Generic Adaptor Signature

Xianrui Qin, Handong Cui, and Tsz Hon Yuen

The University of Hong Kong, Hong Kong
{xrqin,hdcui,thyuen}@cs.hku.hk

Abstract. Adaptor signature is becoming an increasingly important tool in solving the scalability and interoperability issues of blockchain application. It has many useful properties, such as reducing the on-chain communication cost, increasing the fungibility of transactions and circumventing the limitation of the blockchain's scripting language.
In this paper, we propose the first generic construction of adaptor signatures from Type-T canonical identification, which includes discrete-logarithm-based, RSA-based and lattice-based constructions. Our generic construction can be used as a general framework to combine with different privacy-preserving cryptosystems. We propose blind adaptor signature and linkable ring adaptor signature, which are useful in different blockchain applications.

1 Introduction

Blockchains are decentralized networks maintained by miners, where each transaction on the blockchain can be regarded as script-formed applications. A blockchain's scripting language defines possible functionalities that can be implemented on blockchain. Bitcoin, for example, consists of very few scripts, limiting its application to coin transfers. In contrast, Ethereum has a Turing-complete scripting language that allows users to exercute more sophisticated and complicated applications such as smart contracts.

To deploy and execute a transaction, a user should pay a fee to the miners. The fee is determined by the storage and computational costs associated with running each transaction's script. Thus, it is desirable to conduct certain operations off-chain in order to minimize the on-chain fee paid to the miners. In this way, Poelstra developed the idea of scriptless scripts [16], later called adaptor signatures [3, 9].

Adaptor signatures can be seen as an extension over a digital signature. The rough idea is that a "pre-signature" is firstly generated, but still an uncompleted one yet. After being completed a witness of the statement embedded in the pre-signature will be revealed. The verification of the completed adaptor signature is done in the same way as the normal signature, for which the adapting trace is hidden.

Atomic swap is a kind of technique that allows fair exchange of two different cryptocurrencies on distinct blockchains. In other words, an atomic swap protocol ensures either the coins are swapped or the balances are untouched.

A well-known protocol for atomic swaps is the one described in [6] that leverages the Hash Time Locked Contract (HTLC) to perform the swap. Adaptor signatures can be used to support atomic swap, without using smart contract.

There are some papers that instantiate adaptor signatures based on Schnorr and ECDSA digital signatures [3, 14]. However, their constructions are limited to Schnorr and ECDSA signature schemes and therefore is not generic.

1.1 Our Contribution

In this paper, we give a generic construction of adaptor signature. The main contribution of our generic adaptor signature is that it can be applied to various signature schemes used in different cryptocurrencies, making atomic swap between different cryptocurrencies feasible. Our generic adaptor signature can be applied to discrete-logarithm(DL)-based, RSA-based and lattice-based signatures.

Technically, our generic adaptor signature is constructed from a new type of identification scheme called Type-TA. It is derived from a three-move canonical identification with some special properties. Roughly speaking, it requires (1) the commitment algorithm to be (additive/multiplicative) homomorphic (in order to combine the commitment with the statement); (2) the response algorithm is homomorphic with respect to commitment randomness (in order to facilitate the conversion to normal signature); (3) the verification algorithm is composed of running the commitment algorithm on the response (in order to complete the security proof of witness extractability). The Schnorr identification, the RSA-based GQ identification and the lattice-based identification in [8] are some examples of Type-TA identification.

From the application point of view, different types of signature are used in blockchain. For example, linkable ring signature is used in privacy-preserving cryptocurrencies such as Monero [15], in order to hide the public key of the signer. Blind ECDSA signature is recently proposed in [19] to provide anonymity of the recipient address in Bitcoin transaction. In order to provide compatibility with these two variants of signature schemes, we propose the notion of blind adaptor signature and linkable ring adaptor signature and their security models. We give the generic construction for blind adaptor signature and linkable ring adaptor signature. We believe they can be important tools to increase atomicity and privacy for cryptocurrency.

2 Related Work

2.1 Comparison with Lockable Signature

One work that is related to our work is lockable signature [18]. Similar to adaptor signature, lockable signature allows one to compute a lock, which is the analogue of the pre-signature. However, the difference between them is that in adaptor signature, the pre-signature can be computed without knowing the witness of the given relation, while in lockable signature, computing a lock requires the signer's secret key.

2.2 Comparison with a Recent Work [7]

A very recent work by Erwig et al. [7] proposes "Two-Party Adaptor Signatures From Identification Schemes", which shares a very similar idea with ours. We both provide a generic approach to transform three-move type signature schemes to adaptor signature. There are two main differences:

- Our generic construction additionally includes an extra checking step during the verification. It is useful for the lattice-based instantiation (especially for those schemes following the "Fiat-Shamir with abort" paradigm, e.g., [8]). On the other hand, [7] does not consider any lattice-based instantiation.
- We additionally provide a bridge between adaptor signature and other cryptographic building blocks. Concretely, we propose adaptor blind signature and linkable ring adaptor signature. They can be used in many applications, such as increasing atomicity and privacy for cryptocurrency. [7] worked on another notion called two-party signature.

3 Preliminaries

In this paper, we use λ as the security parameter, $\mathsf{negl}(\lambda)$ to represent a negligible function with respect to λ and Δ to represent an appropriate space of randomness defined by the algorithm.

3.1 Type-T Signature and Canonical Identification

Type-T signature is defined in [2], as shown in Algorithm 1.

- The Sign algorithm uses the algorithm A to generate a commitment R using a randomness r (chosen from a randomness domain Δ_r). Then, the message and R are inputted to a function H to obtain a challenge c (within the challenge space Δ_c). Finally, the algorithm uses the function Z to generate the signature using the secret key sk, r and c.
- The Verify algorithm allows the reconstruction of R' from the public key pk, z and c using the function V. The signature is validated by using H on the message and R'.

Schnorr signature [17], Guillou-Quisquater signature [11], Katz-Wang signature [12] and EdDSA [5] are examples of Type-T signatures.

Type-T **Canonical Identification.** Canonical identification [1] is a three-move public-key authentication protocol of a specific form. We first define Type-T canonical identification in Algorithm 2, based on the definition of Type-T signature in [2]. We add the additional checking in line 17 of Algorithm 2, which is useful for lattice-based construction. It is straightforward that after applying the Fiat-Shamir transformation to Type-T canonical identification, we obtain a Type-T signature.

We define the security of *impersonation under key only attack* for Type-T canonical identification.

