A Practical Forward-Secure DualRing

Nan Li[1], Yingjiu Li[2], Atsuko Miyaji[3], Yangguang Tian[3⋆], and Tsz Hon Yuen[4]

[1] The University of Newcastle, Australia Nan.Li@newcastle.edu.au
[2] University of Oregon, USA yingjiul@uoregon.edu
[3] Osaka University, Japan {miyaji,jack}@comm.eng.osaka-u.ac.jp
[4] University of HongKong johnyuenhk@gmail.com

Abstract. Ring signature allows a signer to generate a signature on behalf of a set of public keys, while a verifier can verify the signature without identifying who the actual signer is. In Crypto 2021, Yuen et al. proposed a new type of ring signature scheme called DualRing. However, it lacks forward security. The security of DualRing cannot be guaranteed if the signer's secret key is compromised. In this work, we introduce forward-secure DualRing. The singer can periodically update his secret key using our proposed "split-and-combine" method to mitigate the security risks caused by the leakage of secret keys. We present a practical scheme based on the discrete logarithm assumption. We show a detailed evaluation to validate its practicality.

Keywords: DualRing, Forward Security, Practical Scheme

1 Introduction

Ring signatures [27] allow a signer to sign messages on behalf of a set of public keys, and a verifier cannot identify who the real signer is. Since ring signatures provide anonymity, they are widely used in the privacy-preserving scenarios such as whistleblowing, e-voting, and privacy-preserving cryptocurrencies. The classic ring signature scheme [27] requires a signer first to compute n-1 pseudo-signatures for a set of n public keys PK. Then, the signer generates a real signature on a challenge value c using his signing key. The n signatures together with the challenge value c form a ring signature under PK.

The state-of-the-art ring signature scheme is called DualRing [29] proposed in Crypto 2021. The construction of DualRing takes a different approach, which achieves a significant saving in terms of signature size. Specifically, a signer first chooses n-1 pseudo-challenge values. Next, the signer derives a real challenge value c from the n-1 pseudo-challenge values and a set of n public keys PK. The last step is the signer generating a signature on the challenge value c using his signing key. The resulting DualRing consists of a single signature and n challenge values compared to the classical ring signature that consists of a single challenge value and n signatures. The n challenge values in DualRing can be further compressed to a $O(\log n)$-size argument of knowledge in the discrete logarithm (DL)

⋆ Corresponding author.

setting. However, DualRing lacks forward security. Forward secrecy [13] means that the unforgeability of the message-signature pair generated in the past is still guaranteed after the current secret key is leaked (e.g., due to side-channel attacks).

1.1 Motivations

Forward-secure ring signature is important for privacy-preserving applications. In the case of whistleblowing [1], an employee Alice intends to leak a secret as a whistleblower on behalf of all public keys in her company while she is still in the company, and she does not want to be identified before leaving the company. If Alice uses a ring signature to reveal the secret, the unforgeability of the ring signature assumes that no adversary can obtain any secret key from the members of the ring. However, due to the nature of dynamic ring formation in ring signatures, it is difficult for such assumption to hold over time. In our example, Alice may not know each and every user in the company, and she may not have any control to ensure that all users in her company would keep their secret keys secure over a certain period of time. Therefore, it is beneficial to design a cryptographic solution, such that the unforgeability of a ring signature is guaranteed if and only if the secret keys of the members of the ring are not compromised *at the time of signing*.

Forward-secure ring signatures (FS-RS) can be used in the case of remote (or internet) voting. The internet voting systems like Helios [4], Remotegrity [31], and VOTOR [19] allow anyone to set up an election, invite voters to cast a secret ballot, compute a tally, and provide a verifiable tally for the election. Our forward-secure ring signature scheme is suitable for internet voting for two reasons: 1) On the usability side, the voters register their credentials once to an election authority. The registered credentials can be re-used in different elections without being identified. 2) On the security side, the voters submit their votes in an election and (privately) updates their credentials for future elections. The forward security of our ring signature scheme ensures that no user's updated credentials can be misused by adversary for tracing or revealing the vote submissions of the user, even if the user is under coercion to reveal their updated credentials.

1.2 Overview of Our Construction

In this work, we introduce forward-secure DualRing and extend it to forward-secure linkable DualRing. The proposed construction is built from DualRing [29] and a key update technique [10, 17]. First, we review a Type-T signature (three-move type such as Schnorr signature [28]), which is used in building our scheme. We focus on the DL-based Type-T signature in this work. The signing process of the Type-T signature includes three functions: 1) a commit function, which outputs a commitment $R = g^{\widehat{r}}$, where $\widehat{r}$ denotes a randomness; 2) a hash function, which outputs a challenge $\widehat{c} = \mathtt{H}(R||\mathtt{pk}||m)$, where $\mathtt{pk} = g^{\mathtt{sk}}$ denotes a public key, $\mathtt{sk}$ denotes a secret key, and m denotes the signing message. 3) a

response function, which outputs a response $z = \widehat{r} - \widehat{c} \cdot \mathtt{sk}$. The resulting signature is $\sigma = (\widehat{c}, z)$. For verification, one can check $\widehat{c} \stackrel{?}{=} \mathtt{H}(R' || \mathtt{pk} || m)$, where $R' = g^z \cdot \mathtt{pk}^{\widehat{c}}$.

Second, we show a key update technique, which was used in building the forward-secure schemes [10, 17]. We assume that a secret key at epoch (i.e., a fixed time period) t includes the following elements,

$$\mathtt{sk}_t = (c, d, e_{t+1}, \cdots, e_T) = (g^r, \underline{h^{\mathtt{sk}} \cdot F(t)^r}, h_{t+1}^r, \cdots, h_T^r)$$

where T denotes the upper bound of time periods, r denotes a randomness (due to security reasons), $F(t)$ represents a public function for time t, and $h_{t+1}^r, \cdots, h_T^r$ is used for key updates. The key update process at epoch t' is shown as follows.

$$\mathtt{sk}_{t'} = (c', d', e_{t+2}, \cdots, e_T) = (g^{r+r'}, \underline{h^{\mathtt{sk}} \cdot F(t')^{r+r'}}, h_{t+2}^{r+r'}, \cdots, h_T^{r+r'})$$

where t' denotes a new time period (note that t is a prefix of t'), r' denotes a new randomness. For each key update, it requires a new randomness r' to ensure forward security.

The challenge of designing forward-secure Type-T signature (and forward-secure DualRing) is to replace the static secret key $\mathtt{sk}$ by a time-dependent secret key $\mathtt{sk}_t$ for signing, while the public key $\mathtt{pk}$ is fixed. However, the secret key $\mathtt{sk}_t$ is not suitable to be used directly in generating the response z in forward-secure DualRing because $\mathtt{sk}_t$ consists of group elements that cannot work with the response function on finite field $\mathbb{F}_q$ (q is prime number). We propose a novel technique to apply $\mathtt{sk}_t$ in generating forward-secure DualRing signatures. The key idea is that we use group elements (c, d) as the signing keys, and we use the randomness $\widehat{r}$ involved in the commit function to *link* the signing keys (c, d). We call it "split-and-combine" method. Specifically, we first *split* the randomness $\widehat{r}$ used in the commit function into two shares $(\widehat{r_1}, \widehat{r_2})$, where $\widehat{r} = \widehat{r_1} + \widehat{r_2}$. Then, we use signing keys (c, d) to "sign" two randomness shares $(\widehat{r_1}, \widehat{r_2})$ respectively, and output two response values. The resulting signature includes a challenge $\widehat{c}$ (i.e., the hash function's output) and two response values. The verification of the signature is performed by computing a commitment R' from the two response values, and checking $\widehat{c} \stackrel{?}{=} \mathtt{H}(R' || \mathtt{pk} || m)$. Note that the two randomness shares can be *combined* in the generation of R'. To conclude, this split-and-combine method allows a signer to use the split randomness shares to link group elements (c, d). The linked group elements are used in generating the response values for forward-secure DualRing signatures. In the process of signature verification, the split randomness shares can be combined as a verifier computes a commitment R' from the response values.

1.3 Related Work

Ring Signatures. Ring signatures [27] allow a singer to sign messages over a chosen set of public keys (including his/her own) without revealing who the real signer is. Since ring signatures provide anonymity (i.e., signer-ambiguity),

they can be used in constructing various privacy-preserving protocols, including whistleblowing, electric voting, and privacy-preserving cryptocurrencies (e.g., Monero and Zcash).

Abe et al. [3] introduced a generic framework that allows a signer to choose different types of public keys to form a ring (i.e., public-key set). Specifically, a singer can choose both RSA-keys and Discrete logarithm (DL)-keys to generate ring signatures. The ring signature scheme is efficient if it is used only with a single type of public keys.

Dodis et al. [16] introduced an accumulator-based ring signature scheme. The resulting signature size is constant, which is independent of the size of the ring. Specifically, the proposed scheme allows the signer to "compress" n public keys into a single value, and rely on a *witness* showing that the signer's public key is in the public-key set. However, their scheme requires a trusted setup for generating system parameters.

