1-out-of-$n$ Signatures from a Variety of Keys

Masayuki ABE$^*$, Miyako OHKUBO$^*$, and Koutarou SUZUKI$^*$, Members

SUMMARY This paper addresses how to use public-keys of several different signature schemes to generate 1-out-of-$n$ signatures. Previously known constructions are for either RSA-type keys only or DL-type keys only. We present a widely applicable method to construct a 1-out-of-$n$ signature scheme that allows mixture use of different flavors of keys at the same time. The resulting scheme is more efficient than previous schemes even if it is used only with a single type of keys. With all DL-type keys, it yields shorter signatures than the ones of the previously known scheme based on the witness indistinguishable proofs by Cramer, et al. With all RSA-type keys, it reduces both computational and storage costs compared to that of the Ring signatures by Rivest, et al.

Key words: signer ambiguity, ring signature, discrete-log

1. Introduction

A 1-out-of-$n$ signature convinces a verifier that a document is signed by one of $n$ possible independent signers without allowing the verifier to identify which signer it was. It can be seen as a simple group signature that has no group manager who can revoke the identity of the signer in case of emergency. Such a signature can also be seen as a kind of non-interactive proof that the signer owns a witness (secret-key) that corresponds to one of $n$ commitments (public-keys) or theorems without leaking which one it really is. Such a primitive, as a signature scheme and/or a proof system, plays a central role in variety of applications such as group signatures [6], [9], designated verifier signatures [21], mix-nets [1], electronic voting [13], [14] and so on.

In [12], Cramer, Damgård and Shoenmakers presented a widely applicable, yet efficient construction of $t$-out-of-$n$ witness indistinguishable proofs [17] based on secret sharing and public-coin honest verifier zero-knowledge proofs. It can be transformed into $t$-out-of-$n$ signatures via the Fiat-Shamir technique [16]. It is especially suitable for converting Schnorr signatures [28] and Guillou-Quisquater signatures [20] into $t$-out-of-$n$ signatures. It also allows to involve RSA signature scheme based on a zero-knowledge proof of knowledge about the factors of RSA modulus, e.g. [7], [8], but they are less efficient than the Schnorr or the GQ signatures both in computation and storage. [27] offers more intricate construction of $t$-out-of-$n$ proofs for membership.

In [26], an efficient construction of 1-out-of-$n$ signatures with RSA public-keys was introduced by Rivest, Shamir and Tauman. Called the Ring Signature Scheme, it is based on trapdoor-one-way permutations (TPs for short) and an ideal block cipher that is regarded as a perfectly random permutation. The name reflects its unique structure such that a signer who knows at least one witness (trapdoor information) can connect the head and tail of a series of $n$ random permutations to shape the sequence into a ring. Since the trapdoor is essential in their construction, it is only for the keys like RSA’s and the discrete-log keys are not supported.

There are other solutions that are more efficient but work only in non-separable models where all public-keys are related. For instance, when public-keys of the Schnorr signature scheme are chosen from a common group, one can construct an efficient 1-out-of-$n$ signature scheme as shown in Appendix. Such non-separable but highly efficient schemes may be useful when used within a specific group. In general, however, public-keys are selected independently by each signer. Even key-length would differ from user to user. Constructions based on [12] and [26] suit a separable model where no underlying group are assumed. Hence, they are ‘setup-free’; if one utilizes an existing public-key infrastructure, the key-setup phase only for this purpose is unnecessary. Furthermore, each key can be freely updated whenever each user wishes.

As introduced in [26], one application of 1-out-of-$n$ signatures is to involve somebody else’s public-keys into one’s signature without their agreement. Although there are pros and cons for such usage, it is surely useful for protecting privacy. Unfortunately, all above mentioned known schemes have particular shortcomings for this purpose; What if one is using DSA while others are using RSA? Generating a new RSA key only for this purpose is not a great idea. It is important to have wide freedom for choosing various public-keys to involve.

Our contribution. We present a widely applicable method of constructing 1-out-of-$n$ signature schemes that allows to use several flavors of public-keys such as these for integer factoring based schemes and discrete-
log based schemes at the same time. We describe two classes of signature schemes, which we call trapdoor-one-way type and three-move type, whose public-keys can be used for our construction.

Our approach also has several advantages even for the use with the same kind of keys like the former schemes:
1. When our scheme is used only with public-keys of three-move type signature schemes converted from zero-knowledge proof system, it results in a more efficient scheme than previously known three-move based construction [12] with regard to the size of signatures. For large $n$, it saves signature length about by half. Since this type of schemes includes the discrete-log based public-keys, this can be seen as the first construction of a ring signature scheme based on the discrete logarithm problem.
2. When our scheme is used only with the trapdoor-one-way based public-keys such as RSA, it results in a simplified ring signature scheme. By eliminating the use of block cipher and costly domain adjustment in the former scheme [26], our scheme offers shorter signature and less computation. In particular,
   - The signature size of ours is about 13% less than that of the previous construction when RSA with modulus size 1024 bits are used.
   - Both size of signature and computation in our signature generation is proportional to the average size of the modulus while that of former scheme it is proportional to the maximum size of the modulus. Accordingly, one long modulus does not impact efficiency in our scheme unlike the previous scheme.

We will show several concrete examples following our abstract construction. The security is proven in the random oracle model [2] as well as previously known schemes.

The rest of this paper is organized as follows. Section 2 defines security of 1-out-of-$n$ signatures. We review two constructions that work in the separable model in Sect. 3. Section 4 describes our construction in an abstract way. Some concrete examples are given in Sect. 6. It includes a discrete-log version of the ring signature scheme, improved and simplified version of the RSA-based ring signatures, and small case of mixture use of RSA and DL type signatures. In Sect. 7 the efficiency of some concrete instantiations are analyzed in detail.

2. Definitions

Definition 2.1 (Signer Ambiguous Signature Schemes)

A 1-out-of-$n$ signature scheme $\text{Sig}$ is a triple of polynomial-time algorithms, $\text{Sig} = (G,S,V)$:

$$(sk,pk) \leftarrow G(1^n,\text{type})$$

which is a probabilistic algorithm that takes security parameter $\kappa$ and a key-type specifier type, and outputs secret-key $sk$ and public-key $pk$.

