries which have access to* $|\mathsf{H}\rangle$.

First, we provide a proof intuition of Theorem 1. Let us first recall how the unlinkability property is proven in the classical ROM setting (cf. [DFL19]). Note that the wallet public keys are derived from the wallet state, which is stored within the wallet, hidden from the adversary. The classical adversary can then try to guess one of the states of the wallet and make a "problematic query" to the random oracle H on such a state, in order to derive one of the session public keys generated by the wallet. If the adversary guesses the wallet's state correctly, it can distinguish a public key generated by the wallet from a randomly generated one, and hence the adversary will be able to win the unlinkability game. The classical proof consists of two steps: (1) showing the probability that the adversary makes the above mentioned *problematic* query to the random oracle is negligible, and (2) showing that the adversary has no advantage in winning the unlinkability game conditioned on the event that it does not make any problematic query. Finally, note that while this proof uses the stronger notion of rerandomizable public and secret keys (cf. Definition 3), it is easy to see that it also works with our relaxed definition of rerandomizable public keys (cf. Definition 4). This is because the unlinkability game requires the adversary to distinguish a public key generated by the wallet from a randomly generated public key.

Our proof in the QROM follows the same approach, however, the first step requires a different technique. Recall that the wallet state gets refreshed with every public key query. In [DFL19], the challenger keeps a list of the states of the wallet scheme – starting from the initial state till the one obtained during last public key query. In the analysis a simple comparison allows to check whether a query by the classical adversary is problematic, i.e., whether it coincides with one of the states of the wallet. Since the adversary can access the random oracle H only classically (it can query on exactly one input at a time), hence the challenger can store all these queries in a list. In the QROM, however, we can not keep such a list as the adversary now has quantum computation power, hence can query the random oracle on several, and even all, inputs in superposition.[5] Instead, we consider a game hop which we can bound by the advantage of the adversary in distinguishing two random oracles, which in turn can be bound using the O2H lemma. For the resulting game, the same argument as in the classical proof applies, i.e., the rerandomizable property of the underlying signature schemes ensures that the adversary has no advantage in winning the game.

Proof (of Theorem 1). Throughout, let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an adversary which makes q and q_{PK} queries to its oracles $|\mathsf{H}\rangle$ and PK, respectively. To prove the theorem we use the following two games.

Game G_0: This game is the game WUNL instantiated with $\mathtt{SW}$ (cf. Figure 4).

Game G_1: This game is the same as G_0, except that the randomness ρ and the new state St^*, prior to running $\mathcal{A}_2$ (i.e., Line 13 in Figure 5), are sampled at random, independent of the random oracle. In both games, the randomness and new state are distributed identical. The only difference lies in the random oracle. From the point of view of $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, the random oracle in game G_1 is $|\mathsf{H}_{\mathcal{S}\to\$}\rangle$, i.e., the random oracle that is reprogrammed to random values for every $x \in \mathcal{S}$, where $\mathcal{S}$ contains all pairs of states and IDs prior to running $\mathcal{A}_2$. Hence, we can bound the advantage in distinguishing G_0 and G_1 by the advantage in distinguishing the random oracles $|\mathsf{H}\rangle$ and $|\mathsf{H}_{\mathcal{S}\to\$}\rangle$. Applying the O2H Lemma (cf. Lemma 1) yields

$$\left|\Pr\left[\mathcal{A}^{|\mathsf{H}\rangle} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{|\mathsf{H}_{\mathcal{S}\to\$}\rangle} \Rightarrow 1\right]\right| \leq 2q\sqrt{\Pr\left[x \in \mathcal{S} \colon \mathcal{B}^{|\mathsf{H}\rangle} \Rightarrow x\right]}\,.$$

where $\mathcal{B}$ is the adversary specified in Lemma 1. Note that $\mathcal{A}$ has no information about the states in the set $\mathcal{S}$ until it queries $\mathsf{getState}$ to which only $\mathcal{A}_2$ has access. Furthermore, we have $|\mathcal{S}| \leq q_{\mathsf{PK}} + 2$ at any point in time, q_{PK} from $\mathcal{A}_1$'s queries and 2 from the challenge phase. This yields

$$\Pr\left[x \in \mathcal{S} \colon \mathcal{B}^{|\mathsf{H}\rangle} \Rightarrow x\right] \leq \frac{|\mathcal{S}|}{2^\lambda} \leq \frac{q_{\mathsf{PK}} + 2}{2^\lambda}.$$

Combining the above equations yields that the advantage in distinguishing G_0 and G_1 is negligible in the security parameter λ.

It remains to bound the advantage of $\mathcal{A}$ in game G_1, where the same argument from the classical proof applies. In G_1, the challenge public key $pk^b_{ID^*}$ given to $\mathcal{A}_2$ is independent of the random oracle (as the random oracle is not used in G_1 anymore for deriving keys). Hence, it is irrelevant whether the

[5] We note that, to some extent, the *compressed oracle technique* by Zhandry [Zha19] allows the recording of superposition queries.

adversary makes any query (classical or quantum) to the random oracle. As $\mathtt{RSig}$ is a signature scheme with rerandomizable public keys, the challenge public keys $pk^0_{ID^*}$ and $pk^1_{ID^*}$ are identically distributed, which yields that the adversarial advantage is 0. Combining the above proves the theorem. □

The following theorem shows that the generic construction is unforgeable in the presence of quantum attackers.

Theorem 2. *Let* $\mathtt{RSig}$ *be a signature scheme with rerandomizable public keys (cf. Definition 4) and* H *a random oracle. Then the stateful wallet scheme* $\mathtt{SW}$ *built from* $\mathtt{RSig}$ *and* H *(cf. Figure 4) is pq-unforgeable according to Definition 11, i.e., against quantum adversaries which have access to* $|\mathsf{H}\rangle$*.*

We briefly recap the classical proof in the ROM (cf. [DFL19]), thereby highlighting the challenge when switching to a quantum adversary. Note again, that the classical proof uses the stronger notion of rerandomizable keys (cf. Definition 3) but also holds for the weaker notion of rerandomizable public keys (cf. Definition 4). The proof consists of a game hop in which the adversary loses the game if there is a collision of session keys for different identities. Due to the construction, this occurs if the random oracle outputs a collision which is bound by a simple counting argument. The advantage of an adversary in the resulting game is bound by the security of the underlying signature scheme using a reduction. The crucial part is that the reduction simulates the random oracle H for the adversary using its oracle Rand from the EUF-CMA-HRK game. More precisely, for each query to H by the adversary, the reduction makes a query to Rand.

Our proof in the QROM follows the same idea, however additionally needs to take care of the use of the quantum random oracle $|\mathsf{H}\rangle$ by the adversary. The first part works exactly as in [DFL19], since the access to the oracle PK remains classical even for a quantum adversary. The second part, however, does not work as in [DFL19]. While the adversary can query the quantum random oracle $|\mathsf{H}\rangle$ in superposition, the reduction can query its oracle Rand only classical as it is provided by its (classical) challenger. By querying $|\mathsf{H}\rangle$ on an equal superposition of all (i.e., exponentially many) inputs, the reduction would need exponentially many queries to Rand in order to simulate $|\mathsf{H}\rangle$ for the adversary. Clearly, this would render the reduction useless as it would not be efficient. To tackle this issue, we do an additional game hop in which we switch from a random oracle to an oracle drawn from a small-range distribution. While this affects the advantage of the adversary only negligibly, it allows us to construct a reduction which can simulate the quantum random oracle for the adversary by making a polynomial number of (classical) queries to its oracle Rand.