Groth and Kohlweiss [18] proposed efficient ring signatures based on one-out-of-many proofs. The one-out-of-many approach requires a zero-knowledge proof to prove the knowledge of the secret key with respect to one of the public keys in the ring. The proof size of this scheme is $\mathcal{O}(\log\ n)$, and it is setup-free. The follow-up works are various. For example, Bootle et al. [12] presented an accountable ring signature scheme, which extends Groth and Kohlweiss's scheme to support accountability. Libert et al. [21] introduced a tightly secure ring signature scheme. Their scheme is derived from Groth and Kohlweiss's ring signature scheme and DDH-based Elgamal encryptions. Recently, Lai et al. [20] introduced Omniring (i.e., Ring Confidential Transactions or RingCT) for RingCT, and Yuen et al. [30] proposed a new ring signature scheme for RingCT3.0. Both signature schemes require no trusted setup, and the proof size is $\mathcal{O}(\log\ n)$.

Forward Security. Forward security states that the compromise of entities at the present time will not affect the security of cryptographic primitives in the past. It is regarded as a basic security guarantee for many cryptographic primitives, including encryptions, signatures and key exchanges. Here, we focus on forward-secure signatures. If an attacker compromises a singer (e.g., via side-channel attacks), she cannot forge a signature from the signer at an earlier time. Specifically, when the attacker compromises a signer's signing key for the current time period, the signing keys from earlier time periods cannot be recovered. In this case, a one-way key update (or evolve) process is needed.

Bellare and Miner [7] formalized the security for forward-secure signatures. They also proposed a scheme with a squaring-based key update. So, its forward security is based on the hardness of factoring ($N = pq$, where p, q are two primes). Later, forward-secure ring signature (FS-RS) schemes have been proposed in the literature [23, 24]. However, they have certain limitations. For example, the squaring-based key update in [23] is not suitable for the standard RSA/DL-based forward-secure schemes. The scheme in [24] involves composite-order group operations, thus less practical.

2 Preliminaries

In this section, we present the complexity assumptions and the building blocks for constructing our proposed protocol.

2.1 Complexity Assumptions

Bilinear Maps. We define a group generation as $(q, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, \hat{\mathsf{e}}) \leftarrow \mathsf{GroupGen}(1^\lambda)$, where q is a prime number, g, h are two group generators, $\mathbb{G}$, $\mathbb{H}$ and $\mathbb{G}_T$ are cyclic groups of order q. The asymmetric bilinear map $\hat{\mathsf{e}} : \mathbb{G} \times \mathbb{H} \rightarrow \mathbb{G}_T$ has the following properties: 1) Bilinearity: for $g, h \in \mathbb{G}$ and $a, b \in \mathbb{Z}_q$, we have $\hat{\mathsf{e}}(g^a, h^b) = \hat{\mathsf{e}}(g, h)^{ab}$. 2) Non-degeneracy: $\exists g \in \mathbb{G}$ such that $\hat{\mathsf{e}}(g, h)$ has order q in $\mathbb{G}_T$.
We introduce a variant of wBDHI assumption, which is used in the unforgeability analysis.

Definition 1. *Given group generators $g \in \mathbb{G}, h \in \mathbb{H}$, and $a, b \in \mathbb{Z}_q$, we define the advantage of the adversary $\mathcal{A}$ in solving the wBDHI problem as*

$$\mathtt{Adv}_{\mathcal{A}}^{wBDHI}(\lambda) = \Pr[\mathcal{A}(g, h, g^a, g^b, h^a, h^b, \underline{h^{b^2}, \cdots, h^{b^\ell}}) = \hat{\mathsf{e}}(g, h)^{a \cdot b^{\ell+1}} \in \mathbb{G}_T]$$

The wBDHI assumption is secure if $\mathtt{Adv}_{\mathcal{A}}^{wBDHI}(\lambda)$ is negligible in λ.

The wBDHI assumption was originally defined for Type-1 pairings (i.e., $\mathbb{G}_1 = \mathbb{G}_2$), and its security holds in the generic bilinear group model, as shown in [10]. The wBDHI assumption also holds for Type-3 pairings (i.e., $\mathbb{G}_1 \neq \mathbb{G}_2$), which is shown in [17]. We use wBDHI to represent the variant used in this work. The difference between this variant and the wBHDI assumption for Type-3 pairings is small. If we give $g^{b^2}, \cdots, g^{b^\ell}$ (as well as the above underline part in group $\mathbb{H}$) to $\mathcal{A}$, it is equal to the wBDHI assumption described in [17]. We omit the security analysis of this variant since the reduction is straightforward. The decisional version of the wBDHI problem requires $\mathcal{A}$ to distinguish $\hat{\mathsf{e}}(g, h)^{a \cdot b^{\ell+1}}$ from a random value in $\mathbb{G}_T$.

2.2 DualRing

The DL-based DualRing signature scheme consists of the following algorithms [29].

- $\mathsf{Setup}(1^\lambda)$: It takes a security parameter λ as input, outputs public parameters PP, which are the implicit input for all the following algorithms. It also defines a hash function $\mathtt{H} : \{0,1\}^* \rightarrow \mathbb{Z}_q$.
- $\mathsf{Setup}(\mathsf{PP})$: It takes the public parameters PP as input, output a key pair $(\mathtt{sk}, \mathtt{pk})$, where $\mathtt{pk} = g^{\mathtt{sk}}$.
- $\mathsf{Sign}(\mathsf{PP}, \mathtt{sk}_i, m, \{\mathtt{pk}_j\}^l)$: It takes a signer's key $\mathtt{sk}_i$, a message m, and a set of public keys $\{\mathtt{pk}_j\}^l$ $(j \neq i)$, outputs a signature $\sigma = (z, c, c_1, \cdots, c_l)$. Specifically, the singer $\mathtt{pk}_i$ performs the following operations.

1. Choose $r \in \mathbb{Z}_q$, $\{c_j\}^l \in \mathbb{Z}_q$, and compute a commitment $R = g^r \cdot \sum \mathtt{pk}_j^{c_j}$.
2. Compute a challenge $c = \mathtt{H}(R||\mathsf{PK}||m) - \sum c_j$, where $\mathsf{PK} = (\mathtt{pk}_i||\{\mathtt{pk}_j\})$
3. Compute a response $z = r - \mathtt{sk}_i \cdot c$.

– $\mathsf{Verify}(\mathsf{PP}, \mathsf{PK}, m, \sigma)$: It outputs 1 if $\mathtt{H}(R'||\mathsf{PK}||m) = c + \sum c_j$, where $R' = g^z \cdot \sum(\mathtt{pk}_i^{c_i})_{i=1}^{l+1}$.

We present the high-level idea of DL-based DualRing as follows. First, a signer adds the decoy public keys $\mathtt{pk}_j$ and their corresponding challenge values c_j to the commitment R. Second, after computing a hash value $\mathtt{H}(R||\mathsf{PK}||m)$, the signer with index i can compute a final challenge c from the challenge value $\mathtt{H}(R||\mathsf{PK}||m)$ and the challenge values c_j. Third, the signer computes a response z according to Type-T signature scheme. To verify, the commitment R is reconstructed from all public keys and their corresponding challenge values. The sum of the challenge values is equal to the hash value $\mathtt{H}(R||\mathsf{PK}||m)$. For security, DualRing needs to achieve unforgeability and anonymity. Specifically, unforgeability means that the adversary cannot produce a valid signature without accessing the secret key, even if he/she can adaptively corrupt other honest participants and obtain their secret keys. Anonymity requires that the adversary cannot pinpoint the actual signer given a valid signature and a group of public keys, even if he/she is given all randomness to generate the secret keys. The formal definition is given in [29].

2.3 Forward Security

In this sub-section, we show non-interactive forward security. Non-interactive means that the key holder updates their keys locally, without interacting with any third parties. First, we show the forward security based on the hierarchy identity-based encryption scheme (HIBE) in [10]. Second, we show the forward security with log-linear complexity $\mathcal{O}(\log(T)^2)$ using the binary tree-based approach in [15, 17], where T denotes the upper-bound of time periods (or epochs). We mainly focus on *how to generate and update keys* because they determine the forward security. We assume a secret key $\mathtt{sk}_t$ is of the following form:

$$\mathtt{sk}_t = (g^r, \underline{h^{\mathtt{sk}} \cdot (h_0 \prod h_i^{t_i})^r}, h_{i+1}^r, \cdots, h_T^r)$$

where r is a randomness, $t = t_1||t_2|| \cdots ||t_i$ denotes the current time, $(g, h_0, h_1, \cdots, h_T)$ denotes the public parameters, and $(h_0 \prod h_i^{t_i})^r = (h_0 \cdot h_1^{t_1} \cdot h_2^{t_2} \cdots h_i^{t_i})^r$. To derive a new key $\mathtt{sk}_{t'}$ at the next time $t' = t||t_{i+1}$ from $\mathtt{sk}_t$, the secret key holder performs the following operation on the underlined element above

$$h^{\mathtt{sk}} \cdot (h_0 \prod h_{i+1}^{t_{i+1}})^{r+r'} = \underline{h^{\mathtt{sk}} \cdot (h_0 \prod h_i^{t_i})^r} \cdot h_{i+1}^{r \cdot t_{i+1}} \cdot (h_0 \prod h_{i+1}^{t_{i+1}})^{r'}$$

where r' is a new randomness, and the underline part is the second element of secret key $\mathtt{sk}_t$. For the first and other elements of the new secret key $\mathtt{sk}_{t'}$, the key holder can easily update them by multiplying $g^{r'}, h_{i+2}^{r'}, \cdots, h_T^{r'}$, respectively. So, the new secret key is $\mathtt{sk}_{t'} = (g^{r+r'}, h^{\mathtt{sk}} \cdot (h_0 \prod h_{i+1}^{t_{i+1}})^{r+r'}, h_{i+2}^{r+r'}, \cdots, h_T^{r+r'})$.