$\sigma \leftarrow S(L,sk,m)$, which is a (probabilistic) algorithm that takes a set of public-keys $L$, a secret-key $sk$ whose public-key is included in $L$, and a message $m$, and outputs a signature $\sigma$.

$1/0 \leftarrow V(L,m,\sigma)$, which is an algorithm that takes $L,m$ and signature $\sigma$, and outputs 1 or 0 meaning accept or reject, respectively. We require that $V(L,m,S(L,sk,m)) = 1$ for any message $m$, any $L$ that consists of public-keys generated by $G$ and any $sk$ whose public-key is included in $L$.

$G$ can be seen as a collection of key generators for various signature schemes and type names one of them. If $L$ includes public-keys based on different security parameters, the security of $\text{Sig}$ is set to the smallest one among them.

The security of 1-out-of-$n$ signature schemes has two aspects: signer ambiguity and unforgeability. Informally, the signer ambiguity is that it is infeasible to identify which secret-key is used to generate a signature.

Definition 2.2 (Signer Ambiguity) Let $L = \{pk_1,\ldots,pk_n\}$ be the set of public-keys generated by $(pk_i,sk_i) \leftarrow G(1^n,\text{type}_i)$. $\text{Sig}$ is unconditionally signer ambiguous if for any $L$, $m$, $sk$, and $\sigma \leftarrow S(L,sk,m)$, given $(L,m,\sigma)$, any unbound adversary identifies $pk \in L$ that corresponds to $sk$ with probability exactly $1/|L|$.

Unforgeability is defined similar to the notion of existential unforgeability against adaptive chosen message attacks [19]. To capture the situation that the signer involves public-keys generated by the adversary, we allow the adversary to add arbitrary public-keys to the list of public-keys and to ask the signing oracle to sign with an adaptively chosen subset of the keys. Accordingly, we consider adaptive chosen message and chosen public-key attacks. Formally, we define the security through the following game. Let $(sk_i,pk_i)$ be a key of some type generated by $G$, and let $L_{\text{init}}$ be a collection of $n$ such public-keys the adversary attacks. By $\kappa$ we denote the smallest security parameter used to generate the keys in $L_{\text{init}}$. Let $L_{\text{add}}$ be the list of public-keys added by the adversary, which is empty at the beginning. By $L_{\text{all}}$, we denote $L_{\text{init}} \cup L_{\text{add}}$.

Definition 2.3 (Adaptive Chosen Message and Chosen Public-key Attack)

1. Adversary $A(L_{\text{init}})$ makes the following queries arbitrary times in arbitrary order.

   - $A$ sends $Q_j = (L_j,m_j)$ to the signing oracle $SO$. If $L_j \subseteq L_{\text{all}}$, the oracle returns $\sigma_j$ that
satisfies \( V(L_j, m_j, \sigma_j) = 1 \). Otherwise, \( \perp \) is returned.

- \( A \) issues \((pk_j)\). Public-key \( pk_j \) is then appended to \( L_{add} \).

2. \( A \) outputs \((\bar{L}, \bar{m}, \bar{\sigma})\).

Let \( \{(L_j, m_j, \sigma_j)\} \) denote the history of conversation between \( SO \) and \( A \). We say that \( A \) wins the game if

- \((\bar{L}, \bar{m}, \bar{\sigma}) \notin \{(L_j, m_j, \sigma_j)\}\),
- \( \bar{L} \subseteq L_{init} \), and
- \( V(\bar{L}, \bar{m}, \bar{\sigma}) = 1 \).

The second condition is to prevent the trivial forgery that the adversary yields signatures with the self-generated public-keys.

Definition 2.4 (Unforgeability) \( \text{Sig} \) is existentially unforgeable against adaptive chosen message and chosen public-key attacks if any probabilistic polynomial-time adversary \( A \) wins the above game with probability negligible in \( \kappa \).

It is important to remark that the above definition states that the list of public-keys must not be altered as well as the message. That is, one should not be able to add or remove public-keys associated to the given signatures.

3. Previous Schemes

3.1 Witness Indistinguishable Signatures [12]

Here we show a witness indistinguishable signature scheme in a concrete discrete logarithm setting. The scheme is based on [12] but slightly modified to work in separable settings, i.e. it accepts different groups. Let \( p_i, q_i \) be large primes. Let \( \{g_i\} \) denote a prime subgroup of \( \mathbb{Z}_p^* \) generated by \( g_i \) whose order is \( q_i \). Let \( x_i, y_i \) be \( y_i = g_i^{x_i} \mod p_i \). Here, \( x_i \) is the secret-key and \((y_i, p_i, q_i, g_i)\) is the public-key. Let \( L \) be a set of \( \{y_i, p_i, q_i, g_i\} \) for \( i = 1, \ldots, n \). We assume that all \( q_i \) are in the same size, say \( \ell \) bits. Let \( H : \{0,1\}^* \to \{0,1\}^{\ell - 1} \) be a publicly available hash function.

A signer who owns secret-key \( x_k \) generates a signature for message \( m \) with public-key list \( L \) that includes his own public-key, in the following way.

W-1 (Simulation step): For every \( i \in \{1, \ldots, n\} \setminus \{k\} \), compute \( a_i = g_i^{y_i^m} \mod p_i \) for \( s_i \in \mathbb{Z}_{q_i} \), \( c_i \leftarrow \{0,1\}^{\ell - 1} \).

W-2 (Real proof step): Compute

\[
\begin{align*}
    r_k & \leftarrow \mathbb{Z}_{q_k} \\
    a_k & = g_k^{r_k} \mod p_k \\
    c & = H(L, m, a_1, \ldots, a_n)
\end{align*}
\]

\( c_k = c \oplus c_1 \oplus \cdots \oplus c_{k-1} \oplus c_{k+1} \oplus \cdots \oplus c_n \) (\( \oplus \): bitwise-XOR.)

\( s_k = r_k - c_k \cdot x_k \mod q_k \).

W-3 Output \( \sigma = (c_1, s_1, \ldots, c_n, s_n) \).

A \((L, m, \sigma)\) is valid if \( s_i \in \mathbb{Z}_{q_i} \) and \( c_i \in \{0,1\}^{\ell - 1} \) for all \( i = 1, \ldots, n \) and

\( \oplus_{i=1}^n c_i = H(L, m, g_1^{y_1^m} \mod p_1, \ldots, g_n^{y_n^m} \mod p_n) \).