Proof (of Theorem 2.). Let $\mathcal{A}$ be an adversary which makes q_{H} queries to $|\mathsf{H}\rangle$. The proof consists of the following three games.

Game G_0: This game is the game WUF instantiated with $\mathtt{SW}$ (cf. Figure 4). Assume that $\mathcal{A}$ has non-negligible advantage $\epsilon = \epsilon(\lambda)$ in winning G_0. This means that there exists a polynomial $p = p(\lambda)$ such that $p(\lambda) > \frac{1}{\epsilon(\lambda)}$.

Game G_1: This game is the same as G_0, except the adversary loses when there is a collision of keys for different identities. To detect the change, the adversary has to make queries to PK which result in colliding keys. Note that the adversary only has classical access to PK as it is provided by the classical challenger. Hence the bound from [DFL19] is applicable, which is a simple counting argument over the number of queries to PK. This yields that the advantage of $\mathcal{A}$ in G_1 is $\epsilon - \mathrm{negl}(\lambda)$, i.e., it is negligibly close to its advantage ϵ in G_0.

Game G_2: In this game the adversaries queries to $|\mathsf{H}\rangle$ is simulated using Definition 1 and the Lemma 2. Let $l = 2Cq^3_{\mathsf{H}}p$ with C being the constant from Lemma 2 and p being the polynomial described above. At the start of the game, the challenger will generate l random values and draw the first output (the randomness ρ) of the quantum random oracle $|\mathsf{H}\rangle$ from a small-range distribution using these l samples. The second output (the new state St) is generated just as in G_1. According to Lemma 2, $\mathcal{A}$ can only distinguish this game from the previous one with probability less than $\frac{1}{2p}$. Therefore, Lemma 2 yields that the advantage of $\mathcal{A}$ in this game is at least $\epsilon - \mathrm{negl}(\lambda) - \frac{1}{2p}$.

Bounding the advantage of $\mathcal{A}$ in Game G_2: We now show how to transform an adversary $\mathcal{A}$ playing G_2 into an adversary $\mathcal{B}$ playing $\mathsf{EUF\text{-}CMA\text{-}HRK}$ (where, the underlying signature scheme is $\mathtt{RSig}$). W.l.o.g., we assume that $\mathcal{A}$ never makes a query which results in $\perp$ and that there are no collisions. At the start, $\mathcal{B}$ receives a public key pk. It performs $l = 2Cq_{\mathsf{H}}^3 p$ queries to its oracle Rand and samples an initial state St_0. It invokes $\mathcal{A}$ on input $(mpk = pk, St_0)$.

Simulation of Quantum Random Oracle $|\mathsf{H}\rangle$. $\mathcal{B}$ simulates the first output (the randomness ρ), by using the l samples from Rand drawn from a small-range distribution. Note that Rand internally stores the output ρ in its list RList. For the second output (the new state St), $\mathcal{B}$ simulates it using a $2q_{\mathsf{H}}$-wise independent function which is indistinguishable for an adversary making q_{H} queries [Zha12b].

Simulation of PK *oracle.* When $\mathcal{A}$ queries its oracle PK on ID, $\mathcal{B}$ computes $pk_{ID} \leftarrow \mathtt{RSig.RandPK}(pk; \omega_{ID})$, where $(\omega_{ID}, St^*) \leftarrow \mathsf{H}(St, ID)$, sets $Keys[ID] \leftarrow (pk_{ID}, \omega_{ID})$, and sends pk_{ID} to $\mathcal{A}$.

Simulation of $\mathsf{WalSign}$ *oracle.* When $\mathcal{A}$ makes a query (m, ID) to its oracle $\mathsf{WalSign}$, $\mathcal{B}$ obtains the $(pk_{ID}, \omega_{ID}) = Keys[ID]$, sets $m' \leftarrow (m, pk_{ID})$, queries its own oracle OHR on (m', ω_{ID}), and forwards the response to $\mathcal{A}$. When $\mathcal{A}$ outputs a forgery (m^*, σ^*, ID^*), $\mathcal{B}$ obtains $(pk_{ID^*}, \omega_{ID^*}) = Keys[ID^*]$, sets $\hat{m}^* \leftarrow (m^*, pk_{ID^*})$, and outputs $(\hat{m}^*, \sigma^*, \omega_{ID^*})$.

If $\mathcal{A}$'s forgery (m^*, σ^*, ID^*) is valid in Game G_2, then $\mathcal{B}$'s forgery $(\hat{m}^*, \sigma^*, \omega_{ID^*})$ is also valid in EUF-CMA-HRK. We now show that the output of $\mathcal{B}$ is a valid forgery whenever the output of $\mathcal{A}$ is. First, since (m^*, σ^*, ID^*) is a valid forgery by $\mathcal{A}$, we know that $\mathcal{A}$ never queried (m^*, ID^*) to $\mathsf{WalSign}$. Recall that, for every $\mathsf{WalSign}$ query by $\mathcal{A}$ on any message (m), $\mathcal{B}$ made a OHR query on public key prefixed message $(m' \leftarrow \{m, pk\})$. Since $\mathcal{A}$ never queried $\mathsf{WalSign}$ on input (m^*, ID^*), $\mathcal{B}$ never queried $(\hat{m}^*, \omega_{ID^*})$ to its oracle OHR, where $\hat{m}^* \leftarrow \{m^*, pk\}$. Second, it holds that $\omega_{ID^*} \in \mathsf{RList}$. This follows from the simulation of the quantum random oracle where, for every possible output (ρ, St), ρ is in RList. Third, validity of the forgery by $\mathcal{A}$ yields validity of the forgery by $\mathcal{B}$.

Recall that $l = 2Cq_{\mathsf{H}}^3 p$ and as discussed at the beginning of this game, according to Lemma 2, the advantage of the adversary in this game is equal to $\epsilon - \mathrm{negl}(\lambda) - \frac{1}{2p}$. Assuming the security of the underlying signature scheme $\mathtt{RSig}$, we have that this advantage must be negligible. Combined with $\epsilon > \frac{1}{p}$ (see description of G_0), this yields that $\frac{1}{2p}$ is negligible, resulting in a contradiction. Hence, we conclude that ϵ, the advantage of $\mathcal{A}$, is negligible. □