Algorithm 1: Type-T Signature

```
Procedure Setup($\lambda$):
    return param;
Procedure Sign(sk, $M$):
    $r \leftarrow \Delta_r$;
    $R = A(r)$;
    $c = H(M, R)$;
    $z = Z(\mathsf{sk}, r, c)$;
    return $\sigma = (z, c)$;
Procedure KeyGen():
    return (pk, sk);
Procedure Verify(pk, $\sigma$, $M$):
    parse $\sigma = (z, c)$;
    $R' = V(\mathsf{pk}, z, c)$;
    if $c \neq H(M, R')$ then
        return 0;
    return 1;
```

Algorithm 2: Type-T Canonical Identification

```
Procedure Setup($\lambda$):
    return param;
Procedure KeyGen():
    return (pk, sk);
Procedure Proof1(sk):
    $r \leftarrow_s \Delta_r$;
    $R = A(r)$;
    return $(R, r)$;
Procedure Ch($R$):
    return $c$;
Procedure Proof2(sk, $r$, $c$):
    return $z = Z(\mathsf{sk}, r, c)$;
Procedure Verify(pk, $z$, $c$):
    $R' = V(\mathsf{pk}, c, z)$;
    if $c \neq \textsc{Ch}(R')$ then
        return 0;
    auxiliary checking with $R', c, z$;
    return 1;
```

Definition 1. *A* Type-T *canonical identification is secure against impersonation under key only attack if there is no PPT adversary $\mathcal{A}$ such that $\mathbf{Adv}_{\mathcal{A}}^{\mathsf{imp}}(\lambda)$ is non-negligible, where:*

$$\mathbf{Adv}_{\mathcal{A}}^{\mathsf{imp}}(\lambda) := \Pr[\textsc{Verify}(\mathsf{pk}, z^*, c_{i^*}) = 1 | \mathsf{param} \leftarrow \textsc{Setup}(\lambda), (\mathsf{pk}, \mathsf{sk}) \leftarrow \textsc{KeyGen}(), (c_{i^*}, z^*) \leftarrow \mathcal{A}^{\mathsf{CH}(\cdot)}(\mathsf{param}, \mathsf{pk})].$$

For the i-th query $\mathsf{CH}(R_i)$*, the oracle returns c_i to $\mathcal{A}$, and $i^* \in [1, q_c]$, q_c is the number of query to* CH*.*

3.2 Adaptor Signature

A relation R with a language $L_R := \{Y | \exists y : (Y, y) \in R\}$ is said to be hard if (i) a probabilistic polynomial time (PPT) generator $\mathsf{LockGen}(\lambda)$ that outputs $(Y, y) \in R$, (ii) for every PPT algorithm $\mathcal{A}$, given $Y \in L_R$, the probability of $\mathcal{A}$ outputting y is negligible.

According to [3], an adaptor signature $\prod_{R,\Sigma}$ is defined with respect to a hard relation R and a signature scheme $\Sigma = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$.

Definition 2 (Adaptor Signature Scheme). *An adaptor signature* AS *scheme* $\prod_{R,\Sigma}$ *consists of four algorithms* (PreSign, PreVerify, Adapt, Ext) *defined below.*

Algorithm 3: Experiment $\mathsf{aSignForge}_{\mathcal{A},\prod_{R,\Sigma}}$

```
Procedure aSignForge_{A,∏_{R,Σ}}(λ):
    $\mathcal{Q} := \emptyset$;
    $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$;
    $M^* \leftarrow \mathcal{A}^{O_\mathsf{S},O_\mathsf{pS}}(\mathsf{pk})$ ;
    $(Y,y) \leftarrow \mathsf{LockGen}(\lambda)$;
    $\hat{\sigma} \leftarrow \mathsf{PreSign}((\mathsf{pk},\mathsf{sk}),Y,M^*)$;
    $\sigma^* \leftarrow \mathcal{A}^{O_\mathsf{S},O_\mathsf{pS}}(\hat{\sigma},Y)$;
    return
      $((M^* \notin \mathcal{Q}) \wedge \mathsf{Verify}(\mathsf{pk},\sigma^*,M^*) = 1)$;
Procedure $O_\mathsf{S}(M)$:
    $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk},M)$;
    $\mathcal{Q} := \mathcal{Q} \cup \{M\}$;
    return $\sigma$;
Procedure $O_\mathsf{pS}(M,Y)$:
    $\hat{\sigma} \leftarrow \mathsf{PreSign}((\mathsf{pk},\mathsf{sk}),Y,M)$;
    $\mathcal{Q} := \mathcal{Q} \cup \{M\}$;
    return $\hat{\sigma}$;
```

- $\mathsf{PreSign}((\mathsf{pk},\mathsf{sk}),Y,M)$*: on input a key pair* $(\mathsf{pk},\mathsf{sk})$*, a statement* $Y \in L_R$ *and a message* $M \in \{0,1\}^*$*, outputs a pre-signature* $\hat{\sigma}$*.*
- $\mathsf{PreVerify}(Y,\mathsf{pk},\hat{\sigma},M)$*: on input a statement* $Y \in L_R$*, a pre-signature* $\hat{\sigma}$*, a public key* pk *and a message* M *, outputs a bit* b*.*
- $\mathsf{Adapt}((Y,y),\mathsf{pk},\hat{\sigma},M)$*: on input a statement-witness pair* (Y,y)*, a public key* pk*, a pre-signature* $\hat{\sigma}$ *and a message* M*, outputs a signature* σ*.*
- $\mathsf{Ext}(Y,\sigma,\hat{\sigma})$*: on input a statement* $Y \in L_R$*, a signature* σ *and a pre-signature* $\hat{\sigma}$*, outputs a witness* y *such that* $(Y,y) \in R$*, or* $\perp$*.*

Definition 3 (Pre-signature Adaptability). *An adaptor signature scheme* $\prod_{R,\Sigma}$ *satisfies pre-signature adaptability if for every message* M *in the message space, every statement/witness pair* $(Y,y) \in R$ *and for all pre-signature* $\hat{\sigma}$*, the following holds:*

$$Pr\left[\mathsf{Verify}(\mathsf{pk},\sigma,M) = 1 \middle| \begin{array}{l}(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda),\\ \mathsf{PreVerify}(Y,\mathsf{pk},\hat{\sigma},M) = 1,\\ \sigma \leftarrow \mathsf{Adapt}((Y,y),\mathsf{pk},\hat{\sigma},M).\end{array}\right] = 1.$$

Definition 4 (Unforgeability). *An adaptor signature scheme* $\prod_{R,\Sigma}$ *is* $\mathsf{aEUF\text{-}CMA}$ *secure if for every PPT adversary* $\mathcal{A}$ *runing the experiment* $\mathsf{aSignForge}_{\mathcal{A},\prod_{R,\Sigma}}$ *defined in Algorithm 3,* $\Pr[\mathsf{aSignForge}_{\mathcal{A},\prod_{R,\Sigma}}(\lambda) = 1] \le \mathsf{negl}(\lambda)$*.*

Definition 5 (Witness Extractability). *An adaptor signature scheme* $\prod_{R,\Sigma}$ *is witness extractable if for every PPT adversary* $\mathcal{A}$ *running the experiment* $\mathsf{aWitExt}_{\mathcal{A},\prod_{R,\Sigma}}$ *defined in Algorithm 4,* $\Pr[\mathsf{aWitExt}_{\mathcal{A},\prod_{R,\Sigma}}(\lambda) = 1] \le \mathsf{negl}(\lambda)$*.*

4 Generic Adaptor Signature

In this section, we give a generic construction of adaptor signature, which is built from a new security notion called Type-TA canonical identification.

Algorithm 4: Experiment $\mathsf{aWitExt}_{\mathcal{A},\prod_{R,\Sigma}}$

```
1 Procedure aWitExt_{A,∏_{R,Σ}}(λ):
2     $\mathcal{Q} := \emptyset$;
3     $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$;
4     $(M^*, Y) \leftarrow \mathcal{A}^{O_\mathsf{S},O_\mathsf{pS}}(\mathsf{pk})$ /* $O_\mathsf{S}, O_\mathsf{pS}$ are the same as Algorithm 3 */
5     $\hat{\sigma} \leftarrow \mathsf{PreSign}((\mathsf{pk},\mathsf{sk}), Y)$;
6     $\sigma^* \leftarrow \mathcal{A}^{O_\mathsf{S},O_\mathsf{pS}}(\hat{\sigma})$;
7     $y^* \leftarrow \mathsf{Ext}(Y, \sigma^*, \hat{\sigma})$;
8     return $((M^* \notin \mathcal{Q}) \wedge (Y, y^*) \notin R \wedge \mathsf{Verify}(\mathsf{pk}, \sigma^*, M^*) = 1)$;
```

4.1 Building Block: Type-TA Canonical Identification

We define a new security notion called Type-TA canonical identification, which is a special case of a Type-T canonical identification.