The size of \( \sigma \) is at most \( n(2\ell - 1) \) bits. The security can be proven in the random oracle model by using the rewinding simulation technique [18], [23], [25].

When \( |q_i| \) differs a lot, a modification is needed to retain collision property. That is, \( \ell \) is set to be larger than the largest \( |q_i| \) and \( s_k \) is computed as \( s_k = r_k - x_k \cdot CRH_k(c_k) \mod q_k \) where \( CRH_k : \{0,1\}^* \to \mathbb{Z}_{q_k} \) is a collision-resistant hash function.

3.2 Ring Signatures with Trapdoor-One-Way Permutations [26]

Let \( f_1 : \{0,1\}^* \to \{0,1\}^\ell \) be a trapdoor-one-way permutation where its inverse, \( f_1^{-1} \), can be computed only if the trapdoor information is known. Let \( E,D \) be a symmetric-key encryption and decryption function whose message space is \( \{0,1\}^\ell \). Let \( H \) be a hash function whose output domain matches to the key-space of \( E,D \).

Given \( f_1, \ldots, f_n \), the signer who can compute \( f_k^{-1} \) generates a signature for message \( m \) as follows.

R-1 (Initialization): Compute \( r_n = D_K(c_1) \) where \( K = H_1(m) \) and \( c_1 \leftarrow \{0,1\}^\ell \).

R-2 (Forward sequence): For \( i = 1, \ldots, k-1 \), compute \( c_{i+1} = E_K(c_i \oplus f_i(s_i)) \) for \( s_i \leftarrow \{0,1\}^\ell \).

R-3 (Backward sequence): For \( i = n, \ldots, k+1 \), compute \( r_{i-1} = D_K(r_i \oplus f_i(s_i)) \) for \( s_i \leftarrow \{0,1\}^\ell \).

R-4 (Shaping into a ring): Compute \( s_k' = f_k^{-1}(c_k \oplus r_k) \).

The resulting signature is \((c_1, s_1, s_2, \ldots, s_n)\). A signature-message pair is verified by computing \( K = H(m) \) and \( c_{i+1} = E_K(c_i \oplus f_i(s_i)) \) for \( i = 1, \ldots, n \), and accept if \( c_{n+1} = c_1 \) holds.

In practice, each trapdoor permutation will be defined over individual domain such as \( \mathbb{Z}_{N_i} \). In such a case, the above scheme need to transform such individual functions into common-domain trapdoor permutations. This transformation incurs some overhead. The following method is suggested in [26] to transform \( E_i \) into \( f_i \) defined over common-domain \( \{0,1\}^\ell \) where \( \ell = \max\{|N_i|\} + 160 \). Let \( E_i \) be the RSA encryption function with modulus \( N_i \). Let \( Q \) and \( S \) be positive integers such that \( Q N_i + S = s \) and \( 0 \leq S < N_i \). Define

\[
f_i(s) = \begin{cases} 
    Q N_i + E_i(S) & \text{if } (Q + 1)N_i \leq 2^\ell \\
    s & \text{otherwise.}
\end{cases}
\]
In order for the latter case to happen only with negligible probability, \( \ell \) should be polynomially larger than the size of largest modulus. For instance, if the largest modulus is 2048 bits, \( \ell \) will be 2048 + 160 bits. Accordingly, the resulting signature size is 2208\((n + 1)\) bits. This would be a large overhead when other moduli are all 1024 bits.

The above ring signature is existentially unforgeable against adaptive chosen-message attacks in the ideal cipher model where \( E \) and \( D \) are modeled by truly random permutations.

3.3 Other Related Works

[5] extends the scheme of [26] to a threshold scheme with the cost of \( O(2^t \log n) \) efficiency for threshold \( t \). [22] considers deniable ring authentication that accepts variety of public-keys and a threshold of signers. It however, needs interaction between the signer and the verifier.

4. Our Scheme

4.1 Key Types

We consider two classes of signature schemes characterized by trapdoor-one-way permutations and three-move protocols, which we denote type-OW and type-3M, respectively.

Class type-OW includes schemes such as variants of RSA signatures [3], [10], Rabin signatures and Paillier signatures [24], which use trapdoor-one-way permutations. Let \( F \) be a one-way permutation and \( I \) be its inverse function defined over space \( C \). Computing \( F(pk, r), I(sk, r) \) are easy but computing \( I \) without \( sk \) is infeasible as defined below.

Definition 4.1 (Trapdoor-one-way) For any probabilistic polynomial-time algorithm \( A \), for \((pk, sk) \leftarrow G(1^k)\), and for \( c \leftarrow C \), \( F(pk, A(c, pk)) = c \) happens only with negligible probability in \( \kappa \). Probability is taken over coin flips of \( A \), \( G \), and the choice of challenge \( c \).

Signature \( s \) is issued by inverting encoded message \( e \in \mathcal{E} \) with secret-key \( sk \); i.e., \( s = I(sk, e) \), and verification is done by checking \( e = F(pk, s) \), i.e., recovering the encoded message from signature \( s \) with public-key \( pk \) and comparing it to the given message.

Although trapdoor-one-way is sufficient to our construction, better security bound can be achieved if useful extra property is provided. Here, we assume that \( F(pk, s) \) is chosen from a family of claw-free permutations.

Definition 4.2 (Claw-free) A family \( \mathcal{F}_\kappa \) of pair of functions is claw-free family if, for any probabilistic polynomial-time algorithm \( A \), for \((f, f') \leftarrow \mathcal{F}_\kappa \), probability that, given \((f, f')\), \( A \) outputs \((s, s')\) such that \( f(s) = f'(s') \) only with negligible probability in \( \kappa \). Probability is taken over coin flips of \( A \), and the choice of \((f, f')\).

A pair \((s, s')\) that causes \( f(s) = f'(s') \) is called a claw. For more formal treatment that states random sampling and other standard properties, please refer [19]. We consider that public-key \( pk \) generated by key-generator \( G(1^\kappa) \) uniformly selects a pair of claw-free function from \( \mathcal{F}_\kappa \). Namely, we consider \( f(\cdot) = F(pk, \cdot) \) and \( f'(\cdot) = F'(pk, \cdot) \).