The above approach shows that each key update requires new randomness to unlink the original and the new secret keys, and the complexity is linear to the number of epochs: $\mathcal{O}(T)$. Now, we use a tree-based approach [15, 17] to compress the secret keys down to $\mathcal{O}(\log(T)^2)$. First, we assume the secret key $\mathtt{sk}_t$ for the current time t is of the following form:

$$\mathtt{sk}_t = \tilde{\mathtt{sk}}_t, \tilde{\mathtt{sk}}_{t+1}, \cdots, \tilde{\mathtt{sk}}_T.$$

where each sub-key $\tilde{\mathtt{sk}}_t$ is generated using independent randomness. Second, we explain the tree-based approach. It assumes a binary tree that utilizes all tree nodes (due to the complexity gain from $\mathcal{O}(\log(T))$ to $\mathcal{O}(\log(1))$ in exponentiation). It relies on a tree traversal method, which is described in Figure 1.

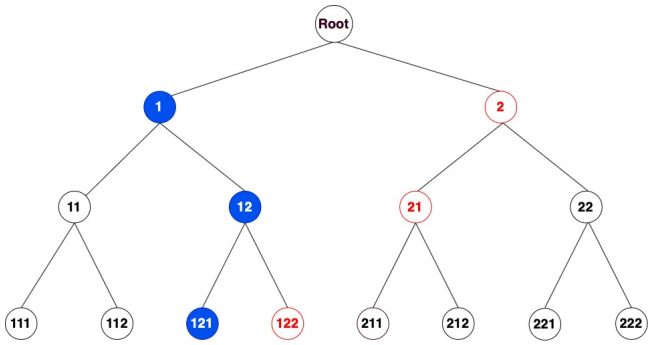

Fig. 1: Tree Traversal Method.

Specifically, a tree of depth $\ell - 1$ consists of $2^\ell - 1$ nodes, which corresponds to time periods in $[1, 2^\ell - 1]$. We use $\{1, 2\}$-string to represent time period, where 1 denotes taking the left branch and 2 denotes taking the right branch. For instance, for $\ell = 4$, the string $(\epsilon, 1, 11, \cdots, 222)$ is corresponding to time period $(1, 2, 3, \cdots, 15)$, where ϵ denotes the root node or the first time period. Suppose the current time is $t = 121$ (a leaf node in color blue in Figure 1), the tree traversal method states that the key holder will use the sub-key $\tilde{\mathtt{sk}}_{121}$ to represent time 121, and locally store the secret keys of the "right siblings" (or siblings on the right) of the nodes on the path from the root to 121 for subsequent key updates. As a result, the key holder stores a set of sub-keys at epoch 121: $\mathtt{sk}_t = (\tilde{\mathtt{sk}}_{121}, \tilde{\mathtt{sk}}_{122}, \tilde{\mathtt{sk}}_{21}, \tilde{\mathtt{sk}}_2)$. In particular, the sub-keys are organized as a stack of node keys, with the sub-key $\tilde{\mathtt{sk}}_{121}$ on top. The sub-keys at epoch 121

are described below

$$\begin{aligned}
\tilde{\mathtt{sk}}_{121} &= (g^{r_{121}}, h^{\mathtt{sk}} \cdot (h_0 \cdot h_1^1 \cdot h_2^2 \cdot h_3^1)^{r_{121}}, \perp) \\
\tilde{\mathtt{sk}}_{122} &= (g^{r_{122}}, h^{\mathtt{sk}} \cdot (h_0 \cdot h_1^1 \cdot h_2^2 \cdot h_3^2)^{r_{122}}, \perp) \\
\tilde{\mathtt{sk}}_{21} &= (g^{r_{21}}, h^{\mathtt{sk}} \cdot (h_0 \cdot h_1^2 \cdot h_2^1 \cdot h_3^0)^{r_{21}}, h_3^{r_{21}}) \\
\tilde{\mathtt{sk}}_{2} &= (g^{r_{2}}, h^{\mathtt{sk}} \cdot (h_0 \cdot h_1^2 \cdot h_2^0 \cdot h_3^0)^{r_{2}}, h_2^{r_2}, h_3^{r_2})
\end{aligned}$$

where $r_{121}, r_{122}, r_{21}, r_2$ are independent randomness, and (g, h_0, h_1, h_2, h_3) are public parameters.

Third, we show the key update from $\mathtt{sk}_t$ to $\mathtt{sk}_{t+1}$. The sub-keys $\tilde{\mathtt{sk}}_{121}, \tilde{\mathtt{sk}}_{122}$ cannot be updated further once they are used because their third elements are empty values $\perp$. But, we can derive a new sub-key $\tilde{\mathtt{sk}}_{211}$ from the sub-key $\tilde{\mathtt{sk}}_{21}$ (which is stored locally) using the following equation.

$$\begin{aligned}
\tilde{\mathtt{sk}}_{211} &= (g^{r_{21}} \cdot g^{r_{211}}, \underline{h^{\mathtt{sk}} \cdot (h_0 \cdot h_1^2 \cdot h_2^1 \cdot h_3^0)^{r_{21}}} \cdot h_3^{r_{21} \cdot 1} \cdot (h_0 \cdot h_1^2 \cdot h_2^1 \cdot h_3^1)^{r_{211}}, \perp) \\
&= (g^{r_{21}+r_{211}}, h^{\mathtt{sk}} \cdot (h_0 \cdot h_1^2 \cdot h_2^1 \cdot h_3^1)^{r_{21}+r_{211}}, \perp)
\end{aligned}$$

where r_{211} is a new randomness used in this key update. We can derive all the following sub-keys shown in the tree using the same method described above. Specifically, $\mathtt{sk}_{212}$ is derived from $\mathtt{sk}_{21}$, and $\mathtt{sk}_{22}, \mathtt{sk}_{221}, \mathtt{sk}_{222}$ are derived from $\mathtt{sk}_2$.

Next, we show the complexity of key updates, which includes storage cost and computational cost. The storage cost is $\mathcal{O}(\log(T)^2)$, meaning that each key $\mathtt{sk}_t$ contains $\mathcal{O}(\log(T))$ sub-keys and each sub-key $\tilde{\mathtt{sk}}_t$ consists of $\mathcal{O}(\log(T))$ group elements. The complexity of computational cost includes multiplications (Mul) and exponentiations (Exp). That is, each key update requires $\mathcal{O}(\log(T))$ Muls, and $\mathcal{O}(1)$ Exps (due to using all tree nodes instead of leaf nodes only).

3 Definition and Models

In this section, we present the definition and the security models of forward-secure ring signature scheme.

3.1 Definition

A forward-secure ring signature (FS-RS) scheme consists of the following algorithms.

- $\mathsf{Setup}(1^\lambda)$: It takes a security parameter λ as input, outputs public parameters PP that include the maximum number of epoch T.
- $\mathsf{KeyGen}(\mathsf{PP})$: It takes public parameters PP as input, outputs an initial key pair $(\mathtt{pk}_i, \mathtt{sk}_{(i,0)})$ for any user. We use $\mathtt{pk}_i$ to represent this user.
- $\mathsf{KeyUp}(\mathsf{PP}, \mathtt{sk}_{(i,t)}, t')$: It takes a user $\mathtt{pk}_i$'s key $\mathtt{sk}_{(i,t)}$ and an epoch t' as input, outputs an updated key $\mathtt{sk}_{(i,t')}$, where $t \leq t'$.

– $\mathsf{Sign}(\mathsf{PP}, \mathtt{sk}_{(i,t)}, m, \mathsf{PK}, t)$: It takes a user $\mathtt{pk}_i$'s key $\mathtt{sk}_{(i,t)}$, a message m, a number of public keys $\{\mathtt{pk}_j\}^l$, and an epoch t as input, outputs a signature σ. We use $\mathsf{PK} = \{\mathtt{pk}_i, \{\mathtt{pk}_j\}^l\}$ to represent a set of public keys.
– $\mathsf{Verify}(\mathsf{PP}, \mathsf{PK}, m, \sigma, t)$: It takes a message-signature pair (m, σ), a public key set PK, and an epoch t as input, outputs 1 to indicate that the signature is valid and 0 otherwise.

Correctness. The FS-RS is *correct* if for all security parameters λ, all public parameters $\mathsf{PP} \leftarrow \mathsf{Setup}(1^\lambda)$, for all keys $(\mathtt{pk}_i, \mathtt{sk}_{(i,0)}) \leftarrow \mathsf{KeyGen}(\mathsf{PP})$, for all $t \leq t'$, $\mathtt{sk}_{(i,t')} \leftarrow \mathsf{KeyUp}(\mathsf{PP}, \mathtt{sk}_{(i,t)}, t')$, for all m and $\mathsf{PK} = (\mathtt{pk}_i, \{\mathtt{pk}_j\})$, $\sigma \leftarrow \mathsf{Sign}(\mathsf{PP}, \mathtt{sk}_{(i,t)}, m, \mathsf{PK}, t)$, we have $1 = \mathsf{Verify}(\mathsf{PP}, \mathsf{PK}, m, \sigma, t)$.

3.2 Security Models

Forward-secure Unforgeability. Informally, an attacker cannot forge a message-signature pair, even if the attacker can adaptively corrupt some honest participants and obtain their epoch-based secret keys. The formal security game between a probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ and a simulator $\mathcal{S}$ is defined as follows.