4 PQ Signatures with Honestly Rerandomizable Public Keys

In this section we propose a lattice-based construction of a signature scheme with honestly rerandomizable public keys (cf. Definition 4). In such a signature scheme, the distribution of honestly rerandomized public keys is identical to the distribution of original public key, while honestly rerandomized secret keys are allowed to be distributed differently from the original secret key. The scheme extends the generic construction of lattice-based signatures from Section 2.5. We analyze the security of our scheme in Section 4.2. In Section 4.3 we discuss alternative ways of key rerandomization in a lattice-based signature scheme and argue why they fall short in building practical hot/cold wallets.

```
RandG(ρ)
  1: $\rho := (\rho_s, \rho_e) \in \{0,1\}^{2o(\lambda)}$
  2: $\mathbf{r} \leftarrow \mathsf{GenG}(k_2, \sigma, S/2, \rho_s)$
  3: $\mathbf{u} \leftarrow \mathsf{GenG}(k_1, \sigma, E/2, \rho_e)$
  4: return $(\mathbf{r}, \mathbf{u})$

RSig.KGen(1^λ)
  1: $(sk, pk) \leftarrow \mathsf{LB}.\mathtt{KGen}(1^\lambda)$
  2: $\mathsf{hpk} \leftarrow \mathsf{F}(pk)$
  3: $sk \leftarrow (\mathsf{hpk}, sk)$
  4: return $(sk, pk)$

RSig.RandPK(pk, ρ)
  1: $(\mathbf{r}, \mathbf{u}) \leftarrow \mathsf{RandG}(\rho)$
  2: $\mathbf{b}' \leftarrow \mathbf{b} + \mathbf{A}\mathbf{r} + \mathbf{u} \pmod q$
  3: $pk' := \mathbf{b}'$
  4: return $pk'$

RSig.RandSK(sk, ρ)
  1: $(\mathbf{r}, \mathbf{u}) \in D^{k_2}_{\mathbb{Z}^n,\sigma} \times D^{k_1}_{\mathbb{Z}^n,\sigma} \leftarrow \mathsf{RandG}(\rho)$
  2: $\mathbf{s}' \in D^{k_2}_{\mathbb{Z}^n,\sqrt{2}\sigma} \leftarrow \mathbf{s} + \mathbf{r}$
  3: $\mathbf{e}' \in D^{k_1}_{\mathbb{Z}^n,\sqrt{2}\sigma} \leftarrow \mathbf{e} + \mathbf{u}$
  4: $\mathbf{b}' \leftarrow \mathbf{A}\mathbf{s}' + \mathbf{e}' \pmod q$
  5: $\mathsf{hpk}' \leftarrow \mathsf{F}(\mathbf{b}')$
  6: $sk' := (\mathsf{hpk}', \mathbf{s}', \mathbf{e}')$
  7: return $sk'$

RSig.Sign(sk, m)
  1: $\mu \leftarrow (m, \mathsf{hpk})$
  2: $(\mathbf{z}_1, \mathbf{z}_2, c) \leftarrow \mathsf{LB}.\mathtt{Sign}(sk, \mu)$
  3: return $(\mathbf{z}_1, \mathbf{z}_2, c)$

RSig.Verify(pk, m, (z_1, z_2, c))
  1: $\mu \leftarrow (m, \mathsf{F}(pk))$
  2: return $\mathsf{LB}.\mathtt{Verify}(pk, \mu, (\mathbf{z}_1, \mathbf{z}_2, c))$
```

Fig. 7. Construction of lattice-based signature scheme with honestly rerandomizable public keys.

4.1 Description of the Scheme

Let $\mathsf{LB}.\Sigma = (\mathsf{LB}.\mathtt{KGen}, \mathsf{LB}.\mathtt{Sign}, \mathsf{LB}.\mathtt{Verify})$ be the lattice-based signature scheme given in Section 2.5, Figure 3, and let $\mathbf{A} \in R_q^{k_1 \times k_2}$ be a uniformly random matrix as defined in Section 2.5, i.e., $\mathbf{A}$ is publicly known and an implicit input to all algorithms. Furthermore, we define the following functions and algorithms:

- Max_j is a function that on input $a \in R$, it outputs the j^{th} largest absolute coefficient of a. This function is used for bounding the secret-related terms, and hence the signatures generated by the algorithm $\mathsf{LB}.\mathtt{Sign}$ (cf. line 6–7 in Figure 3).
- GenG is an algorithm that on input $(\mathsf{dim}, \sigma, \mathsf{bnd}, \mathsf{rnd})$, it outputs a vector $\mathbf{x} = (x_1, \ldots, x_{\mathsf{dim}})$, where $x_i \in R$ are sampled from $D_{\mathbb{Z}^n,\sigma}$ such that $\sum_{j=1}^{\kappa} \mathsf{Max}_j(x_i) \le \mathsf{bnd}$ by using a randomness rnd_i that is extracted from rnd, e.g., via the function E.
- $\mathsf{F} : \{0,1\}^* \longrightarrow \{0,1\}^{o(\lambda)}$ is a collision resistant hash function. It is used to hash the public key in order to prevent related key attacks [MSM+15].

In this section we set the distribution used in the algorithm $\mathsf{LB}.\mathtt{KGen}$ for the secret key to $\chi = D_{\mathbb{Z}^n,\sigma}$. More precisely, we assume that $sk = (\mathbf{s}, \mathbf{e}) \in D^{k_2}_{\mathbb{Z}^n,\sigma} \times D^{k_1}_{\mathbb{Z}^n,\sigma}$, where $\mathbf{s} \leftarrow \mathsf{GenG}(k_2, \sigma, S/2, \mathsf{rnd}_s)$ and $\mathbf{e} \leftarrow \mathsf{GenG}(k_1, \sigma, E/2, \mathsf{rnd}_e)$ for two predefined positive numbers S, E and randomnesses $\mathsf{rnd}_s, \mathsf{rnd}_e$. Setting $\chi = D_{\mathbb{Z}^n,\sigma}$ is essential for rerandomizing the secret key in the construction introduced in this section because the sum of Gaussian distributed elements with standard deviation σ is also Gaussian distributed with standard deviation $\sqrt{2}\sigma$ (cf. Lemma 3).

In the following we describe our signature scheme with honestly rerandomizable public keys. The respective algorithms are formalized in Figure 7. In order to simplify the construction, we first define the algorithm RandG (see Figure 7 for a formal description). This algorithm takes as input a randomness $\rho = (\rho_s, \rho_e) \in \{0,1\}^{o(\lambda)} \times \{0,1\}^{o(\lambda)}$, and outputs two vectors $\mathbf{r}, \mathbf{u}$, which are generated by running the algorithm GenG on input $(k_2, \sigma, S/2, \rho_s)$, $(k_1, \sigma, E/2, \rho_e)$, respectively.

RSig.KGen: The key generation algorithm runs the algorithm LB.KGen to obtain a key pair (sk, pk), where $sk = (\mathbf{s}, \mathbf{e}) \in D^{k_2}_{\mathbb{Z}^n,\sigma} \times D^{k_1}_{\mathbb{Z}^n,\sigma}$ and $pk = \mathbf{b} \in R_q^{k_1}$. Then, it computes $\mathsf{hpk} = \mathsf{F}(pk)$, prepends hpk to sk, and returns the updated (sk, pk).

RSig.RandPK: Given a public key $pk = \mathbf{b}$ and an honestly generated randomness ρ, the algorithm RSig.RandPK runs $\mathsf{RandG}(\rho)$ to generate a pair of Gaussian distributed vectors $(\mathbf{r}, \mathbf{u})$. Then, it computes $\mathbf{b}' = \mathbf{b} + \mathbf{A}\mathbf{r} + \mathbf{u} \pmod q$ and outputs the honestly rerandomized public key $pk' = \mathbf{b}'$.