The RSA family is a concrete candidate of such \( f \). For \( f(s) = s^e \mod n \), define \( f'(s') = rs^e \mod n \) where \( r \) is taken from \( \mathbb{Z}_n^* \). Then, \((f, f')\) is claw-free under the RSA assumption because a claw, say \((s, s')\), which satisfies \( f(s) = s^e \mod n = f'(s') \), allows one to invert \( r \) as \( r^{-1}/e = s/s' \mod n \). In general, claw-free permutations can be constructed from homomorphic or random-self-reducible trapdoor permutations [15].

Class type-3M, typified by the Schnorr signature scheme, includes the ones derived from three-move protocols, e.g., [11], which involve three polynomial-time algorithms, say \( A, Z \) and \( V \) performed by a prover and a verifier. The prover commits to a \( s \leftarrow A(sk; r) \) and answers to randomly chosen challenge \( c \) with \( Z(sk, r, c) \). (By \( A(sk; r) \), we denote that \( r \) is chosen uniformly from its proper domain and given-to probabilistic algorithm \( A \) as a source of randomness. The domain may depend on \( sk \) as well as algorithm \( A \). Similar notation will be used in the rest of paper.) The verifier accepts if \( a = V(pk, s, c) \). The protocol must be honest verifier zero-knowledge. That is, for \( c \) and \( s \) randomly chosen from appropriate distribution, the distribution of \( V(pk, s, c) \) is identical (or statistically close) to that of \( A(sk; r) \) with uniformly chosen \( r \). Due to technical reasons, we demand slightly more. That is, for every \( pk, sk \) and \( c \), distribution of \( V(pk, s, c) \) must be indistinguishable from that of \( Z(sk, r, c) \).

Additionally, the protocols must provide collision property defined as follows.

Definition 4.3 (Collision Property) There exists a polynomial-time algorithm that computes \( sk \) from \( pk \) and two accepting conversations \((a, c, s) \) and \((a, c', s') \) where \((c, s) \neq (c', s') \).

The Fiat-Shamir technique converts three-move protocols to secure digital signature schemes by generating
challenge \(c\) from \(a\) and message \(m\) via an ideal hash function. The resulting signature is \((c, s)\) that can be verified by checking \(c = \text{hash}(V(pk, c, s), m)\).

Some signature schemes are neither type-OW nor type-3M. For such schemes, we consider compatibility among signature schemes. Signature scheme A is compatible with scheme B if 1. A’s secret-keys and public-keys can be issued to users and verify signatures of scheme B, and 2. breaking B (in the sense of existional unforgeable against adaptive chosen message attacks [19]) implies breaking A using the same key. For instance, DSS is not either type but it is compatible with the Schnorr signature scheme of type-3M. Since breaking the Schnorr signature scheme implies that the discrete-log is tractable with regard to the key, it implies DSS is broken, too. Thus, DSS keys can be involved in our scheme with type-3M.

With regard to type-OW schemes, however, special care may be needed. Remember that type-OW only shows the ability of computing \(I(f, \cdot)\) and does not necessarily imply possession of \(sk\). Therefore, it is not sufficient that scheme A’s keys can be used to scheme B, but it has to be true that ability of computing \(I(f, \cdot)\) of scheme B is sufficient to generate signatures of A.

The signature scheme in [4] is an interesting scheme that belongs to type-OW but its keys are also compatible with type-3M ones. In such a case, one can involve the keys as either-type-OW or type-3M keys according to one’s convenience (perhaps by comparing their computational and communication costs, which depend on particular implementation).

### 4.2 Description

#### [Key Generation]

A signer generates his own key pair by using the signature generation function of a signature scheme of his choice: \((sk, pk) \leftarrow G(1^n, \text{type})\).

#### [Public-key Listing]

Compose \(L = \{pk_1, \ldots, pk_n\}\), which is a list of public-keys where the signer’s public-key is located as \(pk_k\). (Corresponding secret-key is referred as \(sk_k\) hereafter.)

We assume that \(L\) is sorted so that the first \(k\) keys are of type-OW and the others are of type-3M. This sorting is only for exposition and irrelevant to user ambiguity.

For \(a, b \in C_i\), let \(a + b\) denote the group operation of Abelian group \(C_i\) and \(a - b\) be the group operation with inverse of \(b\). These binary operators are used without subscripts that denotes each group. Let \(H_i : \{0, 1\}^* \rightarrow C_i\) be a hash function. Domain \(C_i\) depends on \(pk_i\). For \((pk_i, sk_i)\) of type-OW, \(C_i\) corresponds to the domain where \(F(pk_i, \cdot)\) and \(I(sk_i, \cdot)\) are defined. For \((pk_i, sk_i)\) of type-3M, it corresponds to the randomness space of \(A_i(sk_i; r)\).

#### [Signature Generation]

To simplify the description, we define algorithms \(A, V, Z\) for type-OW keys as follows.

\[
\begin{align*}
A(sk; r) & \overset{\text{def}}{=} r \\
V(pk, s, c) & \overset{\text{def}}{=} c + F(pk, s) \\
Z(sk, r, c) & \overset{\text{def}}{=} I(sk, r - c)
\end{align*}
\]

In this way, \(A, V, Z\) can be treated just like those of type-3M. It is easily verified that they provide fixed challenge uniformity, which is necessary to prove the signer ambiguity. In the following, the indices written as subscripts are considered within \{1, \ldots, n\}. Especially, \(n + 1\) is equivalent to 1.

- **G-1 (Initialization):** Compute \(a_k = A_k(sk_k; r)\) and \(a_{k+1} = H_{k+1}(L, m, a_k)\).
- **G-2 (Forward sequence):** For \(i = k + 1, \ldots, n, 1, \ldots, n - 1\), select \(s_i\) randomly and compute
  \[
  a_i = V_i(pk_i, s_i, c_i), \text{ and } c_{i+1} = H_{i+1}(L, m, a_i).
  \]
- **G-3 (Forming the ring):** Compute \(s_k = Z_k(sk_k, r, c_k)\).
- **G-4 (Output):** \(\sigma = (c_1, s_1, s_2, \ldots, s_n)\).