Definition 6. *A* Type-TA *canonical identification is a* Type-T *canonical identification with some additional properties:*

1. *For all $y \in \Delta_r$, $(A(y), y)$ belongs to some hard relation R.*
2. *For all $r_1, r_2 \in \Delta_r$,*

$$A(r_1) \oplus_R A(r_2) = A(r_1 \oplus r_2), \quad A(r_1^{-1}) = (A(r_1))^{-1}.$$

   where $\oplus$ is a group operation in Δ_r, $\oplus_R$ is a group operation in the domain of R. The inverse functions are defined in the corresponding group operations ($\oplus$ and $\oplus_R$).
3. *For all $r_1, r_2 \in \Delta_r$, $c \in \Delta_c$ and secret key* sk*,*

$$Z(\mathsf{sk}, r_1, c) \oplus r_2 = Z(\mathsf{sk}, r_1 \oplus r_2, c).$$

   It implies that $\oplus$ is also a group operation in the domain of z.
4. *For all z, c and public key* pk*, there is a PPT algorithm V' such that:*

$$V(\mathsf{pk}, z, c) = A(z) \oplus_R V'(\mathsf{pk}, c).$$

Looking ahead, the first property is to define the hard relation for an adaptor signature. The second property is to combine the commitment and the statement $A(y)$ and form a new commitment. The third property is for the correctness of the Adapt algorithm. The fourth property is for the proof of witness extractability.

Schnorr Identification [17]. In Schnorr identification, $A(r) = g^r$ is a hard relation if the DL assumption holds. Consider $\oplus_R$ as a multiplication in a cyclic group and $\oplus$ as a modular addition, we have $A(r_1) \cdot A(r_2) := g^{r_1} \cdot g^{r_2} = g^{r_1+r_2} = A(r_1+r_2)$ and $A(-r) = g^{-r} = A(r)^{-1}$. Also $Z(\mathsf{sk}, r_1, c) + r_2 := r_1 + c \cdot \mathsf{sk} + r_2 = (r_1+r_2) + c \cdot \mathsf{sk} = Z(\mathsf{sk}, r_1+r_2, c)$. Finally, $V(\mathsf{pk}, z, c) := g^z \cdot \mathsf{pk}^c = A(z) \cdot V'(\mathsf{pk}, c)$. Hence, Schnorr identification is a Type-TA identification.

GQ Identification [11]. In GQ identification, $A(r) = r^v$ is a hard relation if the standard RSA assumption holds. Consider both $\oplus_R$ and $\oplus$ as multiplication in a cyclic group. Then we have $A(r_1) \cdot A(r_2) := r_1^v \cdot r_2^v = (r_1 \cdot r_2)^v = A(r_1 \cdot r_2)$ and $A(r^{-1}) = r^{-v} = A(r)^{-1}$. Also $Z(\mathsf{sk}, r_1, c) \cdot r_2 := r_1 \mathsf{sk}^c \cdot r_2 = (r_1 \cdot r_2) \cdot \mathsf{sk}^c = Z(\mathsf{sk}, r_1 \cdot r_2, c)$. Finally, $V(\mathsf{pk}, z, c) := z^v \cdot \mathsf{pk}^c = A(z) \cdot V'(\mathsf{pk}, c)$. Hence, GQ identification is a $\mathsf{Type\text{-}TA}$ identification.

Lattice-based Identification [8]. In lattice-based identification [8], $A(\boldsymbol{y}) = \boldsymbol{A}\boldsymbol{y}$ is a hard relation where $|\boldsymbol{y}| \le \beta_{\mathsf{SIS}}$, if the Module-SIS assumption holds. Consider both $\oplus_R$ and $\oplus$ as modular addition. Next we have $A(\boldsymbol{y}_1) + A(\boldsymbol{y}_2) := \boldsymbol{A}\boldsymbol{y}_1 + \boldsymbol{A}\boldsymbol{y}_2 = \boldsymbol{A}(\boldsymbol{y}_1 + \boldsymbol{y}_2) = A(\boldsymbol{y}_1 + \boldsymbol{y}_2)$ where $|\boldsymbol{y}_1 + \boldsymbol{y}_2| \le \beta_{\mathsf{SIS}}$ and $A(-\boldsymbol{y}) = \boldsymbol{A}(-\boldsymbol{y}) = -\boldsymbol{A}\boldsymbol{y} = -A(\boldsymbol{y})$. Also $Z(\mathsf{sk}, \boldsymbol{y}_1, c) + \boldsymbol{y}_2 := \boldsymbol{y}_1 + c \cdot \mathsf{sk} + \boldsymbol{y}_2 = (\boldsymbol{y}_1 + \boldsymbol{y}_2) + c \cdot \mathsf{sk} = Z(\mathsf{sk}, \boldsymbol{y}_1 + \boldsymbol{y}_2, c)$ where $|\boldsymbol{y}_1 + \boldsymbol{y}_2| \le \beta_{\mathsf{SIS}}$. Finally, $V(\mathsf{pk}, \boldsymbol{z}, c) := \boldsymbol{A}\boldsymbol{z} - c \cdot \mathsf{pk} = A(\boldsymbol{z}) + V'(\mathsf{pk}, c)$. Although we add an additional condition to bound the $|\boldsymbol{y}| \le \beta_{\mathsf{SIS}}/2$, if we see it as a inherent requirement of lattice cryptography, lattice-based identification in [8] is also a $\mathsf{Type\text{-}TA}$ identification.

Algorithm 5: Generic adaptor signature (AS) from a $\mathsf{Type\text{-}TA}$ identification scheme ID for the language $L := \{Y : \exists y \in \Delta_r : Y = A(y)\}$.

```
1  Procedure Setup($\lambda$):
2      define hash $H : \{0,1\}^* \to \Delta_c$;
3      $\mathsf{param}_I \leftarrow$ ID.SETUP($\lambda$);
4      return ($\mathsf{param}_I, H$);
5  Procedure KeyGen():
6      return ID.KEYGEN();
7  Procedure PreSign(($\mathsf{pk}, \mathsf{sk}$), $Y, M$):
8      $r \leftarrow \Delta_r$;
9      $R = A(r) \oplus_R Y$;
10     $c = H(M, R)$;
11     $\hat{z} = Z(\mathsf{sk}, r, c)$;
12     return $\hat{\sigma} = (\hat{z}, c)$;
13 Procedure PreVerify($Y, \mathsf{pk}, \hat{\sigma}, M$):
14     parse $\hat{\sigma} = (\hat{z}, c)$;
15     $R' = V(\mathsf{pk}, \hat{z}, c) \oplus_R Y$;
16     if $c \neq H(M, R')$ then
17         return 0;
18     auxiliary checking with $R', c, z$;
19     return 1;
20 Procedure Adapt(($Y, y$), $\mathsf{pk}, \hat{\sigma}, M$):
21     parse $\hat{\sigma} = (\hat{z}, c)$;
22     $z = \hat{z} \oplus y$;
23     return $\sigma = (z, c)$;
24 Procedure Ext($Y, \hat{\sigma}, \sigma$):
25     parse $\hat{\sigma} = (\hat{z}, \hat{c})$ and $\sigma = (z, c)$;
26     if $\hat{c} = c$ then
27         return $y = (\hat{z})^{-1} \oplus z$;
28     return $\perp$;
```

4.2 Our Construction

Then we give the generic adaptor signature in Algorithm 5.