RSig.RandSK: Given a secret key $sk = (\mathsf{hpk}, \mathbf{s}, \mathbf{e})$ and an honestly generated randomness $\rho \in \{0,1\}^{2o(\lambda)}$, the algorithm RSig.RandSK runs RandG to obtain $(\mathbf{r}, \mathbf{u}) \in D^{k_2}_{\mathbb{Z}^n,\sigma} \times D^{k_1}_{\mathbb{Z}^n,\sigma}$. Then, it computes $\mathbf{s}' = \mathbf{s}+\mathbf{r}$ and $\mathbf{e}' = \mathbf{e}+\mathbf{u}$. Note that by Lemma 3, the pair $(\mathbf{s}', \mathbf{e}')$ is distributed as $D^{k_2}_{\mathbb{Z}^n,\sqrt{2}\sigma} \times D^{k_1}_{\mathbb{Z}^n,\sqrt{2}\sigma}$. Finally, the algorithm computes $\mathsf{hpk}' = \mathsf{F}(\mathbf{b}')$ and outputs the honestly rerandomized secret key $sk' = (\mathsf{hpk}', \mathbf{s}', \mathbf{e}')$.

RSig.Sign: The algorithm RSig.Sign returns the signature obtained by calling the algorithm LB.Sign on message $\mu = (m, \mathsf{hpk})$. Signing messages together with the hash value of (honestly rerandomized) public keys ensures security under related key attacks [MSM+15].

RSig.Verify: The algorithm RSig.Verify returns the bit obtained by running LB.Verify(pk, μ), where $\mu = (m, \mathsf{F}(pk))$.

We note that rerandomizing sk must be carried out only with the original secret key, i.e., a rerandomized secret key cannot be used to generate a new rerandomized one. This ensures that all honestly rerandomized secret keys have identical distribution, i.e., $D^{k_2}_{\mathbb{Z}^n,\sqrt{2}\sigma} \times D^{k_1}_{\mathbb{Z}^n,\sqrt{2}\sigma}$. Furthermore, signatures generated using honestly rerandomized keys have different distribution from signatures generated using the original key pair. More precisely, the pair $(\mathbf{z}_1, \mathbf{z}_2)$ is distributed uniformly at random over $R^{k_2}_{B_1} \times R^{k_1}_{B_2}$, where

$$B_1 = \begin{cases} Y - S/2 & \text{if } sk \leftarrow \texttt{LB.KGen} \\ Y - S & \text{if } sk \leftarrow \texttt{RSig.RandSK} \end{cases}$$

$$B_2 = \begin{cases} Y - E/2 & \text{if } sk \leftarrow \texttt{LB.KGen} \\ Y - E & \text{if } sk \leftarrow \texttt{RSig.RandSK} \end{cases}$$

The bound Y of the masking pair $(\mathbf{y}_1, \mathbf{y}_2)$ is chosen such that the probability of generating valid signatures (cf. Section 2.5) is at least $1/M$, i.e.,

$$\left(\frac{2B_1+1}{2Y+1}\right)^{k_2 n} \cdot \left(\frac{2B_2+1}{2Y+1}\right)^{k_1 n} \geq 1/M,$$

where $M = O(1)$ is the repetition rate of the signing algorithm.

4.2 Security Analysis

In this section we analyze the EUF-CMA-HRK security of the scheme introduced in Section 4.1 in the QROM. More precisely, we reduce its EUF-CMA-HRK security to the EUF-CMA security of the lattice-based signature scheme $\mathsf{LB}.\Sigma = (\texttt{LB.KGen}, \texttt{LB.Sign}, \texttt{LB.Verify})$ described in Section 2.5. The correctness of the scheme directly follows from the correctness of $\mathsf{LB}.\Sigma$. Note that rerandomizability of public keys (see Definition 4) follows from the MLWE assumption [LS15]. That is, for any public key $\mathbf{b}$ and any honestly rerandomized public key $\mathbf{b}'$ both pairs $(\mathbf{A}, \mathbf{b})$, $(\mathbf{A}, \mathbf{b}')$ are indistinguishable from the uniform distribution over $R_q^{k_1 \times k_2} \times R_q^{k_1}$. Therefore, $\mathbf{b}$ and $\mathbf{b}'$ are identically distributed.

Theorem 3 (EUF-CMA-HRK Security). *The signature scheme with honestly rerandomizable public keys depicted in Figure 7 is EUF-CMA-HRK secure in the* QROM *if scheme* $\mathsf{LB}.\Sigma = (\mathsf{LB}.\texttt{KGen}, \mathsf{LB}.\texttt{Sign}, \mathsf{LB}.\texttt{Verify})$ *described in Figure 3 is EUF-CMA secure in the* QROM.

```
Reduction $\mathcal{D}(pk)$
1: $\mathsf{RList} := \varnothing$
2: $\mathcal{Q} := \varnothing$
3: $(m, ((\mathbf{z}_1, \mathbf{z}_2, c), \rho)) \leftarrow \mathcal{A}^{\mathsf{H}', \mathsf{Rand}, \mathsf{OHR}}(pk)$
4: if $(\rho = \mathsf{NULL})$ then
5:     $\mathsf{hpk} \leftarrow \mathsf{F}(pk)$
6:     $\mu \leftarrow (m, \mathsf{hpk})$
7:     return $(\mu, (\mathbf{z}_1, \mathbf{z}_2, c))$
8: if $(\rho \neq \mathsf{NULL})$ then
9:     $pk' \leftarrow \mathtt{RSig.RandPK}(pk, \rho)$
10:    $\mathsf{hpk}' \leftarrow \mathsf{F}(pk')$
11:    $\mu' \leftarrow (m, \mathsf{hpk}')$
12:    $(\mathbf{r}, \mathbf{u}) \in D^{k_2}_{\mathbb{Z}^n,\sigma} \times D^{k_1}_{\mathbb{Z}^n,\sigma} \leftarrow \mathsf{RandG}(\rho)$
13:    $\mathbf{z}'_1 \leftarrow \mathbf{z}_1 - \mathbf{r}c$
14:    $\mathbf{z}'_2 \leftarrow \mathbf{z}_1 - \mathbf{u}c$
15: return $(\mu', (\mathbf{z}'_1, \mathbf{z}'_2, c))$
```

Fig. 8. Reduction from the EUF-CMA security of LB.Σ (Figure 3) to EUF-CMA-HRK security of signature scheme with honestly rerandomizable public keys (Figure 7). Queries to OHR, H', and Rand are answered as shown in Figure 9.