#### [Signature Verification]

Given \((L, m, \sigma)\), purse \(\sigma\) as above and compute the following. For \(i = 1, \ldots, n\), compute

\[
\begin{align*}
a_i &= V_i(pk_i, s_i, c_i) \\
c_{i+1} &= H_{i+1}(L, m, a_i)
\end{align*}
\]

Accept if \(c_{n+1}\) computed at the last moment of the above loop is identical to the input \(c_1\). Reject otherwise.

### 5. Security

**Theorem 5.1** The proposed scheme is unconditionally signer-ambiguous.

**Proof.** We show that distribution of signature \(\sigma\) is independent of \(k\). Observe that for every \(i \in \{1, \ldots, n\}\) except for \(i = k, s_i\) is uniformly chosen from appropriate domain. For \(i = k, s_k\) also distributes uniformly due to the uniform choice of \(r\) and the fixed challenge uniformity of \(Z\). Next, we consider the distribution of \(c_k\). Observe that \(c_k\) distributes according to \(a_n\). If \(k \neq n, a_n\) distributes as if it were generated by \(A_n(sk_n, r_n)\) with uniformly chosen \(r_n\) due to the uniform choice of \(s_n\) and the fixed challenge uniformity of \(V\). Otherwise,
if \( k = n \), \( a_n \) is indeed generated by \( A_n(sk_n;r) \). Accordingly, distribution of \( c_1 \) is independent of \( k \).

\( \square \)

Note that the signer ambiguity does not rely on ideal randomness assumption on the hash function. Next, we claim unforgeability.

**Theorem 5.2 (Unforgeability)** The proposed scheme is existentially unforgeable against adaptive chosen message and chosen public-key attacks.

**Proof.** Let \( \mathcal{A} \) be \((\epsilon, \tau, q_s, q_h, q_a)\)-adversary that wins the game defined in Definition 2.3, with a forged signature \((\hat{L}, \hat{m}, \hat{c}_1, \hat{s}_1, \ldots, \hat{s}_{|L|})\) with probability at least \( \epsilon \), running time at most \( \tau \), at most \( q_a \) signature requests, \( q_s \) add requests. Here, \( q_h \) denotes the upper bound of random oracle calls made during the game, i.e., calls from the adversary, from the signing oracle, and for verification of the forged signature. For each \( F_i(pk_i, \cdot) \), let \( F'(pk_i, \cdot) \) be a partner of claw-free permutations as defined in Definition 4.2. We prove the theorem by proving the following lemma, which is the quantitative version of the theorem.

**Lemma 5.3** If there exists \((\epsilon', \tau', q_s, q_h, q_a)\)-adversary \( \mathcal{A} \), then, there exists \((\epsilon', \tau')\)-simulator \( \mathcal{S} \) that takes \((Linit, F_1, F_2, \ldots, F_{|L|})\) as an input and finds a secret-key of type-3M in \( Linit \) or a claw for one of the claw-free permutations, with probability \( \epsilon' \) and running time \( \tau' \) where

\[
\epsilon' > \frac{9}{50}, \quad \tau' < \frac{q_h}{\epsilon} \left( \frac{1}{(1 - \frac{2a}{q})qs} + \frac{2}{(1 - \frac{2a}{q})qs + 1} \right) \tau
\]

**Proof.** Let \( \tau \) and \( \epsilon \) be the running time and success probability of \( \mathcal{A} \). Let \( q_a \) be the maximum number of signing oracle queries made by \( \mathcal{A} \). Let \( q_h \) be the maximum number of hash queries made during the attack. Note that \( q_h \) includes the ones from \( \mathcal{S} \) for signing oracle simulation, verification, and etc. Namely, \( q_h \) denotes the total length of the tables that defines input/output correspondence of the random oracles. By \( q_s \), we denote the size of smallest \( C_i \) (regardless of the types of keys) defined by \( Linit \).

We first show that, by treating all hash functions as random oracles, signatures can be generated without knowing any secret-key at all. Let \((L_j, m_j)\) be a query to the signing oracle and let \( n_j \) denote \(|L_j|\). In the following, \( pk_i \) means the \( i \)-th public-key stored in \( L_j \) and all functions and variables accompanied by suffix \( i \) are defined relative to \( pk_i \).

**[Signing Oracle Simulation]**

\( \text{S-1.} \) Choose \( c_1, s_1 \) randomly and compute \( a_1 = V_1(pk_1, s_1, c_1) \).

\( \text{S-2.} \) For \( i = 2, \ldots, n_j \), choose \( s_i \) randomly and compute

\[
c_i = H_i(L_j, m_j, a_{i-1}), \quad a_i = V_i(pk_i, s_i, c_i)
\]

\( \text{S-3.} \) Assign \( c_i \) to the value of \( H_i(L_j, m_j, a_{n_j}) \).

\( \text{S-4.} \) Output \( \sigma_j = (c_1, s_1, \ldots, s_{n_j}) \).

Observe that back-patching of hash value is done only once for \( H_1 \) at step 3 to simulate one signature. It fails only if exactly the same query \((L,j, m_j, a_{n_j})\) has been already made to \( H_1 \). Since \( a_{n_j} \) distributes uniformly due to the fixed-challenge uniformity of \( V_i \) and uniform choice of \( s_{n_j} \) in step 2, such \( a_{n_j} \) is included in the past < \( q_h \) queries to \( H_1 \) with probability at most \( \frac{q_h}{q} \). Thus, step 3 is successful with probability at least \( 1 - \frac{q_h}{q} \) and \( q_s \) signatures are successfully simulated with probability at least \( (1 - \frac{q_h}{q})^{q_s} \). Except for such negligible failure cases, the simulated signatures distribute as well as the real ones. Since \( q \) is supposed to be exponentially large while other parameters in the above formula are polynomially bounded, the simulated signature are statistically close to those in the real run.

We proceed to construction of \( \mathcal{S} \). For simplicity, we wrap the random oracles \( H_1, H_2, \ldots \) to a single random oracle whose \( j \)-th query is formatted as \( Q_j = (pk_i, L_j, m_j, a_j) \). If \( pk_i \in L_j \), the unified random oracle returns a random value that defines \( H_j(L_j, m_j, a_j) \). (Note that, since \( pk_i \) is used only as a label to identify individual hash function \( H_i \), it does not matter even if \( pk_i \not\in L_j \) and/or \( L_j \not\subseteq L_{\text{all}} \).) If \( pk_i \not\in L_{\text{all}} \), it returns \( \perp \) (since no hash function is defined for such public-key). It returns the same value for duplicated queries. (We do not explicitly describe such ineffective interactions in the following for conciseness.)