Security Proof. The correctness of the generic adaptor signature is straightforward.

Theorem 1. *Our generic adaptor signature has pre-signature adaptability.*

Proof. Next we show that it has pre-signature adaptability. For a pre-signature $\hat{\sigma} = (\hat{z}, c)$ which passes the $\mathsf{PreVerify}$ algorithm, we have $R = V(\mathsf{pk}, \hat{z}, c) \oplus_R Y$ and $c = H(M, R)$. By the Adapt algorithm, we have $\sigma = (z = \hat{z} \oplus y, c)$. We further have

$$\begin{aligned} c &= H(M, R) \\ &= H(M, V(\mathsf{pk}, \hat{z}, c) \oplus_R Y) \\ &= H(M, A(\hat{z}) \oplus_R V'(\mathsf{pk}, c) \oplus_R Y) \\ &= H(M, A(z \ominus y) \oplus_R V'(\mathsf{pk}, c) \oplus_R Y) \\ &= H(M, A(z) \ominus_R A(y) \oplus_R V'(\mathsf{pk}, c) \oplus_R Y) \\ &= H(M, A(z) \ominus_R Y \oplus_R V'(\mathsf{pk}, c) \oplus_R Y) \\ &= H(M, A(z) \oplus_R V'(\mathsf{pk}, c)) \\ &= H(M, V(\mathsf{pk}, z, c)) \end{aligned} \tag{1}$$

It follows that σ is valid, i.e., $\mathsf{Verify}(\mathsf{pk}, \sigma, M) = 1$.

Theorem 2. *Our generic adaptor signature is aEUF-CMA secure in the random oracle model if the identification scheme* ID *is secure against impersonation under key only attack.*

Proof. Suppose that there is a PPT adversary $\mathcal{A}$ breaking the aEUF-CMA security of our generic adaptor signature. We build an algorithm $\mathcal{B}$ to break the security of ID. First, the challenger $\mathcal{C}$ of ID gives param and pk to $\mathcal{B}$. $\mathcal{B}$ forwards param and pk to $\mathcal{A}$.

For all signing oracle queries from $\mathcal{A}$ on a message M, $\mathcal{B}$ picks a random z and c from their corresponding domain and computes $R = V(\mathsf{pk}, z, c)$. $\mathcal{B}$ sets $c = H(M, R)$ in the random oracle H. $\mathcal{B}$ returns (z, c) to $\mathcal{A}$.

For the pre-signing oracle queries from $\mathcal{A}$ with input (M, Y), $\mathcal{B}$ picks a random $\hat{z}$ and c from their corresponding domain and computes $R = V(\mathsf{pk}, \hat{z}, c) \oplus_R Y$. $\mathcal{B}$ sets $c = H(M, R)$ in the random oracle H. $\mathcal{B}$ returns $(\hat{z}, c)$ to $\mathcal{A}$.

For all random oracle queries $H(M, R)$, $\mathcal{B}$ queries the oracle $\mathrm{CH}(R)$ from $\mathcal{C}$ and obtains c. $\mathcal{B}$ returns c to $\mathcal{A}$.

If $\mathcal{A}$ outputs a valid forgery (z^*, c^*) on a message M^*, we have $R^* = V(\mathsf{pk}, z^*, c^*), c^* = H(M^*, R^*)$. Then $\mathcal{B}$ returns (c^*, z^*) as the attack to $\mathcal{C}$. □

Theorem 3. *Our generic adaptor signature is witness extractable.*

Proof. Suppose that there is a PPT adversary $\mathcal{A}$ breaking the witness extractability of our generic adaptor signature. All oracle queries can be simulated by using the secret key.

In the challenge phase, $\mathcal{A}$ is given a pre-signature $\hat{\sigma} = (\hat{z}, c)$ for a message M^* and a statement Y, where $\hat{z} = Z(\mathsf{sk}, r, c)$ and $c = H(M^*, R)$. Then $\mathcal{A}$ outputs a full signature $\sigma^* = (z^*, c^*)$, where $R^* = V(\mathsf{pk}, z^*, c^*)$. If $\mathcal{A}$ wins, it implies that $\mathsf{Ext}(Y, \sigma^*, \hat{\sigma})$ did not output $\perp$. Hence $c = c^*$. By the collision resistant property of H, then $R^* = R$. It implies $V(\mathsf{pk}, z^*, c^*) = A(r) \oplus_R Y$. By the Ext algorithm, we can compute $y = (\hat{z})^{-1} \oplus z^*$. Hence we have:

$$V(\mathsf{pk}, \hat{z} \oplus y, c^*) = A(r) \oplus_R Y = V(\mathsf{pk}, \hat{z}, c^*) \oplus_R Y.$$

Also by the property 2 and 4 of the $\mathsf{Type\text{-}TA}$ identification, we have:

$$V(\mathsf{pk}, \hat{z} \oplus y, c^*) = A(\hat{z}) \oplus_R A(y) \oplus_R V'(\mathsf{pk}, c^*) = V(\mathsf{pk}, \hat{z}, c^*) \oplus_R A(y).$$

Hence we can extract y such that $A(y) = Y$. □

4.3 Discussion on ECDSA Adaptor Signature

ECDSA is the most commonly used signature scheme in cryptocurrency. ECDSA does not fall into the category of $\mathsf{Type\text{-}TA}$ signature and hence cannot be used with our generic construction. In particular, the inverse computation in ECDSA makes it difficult to compute an adaptor signature. The first provably secure ECDSA adaptor signature is given in [14]. Nevertheless, the language $L := \{Y : \exists y : Y = g^y\}$ used in the ECDSA adaptor signature is the same as the language used in our Schnorr-based instantiation.

5 Blind Adaptor Signature

In this section, we propose the notion of blind adaptor signature. In a blind signature scheme, a user can obtain a signature from a signer on a message M such that: (1) the signer cannot recognize the signature later (blindness, which implies that the message M is unknown to the signer) and (2) the user can compute a single signature per interaction with the signer (one-more unforgeability). Blind signature is used to provide private fiat-to-cryptocurrency swap in [19].

5.1 Security Notions

A blind signature scheme BS consists of the following algorithms:

- $\mathsf{Setup}(\lambda)$: It takes the security parameter 1^λ and returns public parameters param.
- $\mathsf{KeyGen}(\mathsf{param})$: It takes the public parameters param and returns a secret/public key pair $(\mathsf{sk}, \mathsf{pk})$.
- $\mathsf{Sign}(\mathsf{sk}), \mathsf{User}(\mathsf{pk}, M)$: an interactive protocol is run between the signer with private input a secret key sk and the user with private input a public key pk and a message M. The signer outputs $b = 1$ if the interaction completes successfully and $b = 0$ otherwise, while the user outputs a signature σ if it ends correctly, and $\perp$ otherwise.
- $\mathsf{Verify}(\mathsf{pk}, M, \sigma)$: it takes a public key pk, a message M, and a signature σ, and returns 1 if σ is valid on M under pk and returns 0 otherwise.