– $\mathcal{S}$ sets up the game by creating n users with the corresponding key pairs $\{(\mathtt{pk}_i, \mathtt{sk}_i)\} \leftarrow \mathsf{KeyGen}(\mathsf{PP})$, where $\mathsf{PP} \leftarrow \mathsf{Setup}(1^\lambda)$. Next, $\mathcal{S}$ generates the initial secret keys $\{\mathtt{sk}_{(i,0)}\}$ for n users. For each user $\mathtt{pk}_i$, $\mathcal{S}$ can update the epoch-based secret keys to $\{\mathtt{sk}_{(i,t)}\}_{t=1}^{T}$. Eventually, $\mathcal{S}$ returns all public keys to $\mathcal{A}$, and $\mathcal{S}$ maintains a set $\mathcal{Q}$ to record the corrupted users.
– During the game, $\mathcal{A}$ can make the following queries to $\mathcal{S}$.
  - Key Update. If $\mathcal{A}$ issues a key update query with respect to a user $\mathtt{pk}_i$ at epoch t, then $\mathcal{S}$ updates the key $\mathtt{sk}_{(i,t)}$ to $\mathtt{sk}_{(i,t+1)}$ and increases t, where $t \leq T$. The key update should be queried in a non-decreasing order of epoch.
  - Signing. If $\mathcal{A}$ issues a signing query on a message m and a public key set $\mathsf{PK} = \{\mathtt{pk}_i, \cdots\}$ at epoch t, then $\mathcal{S}$ computes a signature σ using the secret key $\mathtt{sk}_{(i,t)}$ and returns it to $\mathcal{A}$.
  - Break In. If $\mathcal{A}$ issues a break-in query at epoch $\bar{t}$ with respect to a user $\mathtt{pk}_i$, then $\mathcal{S}$ returns the corresponding secret key $\mathtt{sk}_{(i,\bar{t})}$ to $\mathcal{A}$. This query can be issued once for each user, and after this query, $\mathcal{A}$ can make no further key update or signing queries to that user. In addition, we allow $\mathcal{A}$ to issue different break-in queries with respect to different users.
  - Corrupt. If $\mathcal{A}$ issues a corrupt query on a user $\mathtt{pk}_i$, then $\mathcal{S}$ returns the user's initial secret key $\mathtt{sk}_{(i,0)}$ to $\mathcal{A}$, and updates the set $\mathcal{Q}$ by including the corrupted public key $\mathtt{pk}_i$.
– At some point, $\mathcal{A}$ outputs a forgery $(\mathsf{PK}^*, t^*, m^*, \sigma^*)$. $\mathcal{A}$ wins the game if the following conditions hold.
  1. The signature σ^* is a valid under PK^* for t^* and m^*.
  2. The public key set satisfies $\mathsf{PK}^* \notin \mathcal{Q}$.
  3. The forgery $(\mathsf{PK}^*, t^*, m^*, \cdot)$ was not previously queried to the signing oracle.

4. If the break in oracle has been queried at epoch $\bar{t}$ with respect to any user in PK^*, the break in epoch must satisfy $\bar{t} > t^*$.

We define the advantage of $\mathcal{A}$ in the above game as

$$\mathtt{Adv}_{\mathcal{A}}(\lambda) = |\Pr[\mathcal{A}\ wins]|.$$

Definition 2. *The forward-secure ring-signature scheme is unforgeable if for any PPT $\mathcal{A}$, $\mathtt{Adv}_{\mathcal{A}}(\lambda)$ is a negligible function in λ.*

Strong Anonymity. Informally, an attacker cannot identify a specific signer given a valid signature and a set of public keys, even if the attacker can access all randomnesses that were used in generating each user's secret key. Note that we consider a strong anonymity model in [9] that the attacker can access all randomnesses that were used in generating each user's secret key (i.e., full key exposure). The formal security game between a PPT adversary $\mathcal{A}$ and a simulator $\mathcal{S}$ is defined as follows.

- $\mathcal{S}$ sets up the game using the same method described in the above unforgeability game except the following differences. First, $\mathcal{S}$ generates a user's key pair as $(\mathtt{pk}_i, \mathtt{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{PP}; w_i)$, where w_i denotes the randomness used in generating user's secret key. Second, $\mathcal{S}$ returns all users' public keys to $\mathcal{A}$, and tosses a random coin b which is used later in the game.
- $\mathcal{A}$ can make signing queries to $\mathcal{S}$ during the training phase. In the end, $\mathcal{A}$ outputs two indices (i_0, i_1).
- During the challenge phase, $\mathcal{A}$ can issue a signing query on a message m^* under a public key set $\mathsf{PK}^* = \{\mathtt{pk}_{i_0}, \mathtt{pk}_{i_1}, \cdots\}$ at epoch t^*, then $\mathcal{S}$ returns a signature $\sigma \leftarrow \mathsf{Sign}(\mathsf{PP}, \mathtt{sk}_{(i_b, t^*)}, m^*, \mathsf{PK}^*, t^*)$ and all witness $\{w_i\}$ to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs b' as its guess for b. If $b' = b$, then $\mathcal{S}$ outputs 1; Otherwise, $\mathcal{S}$ outputs 0. We define the advantage of $\mathcal{A}$ in the above game as

$$\mathtt{Adv}_{\mathcal{A}}(\lambda) = |\Pr[\mathcal{S} \rightarrow 1] - 1/2|.$$

Definition 3. *The forward-secure ring-signature scheme is anonymous if for any PPT $\mathcal{A}$, $\mathtt{Adv}_{\mathcal{A}}(\lambda)$ is a negligible function in λ.*

4 Our Construction

We denote an epoch-based function as $F(t) = h_0 \prod h_i^{t_i}$, where $t = t_1||t_2||\cdots||t_i = \{1,2\}^i$. Let $\mathtt{H} : \{0,1\}^* \rightarrow \mathbb{Z}_q$ be a collision-resistant hash function. Below, we show a construction with linear-complexity; one can use the tree-based approach described in Section 2.3 to convert it to achieve log-linear complexity.

- $\mathsf{Setup}(1^\lambda)$: Let $\hat{\mathsf{e}} : \mathbb{G} \times \mathbb{H} \rightarrow \mathbb{G}_T$ be a bilinear pairing. The common system parameters include $\mathsf{PP} = (g, h, T, \{h_i\}^\ell)$, where $g \in \mathbb{G}$, $h, \{h_i\}^\ell \in \mathbb{H}$, and $T = 2^\ell - 1$ denotes the upper bound of epochs. The first epoch is $\epsilon = 0$, and the last epoch is $t_1||\cdots||t_{\ell-1}$.

– $\mathsf{KeyGen}(\mathsf{PP})$: A user chooses a secret key $\mathtt{sk}_i$ and computes $h^{\mathtt{sk}_i}$. It computes an initial key as $\mathtt{sk}_{(i,\epsilon)} = (g^{r_0}, h^{\mathtt{sk}_i} \cdot h_0^{r_0}, h_1^{r_0}, \cdots h_\ell^{r_0})$, where $r_0 \in \mathbb{Z}_q$. It also sets its public key as $\mathtt{pk}_i = g^{\mathtt{sk}_i}$. We denote the second element of $\mathtt{sk}_{(i,\epsilon)}$ as $\mathtt{sk}_{(i,\epsilon,2)}$.
– $\mathsf{KeyUp}(\mathsf{PP}, \mathtt{sk}_{(i,t)}, t')$: Given a key $\mathtt{sk}_{(i,t)} = (g^r, h^{\mathtt{sk}_i} \cdot F(t)^r, h_{i+1}^r, \cdots, h_\ell^r)$, where $t = t_1||\cdots||t_i$, the user creates a new key $\mathtt{sk}_{(i,t')} = (g^r \cdot g^z, \mathtt{sk}_{(i,t,2)} \cdot h_{i+1}^{r \cdot t_{i+1}} \cdot (h_0 \cdot \prod h_{i+1}^{t'})^z, h_{i+2}^r \cdot h_{i+2}^z, \cdots, h_\ell^r \cdot h_\ell^z)$, where $z \in \mathbb{Z}_q$, and epoch $t' = t_1||\cdots||t_i||t_{i+1}$.
– $\mathsf{Sign}(\mathsf{PP}, \mathtt{sk}_{(i,t)}, m, \mathsf{PK}, t)$: Given a signing key $\mathtt{sk}_{(i,t)}$, a message m, and a set of public keys $\{\mathtt{pk}_j\}^l$ (note that $l < \ell$), a signer performs the following operations.
  1. Choose challenge values $\{c_j\}^l \in \mathbb{Z}_q$, and compute a commitment value $R = \hat{\mathsf{e}}(\prod \mathtt{pk}_j^{c_j}, h)/\hat{\mathsf{e}}(g, F(t))^{\widehat{r}}$, where $\widehat{r} = \widehat{r_1} + \widehat{r_2}$, and $\widehat{r_1}, \widehat{r_2} \in \mathbb{Z}_q$.
  2. Compute a challenge value $c = \mathtt{H}(R||m||\mathsf{PK}) - \sum c_j$, where $\mathsf{PK} = (\mathtt{pk}_i||\{\mathtt{pk}_j\}_{j \neq i}^l)$.
  3. Output a ring signature $\sigma = (\sigma_1, \sigma_2, \{c, c_j\}^{l+1})$, where $\sigma_1 = [h^{\mathtt{sk}_i} \cdot F(t)^r]^c \cdot F(t)^{\widehat{r_1}}$ and $\sigma_2 = g^{\widehat{r_2}}/g^{r \cdot c}$.
– $\mathsf{Verify}(\mathsf{PP}, \mathsf{PK}, m, \sigma, t)$: Anyone can verify $\mathtt{H}(R'||m||\mathsf{PK}) \stackrel{?}{=} c + \sum c_j$, where R' is computed as follows.