A key observation is the following. Suppose that \( \mathcal{A} \) outputs forged signature \((\hat{L}, \hat{m}, \hat{c}_1, \hat{s}_1, \ldots, \hat{s}_{|L|})\). Then, for each \( pk_i \in \hat{L} \) (from now, \( pk_i \) denotes \( i \)-th public-key in \( \hat{L} \) and all functions and variables with suffix \( i \) are defined relative to \( pk_i \)), random oracle query \((pk_i, L, \hat{m}, V_i(pk_i, \hat{s}_i, \hat{c}_i)) \) must be made. Since these hash values form a ring, there exist \( \hat{c}_i \in \{1, \ldots, |\hat{L}|\} \) and a pair of indices \((u, v)\) such that \( Q_u = (pk_i, \hat{L}, \hat{m}_u, u) \), \( Q_v = (pk_i, \hat{L}, \hat{m}_v, a_u) \) and \( u \leq v \). Namely, \( pk_i \) lies in between the gap of hash query order. We call such \((u, v)\) a gap index pair. Note that \( u = v \) happens only if the resulting \( \hat{L} \) contains only one public-key. If there are two or more gap indices with, regard'to the signature, only the smallest one is considered.

Given \( L_{\text{init}} \) and pairs of claw-free permutations as described in Lemma 5.3, \( \mathcal{S} \) works as follows.

\( \text{A-1.} \) Choose an index \( u \in \{1, \ldots, q_h\} \) (expecting that \( u \) is to be the smaller one of the gap index pair \((u, v)\)).

\( \text{A-2.} \) Run \( \mathcal{A} \) and simulates the environment of the
A-3. A outputs \((\bar{L}, \bar{m}, \bar{c}, \bar{s}, \ldots, \bar{s}_{|\bar{L}|})\) that satisfies all winning conditions of the forgery game and corresponds to gap index \((u, v)\) for some \(v \geq u\). Otherwise, restart from A-1.

B-1. If \(i'\)-th public-key in \(\bar{L}\) is \(p_k^i\) and it is of type-OW, it holds that \(a_u = H_i(\bar{L}, \bar{m}, a_v) + F'(p_k^i, \bar{s}_{i'})\) as defined by verification procedure. Since \(H_i(\bar{L}, \bar{m}, a_v) = a_u - F'(p_k^i, \bar{s}_{i'})\) is set in step A-2, we have \(F'(p_k^i, \bar{s}_{i'}) = F'((p_k^i, \bar{s}_{i'})\), which is a claw. \(SIM\) thus completes the simulation by outputting the claw \((\bar{s}_{i'}, \bar{s}_{i'})\). If \(p_k^i\) is type-3M, otherwise, \(SIM\) proceeds to phase C.

C-1. Restart \(A\) handling its queries exactly in the same way as done in phase A making exactly the same random choices until \(Q_u\) is made. For \(Q_u\) and all subsequent queries, simulation proceeds in the same way as phase A but using a new independent random choices.

C-2. A outputs \((\bar{L}, \bar{m}, \bar{c}, \bar{s}, \ldots, \bar{s}_{|\bar{L}|})\) as well as step A-3 that corresponds to gap index \((u, v')\) for some \(v' \geq u\). Let \(c\) and \(c'\) be replies to \(Q_v\) in phase A and \(Q_{v'}\) in phase C respectively. If \(c \neq c'\), proceed to phase D. Otherwise, restart from C-1.

D-1. \(SIM\) now extracts \(sk^*\) using \((a_u, c, \bar{s}_{i'})\) and \((a_u, c', \bar{s}_{i'})\), that forms a collision with regard to \(pk^*\). \(SIM\) outputs \((sk^*_{i'}, pk^*)\) and stops.

Now we evaluate the above reduction. Let \(\epsilon_1\) denote the probability that the expected event in A-3 happens. We then have

\[
\epsilon_1 > \frac{1}{q_h} \left(1 - \frac{q_h}{q}\right)^{q_h} \epsilon_1^* \tag{1}
\]

where factor \(\frac{1}{q_h}\) is for the cost of guessing correct index \(u\), and factor \(\left(1 - \frac{q_h}{q}\right)^{q_h}\) is for the signing oracle simulation. By repeating phase A at most \(\tau_1 = \frac{1}{\epsilon_1}\) times, the event at A-3 happens with probability \(1 - (1 - \epsilon_1)^{\frac{1}{\epsilon_1}} > 1 - \exp(-1) > 3/5\).

If \(L_{init}\) contains only type-OW keys, \(SIM\) does find a claw in B-1. In such a case, \(SIM\) runs in time \(\tau_1\) and success probability at least \(3/5\). Except for such a special case, we proceed evaluation assuming that the favorable result in B-1 does not come out.

To evaluate phase C, we apply classical heavy-row argument [18]. Consider a binary matrix whose row corresponds to the random choices (including the entire random tape of \(A\)) made before query \(Q_u\) is answered, and whose column corresponds to the random choices after that. Each cell contains \(\frac{q_h}{q}\) if \(A\) causes the expected event of A-3. Otherwise it contains 0. The probability that a randomly chosen cell contains 1 is \(\epsilon_1\). A heavy row is a row that contains 1 more than \(\epsilon_1/2\) of its cells. Now, due to the heavy-row-lemma, the random choices made during the successful run in phase A is in a heavy row with probability at least \(1/2\). Therefore, the successful case in C-2 happens with probability at least \(\epsilon_1/2\). Besides the probability that random oracle query \(Q_v\) in phase A and \(Q_{v'}\) in phase C results in an identical hash value is bound by \(\epsilon_1/q\). Accordingly, we need to bound this probability by the probability that the value replied to \(Q_v\) in phase A also appears in \(Q_{v'}\) answers of phase C.) Therefore, the total successful case in C is \(\frac{q_h}{q} \left(1 - \frac{q_h}{q}\right)^{q_h} \epsilon_1^* \frac{q_h}{q} \left(1 - \frac{q_h}{q}\right)^{q_h} \epsilon_1^* \).