Definition 7 (Blind Adaptor Signature Scheme). *A blind adaptor signature (*BAS*) scheme* $\prod_{R,\mathsf{BS}}$ *with respect to a hard relation* R *with a language* $L_R := \{Y|\exists y : (Y, y) \in R\}$ *and a blind signature scheme* BS *consists of four algorithms* $(\mathsf{PreSign}, \mathsf{PreVerify}, \mathsf{Adapt}, \mathsf{Ext})$ *defined below.*

– $\mathsf{PreSign}(\mathsf{sk}, Y), \mathsf{User}(\mathsf{pk}, M, Y)$ *an interactive protocol is run between the signer with private input a secret key* sk *and the user with private input a public key* pk *and a message* M*. A statement* $Y \in L_R$ *is the public input. The signer outputs* $b = 1$ *if the interaction completes successfully and* $b = 0$ *otherwise, while the user outputs a pre-signature* $\hat{\sigma}$ *if it ends correctly, and* $\perp$ *otherwise.*
– $\mathsf{PreVerify}$*,* Adapt *and* Ext *are the same as the adaptor signature.*

5.2 Security Models

For the security models of BAS, we follow the security requirements of blind signature in [10] to define *one-more unforgeability* and *blindness*. We also define *pre-signature adaptability* and *witness extractability* as the adaptor signature.

In other to define the security model for BAS, suppose that there are N_s (resp. N_p) interactions by the signer in the $\mathsf{Sign}(\mathsf{sk})$ (resp. $\mathsf{PreSign}(\mathsf{sk}, Y)$) algorithm. We use $(m', st_1) \leftarrow \mathsf{Sign}_1(\mathsf{sk}, m)$ (resp. $(m', st_1) \leftarrow \mathsf{PreSign}_1(\mathsf{sk}, Y, m)$) to represent the first interaction, where m is the message received by the signer, m' is the message output and st_1 is the internal state. We use $(m', st_i) \leftarrow \mathsf{Sign}_i(st_{i-1}, m)$ (resp. $(m', st_1) \leftarrow \mathsf{PreSign}_i(st_{i-1}, m)$) to represent the i-th interaction, for $i \in [2, N_s - 1]$ (resp. $i \in [2, N_p - 1]$). We use $(m', b) \leftarrow \mathsf{Sign}_{N_s}(st_{N_s-1}, m)$ (resp. $(m', b) \leftarrow \mathsf{PreSign}_{N_p}(st_{N_p-1}, m)$) to represent the last interaction, where b is a bit.

One-more Unforgeability. The unforgeability model is defined to capture the attack that the adversary returns n distinct message-signature pairs when he is only given $k_2 < n$ pairs during the oracle queries. It is commonly known as the one-more unforgeability in blind signature [10].

Definition 8 (One-more Unforgeability). *A* BAS *scheme* $\Xi_{R,\Sigma}$ *is* $\mathsf{omaEUF{-}CMA}$ *secure if for every PPT adversary* $\mathcal{A}$ *running the experiment* $\mathsf{omaSignForge}_{\mathcal{A},\Xi_{R,\Sigma}}$ *defined in Algorithm 16,* $\Pr[\mathsf{omaSignForge}_{\mathcal{A},\Xi_{R,\Sigma}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$*.*

Blindness. The blindness security model of BAS is the same as that of blind signature in [10], except that the algorithm User takes an extra input Y. It is because BAS mainly modifies the algorithms in the signer side.
Pre-signature Adaptability. The pre-signature adaptability of BAS is the same as that of an adaptor signature. It is because the $\mathsf{PreSign}$ algorithm is not involved in the model.
Witness extractability. The witness extractability of BAS is different from that of the adaptor signature. It is because the message signed by the oracles O_S and O_{pS} is unknown to the challenger. Hence, we have to change the definition of the oracles to avoid giving a full signature to the adversary.

Definition 9 (Witness Extractability). *A* BAS *scheme* $\prod_{R,\Sigma}$ *is witness extractable if for every PPT adversary* $\mathcal{A}$ *running the experiment* $\mathsf{baWitExt}_{\mathcal{A},\prod_{R,\Sigma}}$ *defined in Algorithm 17,* $\Pr[\mathsf{baWitExt}_{\mathcal{A},\prod_{R,\Sigma}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$.

Algorithm 16 and Algorithm 17 can be found in Appendix C.

5.3 Clause Blind Schnorr Adaptor Signature

The clause blind Schnorr signature [10] can be viewed as the Schnorr identification with a crafted challenge, such that the message to be signed is hidden from the signer and the signer eventually signs one out of the two commitments. It is secure against the recent attack on the ROS problem [4]. Following our generic construction of adaptor signature in the last section, the clause blind Schnorr adaptor signature can be constructed as in Algorithm 11 in appendix A.

The clause blind Schnorr adaptor signature is omaEUF-CMA secure, has perfect blindness and pre-signature adaptability.

5.4 Security Proofs of Clause Blind Schnorr Adaptor Signature

Theorem 4. *The clause blind Schnorr adaptor signature is omaEUF-CMA secure if the clause blind Schnorr signature is omEUF-CMA secure.*

Proof. Suppose that there is a PPT adversary $\mathcal{A}$ breaking the aEUF-CMA security of the clause blind Schnorr adaptor signature. We build an algorithm $\mathcal{B}$ to break the security of the clause blind Schnorr signature. First, the challenger $\mathcal{C}$ of the clause blind Schnorr signature gives param and pk to $\mathcal{B}$. $\mathcal{B}$ forwards param and pk to $\mathcal{A}$.

For all signing oracle queries from $\mathcal{A}$ on a message M, $\mathcal{B}$ forwards them to the signing oracle of $\mathcal{C}$ to get the answer.

For the pre-signing oracle queries from $\mathcal{A}$ with input Y and session ID k, $\mathcal{B}$ asks the signing oracle of $\mathcal{C}$ and obtains (R_0, R_1). $\mathcal{B}$ returns $\hat{R}_0 = R_0 \cdot Y$ and $\hat{R}_1 = R_1 \cdot Y$ to $\mathcal{A}$. After that when $\mathcal{A}$ sends $\hat{c}_0, \hat{c}_1$ to $\mathcal{B}$ with the session ID k, $\mathcal{B}$ forwards the query to $\mathcal{C}$ and obtains (z, b). $\mathcal{B}$ answers $\mathcal{A}$ by (z, b).

After running $n-1$ queries to the signing oracle or the pre-signing oracle, $\mathcal{A}$ outputs n distinct message-signature pairs $(M_j, \sigma_j$
$= (z_j, c_j))$ for $j \in [1, n]$ such that $R_j = g^{z_j} \cdot \mathsf{pk}^{c_j}, c_j = H(M_j, R_j)$. Then $\mathcal{B}$ forwards them as the attack to $\mathcal{C}$.

Theorem 5. *The clause blind Schnorr adaptor signature has prefect blindness.*

Proof. (Sketch) Observe that the blinding step of the user during the $\mathsf{PreSign}$ algorithm is the same as the Sign algorithm in the original blind signature scheme. Hence, the clause blind Schnorr adaptor signature has the same level of blindness as the clause blind Schnorr signature, i.e., prefect blindness.

Theorem 6. *The clause blind Schnorr adaptor signature has pre-signature adaptability.*

Proof. If a pre-signature $\hat{\sigma} = (\hat{z}, c)$ passes $\mathsf{PreVerify}$, we have:

$$R' = g^{\hat{z}} Y \mathsf{pk}^c = g^{\hat{z}+y} \mathsf{pk}^c, \quad c = H(M, R').$$

After Adapt, we have the full signature $\sigma = (z = \hat{z} + y, c)$. Then σ can pass Verify since $g^z \mathsf{pk}^c = g^{\hat{z}+y} \mathsf{pk}^c = R'$.