Proof. Let $\mathcal{A}$ be an adversary that is able to generate valid forgeries under the signature scheme with honestly rerandomizable public keys, i.e., $\mathcal{A}$ is able to win the game $\mathsf{EUF\text{-}CMA\text{-}HRK}^{\mathcal{A}}_{\mathtt{RSig}}$ (cf. Definition 6). We construct an algorithm $\mathcal{D}$ that runs $\mathcal{A}$ as subroutine in order to win the game $\mathsf{EUF\text{-}CMA}^{\mathcal{D}}_{\mathsf{LB}.\Sigma}$ (see Definition 5) against the scheme LB.Σ. According to the security model, $\mathcal{A}$ has quantum access to a random oracle H' and classical access to a random oracle Rand in addition to classical access to the signing oracle OHR. The reduction $\mathcal{D}$ has quantum access to the random oracle H and classical access to the signing oracle O, which returns to $\mathcal{D}$ signatures generated by LB.Σ. The algorithm $\mathcal{D}$ is described in Figure 8. $\mathcal{D}$ initializes two empty lists RList, $\mathcal{Q}$. These are used by $\mathcal{D}$ to store queries to Rand and OHR, respectively. Simulation of OHR, H', and Rand is given in Figure 9.

Analysis. Let $(m, ((\mathbf{z}_1, \mathbf{z}_2, c), \rho))$ be a valid forgery output by the adversary $\mathcal{A}$. This means that $m \notin \mathcal{Q}$ and $\mathtt{RSig.Verify}(pk, m, (\mathbf{z}_1, \mathbf{z}_2, c)) = 1$. Moreover, $\rho \in \mathsf{RList}$ in case randomness $\rho \neq \mathsf{NULL}$.

We first analyze the case that $\rho = \mathsf{NULL}$. The signature satisfies $(\mathbf{z}_1, \mathbf{z}_2) \in R^{k_2}_{Y-\frac{S}{2}} \times R^{k_1}_{Y-\frac{E}{2}}$ and $c = \mathsf{H}'(\mathbf{w}, m, \mathsf{hpk}) = \mathsf{H}(\mathbf{w}, m, \mathsf{hpk})$, where $\mathbf{w} = \mathbf{A}\mathbf{z}_1 + \mathbf{z}_2 - \mathbf{b}c \pmod q$. Hence, this forgery constitutes a valid signature under LB.Σ on message $\mu = (m, \mathsf{hpk})$. Note that if c was not queried by some input, then $\mathcal{A}$ produces such c only with negligible probability, i.e., $1/|\mathbb{T}^n_\kappa|$. Thus, with probability of $1 - 1/|\mathbb{T}^n_\kappa|$, the value c must be a random oracle answer to a query made by $\mathcal{A}$, where $|\mathbb{T}^n_\kappa| = 2^\kappa \binom{n}{\kappa}$ and κ is chosen such that $|\mathbb{T}^n_\kappa| \geq 2^{2\lambda}$. This ensures that the probability of mapping two different values to the same output of H is at most $2^{-2\lambda}$.

Next, we assume that $\mathcal{A}$ outputs a valid forgery $(m, (\mathbf{z}_1, \mathbf{z}_2, c), \rho)$ under honestly rerandomized public key $\mathbf{b}'$ and $\rho \neq \mathsf{NULL}$. This means that $(\mathbf{z}_1, \mathbf{z}_2) \in R^{k_2}_{Y-S} \times R^{k_1}_{Y-E}$. In this case $\mathcal{D}$ transforms this signature into a forgery under the original public key $\mathbf{b}$ as follows: $\mathcal{D}$ runs $\mathsf{RandG}(\rho)$ to obtain $(\mathbf{r}, \mathbf{u})$. Then, it computes the vectors $\mathbf{z}'_1 = \mathbf{z}_1 - \mathbf{r}c$ and $\mathbf{z}'_2 = \mathbf{z}_2 - \mathbf{u}c$. Note that

$$\begin{aligned} \|\mathbf{z}'_1\|_\infty &\leq \|\mathbf{z}_1\|_\infty + \|\mathbf{r}c\|_\infty \leq\ Y - S + S/2 = Y - S/2, \\ \|\mathbf{z}'_2\|_\infty &\leq \|\mathbf{z}_2\|_\infty + \|\mathbf{u}c\|_\infty \leq Y - E + E/2 = Y - E/2\,. \end{aligned}$$

```
$\boxed{\mathsf{Sim}(pk, m, \rho)}$
1: if ($\rho = \mathsf{NULL}$) then
2:     $\mathcal{Q} := \mathcal{Q} \cup \{m\}$
3:     return $\mathsf{SimNoR}(pk, m)$
4: if ($\rho \neq \mathsf{NULL}$) then
5:     if ($\rho \notin \mathsf{RList}$) then
6:         return $\perp$
7:     $\mathcal{Q} := \mathcal{Q} \cup \{m\}$
8:     return $\mathsf{SimR}(pk, m, \rho)$

$\boxed{\mathsf{SimNoR}(pk, m)}$
1: $\mathsf{hpk} \leftarrow \mathsf{F}(\mathbf{b})$
2: $\mu \leftarrow (m, \mathsf{hpk})$
3: $(\mathbf{z}_1, \mathbf{z}_2, c) \leftarrow \mathsf{O}(\mu)$
4: return $(\mathbf{z}_1, \mathbf{z}_2, c)$

$\boxed{\mathsf{H}'(\cdot)}$
1: return $\mathsf{H}(\cdot)$

$\boxed{\mathsf{SimR}(pk, m, \rho)}$
1: $(\mathbf{r}, \mathbf{u}) \leftarrow \mathsf{RandG}(\rho)$
2: $pk' := \mathbf{b}' \leftarrow \mathtt{RSig.RandPK}(pk, \rho)$
3: $\mathsf{hpk}' \leftarrow \mathsf{F}(\mathbf{b}')$
4: $\mu' \leftarrow (m, \mathsf{hpk}')$
5: $(\mathbf{z}'_1, \mathbf{z}'_2, c) \leftarrow \mathsf{O}(\mu')$
6: $\mathbf{z}_1 \leftarrow \mathbf{z}'_1 + \mathbf{r}c$
7: $\mathbf{z}_2 \leftarrow \mathbf{z}'_2 + \mathbf{u}c$
8: if $\left((\mathbf{z}_1, \mathbf{z}_2) \notin R^{k_2}_{Y-S} \times R^{k_1}_{Y-E}\right)$ then
9:     goto 5
10: return $(\mathbf{z}_1, \mathbf{z}_2, c)$

$\boxed{\mathsf{Rand}()}$
1: $\rho \leftarrow_\$ \{0,1\}^{2o(\lambda)}$
2: $\mathsf{RList} \leftarrow \mathsf{RList} \cup \{\rho\}$
3: return $\rho$
```

Fig. 9. Description of algorithm Sim, which simulates signing queries to OHR. The algorithms SimNoR, SimR are subroutines used by Sim. The first one is called when signing query does not include randomness ρ, while the latter one is called when signing query includes honestly generated randomness $\rho \neq \mathsf{NULL}$. Queries to H' made by adversary $\mathcal{A}$ are redirected to the random oracle H, to which reduction $\mathcal{D}$ has access. Queries to Rand are answered locally by $\mathcal{D}$.