By repeating phase C at most \(\tau_2 = \frac{q_h}{q} \left(1 - \frac{q_h}{q}\right)^{q_h} \epsilon_1^* \frac{q_h}{q} \left(1 - \frac{q_h}{q}\right)^{q_h} \epsilon_1^* \) times, the event at C happens with probability greater than 3/5 as well as phase A.

Finally, we analyse phase D. If \(c \neq c'\), \((a_u, c, \bar{s}_{i'})\) and \((a_u, c', \bar{s}_{i'})\) forms a collision with regard to \(pk^*_i\), and the simulator can compute secret-key \(sk^*_i\) due to the collision property. Since \(L \subseteq L_{init}\) should hold by the winning condition of the game, \(pk^*_i\) is included in \(L_{init}\). Thus the simulator is successful in computing a secret-key that corresponds to one of the public-keys in \(L_{init}\).

The total running time is upper bounded by

\[
\tau_1 + \tau_2 = \left(\frac{1}{\epsilon_1} + \frac{2}{\left(1 - \frac{q_h}{q}\right)^{q_h}}\right) \frac{q_h}{\epsilon} \left(\frac{1}{\left(1 - \frac{q_h}{q}\right)^{q_h} + \frac{2}{\left(1 - \frac{q_h}{q}\right)^{q_h} + 1}}\right) \tau.
\]

When \(q_h < q\), the central term of the above formula is very close to 3, and the running time \(\tau_1 + \tau_2\) is approximately \(3q_h\), assuming that the running time of \(A\) is the dominant factor compared to other operations such as \(A, V, F\) and hashing. The success probability is at least

\[
\begin{align*}
3 & \quad \frac{1}{5} \cdot \frac{3}{2} = \frac{9}{50} \\
5 & \quad \frac{1}{5} \cdot \frac{3}{5} = \frac{9}{50}
\end{align*}
\]

Therefore, if \(q\) is exponential, \(\epsilon\) is not negligible, and \(q_h\) and \(q_h\) are bound by polynomial in the security parameter, \(SIM\) runs in polynomial-time with sufficiently large success probability. \(\square\)

Due to the lemma 5.3, Theorem 5.2 can be proven.
Because if there exist \( SIM \) that runs in polynomial-time, it allows one to compute a claw for at least one claw-free family or extracts at least one secret-key that corresponds to type-3M public-key in \( L_{\text{init}} \). It contradicts the assumptions. □

6. Concrete Examples

6.1 All Discrete-Log Case

For \( i = 1, \ldots, n \), let \((y_i, p_i, q_i, g_i)\) be DL public-keys as described in Sect. 3.1 and \( H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_{q_i} \) be hash functions. Let \( L \) be a list of these public-keys. A signer who has secret-key \( x_k \) generates a signature for message \( m \) as follows.

[Signature Generation]

\[ \text{D-1 (Initialization): Select } r \leftarrow \mathbb{Z}_{q_k} \text{ and compute } c_{k+1} = H_{k+1}(L, m, g_k^r \mod p_k). \]

\[ \text{D-2 (Forward sequence): For } i = k+1, \ldots, n, 1, \ldots, k-1, \text{ select } s_i \leftarrow \mathbb{Z}_{q_i}, \text{ and compute } c_{i+1} = H_{i+1}(L, m, g_i^{s_i} y_i^r \mod p_i). \]

\[ \text{D-3 (Forming the ring): Compute } s_k = r - x_k c_k \mod q_k. \]

The resulting signature for \( m \) and \( L \) is \((c_1, s_1, s_2, \ldots, s_n)\).

[Signature Verification]

For \( i = 1, \ldots, n \), compute \( c_i = H_1(L, m, g_i^{s_i} y_i^r) \mod p_i \) and \( c_{i+1} = H_{i+1}(L, m, c_i + s_i^r \mod N_i) \). Accept if \( c_{n+1} \) computed at the last moment of the above loop is identical to the input \( c_1 \). Reject otherwise.

6.2 All RSA Case

For \( i = 1, \ldots, n \), let \((e_i, N_i)\) be RSA public-keys and \( H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_{N_i}^* \) be hash functions. Let \( L \) be a list of these public-keys. A signer who has secret-key \( d_k \) generates a signature for message \( m \) as follows.

[Signature generation]

\[ \text{T-1 (Initialization): Select } r_k \leftarrow \mathbb{Z}_{N_k} \text{ and compute } c_{k+1} = H_{k+1}(L, m, r_k). \]

\[ \text{T-2 (Forward sequence): For } i = k+1, \ldots, n, 1, \ldots, k-1, \text{ select } s_i \leftarrow \mathbb{Z}_{N_i}, \text{ and compute } c_{i+1} = H_{i+1}(L, m, c_i + s_i^r \mod N_i). \]

\[ \text{T-3 (Shaping into a ring): Compute } s_k = (r_k - c_k)^d_k \mod N_k. \]

The resulting signature for \( m \) and \( L \) is \((c_1, s_1, s_2, \ldots, s_n)\).

[Signature Verification]

For \( i = 1, \ldots, n \), compute \( r_i = c_i + s_i^e_i \mod N_i \) and then \( c_{n+1} = H_{n+1}(L, m, r_n) \). Accept if \( c_{n+1} \) computed at the last moment of the above loop is identical to the input \( c_1 \). Reject otherwise.

6.3 Mixture Case: RSA and Schnorr

We finally show a small example for involving both RSA-type and DL-type keys. For simplicity, we consider the case \( n = 2 \), i.e., only two public-keys are involved. Let \( L \) consist of RSA-public-key \((e, N)\) and one Schnorr public-key \((y, g, p, q)\). Let \( H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_N \) and \( H_2 : \{0,1\}^* \rightarrow \mathbb{Z}_q \) be hash functions. A signer who has the RSA-type secret-key, \( d \), generates a signature for message \( m \) as follows.

[Signature generation]

\[ \text{M-1 (Initialization): Select } r \leftarrow \mathbb{Z}_N \text{ and compute } c_2 = H_2(L, m, r). \]

\[ \text{M-2 (Forward sequence): Select } s_2 \leftarrow \mathbb{Z}_q \text{ and compute } c_1 = H_1(L, m, g^{s_2} y^{s_2} \mod p). \]

\[ \text{M-3 (Shaping into a ring): Compute } s_1 = (r - c_1)^d \mod N. \]

The resulting signature is \((c_1, s_1, s_2)\).