$$
\begin{aligned}
A &= \hat{\mathsf{e}}(g, \sigma_1) = \hat{\mathsf{e}}(g, h^{\mathtt{sk}_i})^c \cdot \hat{\mathsf{e}}(g, F(t)^r)^c \cdot \hat{\mathsf{e}}(g, F(t))^{\widehat{t_1}} \\
B &= \hat{\mathsf{e}}(\sigma_2, F(t)) = \hat{\mathsf{e}}(g^{\widehat{r_2}}, F(t))/\hat{\mathsf{e}}(g^{rc}, F(t)) \\
AB &= \hat{\mathsf{e}}(g, h^{\mathtt{sk}_i})^c \cdot \hat{\mathsf{e}}(g^{\widehat{r}}, F(t)), \triangleright\ \widehat{r} = \widehat{r_1} + \widehat{r_2} \\
C &= \hat{\mathsf{e}}(\mathtt{pk}_i^c \prod \mathtt{pk}_j^{c_j}, h) \\
R' &= C/AB = \hat{\mathsf{e}}(\prod \mathtt{pk}_j^{c_j}, h)/\hat{\mathsf{e}}(g, F(t))^{\widehat{r}}
\end{aligned}
$$

Correctness. We associate a user $\mathtt{pk}_i$'s signing key at epoch $t = t_1||\cdots||t_i$ of the form

$$\mathtt{sk}_{(i,t)} = (c, d, e_{i+1}, \cdots, e_\ell) = (g^r, h^{\mathtt{sk}_i} \cdot (h_0 \cdot \prod h_i^{t_i})^r, h_{i+1}^r, \cdots, h_\ell^r) \quad (1)$$

where r is an independent uniformly distributed exponent. We say that a signing key $\mathtt{sk}_{(i,t)}$ is *well-formed* if it satisfies the equation (1). Now, we show the honestly generated and updated secret keys are well-formed. For simplicity, we assume a key update from epoch $t = t_1||\cdots||t_i$ to $t' = t_1||\cdots||t_i||t_{i+1}$, where t' contains t as a prefix (e.g., $t = 12$ and $t' = 121$ or $t' = 122$). Note that the epoch cannot contain bit 0 due to technical reasons (e.g., $h_i^0 = 1$).

First, the initial key $\mathtt{sk}_{(i,\epsilon)}$ for $\epsilon = 1$ is trivially well-formed. Then, we show that the key $\mathtt{sk}_{(i,t')}$ is also well-formed after a key update from t to t'. Specifically, we show two cases of key update. The first case is of the form

$$\mathtt{sk}_{(i,t')} = (c, d \cdot e_{i+1}^{t_{i+1}}, e_{i+2}, \cdots, e_\ell) = (g^r, h^{\mathtt{sk}_i} \cdot (h_0 \cdot \prod h_{i+1}^{t_{i+1}})^r, h_{i+2}^r, \cdots, h_\ell^r)$$

which satisfies equation (1) with an independent randomness r. The second case is of the form

$$\begin{aligned}\mathtt{sk}_{(i,t')} &= (c \cdot g^z, d \cdot e_{i+1}^{t_{i+1}} \cdot (h_0 \cdot \prod h_{i+1}^{t'})^z, e_{i+2} \cdot h_{i+2}^z, \cdots, e_\ell \cdot h_\ell^z) \\ &= (g^{r+z}, h^{\mathtt{sk}_i} \cdot (h_0 \cdot \prod h_i^{t_i})^r \cdot h_{i+1}^{r \cdot t_{i+1}} \cdot (h_0 \cdot \prod h_{i+1}^{t'})^z, h_{i+2}^{r+z}, \cdots, h_\ell^{r+z}) \\ &= (g^{r+z}, h^{\mathtt{sk}_i} \cdot (h_0 \cdot \prod h_{i+1}^{t'})^{r+z}, h_{i+2}^{r+z}, \cdots, h_\ell^{r+z}).\end{aligned}$$

The above form also satisfies equation (1) with randomness $r+z$, which is an independent exponent due to the uniform choice of z. The last step to obtain $\mathtt{sk}_{(i,t')}$ is crucial to forward security, the signer deletes $\mathtt{sk}_{(i,t)}$ and the re-randomization exponent z used in the second case of key update. The verification of signatures for epoch $t' = t_1 || \cdots || t_{i+1}$ is straightforward. The signer generates a signature using $F(t')$, while the verifier computes $F(t')$ and uses it in computing B of the Verify algorithm.

4.1 Security Analysis

Theorem 1. *The FS-DR signature scheme Σ is EUF-CMA secure if the wB-DHI assumption holds in the underlying asymmetric groups.*

Proof. We define a sequence of games $\mathbb{G}_i$, $i = 0, \cdots, 2$ and let $\mathtt{Adv}_i^\Sigma$ denote the advantage of the adversary in game $\mathbb{G}_i$. Assume that $\mathcal{A}$ issues at most q signing queries in each game.

- $\mathbb{G}_0$: This is original unforgeability game.
- $\mathbb{G}_1$: This game is identical to game $\mathbb{G}_0$ except the following difference: $\mathcal{S}$ randomly chooses g as a guess for a forgery at epoch t^* with respect to user $\mathtt{pk}_i$. $\mathcal{S}$ will output a random bit if $\mathcal{A}$'s forgery does not occur in the g-th query. In this game, $\mathcal{S}$ honestly generates all initial signing keys during setup. In particular, $\mathcal{S}$ sets the break in epoch as $\bar{t} = t^* + 1$. If $\mathcal{A}$ issues a break-in query at epoch $\bar{t}'$ with respect to user $\mathtt{pk}_i$, such that $\bar{t}' \geq \bar{t}$, then $\mathcal{S}$ returns $\mathtt{sk}_{(i,\bar{t}')}$ to $\mathcal{A}$. Since at most T epochs and n users exist in the system, we have
$$\mathtt{Adv}_0^\Sigma = q \cdot T \cdot n \cdot \mathtt{Adv}_1^\Sigma$$
- $\mathbb{G}_2$: This game is identical to game $\mathbb{G}_1$ except that in the g-th session, $\mathcal{S}$ outputs a random bit if a **Forge** event happens where $\mathcal{A}$'s forgery is valid at epoch t^* under a public key set PK^* (that includes $\mathtt{pk}_i$) while the corresponding signing key $\mathtt{sk}_{(i,t^*)}$ is not corrupted. Then we have
$$\left|\mathtt{Adv}_1^\Sigma - \mathtt{Adv}_2^\Sigma\right| \leq \Pr[\textbf{Forge}].$$
Let $\mathcal{S}$ be a challenger, who is given $(g, h, g^a, g^b, h^a, h^b, \cdots, h^{b^\ell}, \hat{\mathtt{e}})$, aiming to compute $\hat{\mathtt{e}}(g,h)^{ab^{\ell+1}}$. $\mathcal{S}$ sets up the game for $\mathcal{A}$ by creating n users and T

epochs. We assume that each epoch $t = t_1||t_2||\cdots||t_\ell$ is a $\{1,2\}$-string of length ℓ. $\mathcal{S}$ pads zeros if an epoch's length is less than ℓ. $\mathcal{S}$ randomly selects a challenge user and sets its public key as $\mathtt{pk}_i = g^b$. $\mathcal{S}$ honestly generates key pairs for n-1 users. To complete the setup, $\mathcal{S}$ computes the system parameters as $h = h^{b^\ell} \cdot \bar{H}^\gamma$, $h_1 = \bar{H}^{\gamma_1}/h^{b^\ell}, \cdots, h_\ell = \bar{H}^{\gamma_\ell}/h^b$, and $h_0 = \bar{H}^\delta \cdot h^{b^\ell \cdot t_1^*} \cdot \cdots h^{b \cdot t_\ell^*}$, where $t^* = t_1^*||t_2^*||\cdots||t_\ell^*$, and $\gamma, \gamma_1, \cdots \gamma_\ell, \delta, \bar{z} \in \mathbb{Z}_q, \bar{H} = h^{\bar{z}} \in \mathbb{H}$. $\mathcal{S}$ also sets the commitment involved in the g-th session as $R^* = \hat{\mathsf{e}}(g^a, F(t^*)^{\widehat{r_1^*}})$, where $F(t^*) = h_0 \cdot h_1^{t_1^*} \cdots h_\ell^{t_\ell^*}$. Note that $\widehat{r_2^*}$ is implicitly set as a, and $\widehat{r_1^*}$ is chosen by $\mathcal{A}$. Also note that the value $h^{b^{\ell+1}} \cdot \bar{H}^{b \cdot \gamma}$ associated with user $\mathtt{pk}_i$'s signing key is unknown to $\mathcal{S}$.