Theorem 7. *The clause blind Schnorr adaptor signature has witness extractability.*

Proof. Suppose that there is a PPT adversary $\mathcal{A}$ breaking the witness extractability of our clause blind Schnorr adaptor signature. All oracle queries can be simulated by using the secret key.

In the challenge phase, $\mathcal{A}$ is given a pre-signature $\hat{\sigma} = (\hat{z}, c)$ for a message M^* and a statement Y, where $\hat{z} = r - \hat{c} \cdot \mathsf{sk} + \alpha$ and $c = H(M^*, R') + \beta$. Then $\mathcal{A}$ outputs a full signature $\sigma^* = (z^*, c^*)$, where $R^* = g^{z^*} \mathsf{pk}^{c^*}$. If $\mathcal{A}$ wins, then $\mathsf{Ext}(Y, \sigma, \hat{\sigma})$ did not output $\perp$. It implies $c^* = c - \beta$. By the collision resistant property of H, then $R^* = R'$. It implies $g^{z^*} \mathsf{pk}^{c^*} = g^r \cdot Y \cdot g^\alpha \cdot \mathsf{pk}^{-\beta}$. By the $\textsc{Ext}$ function, $\mathcal{B}$ computes $y = z^* - \hat{z}$. Hence we have:

$$g^{z^*} \mathsf{pk}^{c^*} = g^{y+\hat{z}} \mathsf{pk}^c = g^{y+r-\hat{c}\cdot\mathsf{sk}+\alpha} \mathsf{pk}^{\hat{c}-\beta} = g^r \cdot g^y \cdot g^\alpha \cdot \mathsf{pk}^{-\beta}.$$

Hence $\mathcal{B}$ can extract y such that $A(y) = Y$.

5.5 Discussion on Blind ECDSA Signature

ECDSA is commonly used in cryptocurrencies such as Bitcoin and Ethereum. Recently, blind ECDSA signature is proposed in [19] to provide recipient anonymity. However, there is no discussion about one-more unforgeability in [19]. It is not clear if any additional model is needed (e.g., the generic group model). Furthermore, the security proof for unforgeability in [19] is not rigorous: it is not clear how the signing oracle can be simulated without knowing the secret key. Therefore, we left the construction of blind ECDSA (adaptor) signature secure against one-more unforgeability as an interesting open problem.

6 Linkable Ring Adaptor Signature

In a linkable ring signature scheme [13], a signer has anonymity by hiding himself among a set of verification keys. However, if he signed twice, the two signatures are linked. It is useful in privacy-preserving cryptocurrency (e.g., Monero) to provide sender anonymity, while detecting double spending is feasible.

We will show that our generic adaptor signature can be applied to the linkable ring signature. In particular, we introduce the notion of *Linkable Ring Adaptor Signature* and define the corresponding security model. We will give a concrete construction based on the recent RingCT3.0 protocol [20].

6.1 Security Notions

A linkable ring adaptor signature (LRAS) scheme with respect to a hard relation R with a language $L_R := \{Y | \exists y : (Y, y) \in R\}$ and a linkable ring signature scheme $\Sigma = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Link})$ consists of four algorithms $\Xi_{R,\Sigma} = (\mathsf{PreSign}, \mathsf{PreVerify}, \mathsf{Adapt}, \mathsf{Ext})$ defined as:

- $\mathsf{PreSign}(\mathsf{sk}, m, \vec{\mathsf{vk}}, Y)$: On input a secret key sk, a message m, a set of verification keys $\vec{\mathsf{vk}} = (\mathsf{vk}_1, \ldots, \mathsf{vk}_\mathsf{n})$ and a statement $Y \in L_R$, outputs a pre-signature $\hat{\sigma}$.
- $\mathsf{PreVerify}(m, \vec{\mathsf{vk}}, Y, \hat{\sigma})$: On input a message $m \in \{0,1\}^*$, a set of verification keys $\vec{\mathsf{vk}} = (\mathsf{vk}_1, \ldots, vk_n)$, a statement $Y \in L_R$ and a pre-signature $\hat{\sigma}$, outputs 1 for valid pre-signature or 0 otherwise.
- $\mathsf{Adapt}((Y, y), \vec{\mathsf{vk}}, \hat{\sigma}, m)$: On input $(Y, y) \in R$, a set of verification keys $\vec{\mathsf{vk}}$, a pre-signature $\hat{\sigma}$ and a message m, outputs a signature σ.
- $\mathsf{Ext}(Y, \hat{\sigma}, \sigma)$: on input a statement $Y \in L_R$, a signature σ and a pre-signature $\hat{\sigma}$, outputs a witness y such that $(Y, y) \in R$, or $\perp$.

Algorithm 6: Experiment $\mathsf{aSignForge}_{\mathcal{A},\Xi_{R,\Sigma}}$

```
1  Procedure $\mathit{aSignForge}_{\mathcal{A},\Xi_{R,\Sigma}}(\lambda)$:
2      $\mathcal{Q} := \emptyset$ , $\mathcal{F} := \emptyset$, $\mathcal{T} := \emptyset$;
3      $(\mathsf{vk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(1^\lambda)$, for i$\in[1, N]$;
4      $\vec{\mathsf{vk}} = \{\mathsf{vk}_1, \mathsf{vk}_2, ..., \mathsf{vk}_N\}$;
5      $(m^*, \mathsf{vk}_{i^*}, \hat{\mathbf{vk}}) \leftarrow \mathcal{A}^{O_\mathsf{S}(\cdot), O_\mathsf{pS}(\cdot,\cdot), O_\mathsf{Corrupt}(\cdot)}(\vec{\mathsf{vk}})$, where $\mathsf{vk}_{i^*} \in \hat{\mathbf{vk}} \subseteq \vec{\mathsf{vk}}$ ;
6      $(\mathsf{Y}, \mathsf{y}) \leftarrow \mathsf{LockGen}(\lambda)$;
7      $\hat{\sigma} \leftarrow \mathsf{PreSign}(\mathsf{sk}_{i^*}, m^*, \hat{\mathbf{vk}}, \mathsf{Y})$;
8      $(\tilde{\mathbf{vk}}, \sigma^*) \leftarrow \mathcal{A}^{O_\mathsf{S}(\cdot), O_{\mathsf{pS}(\cdot,\cdot)}, O_\mathsf{Corrupt}(\cdot)}(\hat{\sigma}, \mathsf{Y})$;
9      return $((\tilde{\mathbf{vk}} \subseteq \vec{\mathsf{vk}} \backslash \mathcal{F}) \wedge ((\star, m^*, \tilde{\mathbf{vk}}) \notin \mathcal{Q}) \wedge \mathsf{Verify}(m^*, \tilde{\mathbf{vk}}, \sigma^*) = 1)$;
10 Procedure $O_\mathsf{Corrupt}(j)$:
11     $\mathcal{F} := \mathcal{F} \cup \{\mathsf{vk}_j\}$;
12     return $\mathsf{sk}_j$;
13 Procedure $O_\mathsf{S}(m, i, \overline{\vec{\mathsf{vk}}})$:
14     $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_i, m, \overline{\vec{\mathsf{vk}}})$ /* Require that $\mathsf{vk}_i \in \overline{\vec{\mathsf{vk}}}$ */
15     $\mathcal{Q} := \mathcal{Q} \cup \{i, m, \overline{\vec{\mathsf{vk}}}\}$ ;
16     return $\sigma$;
17 Procedure $O_\mathsf{pS}(m, i, \overline{\vec{\mathsf{vk}}}, Y)$:
18     $\hat{\sigma} \leftarrow \mathsf{PreSign}(\mathsf{sk}_i, m, \overline{\vec{\mathsf{vk}}}, Y)$ /* Require that $\mathsf{vk}_i \in \overline{\vec{\mathsf{vk}}}$ and $Y \in L_R$
           */
19     $\mathcal{Q} := \mathcal{Q} \cup \{i, m, \overline{\vec{\mathsf{vk}}}\}$ ;
20     return $\hat{\sigma}$;
```