[Signature Verification]

Given \((L, m, c_1, s_1, s_2)\), compute \( c_2 = H_2(L, m, c_1 + s_1^e \mod N) \). Accept if \( c_2 \). Reject otherwise.

The signature can be shorten by selecting \((c_2, s_1, s_2)\) as a signature because \( |c_1| \) is the size of RSA modulus typically > 1024 bits while \( |c_2| \) is the size of \( q \) typically > 160 bits.

7. Efficiency

We compare our ring signature scheme with the existing schemes using DL, ECDL (elliptic curve DL) and RSA trapdoor functions, in terms of the length of a signature and the computational cost of signature generation and verification. We refer the scheme in Sect. 3.1 by “WI signatures” and the scheme in Sect. 3.2 with RSA trapdoor function by “RSA ring signatures,” hereafter. Let \( n \) be the number of signers of ring signature.

Table 1 shows the comparison in terms of the length of the signature. Here, \( L(DL) \) is the length of exponent of DL signature, and is typically 1024-bit. \( L(RSA) \) is the length of modulus of RSA signature, and is typically 1024-bit. \( L(EC) \) is the length of the size of cyclic subgroup in elliptic curve, and is typically 160-bit. From the table, we can see that the length of our
signature with DL is one half of WI signature for large \( n \), and that the length of our signature with RSA is 13% less than the length of RSA ring signature.

Table 2 shows the comparison in terms of the computational costs of signature generation and verification. Here, \( T(DL) \), \( T(EC) \), \( T(RSA^{-1}) \) and \( T(RSA) \) are the computational costs of modular exponentiation, scalar multiplication on elliptic curve, inverse RSA function and RSA function, respectively. Typically, \( T(DL) = T((1024)^{16}) \), \( T(EC) = T((160) \cdot (EC160)) \), \( T(RSA^{-1}) = T((1024)^{1024}) \) and \( T(RSA) = T((1024)^{16}) \). Here, \( T(x)^{y} \) is the number of \( y \)-bit exponentiation with \( x \)-bit modulus. RIPEMD-160 function is used for SHA-1 function. The computational costs of exponentiation with two basis is 5/4 of exponentiation with single basis, using two basis binary method. From these tables, we can see that the computational costs of our signature with DL is as same as WI signature, and that the computational costs of our signature with RSA is as same as RSA ring signature.

Notice that in known schemes the length and the computational costs of signature is proportional to the maximum of the length of DL exponent / RSA modulus. In our scheme, the length and the computational costs of signature is proportional to the average of the length of DL exponent / RSA modulus, since our scheme need not to round up the length to the maximum length.

References

yi = gXi mod p.

Let p, q be large primes. Let (g) denote a prime subgroup in Zp generated by g whose order is q. Let xi, yi be yi = gxi mod p. Here xi is the secret-key and (yi, p, g, g) is the public-key. All member use common p, q, g in the non-separable model. So only yi is different in public-keys for each member. Let L be a set of (yi, p, g, g) for i = 1, ..., n. Let H : {0, 1}∗ → Zq be a hash function.

A signer who owns secret-key xk generates a signature for message m with public-key list L that includes yk, in the following way.

S-1 Select ci, ci ← Zq for i = 1, ..., n, i = k, and compute a = gα = 1,...,t, [1]−1, i ≠ k yi mod p.

S-2 Compute

\[ c = H(L, m, a) \]
\[ c_k = c - (c_1 + ... + c_{k-1} + c_{k+1} + ... + c_n) \mod q \]
\[ s = a - c_k \cdot x_k \mod q. \]

The resulting signature is σ = (s, c1, ..., cn). (L, m, σ) is valid if

\[ \sum_{i=1}^{n} c_i \equiv H(L, m, g^s y_1^c_1 \cdots y_n^c_n) \mod q. \]

The security of this scheme can be reduced to the discrete-log problem (representation problem) by standard rewinding simulation.

Appendix: An Efficient Scheme in Non-separable Model

Let p, q be large primes. Let (g) denote a prime subgroup in Zp generated by g whose order is q. Let xi, yi be yi = gxi mod p. Here xi is the secret-key and (yi, p, g, g) is the public-key. All member use common p, q, g in the non-separable model. So only yi is different in public-keys for each member. Let L be a set of (yi, p, g, g) for i = 1, ..., n. Let H : {0, 1}∗ → Zq be a hash function.

A signer who owns secret-key xk generates a signature for message m with public-key list L that includes yk, in the following way.

S-1 Select ci, ci ← Zq for i = 1, ..., n, i = k, and compute a = gα = 1,...,t, [1]−1, i ≠ k yi mod p.

S-2 Compute

\[ c = H(L, m, a) \]
\[ c_k = c - (c_1 + ... + c_{k-1} + c_{k+1} + ... + c_n) \mod q \]
\[ s = a - c_k \cdot x_k \mod q. \]

The resulting signature is σ = (s, c1, ..., cn). (L, m, σ) is valid if

\[ \sum_{i=1}^{n} c_i \equiv H(L, m, g^s y_1^c_1 \cdots y_n^c_n) \mod q. \]

The security of this scheme can be reduced to the discrete-log problem (representation problem) by standard rewinding simulation.

\[ c = H(L, m, a) \]
\[ c_k = c - (c_1 + ... + c_{k-1} + c_{k+1} + ... + c_n) \mod q \]
\[ s = a - c_k \cdot x_k \mod q. \]

The resulting signature is σ = (s, c1, ..., cn). (L, m, σ) is valid if

\[ \sum_{i=1}^{n} c_i \equiv H(L, m, g^s y_1^c_1 \cdots y_n^c_n) \mod q. \]

The security of this scheme can be reduced to the discrete-log problem (representation problem) by standard rewinding simulation.

\[ c = H(L, m, a) \]
\[ c_k = c - (c_1 + ... + c_{k-1} + c_{k+1} + ... + c_n) \mod q \]
\[ s = a - c_k \cdot x_k \mod q. \]

The resulting signature is σ = (s, c1, ..., cn). (L, m, σ) is valid if

\[ \sum_{i=1}^{n} c_i \equiv H(L, m, g^s y_1^c_1 \cdots y_n^c_n) \mod q. \]