Hence, $(\mathbf{z}'_1, \mathbf{z}'_2) \in R^{k_2}_{Y-\frac{S}{2}} \times R^{k_1}_{Y-\frac{E}{2}}$. Furthermore, we have

$$\mathbf{w} = \mathbf{A}\mathbf{z}'_1 + \mathbf{z}'_2 - \mathbf{b}c = \mathbf{A}(\mathbf{z}_1 - \mathbf{r}c) + \mathbf{z}_2 - \mathbf{u}c - \mathbf{b}c = \mathbf{A}\mathbf{z}_1 + \mathbf{z}_2 - \mathbf{b}'c \pmod q .$$

Therefore, it holds that $c = \mathsf{H}'(\mathbf{w}, m, \mathsf{hpk}') = \mathsf{H}(\mathbf{w}, m, \mathsf{hpk}')$. Hence, the forgery output by $\mathcal{A}$ can be turned into a valid forgery under the original public key $\mathbf{b}$ for message $\mu' = \mathsf{F}(m, \mathsf{hpk}')$, i.e., it is a forgery under $\mathsf{LB}.\Sigma$.

Finally, we note that the environment of $\mathcal{A}$ is perfectly simulated, and whenever $\mathcal{A}$ wins the game $\mathsf{EUF\text{-}CMA\text{-}HRK}^{\mathcal{A}}_{\mathtt{RSig}}$, $\mathcal{D}$ wins the game $\mathsf{EUF\text{-}CMA}^{\mathcal{D}}_{\mathsf{LB}.\Sigma}$. The number of signing queries made by $\mathcal{D}$ is at most $M \cdot Q$, where $M = O(1)$ is the repetition rate[6] of $\mathsf{LB}.\Sigma$ and Q is the number of signing queries made by $\mathcal{A}$. $\square$

4.3 Alternative Methods for Rerandomization

In this section we describe alternative approaches for rerandomizing keys in the lattice setting and show why our scheme introduced in Section 4.1 is the most suitable option in the context of hot/cold wallets. First, we recall that our construction from the previous section assumes that the distribution of the secrets used in the key generation algorithm are from the Gaussian distribution, i.e., $\chi = D_{\mathbb{Z}^n,\sigma}$. This allows us to use Lemma 3 in order to obtain rerandomized secret keys that are also Gaussian distributed but with a slightly different standard deviation, i.e., $D_{\mathbb{Z}^n,\sqrt{2}\sigma}$. The key generation of our scheme cannot use uniformly distributed secrets over a small subset R_d from R, where R_d is the set of all polynomials from R with ℓ_∞ norm bounded by some integer $d \geq 1$. This is because the sum

[6] In practice, the repetition rate of the signing algorithm of standard lattice-based signature schemes is strictly smaller than 4 (e.g., see [DKL+18, ABB+20]).

of two uniformly random polynomials over R_d does not yield a polynomial that follows the uniform distribution over a subset $S \subseteq R_d$. Using a uniformly random sk for rerandomization would yield rerandomized secret keys with unknown distribution, and hence the hardness of the computational assumption underlying the rerandomized key pairs would be unclear. Let us now discuss the alternative approaches.

Rerandomizability of Gaussian distributed secret keys. It is (theoretically) possible to rerandomize key pairs such that the rerandomized secret keys have the same distribution as the original secret key. More precisely, assume that sk is Gaussian distributed with standard deviation σ. Given a randomness ρ, a rerandomized secret key is computed as $sk' = sk + \rho$. Due to [GKPV10, Lemma 3], sk' is Gaussian distributed with the same σ when σ is of a super-polynomial size in the security parameter λ. In other words, we must select σ large enough in order to make the statistical distance between the distribution of sk and sk' negligible in λ. This value of σ gives secret keys of very large size, and requires to increase the size of the masking vectors used in the signing algorithm. Hence, we obtain signatures of very large size, which rules out using the resulting scheme in practice.

Rerandomizability of uniform distributed secret keys. In theory, it is possible to use uniformly distributed rather than Gaussian distributed secret keys as follows. Assume that $\chi = R_d$ and $\rho \in R_1$. The rerandomized secret key $sk' = sk + \rho$ is uniformly distributed over R_{d-1} with probability $\left(\frac{2d-1}{2d+1}\right)^{(k_1+k_2)n}$, where $(k_1 + k_2)n$ is the dimension of sk'. Therefore, for a very large d this probability would be overwhelming in λ.

Example 1. By considering the parameters of Dilithium [DKL+18] proposed for $\lambda = 128$, we have $k_1 = 5$, $k_2 = 4$, and $n = 256$. Hence, we have to set $d \approx 2^{139}$ in order to make the previously stated probability at least $1 - 2^{-128}$. This value of d yields a secret key of size $\approx 2^{147}$ Bytes.

The above given example shows that this approach is merely of theoretical interest only and is not suitable for practical applications as it requires huge sizes of keys, and hence signatures.

Allowing rerandomization algorithms to communicate. Consider an application, in which the rerandomization algorithms (i.e., `RSig.RandSK` and `RSig.RandPK`) synchronize after each invocation of `RSig.RandSK`. Given sk and ρ, the algorithm `RSig.RandSK` uses ρ together with a counter ctr in order to deterministically generate a randomness ρ', e.g., by using the function E on input (ρ, ctr). Then, it computes $sk' = sk + \rho'$ and outputs the rerandomized secret key sk' only after verifying that it has the correct distribution. If this is not the case, it increases ctr by 1 and repeats this process. The algorithm `RSig.RandPK` needs to receive the corresponding ctr from `RSig.RandPK` in order to generate the rerandomized public key related to sk'. Note that if sk is Gaussian distributed, then we even obtain a scheme with rerandomizable public and secret keys as defined in [FKM+16]. While this method is practical and may be applicable in the construction of sanitizable signatures proposed in [FKM+16], it cannot be used in the setting of hot/cold wallets due to the fact that in each signing process `RSig.RandPK` must obtain the correct ctr that were used to generate sk'. This synchronization requirement undermines the main concept of hot/cold wallets, namely the fact that hot and cold wallets do not communicate with each other (except when they are being initialized).

5 Practical Instantiation

In this section we present an efficiency analysis of the wallet scheme introduced in Section 3. To this end, we instantiate the signature scheme presented in Section 4 with a concrete lattice-based signature scheme. The most recent Fiat-Shamir constructions of lattice-based signatures are Dilithium [DKL+18] and qTESLA [ABB+20]. We consider the latter scheme, since the hard lattice problem underlying its key generation algorithm uses Gaussian distributed secrets. This is essential for rerandomizing the secret key in our setting, and hence is sufficient for our scheme with honestly rerandomizable public keys described in Figure 7. On the other hand, Dilithium's key generation uses uniformly distributed secrets for the underlying lattice problem, instead of Gaussian distributed secrets, which is not suitable in our wallet setting (see Section 4.3). Employing the Gaussian distribution in the key generation algorithm of Dilithium instead, requires to adjust the security analysis of Dilithium and to choose new parameters. The resulting scheme would be similar to qTESLA, with slight differences in how signatures are compressed. We choose not to modify Dilithium's original design but stick to qTESLA, which does not need any modification for our setting and is well-studied in comparison to a modified version of Dilithium.