Algorithm 7: Experiment $\mathsf{aAnon}_{\mathcal{A},\Xi_{R,\Sigma}}$

```
1  Procedure $\mathsf{aAnon}_{\mathcal{A},\Xi_{R,\Sigma}}(\lambda)$:
2      $\mathcal{Q} := \emptyset$;
3      $(\mathsf{vk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(1^\lambda)$, for i$\in[1, N]$;
4      $\mathbf{vk} = \{\mathsf{vk}_1, \mathsf{vk}_2, ..., \mathsf{vk}_N\}$;
5      $(m^*, i_0, i_1, \tilde{\mathbf{vk}}, Y) \leftarrow \mathcal{A}^{O_\mathsf{S}(\cdot), O_\mathsf{pS}(\cdot,\cdot)}(\mathbf{vk})$, where $\mathsf{vk}_{i_0}, \mathsf{vk}_{i_1} \in \tilde{\mathbf{vk}} \cup \mathbf{vk}$
          /* $O_\mathsf{S}, O_\mathsf{pS}$ are the same as Algorithm 6 */
6      $b \xleftarrow{\$} \{0, 1\}$;
7      $\hat{\sigma} \leftarrow \mathsf{PreSign}(\mathsf{sk}_{i_b}, m^*, \tilde{\mathbf{vk}}, Y)$;
8      $b' \leftarrow \mathcal{A}^{O_\mathsf{S}, O_\mathsf{pS}}(\hat{\sigma}, Y)$;
9      if $(b = b' \wedge \{i_0, \star, \star\} \notin \mathcal{Q} \wedge \{i_1, \star, \star\} \notin \mathcal{Q})$ then
10         return 1;
11     return 0;
```

6.2 Security Models

The security models for LRAScombines the security requirements from both linkable ring signatures (unforgeability, linkable anonymity, non-slanderability) and adaptor signatures (unforgeability, pre-signature adaptability, witness extractability). They are formally given in the following definitions.

Definition 10 (aEUF–CMA security). *A* LRAS *scheme* $\Xi_{R,\Sigma}$ *is* aEUF–CMA *secure if for every PPT adversary* $\mathcal{A}$ *runing the experiment* $\mathsf{aSignForge}_{\mathcal{A},\Xi_{R,\Sigma}}$ *defined in Algorithm 6,* $\Pr[\mathsf{aSignForge}_{\mathcal{A},\Xi_{R,\Sigma}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$.

Definition 11 (Pre-signature anonymity). *A* LRAS *scheme* $\Xi_{R,\Sigma}$ *achieves linkable anonymity if for any PPT adversary* $\mathcal{A}$ *running the experiment* $\mathsf{aAnon}_{\mathcal{A},\Xi_{R,\Sigma}}$ *defined in Algorithm 7,* $\left|\Pr[\mathsf{aAnon}_{\mathcal{A},\Xi_{R,\Sigma}}(\lambda) = 1] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda)$.

Definition 12 (Linkability). *A* LRAS *scheme* $\Xi_{R,\Sigma}$ *satisfies pre-signature linkability w.r.t. insider corruption if for any PPT adversary* $\mathcal{A}$ *running the experiment* $\mathsf{aLink}_{\mathcal{A},\Xi_{R,\Sigma}}$ *defined in Algorithm 8,* $\left|\Pr[\mathsf{aLink}_{\mathcal{A},\Xi_{R,\Sigma}}(\lambda) = 1] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda)$.

Definition 13 (Non-Slanderability). *A* LRAS *scheme* $\Xi_{R,\Sigma}$ *satisfies non-slanderability if for any PPT adversary* $\mathcal{A}$ *running the experiment* $\mathsf{aSlan}_{\mathcal{A},\Xi_{R,\Sigma}}$ *defined in Algorithm 9,* $\left|\Pr[\mathsf{aNSlan}_{\mathcal{A},\Xi_{R,\Sigma}}(\lambda) = 1] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda)$.

Algorithm 8: Experiment $\mathsf{aLink}_{\mathcal{A},\Xi_{\mathsf{R},\Sigma}}$

```
1 Procedure aLink$_{\mathcal{A},\Xi_{R,\Sigma}}(\lambda)$:
2     $\mathcal{Q} := \emptyset$, $\mathcal{F} := \emptyset$;
3     $(\mathsf{vk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(1^\lambda)$, for i$\in[1, N]$;
4     $\mathbf{vk} = \{\mathsf{vk}_1, \mathsf{vk}_2, ..., \mathsf{vk}_N\}$;
5     $(m_i, \tilde{\mathbf{vk}}_i, \sigma_i)_{i=1,2} \leftarrow \mathcal{A}^{O_\mathsf{S}, O_\mathsf{pS}, O_\mathsf{Corrupt}}(\mathbf{vk})$ /* $O_\mathsf{S}, O_\mathsf{pS}, O_\mathsf{Corrupt}$ are the same
          as Algorithm 6                                    */
6     $b_1 := \mathsf{Verify}(m_i, \tilde{\mathbf{vk}}_i, \sigma_i)_{i=1,2}$;
7     $b_2 := \mathsf{Link}((m_1, \tilde{\mathbf{vk}}_1, \sigma_1), (m_2, \tilde{\mathbf{vk}}_2, \sigma_2))$;
8     $b_3 := \{\star, m_i, \tilde{\mathbf{vk}}_i\}_{i=1,2} \notin \mathcal{Q}$;
9     $b_4 := \left|((\tilde{\mathbf{vk}}_1 \cup \tilde{\mathbf{vk}}_2) \cap \mathcal{F}) \cup ((\tilde{\mathbf{vk}}_1 \cup \tilde{\mathbf{vk}}_2) \backslash \mathbf{vk})\right|$;
10    return$(b_1 = 1 \wedge b_2 = 0 \wedge b_3 = 1 \wedge b_4 \leq 1)$;
```

Algorithm 9: Experiment $\mathsf{aNSlan}_{\mathcal{A},\Xi_{\mathsf{R},\Sigma}}$

```
1 Procedure aNSlan$_{\mathcal{A},\Xi_{R,\Sigma}}(\lambda)$:
2     $\mathcal{Q} = \emptyset$, $\mathcal{F} = \emptyset$;
3     $(\mathsf{vk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(1^\lambda)$, for $i\in[1, N]$ ;
4     $\boldsymbol{vk} = \{\mathsf{vk}_1, \mathsf{vk}_2, ..., \mathsf{vk}_N\}$;
5     $(m, \tilde{\boldsymbol{vk}}, \sigma) \leftarrow \mathcal{A}^{O_\mathsf{S}, O_\mathsf{pS}, O_\mathsf{Corrupt}}(\boldsymbol{vk})$ /* $O_\mathsf{S}, O_\mathsf{pS}, O_\mathsf{Corrupt}$ are the same
          as Algorithm 6                                    */
6     $b_1 := \mathsf{Verify}(m, \tilde{\boldsymbol{vk}}, \sigma)$;
7     $b_2 := \{\star, \star, \sigma\} \notin \mathcal{Q}$;
8     $b_3 := \mathsf{Link}((m, \tilde{\boldsymbol{vk}}, \sigma), (\hat{m}, \hat{\boldsymbol{vk}}, \hat{\sigma})) = 1, \exists\{\hat{m}, \hat{\boldsymbol{vk}}, \hat{\sigma}\} \in \mathcal{Q}$;
9     $b_4 := (\boldsymbol{vk} \cap \hat{\boldsymbol{vk}} \cap (\tilde{\boldsymbol{vk}} \backslash \mathcal{F})) \neq \emptyset$;
10    return $(b_1 = 1 \wedge b_2 = 1 \wedge b_3 = 1 \wedge b_4 = 1)$;
```