During the game, $\mathcal{S}$ can honestly answer $\mathcal{A}$'s corrupt queries with respect to all users except the challenge user $\mathtt{pk}_i$. If $\mathcal{A}$ queries corrupt oracle on $\mathtt{pk}_i$, $\mathcal{S}$ aborts. Next, we show $\mathcal{S}$ can simulate a signing key at epoch $t = t_1||\cdots||t_k||\cdots||t_\ell$, where $k \in [1, \ell]$. Note that $t_k \neq t_k^*$ means that t is not prefix of t^*, and k is the smallest index at epoch t.

Specifically, $\mathcal{S}$ first chooses $z \in \mathbb{Z}_q$, and sets $r = \frac{b^k}{t_k - t_k^*} + z$. Then, $\mathcal{S}$ computes a signing key with the following form

$$(g^r, h^b \cdot \underline{(h_0 \cdot h_1^{t_1} \cdots h_k^{t_k})^r}, h_{k+1}^r, \cdots, h_\ell^r) \tag{2}$$

This is a well-formed key for epoch $t = t_1||\cdots||t_k$. We show that $\mathcal{S}$ can compute the underline term in (2).

$$\begin{aligned}
(h_0 \cdot h_1^{t_1} \cdots h_k^{t_k})^r &= [\bar{H}^\delta \cdot h^{b^\ell \cdot t_1^*} \cdot \cdots h^{b \cdot t_\ell^*} \cdot (\bar{H}^{\gamma_1}/h^{b^\ell})^{t_1} \cdots (\bar{H}^{\gamma_k}/h^{b^{\ell-k+1}})^{t_k}]^r \\
&= [\bar{H}^{\delta + \Sigma_{i=1}^k t_i \cdot \gamma_i} \cdot \prod_{i=1}^{k-1} h_{\ell-i+1}^{t_i^* - t_i} \cdot h_{\ell-k+1}^{t_k^* - t_k} \cdot \prod_{i=k+1}^{\ell} h_{\ell-i+1}^{t_i^*}]^r \\
&= Z \cdot h_{\ell-k+1}^{r(t_k^* - t_k)}
\end{aligned}$$

where Z is shown as follows

$$Z = [\bar{H}^{\delta + \Sigma_{i=1}^k t_i \cdot \gamma_i} \cdot \underline{\prod_{i=1}^{k-1} h_{\ell-i+1}^{t_i^* - t_i}} \cdot \prod_{i=k+1}^{\ell} h_{\ell-i+1}^{t_i^*}]^r$$

$\mathcal{S}$ can compute all the terms in Z and the underline term in Z is equal to 1 because $t_i = t_i^*$ for all $i < k$. The remaining term in $(h_0 \cdot h_1^{t_1} \cdots h_k^{t_k})^r$ is $h_{\ell-k+1}^{r(t_k^* - t_k)}$. Since we set $r = \frac{b^k}{t_k - t_k^*} + z$, we rewrite it as follows

$$h_{\ell-k+1}^{r \cdot (t_k^* - t_k)} = h_{\ell-k+1}^{z(t_k^* - t_k)} \cdot h_{\ell-k+1}^{(t_k^* - t_k)\frac{b^k}{t_k - t_k^*}} = \frac{h_{\ell-k+1}^{z(t_k^* - t_k)}}{h^{b^{\ell+1}}}$$

Hence, the second element in (2) is equal to

$$h^b \cdot \underline{(h_0 \cdot h_1^{t_1} \cdots h_k^{t_k})^r} = h^{b^{\ell+1}} \cdot \bar{H}^{b \cdot \gamma} \cdot Z \cdot \frac{h_{\ell-k+1}^{z(t_k^* - t_k)}}{h^{b^{\ell+1}}} = \bar{H}^{b \cdot \gamma} \cdot Z \cdot h_{\ell-k+1}^{z(t_k^* - t_k)}$$

To this end, $\mathcal{S}$ can simulate the second element in (2) because the unknown value $h^{b^{\ell+1}}$ is cancelled out. Besides, the first element g^r in (2), and other elements $(h_{k+1}^r, \cdots, h_\ell^r)$ can be easily computed by $\mathcal{S}$ since they do not involve $h^{b^{\ell+1}}$. This completes the simulation of signing key at epoch $t \neq t^*$. $\mathcal{S}$ can simulate signing queries on various messages using the simulated signing keys at epoch $t \neq t^*$.

Another case is that $\mathcal{S}$ can simulate message-signature pairs at epoch t^*. If $\mathcal{A}$ issues a signing query on a message m for a public key set $\mathsf{PK} = \{\mathtt{pk}_i || \{\mathtt{pk}_j\}^l\}$ (note that if $\mathtt{pk}_i \notin \mathsf{PK}$, $\mathcal{S}$ aborts) at epoch t^*, $\mathcal{S}$ simulates a valid signature using a similar approach described in the simulation of Schnorr signature. $\mathcal{S}$ performs the following operations.

- Choose $c, \{c_j\}^l, \widehat{r_1}, \widehat{r_2} \in \mathbb{Z}_q$ and $h^* \in \mathbb{H}$, compute $\sigma_1 = h^* \cdot F(t^*)^{\widehat{r_1}}, \sigma_2 = g^{\widehat{r_2}}$.
- Set $c = \mathtt{H}(R||m||\mathsf{PK}) - \sum c_j$, where $R = \frac{\hat{\mathsf{e}}(\mathtt{pk}_i^c \cdot \prod \mathtt{pk}_j^{c_j}, h)}{\hat{\mathsf{e}}(g, h^* \cdot F(t^*)^{\widehat{r}})}$ and $\widehat{r} = \widehat{r_1} + \widehat{r_2}$.
- Return (m, σ) to $\mathcal{A}$, where $\sigma = (c, \{c_j\}^l, \sigma_1, \sigma_2)$.

For key update, $\mathcal{S}$ keeps track of the current epoch t without returning anything to $\mathcal{A}$. For break in query, $\mathcal{S}$ needs to simulate a signing key $\mathtt{sk}_{(i,\bar{t})}$ with respect to user $\mathtt{pk}_i$, such that $t^* < \bar{t}$. $\mathcal{S}$ can simulate $\mathtt{sk}_{(i,\bar{t})}$ using the same method described in the case of $t \neq t^*$, and return it to $\mathcal{A}$.

At some point, if $\mathcal{A}$ outputs a forgery on a message m^* for a public key set PK^* and t^* in the form of $(m^*, c^*, \{c_j^*\}^l, \sigma_1^*, \sigma_2^*)$, such that

$$\sigma_1^* = [h^{b^{\ell+1}} \cdot \bar{H}^{b \cdot \gamma} \cdot \bar{H}^{r^*(\delta + \sum_{i=1}^{|t^*|} \gamma_i \cdot t_i^*)}]^{c^*} \cdot (\bar{H}^{\delta + \sum_{i=1}^{|t^*|} \gamma_i \cdot t_i^*})^{\widehat{r_1^*}}$$
$$\sigma_2^* = g^a / g^{r^* \cdot c^*}$$

where $c^* = \mathtt{H}(R^*||m^*||\mathsf{PK}^*) - \sum c_j^*, R^* = \hat{\mathsf{e}}(\prod \mathtt{pk}_j^{c_j^*}, h) \cdot \hat{\mathsf{e}}(g^a, (\bar{H}^{\delta + \sum_{i=1}^{|t^*|} \gamma_i \cdot t_i^*})^{\widehat{r_1^*}})$, and $r^*, \widehat{r_1^*}$ are chosen by $\mathcal{A}$, then $\mathcal{S}$ checks the following conditions.

- The forgery occurs on the g-th session.
- The public key set PK^* includes the challenge user $\mathtt{pk}_i$.
- The message-signature pair $(m^*, c^*, \{c_j^*\}^l, \sigma_1^*, \sigma_2^*)$ was not previously generated by $\mathcal{S}$.
- The signature (σ_1^*, σ_2^*) is valid on message m^* and public key set PK^* according to the Verify process.

If all the above conditions hold, $\mathcal{S}$ regards it as a valid forgery. The next step is that $\mathcal{S}$ rewinds the game according to the forking lemma [8], and obtains another valid forgery (σ_1', σ_2') with a different $c^{*'} = \mathtt{H}(R^*||m^*||\mathsf{PK}^*) - \sum c_j^*$ (note that the different value $c^{*'}$ happens with probability $1/n$). Eventually,

$\mathcal{S}$ computes the following equations

$$
\begin{aligned}
E &= (\sigma_1/\sigma_1')^{1/(c^*-c^{*'})} = h^{b^{\ell+1}} \cdot \bar{H}^{b\cdot\gamma} \cdot \bar{H}^{r^*(\delta+\sum_{i=1}^{|t^*|}\gamma_i\cdot t_i^*)} \\
F &= (\sigma_2'/\sigma_2)^{1/(c^{*'}-c^*)} = g^{r^*} \\
D &= \frac{\hat{\mathsf{e}}(g^a, E)}{\hat{\mathsf{e}}(g^a, \bar{H}^{b\cdot\gamma})\hat{\mathsf{e}}(F, h^{a\cdot(\delta+\sum_{i=1}^{|t^*|}\gamma_i\cdot t_i^*)})} \\
&= [\frac{\hat{\mathsf{e}}(g^a, h^{b^{\ell+1}})\hat{\mathsf{e}}(g^a, \bar{H}^{b\cdot\gamma})\hat{\mathsf{e}}(g^a, \bar{H}^{r^*(\delta+\sum_{i=1}^{|t^*|}\gamma_i\cdot t_i^*)})}{\hat{\mathsf{e}}(g^a, h^{b\cdot\bar{r}\cdot\gamma})\hat{\mathsf{e}}(g^{r^*}, h^{a\cdot(\delta+\sum_{i=1}^{|t^*|}\gamma_i\cdot t_i^*)})}] \\
&= \hat{\mathsf{e}}(g,h)^{ab^{\ell+1}}
\end{aligned}
$$

It is easy to see that D is the solution to the wBDHI problem. Therefore, we have

$$|\Pr[\mathbf{Forge}]| \leq \mathtt{Adv}_{\mathcal{A}}^{\mathrm{wBDHI}}(\lambda).$$

By combining the above results together, we have

$$\mathtt{Adv}_{\mathcal{A}}^{\Sigma}(\lambda) \leq q \cdot T \cdot n \cdot \mathtt{Adv}_{\mathcal{A}}^{\mathrm{wBDHI}}(\lambda).$$

Theorem 2. *The FS-DR signature scheme Σ is anonymous in the random oracle model.*

Proof. The simulation is performed between an adversary $\mathcal{A}$ and a simulator $\mathcal{S}$. The goal of simulator $\mathcal{S}$ is to break the strong anonymity. In this simulation, $\mathcal{S}$ simulates $\mathtt{H}$ as a random oracle.