5.1 An Instantiation with qTESLA

In this section we show how the signature scheme with honestly rerandomizable public keys introduced in Section 4 can be instantiated with qTESLA. We note that the parameters of qTESLA were selected according to the security reduction from the RLWE problem. This approach has two different aspects: On the one hand, it guarantees that qTESLA has the security level as long as the underlying RLWE instance is hard enough. On the other hand, this approach affects the performance and sizes of keys and signatures, because larger parameters are required to achieve the desired security level. The main goal of our choice is to demonstrate that our wallet scheme can be instantiated with state-of-the-art lattice-based signature schemes without taking into account any of the two different aspects mentioned above.

The design of our scheme is based on lattices over modules. In order to employ qTESLA in our construction we set $k_2 = 1$ to obtain a variant based on lattices over ideals, and security based on the hardness of RLWE. The (master) secret key includes polynomials $s, e_1, \ldots, e_{k_1}$ sampled from $D_{\mathbb{Z}^n,\sigma}$. The polynomial s is bounded by $S/2$ using the function Max_j defined in Section 2.5, while $e_1, \ldots, e_{k_1}$ are each bounded by $E/2$ using Max_j. In qTESLA the bounds are S and E, respectively. However, our wallet scheme uses the master key pair only for rerandomization, and signatures are generated using honestly rerandomized key pairs, which already satisfy the bounds S and E. Therefore, we can use exactly the same parameters proposed for qTESLA in [ABB+20, Table 4]).

Note that in comparison to the generic signature scheme shown in Figure 3, Section 2.5, the signature scheme qTESLA [ABB+20] compresses signatures by employing the technique of [BG14]. In this technique the signer proves knowledge of only the secret polynomial s rather than s and $e_1, \ldots, e_{k_1}$. Therefore, signatures are of the form $(z_1, c) \in R_Y \times \mathbb{T}_\kappa^n$ rather than $(z_1, \mathbf{z}_2, c) \in R_Y \times R_Y^{k_1} \times \mathbb{T}_\kappa^n$. This approach does not affect the EUF-CMA-HRK security of the signature scheme with honestly rerandomizable public keys. That is, the reduction given in Figure 8 remains the same. Only simulating the signing oracle (cf. Figure 9) requires to include an additional check to ensure the correctness of simulated signatures. More concretely, after step 9 of algorithm SimR (see Figure 9) we add the last **for** loop of qTESLA's signature generation algorithm [ABB+20, Algorithm 4]. However, we have in our setting

$$w_i = a_i z_1 - b_i' c - r_i c \; (\mathsf{mod}^{\pm} q) \quad \text{for all} \quad i = 1, \ldots, k_1,$$

where a_i, b_i', and r_i are the entries of the public vector $\mathbf{a}$ (replaced by the matrix $\mathbf{A}$, since $k_2 = 1$), rerandomized public key $\mathbf{b}'$, and the vector $\mathbf{r}$, respectively.

5.2 Deploying PQ Wallets over Blockchains

In this section we give an overview of the transaction throughput that can be achieved in a cryptocurrency system using our signature scheme instantiated with qTESLA.

A simple transaction in most cryptocurrency networks transfers coins from one party to another. Such transactions must usually include the public key pk and the signature σ of the sender such that the validity of the transaction can be verified. In order to give an estimated transaction throughput, we use the raw transaction size of a regular Bitcoin transaction (i.e., without the size of pk and σ) and then add the size of pk and σ of our scheme to it. The raw transaction size of a Bitcoin is roughly 100 Bytes (B) [Bit19]. Hence, when instantiating our wallet scheme with qTESLA, we can take the corresponding signature size (2,592 B) and public key size (14,880 B) [ABB+20, Table 4] for a post-quantum security level of 95 bits[7] and add those to the rough raw transaction size of 100 B. The size of a transaction would then result in $100\text{ B} + 14,880\text{ B} + 2,592\text{ B} \approx 17.5\text{ KB}$. We note that it is possible for a party to send coins to multiple receivers in a single transaction which would essentially allow for transactions to be aggregated and increase efficiency.

Many cryptocurrencies (including Bitcoin and Ethereum) currently use the classical signature scheme ECDSA. For the sake of drawing a comparison, note that the size of the ECDSA public key and signature in Bitcoin is approximately 65 B and 73 B [ECD19], respectively, which results in more compact transactions (minimum size of a transaction being $100B + 65B + 73B \approx 240B$), and hence higher transaction throughput.

Naturally, there are various ways to improve the transaction throughput such as increasing block size and the rate at which blocks are produced. For example, in a Bitcoin-like currency new blocks are created roughly every 10 minutes, which tremendously limits the throughput and scalability of the network. In contrast, one can consider a system with a block rate of a few seconds, say 15-20 seconds (e.g., this is the case for the Ethereum blockchain). This significantly increases transaction throughput, and hence compensates for larger sizes of pk and σ. Yet these solutions are ad-hoc, while a more interesting direction for future work is to design further efficient post-quantum secure signature schemes with rerandomizable keys.

Acknowledgments

The authors are grateful to the anonymous reviewers of ACM CCS'20 for their valuable comments. This work is supported by the German Research Foundation (DFG) Emmy Noether Program FA 1320/1-1, by the German Research Foundation DFG - SFB 1119 - 236615297 (CROSSING Projects P1 and S7), by the German Federal Ministry of Education and Research (BMBF) *iBlockchain Project* (grant nr. 16KIS0902), by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the *National Research Center for Applied Cybersecurity ATHENE*.

References

ABB+20. Erdem Alkim, Paulo S. L. M. Barreto, Nina Bindel, Juliane Krämer, Patrick Longa, and Jefferson E. Ricardini. The lattice-based digital signature scheme qTESLA. In *Applied Cryptography and Network Security - 18th International Conference, ACNS 2020*, Lecture Notes in Computer Science, 2020. 4, 5, 10, 11, 22, 23, 24, 25

ABL+18. Divesh Aggarwal, Gavin Brennen, Troy Lee, Miklos Santha, and Marco Tomamichel. Quantum attacks on bitcoin, and how to protect against them. *Ledger*, 3, 2018. 2

AHU19. Andris Ambainis, Mike Hamburg, and Dominique Unruh. Quantum security proofs using semi-classical oracles. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology*

[7] Note that qTESLA proposes only two parameter sets, chosen with respect to a conservative cost model: qTESLA-p-I with 95 bits of post-quantum security and qTESLA-p-III with 160 bits of post-quantum security [ABB+20, Section 4.3].

Conference, 2019, volume 11693 of *Lecture Notes in Computer Science*, pages 269–295. Springer, 2019. 7

BDF$^+$11. Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security*, volume 7073, pages 41–69. Springer, 2011. 6

BF11. Dan Boneh and David Mandell Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In *Public Key Cryptography - PKC 2011*, pages 1–16. Springer, 2011. 10

BG14. Shi Bai and Steven D Galbraith. An improved compression technique for signatures based on learning with errors. In *Cryptographers' Track at the RSA Conference*, pages 28–47. Springer, 2014. 5, 11, 24

BIP17. Bitcoin bip32 specification. https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki, Feb. 24 2017. Accessed: 2020-09-15. 2, 5

Bit. Bitcoin post-quantum. https://bitcoinpq.org/. 2, 5

Bit18. BitcoinExchangeGuide. CipherTrace Releases Report Exposing Close to $1 Billion Stolen in Crypto Hacks During 2018. https://bitcoinexchangeguide.com/ciphertrace-releases-report-exposing-close-to-1-billion-stolen-in_-crypto-hacks-during-2018/, 2018. 2

Bit19. Bitcoin wiki transaction format. https://en.bitcoin.it/wiki/Transaction, Dec. 2019. Accessed: 2020-05-04. 25

Blo18. Bloomberg. How to Steal $500 Million in Cryptocurrency. http://fortune.com/2018/01/31/coincheck-hack-how/, 2018. 2

BR93. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, 1993*, pages 62–73. ACM, 1993. 6

CGK$^+$19. Alexandru Cojocaru, Juan A. Garay, Aggelos Kiayias, Fang Song, and Petros Wallden. The bitcoin backbone protocol against quantum adversaries. Cryptology ePrint Archive, Report 2019/1150, 2019. https://eprint.iacr.org/2019/1150. 2

DDLL13. Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal Gaussians. In *Advances in Cryptology–CRYPTO 2013*, pages 40–56. Springer, 2013. 5

DFL19. Poulami Das, Sebastian Faust, and Julian Loss. A formal treatment of deterministic wallets. In *ACM SIGSAC Conference on Computer and Communications Security - CCS 2019*, pages 651–668. ACM, 2019. 1, 2, 3, 5, 9, 11, 13, 14, 15, 16, 17

DFMS19. Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Security of the Fiat-Shamir transformation in the quantum random-oracle model. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference*, volume 11693, pages 356–383. Springer, 2019. 5, 11

DKL$^+$18. Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium: A lattice-based digital signature scheme. *Transactions on Cryptographic Hardware and Embedded Systems - TCHES 2018*, (1):238–268, 2018. 5, 10, 11, 22, 23

ECD19. Bitcoin wiki: Elliptic curve digital signature algorithm, Nov. 2019. https://en.bitcoin.it/wiki/Elliptic_Curve_Digital_Signature_Algorithm. 25

eth15. ethereum.org. Ethereum. https://ethereum.org/, 2015. 1

EZS$^+$19. Muhammed F Esgin, Raymond K Zhao, Ron Steinfeld, Joseph K Liu, and Dongxi Liu. Matrict: Efficient, scalable and post-quantum blockchain confidential transactions protocol. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 567–584, 2019. 1, 2, 5

FKM$^+$16. Nils Fleischhacker, Johannes Krupp, Giulio Malavolta, Jonas Schneider, Dominique Schröder, and Mark Simkin. Efficient unlinkable sanitizable signatures from signatures with re-randomizable keys. In *Public Key Cryptography - PKC 2016*, pages 301–330. Springer, 2016. 3, 4, 5, 7, 9, 15, 23

FTS$^+$19. Chun-I Fan, Yi-Fan Tseng, Hui-Po Su, Ruei-Hau Hsu, and Hiroaki Kikuchi. Secure hierarchical bitcoin wallet scheme against privilege escalation attacks. *International Journal of Information Security*, pages 1–11, 2019. 1, 5

GKPV10. Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Robustness of the learning with errors assumption. In *Innovations in Computer Science - ICS 2010*, pages 230–240. Tsinghua University Press, 2010. 23

GS15. Gus Gutoski and Douglas Stebila. Hierarchical deterministic bitcoin wallets that tolerate key leakage. In *Financial Cryptography and Data Security - 19th International Conference, FC 2015*, volume 8975, pages 497–504. Springer, 2015. 1, 5

HMW18. Timo Hanke, Mahnush Movahedi, and Dominic Williams. DFINITY technology overview series, consensus system. *CoRR*, abs/1805.04548, 2018. 1

KLS18. Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner. A concrete treatment of fiat-shamir signatures in the quantum random-oracle model. In *Advances in Cryptology–EUROCRYPT 2018*, pages 552–586. Springer, 2018. 5, 11

LPR10. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Advances in Cryptology–EUROCRYPT 2010*, pages 1–23. Springer, 2010. 10

LS15. Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, 75(3):565–599, 2015. 10, 20

Lyu09. Vadim Lyubashevsky. Fiat-shamir with aborts: Applications to lattice and factoring-based signatures. In *Advances in Cryptology–ASIACRYPT 2009*, pages 598–616. Springer, 2009. 5

LZ19. Qipeng Liu and Mark Zhandry. Revisiting post-quantum Fiat-Shamir. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference*, volume 11693 of *Lecture Notes in Computer Science*, pages 326–355. Springer, 2019. 5, 11

MP12. Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *Advances in Cryptology - EUROCRYPT 2012*, pages 700–718. Springer, 2012. 10

MR04. Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. In *Symposium on Foundations of Computer Science (FOCS 2004)*, pages 372–381. IEEE Computer Society, 2004. 10

MSM$^+$15. Hiraku Morita, Jacob C. N. Schuldt, Takahiro Matsuda, Goichiro Hanaoka, and Tetsu Iwata. On the security of the schnorr signature scheme and DSA against related-key attacks. In *Information Security and Cryptology - ICISC 2015*, volume 9558, pages 20–35. Springer, 2015. 14, 19, 20

Nak09. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009. http://bitcoin.org/bitcoin.pdf. 1

NC11. Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, New York, NY, USA, 10th edition, 2011. 6

Noe15. Shen Noether. Ring signature confidential transactions for monero. Cryptology ePrint Archive, Report 2015/1098, 2015. http://eprint.iacr.org/2015/1098. 1, 5

QRL. Quantum resistant ledger (qrl). https://github.com/theQRL/Whitepaper/blob/master/QRL_whitepaper.pdf. 2, 5

Sch91. Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991. 3

Sho94. Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science*, pages 124–134. IEEE Computer Society, 1994. 2

Ske18. Rhys Skellern. Cryptocurrency Hacks: More Than $2b USD lost between 2011-2018. https://medium.com/ecomi/cryptocurrency-hacks-more-than-2b-usd-lost-between-2011-2018_-67054b342219, 2018. 2

TVR16. Mathieu Turuani, Thomas Voegtlin, and Michael Rusinowitch. Automated verification of electrum wallet. In *International Conference on Financial Cryptography and Data Security*, pages 27–42. Springer, 2016. 1, 5

Unr15. Dominique Unruh. Revocable quantum timed-release encryption. *J. ACM*, 62(6):49:1–49:76, 2015. 6

Unr17. Dominique Unruh. Post-quantum security of Fiat-Shamir. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security*, volume 10624, pages 65–95. Springer, 2017. 5, 11

Zha12a. Mark Zhandry. How to construct quantum random functions. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012*, pages 679–687. IEEE Computer Society, 2012. 7

Zha12b. Mark Zhandry. Secure identity-based encryption in the quantum random oracle model. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference*, volume 7417, pages 758–775. Springer, 2012. 18

Zha19. Mark Zhandry. How to record quantum queries, and applications to quantum indifferentiability. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference*, volume 11693, pages 239–268. Springer, 2019. 16