$\mathcal{S}$ setups the game for $\mathcal{A}$ by creating n users with the corresponding key pairs $\{(\mathtt{pk}_i, \mathtt{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{PP}; w_i)\}$, where $\mathsf{PP} \leftarrow \mathsf{Setup}(1^\lambda)$. $\mathcal{S}$ gives $\{\mathtt{pk}_i\}^n$ to $\mathcal{A}$ to $\mathcal{A}$. $\mathcal{S}$ also chooses a random bit b.

During the training phase, if $\mathcal{A}$ issues a signing query on a message m, a set of public keys PK with the signer index j at epoch t, then $\mathcal{S}$ generates $\sigma \leftarrow \mathsf{Sign}(\mathsf{PP}, \mathtt{sk}_{(j,t)}, m, \mathsf{PK}, t)$ and returns it to $\mathcal{A}$.

During the challenge phase, if $\mathcal{A}$ issues a signing query on a message m^*, a set of public keys PK^*, two indices (i_0, i_1) and an epoch t^*, then $\mathcal{S}$ simulates the signature $\sigma^* = (\sigma_1^*, \sigma_2^*, c^*, \{c_j\}^l)$ using the same method described in the above game $\mathbb{G}_2$ (i.e., the case of $t = t^*$). Eventually, $\mathcal{S}$ returns σ^* and $\{w_i\}^n$ to $\mathcal{A}$. Recall that in the simulation of signature σ^*, $\mathcal{S}$ picks $c^*, \{c_j\}^l$ at random in $\mathbb{Z}_q$, and sets $c = \mathtt{H}(R^*||m^*||\mathsf{PK}^*) - \sum_{j=1}^{l} c_j$ in the random oracle. The distribution of message-signature pair (m^*, σ^*) is correct. Note that the commutative operation $c + \sum_{j=1}^{l} c_j$ is also uniformly distributed in $\mathbb{Z}_q$, and $\mathcal{S}$ aborts if the hash value $\mathtt{H}(R^*||m^*||\mathsf{PK}^*)$ is already set by the random oracle $\mathtt{H}$.

Finally, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs. Since b is not used in the simulation of message-signature pair in the challenge phase (i.e., $\mathcal{S}$ simulates a valid signature without using the signing key $\mathtt{sk}_{(i_b,t^*)}$), $\mathcal{A}$ wins only with probability $1/2$.

5 Extension

Extending our construction, we now introduce a forward-secure linkable Dual-Ring. The linkability means that anyone can link multiple signatures generated by a same signer. Based on the technique used in [25], we adapt the proposed FS-DR as follows

- The setup is almost same as FS-DR, except that the algorithm additionally generates a one-time signature scheme $\Sigma_{ots} = (\mathsf{OKGen}, \mathsf{OSig}, \mathsf{OVer})$.
- The key generation proceeds as follows. A user generates a key pair $(\mathtt{osk}, \mathtt{opk}) \leftarrow \mathsf{OKGen}(1^\lambda)$, computes a linkability tag $R_i = \mathtt{H}(\mathtt{opk})$. The user's secret key is of the form $\mathtt{sk}_{(i,t)} = (g^r, h^{\mathtt{sk}_i + R_i} \cdot h_0^r, h_1^r, \cdots h_\ell^r)$, where $r \in \mathbb{Z}_q$. The user's public key is $\mathtt{pk}_i = g^{\mathtt{sk}_i + R_i}$. The key update remains the same as FS-DR.
- For signing, a signer with a signing key $\mathtt{sk}_{(i,t)}$, a message m, and a set of public keys $\{\mathtt{pk}_j\}^l$, performs the following operations.
  1. Generate a new set of public keys using its linkability tag R_i, such that $\mathtt{pk}'_i = \mathtt{pk}_i / g^{R_i} = g^{\mathtt{sk}_i}$, and $\mathtt{pk}'_j = \mathtt{pk}_j / g^{R_i} = g^{\mathtt{sk}_j - R_i}$.
  2. Choose challenge values $\{c_j\}^l \in \mathbb{Z}_q$, and compute a commitment value $R = \hat{\mathsf{e}}(\prod \mathtt{pk}'^{c_j}_j, h) / \hat{\mathsf{e}}(g, F(t))^{\widehat{r}}$, where $\widehat{r} = \widehat{r_1} + \widehat{r_2}$, and $\widehat{r_1}, \widehat{r_2} \in \mathbb{Z}_q$.
  3. Compute a challenge value $c = \mathtt{H}(R || m || \mathsf{PK}') - \sum c_j$, where $\mathsf{PK}' = (\mathtt{pk}'_i || \{\mathtt{pk}'_j\}^l_{j \neq i})$.
  4. Generate a ring signature $\sigma = (\sigma_1, \sigma_2, \{c, c_j\}^{l+1})$, where $\sigma_1 = [h^{\mathtt{sk}_i} \cdot F(t)^r]^c \cdot F(t)^{\widehat{r_1}}$ and $\sigma_2 = g^{\widehat{r_2}} / g^{r \cdot c}$. Note that $h^{\mathtt{sk}_i} \cdot F(t)^r = \frac{h^{\mathtt{sk}_i + R_i} \cdot F(t)^r}{h^{R_i}}$.
  5. Generate a one-time signature $s \leftarrow \mathsf{OSig}(\mathtt{osk}; m, \sigma, \mathsf{PK})$.
  6. Output $(\mathsf{PK}, m, \sigma, \mathtt{opk}, s)$.
- For verification, anyone first computes $\mathsf{PK}' = (\mathtt{pk}'_i || \{\mathtt{pk}'_j\}^l_{j \neq i})$ from the public-key set $\mathsf{PK} = (\mathtt{pk}_i, \{\mathtt{pk}_j\}^l)$ and g^{R_i}, where $R_i = \mathtt{H}(\mathtt{opk})$. Next, the user runs the Verify algorithm described in FS-DR under public key set PK'. Last, the user verifies the signature $1 \leftarrow \mathsf{OVer}(\mathtt{opk}; m, \sigma, \mathsf{PK})$.
- The link process takes two message-signature pairs $(\mathsf{PK}_1, m_1, \sigma_1, \mathtt{opk}_1, s_1)$, $(\mathsf{PK}_2, m_2, \sigma_2, \mathtt{opk}_2, s_2)$ as input, output either *linked* or *unlinked*. Specifically, the algorithm first verify (m_1, σ_1) under PK_1 and (m_2, σ_2) under PK_2, respectively. Then, the algorithm outputs *linked* if $\mathtt{opk}_1 = \mathtt{opk}_2$. Otherwise, it outputs *unlinked*.

Correctness and Security. The correctness of linkable FS-DR is held if: 1) the FS-DR and the one-time signature Σ_{ots} are correct. 2) two legally signed signatures are linked if they share a same signer. The security of linkable FS-DR should include the following aspects.

- *Forward-secure Unforgeability.* The forward-secure unforgeability for linkable FS-DR remains the same as in Section 3.2.
- *Forward-secure Anonymity.* Informally, an attacker cannot identify a specific signer given a valid signature and a set of public keys at epoch t^*, even if the attacker can corrupt all users' secret keys after t^*. The formal definition is adopted from Boyen and Haines [14]. We claim that, the linkable FS-DR

is forward-secure anonymous in the random oracle model if the decisional wBDHI is held in the asymmetric pairing group. The security proof is similar to Theorem 5 described in [14], except that the hard problem is replaced by the decisional wBDHI problem.

– *Linkability and Non-slanderability.* Linkability means that the link process always outputs "linked" for two signatures generated by a same signer. The non-slanderability states that a signer cannot frame other honest signers for generating a signature linked with another signature not signed by the signer. The formal definitions are adopted from Liu et al. [22]. We claim that, the linkable FS-DR is linkable and non-slanderable in the random oracle model if the FS-DR scheme and the one-time signature scheme Σ_{ots} are unforgeable. This assumption is valid because if a linkable ring signature scheme is linkable and non-slanderable, it is also unforgeable [6]. The security proofs are similar to Theorem 4 and 5 described in [25].

6 Implementation and Evaluation

In this section, we focus on the implementation and evaluation. Specifically, we compare the proposed scheme with a closely related research work [14] in terms of execution time and storage cost. First, we remove the linkability described in [14] for a fair comparison. We stress that the extension to a linkable FS-DualRing is not our major contribution. Second, we remove the implementation of the forward-secure key update described in [14]. They use multilinear maps [11] to update key pairs for different time periods or epochs. But, multilinear maps are not available in practice due to various attacks [14]. Therefore, they suggest using (symmetric) bilinear maps to give a forward-secure scheme that supports a key update for two epochs.

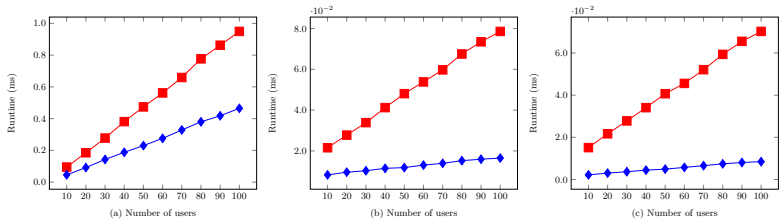

Fig. 2: Execution time of KeyGen, Sign, Verify algorithms. Red (with solid square) is for our scheme, Blue (with solid dot) is for [14].

We implement our proposed scheme using Charm framework [5] and evaluate its performance on a PC with Intel Core i9. We use MNT224 curve [26] for pairing, which is the commonly-used asymmetric pairing in PBC library, and it has around 100-bit security level. Our source code is available at Github [2].

First, we provide a performance comparison between our scheme and [14]. The execution time of KeyGen, Sign, and Verify algorithms are shown in Figure 2. The execution time of our scheme is relatively slow compared to [14]. This is because their pairing relies on the symmetric SS512 curve, which is short (thus fast). But, the security level of SS512 is incomparable to MNT224 (note that our scheme is insecure in the symmetric pairing setting). We stress that our scheme's execution time is acceptable. The signing and verifying processes take approximately 0.07ms to handle a group of 100 public keys.

Second, we focus on the storage cost and provide a comparison in Table 1. We evaluate the size of the signing key, public key, and signature. One can see that the public/signing key size between [24] and our proposed scheme are close because these two works rely on the same forward-secure technique described in Section 2.3. However, our scheme's signature size is much smaller compared to [24]. The signature size in [24] is linear to the number of public keys because their construction is based on the classic ring signature scheme [27]. Our proposed scheme can save the storage cost significantly for ring signature schemes with relatively small challenges and large signatures. Further, our scheme ensures a $\mathcal{O}(\log n)$-size argument of knowledge in the DL setting. The signature size in [14] is also larger than our proposed scheme because their resulting signature involves group elements of $\mathbb{G}_T$ (note that the size of an element of $\mathbb{G}_T$ is usually larger than the size of $\mathbb{G}/\mathbb{H}$ on commonly-used curves). Besides, the public key in [14] is not a single group element as in [24] and our proposed scheme. To conclude, our proposed scheme is practical in terms of execution time and storage cost.

Table 1: Storage comparison with two existing works. n denotes the number of public keys involved in a signature. T denotes the upper bound for time periods. Subgroup means subgroup decision problem. (k_1, b)-GMDP means generalized multilinear decoding problem, where k_1 is a combinatorial constant, and b (we assume $b = 1$ here) is an initial public key level. (k_1, b)-(GMDDH) means that generalized sub-exponent multilinear decisional Diffie–Hellman problem.

	$\lvert\mathtt{pk}\rvert$	$\lvert\mathtt{sk}\rvert$	$\lvert\sigma\rvert$	Assumption
[24]:	$\mathbb{G}$	$(2 + (\log(T))^2)\mathbb{G}$	$(2n + 3)\mathbb{G}$	CDH/Subgroup
[14]:	$k_1\mathbb{G}$	$k_1\mathbb{Z}_q + 2k_1\mathbb{G}$	$\mathbb{Z}_q + \mathbb{G}_T + n\mathbb{Z}_q$	(k_1, b)-GMDP/GMDDH
Ours:	$\mathbb{G}$	$\mathbb{G} + \mathbb{H} + (\log(T))^2\mathbb{H}$	$\mathbb{G} + \mathbb{H} + n\mathbb{Z}_q$	wBDHI

7 Conclusion

In this work, we proposed a forward-secure DualRing scheme and extended it to a forward-secure linkable DualRing scheme. We relied on a non-interactive key update mechanism described in the hierarchy identity-based encryption (HIBE) [15, 17] to ensure forward security. We proposed a novel "split-and-combine" method in building our schemes. This method is suitable for the Type-T based (or random oracle based) signature schemes such as DualRing [29].

References

1. Facebook Whistleblower. https://www.nbcnews.com/tech/tech-news/facebook-whistleblower-documents-detail-deep-look-facebook-rcna3580.
2. Our Source Code. https://github.com/SMC-SMU/Forward-secure-DualRing.
3. M. Abe, M. Ohkubo, and K. Suzuki. 1-out-of-n signatures from a variety of keys. In *ASIACRYPT*, pages 415–432, 2002.
4. B. Adida. Helios: Web-based open-audit voting. In *USENIX security symposium*, volume 17, pages 335–348, 2008.
5. J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin. Charm: a framework for rapidly prototyping cryptosystems. *Journal of Cryptographic Engineering*, 3(2):111–128, 2013.
6. M. H. Au, W. Susilo, and S.-M. Yiu. Event-oriented k-times revocable-iff-linked group signatures. In *ACISP*, pages 223–234, 2006.
7. M. Bellare and S. K. Miner. A forward-secure digital signature scheme. In *CRYPTO*, pages 431–448, 1999.
8. M. Bellare and G. Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *CCS*, pages 390–399, 2006.
9. A. Bender, J. Katz, and R. Morselli. Ring signatures: Stronger definitions, and constructions without random oracles. In *TCC*, pages 60–79, 2006.
10. D. Boneh, X. Boyen, and E.-J. Goh. Hierarchical identity based encryption with constant size ciphertext. In *CRYPTO*, pages 440–456, 2005.
11. D. Boneh and A. Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics*, 324(1):71–90, 2003.
12. J. Bootle, A. Cerulli, P. Chaidos, E. Ghadafi, J. Groth, and C. Petit. Short accountable ring signatures based on ddh. In *ESORICS*, pages 243–265, 2015.
13. C. Boyd and K. Gellert. A modern view on forward security. *The Computer Journal*, 64(4):639–652, 2021.
14. X. Boyen and T. Haines. Forward-secure linkable ring signatures. In *ACISP*, pages 245–264, 2018.
15. R. Canetti, S. Halevi, and J. Katz. A forward-secure public-key encryption scheme. In *EUROCRYPT*, pages 255–271, 2003.
16. Y. Dodis, A. Kiayias, A. Nicolosi, and V. Shoup. Anonymous identification in ad hoc groups. In *EUROCRYPT*, pages 609–626, 2004.
17. M. Drijvers, S. Gorbunov, G. Neven, and H. Wee. Pixel: Multi-signatures for consensus. In *USENIX*, pages 2093–2110, 2020.
18. J. Groth and M. Kohlweiss. One-out-of-many proofs: Or how to leak a secret and spend a coin. In *EUROCRYPT*, pages 253–280, 2015.
19. T. Haines and X. Boyen. Votor: conceptually simple remote voting against tiny tyrants. In *Proceedings of the Australasian Computer Science Week Multiconference*, pages 1–13, 2016.
20. R. W. Lai, V. Ronge, T. Ruffing, D. Schröder, S. A. K. Thyagarajan, and J. Wang. Omniring: Scaling private payments without trusted setup. In *ACM CCS*, pages 31–48, 2019.
21. B. Libert, T. Peters, and C. Qian. Logarithmic-size ring signatures with tight security from the ddh assumption. In *ESORICS*, pages 288–308, 2018.
22. J. K. Liu, M. H. Au, W. Susilo, and J. Zhou. Linkable ring signature with unconditional anonymity. *IEEE Transactions on Knowledge and Data Engineering*, 26(1):157–165, 2013.

23. J. K. Liu and D. S. Wong. Solutions to key exposure problem in ring signature. *Int. J. Netw. Secur.*, 6(2):170–180, 2008.
24. J. K. Liu, T. H. Yuen, and J. Zhou. Forward secure ring signature without random oracles. In *ICICS*, pages 1–14, 2011.
25. X. Lu, M. H. Au, and Z. Zhang. Raptor: a practical lattice-based (linkable) ring signature. In *ACNS*, pages 110–130, 2019.
26. A. Miyaji, M. Nakabayashi, and S. Takano. Characterization of elliptic curve traces under fr-reduction. In *ICISC*, pages 90–108, 2000.
27. R. L. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In *ASIACRYPT*, pages 552–565, 2001.
28. C.-P. Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.
29. T. H. Yuen, M. F. Esgin, J. K. Liu, M. H. Au, and Z. Ding. Dualring: Generic construction of ring signatures with efficient instantiations. In *CRYPTO*, pages 251–281, 2021.
30. T. H. Yuen, S.-f. Sun, J. K. Liu, M. H. Au, M. F. Esgin, Q. Zhang, and D. Gu. Ringct 3.0 for blockchain confidential transaction: Shorter size and stronger security. In *FC*, pages 464–483, 2020.
31. F. Zagórski, R. T. Carback, D. Chaum, J. Clark, A. Essex, and P. L. Vora. Remotegrity: Design and use of an end-to-end verifiable remote voting system. In *ACNS*, pages 441–457, 2013.