Definition 14 (Pre-signature adaptability). *A* LRAS *scheme* $\Xi_{R,\Sigma}$ *satisfies pre-signature adaptability if for any* $n \in \mathbb{N}$, *any message* $m \in \{0,1\}^*$, *any statement/witness pair* $(T, t) \in R$, *any key pair* $(\mathsf{vk}, \mathsf{sk}) \overset{\$}{\leftarrow} \mathsf{KeyGen}(1^\lambda)$, *any set of verification keys* $\boldsymbol{vk} = \{\mathsf{vk}_1, ..., \mathsf{vk}_N\}$ *and any pre-signature* $\hat{\sigma} \leftarrow \{0,1\}^*$ *with* $\mathsf{PreVerify}(m, \boldsymbol{vk}, T, \hat{\sigma}) = 1$, *we have* $\Pr[\mathsf{Verify}(m, \boldsymbol{vk}, \mathsf{Adapt}((T, t), \boldsymbol{vk}, \hat{\sigma}, M)) = 1] = 1$.

Definition 15 (Witness extractability). *A linkable ring adaptor signature scheme* $\Xi_{R,\Sigma}$ *is witness extractable if for every PPT adversary* $\mathcal{A}$ *running the experiment* $\mathsf{aWitExt}_{\mathcal{A},\Xi_{R,\Sigma}}$ *defined in Algorithm 10,* $\Pr[\mathsf{aWitExt}_{\mathcal{A},\Xi_{R,\Sigma}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$.

Algorithm 10: Experiment $\mathsf{aWitExt}_{\mathcal{A},\Xi_{R,\Sigma}}$

```
1 Procedure aWitExt_{A,Ξ_{R,Σ}}(λ):
2     $\mathcal{Q} = \emptyset$;
3     $(\mathsf{vk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(1^\lambda), for\ i \in [1, N]$ ;
4     $\boldsymbol{vk} = \{\mathsf{vk}_1, \mathsf{vk}_2, ..., \mathsf{vk}_N\}$;
5     $(m^*, i^*, Y^*, \tilde{\boldsymbol{vk}}) \leftarrow \mathcal{A}^{O_\mathsf{S}, O_\mathsf{pS}}(\boldsymbol{vk})$ /* $O_\mathsf{S}, O_\mathsf{pS}$ are the same as
        Algorithm 6                                                  */
6     $\hat{\sigma} \leftarrow \mathit{PreSign}(\mathsf{sk}_{i^*}, m^*, \tilde{\boldsymbol{vk}}, Y^*)$;
7     $\sigma \leftarrow \mathcal{A}^{O_\mathsf{S}, O_\mathsf{pS}}(\hat{\sigma})$;
8     $t' := \mathsf{Ext}(Y^*, \sigma, \hat{\sigma})$;
9     return $(m^* \notin \mathcal{Q} \wedge (Y^*, y') \notin R \wedge \mathsf{Verify}(m^*, \tilde{\boldsymbol{vk}}, \sigma) = 1)$;
```

6.3 Generic Construction of LRAS

We first give a definition of Type-T Linkable Ring Canonical Identification in Algorithm 12. Then Type-TA Linkable Ring Canonical Identification can be defined in the same way as the previous section.

Then, we can build a generic construction of LRAS in Algorithm 15. The security of the generic construction of LRAS can be reduced to the underlying Type-TA linkable ring identification scheme or the generic adaptor signature. The security proofs are almost the same as the proofs of the generic adaptor signature, and hence they are omitted.

Observe that the linkable ring signature schemes in RingCT [15] and RingCT3.0 [20] are both Type-TA linkable ring canonical identification. Hence we can construct a concrete linkable ring adaptor signature similarly. Details are given in the Appendix B.

7 Conclusion

Adaptor signatures are a novel cryptographic primitive with important applications for cryptocurrencies. In this paper, we propose the first generic construction of adaptor signature. It can be combined with a number of different cryptographic protocols, such as blind adaptor signature and linkable ring adaptor signature. An interesting open question is that whether we can give a more generalized version of adaptor signature that can include non-Type-T signatures such as ECDSA. We leave it as the future work.

References

1. Abdalla, M., An, J.H., Bellare, M., Namprempre, C.: From identification to signatures via the fiat-shamir transform: Necessary and sufficient conditions for security and forward-security. IEEE Trans. Information Theory **54**(8), 3631–3646 (2008)

2. Abe, M., Ohkubo, M., Suzuki, K.: 1-out-of-n signatures from a variety of keys. In: Zheng, Y. (ed.) ASIACRYPT 2002. Lecture Notes in Computer Science, vol. 2501, pp. 415–432. Springer (2002)
3. Aumayr, L., Ersoy, O., Erwig, A., Faust, S., Hostakova, K., Maffei, M., Moreno-Sanchez, P., Riahi, S.: Generalized bitcoin-compatible channels. Cryptology ePrint Archive, Report 2020/476 (2020), `https://eprint.iacr.org/2020/476`
4. Benhamouda, F., Lepoint, T., Loss, J., Orrù, M., Raykova, M.: On the (in)security of ros. Cryptology ePrint Archive, Report 2020/945 (2020), `https://eprint.iacr.org/2020/945`
5. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.: High-speed high-security signatures. In: Preneel, B., Takagi, T. (eds.) CHES 2011. Lecture Notes in Computer Science, vol. 6917, pp. 124–142. Springer (2011)
6. Bowe, S., Hopwood, D.: "hashed time-locked contract transactions". `https://github.com/bitcoin/bips/blob/master/bip-0199.mediawiki/`
7. Erwig, A., Faust, S., Hostáková, K., Maitra, M., Riahi, S.: Two-party adaptor signatures from identification schemes. Cryptology ePrint Archive, Report 2021/150 (2021), `https://eprint.iacr.org/2021/150`. To appear in PKC 2021.
8. Esgin, M.F., Ersoy, O., Erkin, Z.: Post-quantum adaptor signatures and payment channel networks. In: Chen, L., Li, N., Liang, K., Schneider, S.A. (eds.) ESORICS 2020. Lecture Notes in Computer Science, vol. 12309, pp. 378–397. Springer (2020)
9. Fournier, L.: One-time verifiably encrypted signatures a.k.a. adaptor signatures. https://github.com/LLFourn/one-time-VES/blob/master/main.pdf (October 2019) (2019)
10. Fuchsbauer, G., Plouviez, A., Seurin, Y.: Blind schnorr signatures and signed elgamal encryption in the algebraic group model. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. Lecture Notes in Computer Science, vol. 12106, pp. 63–95. Springer (2020)
11. Guillou, L.C., Quisquater, J.: A "paradoxical" indentity-based signature scheme resulting from zero-knowledge. In: Goldwasser, S. (ed.) CRYPTO '88. Lecture Notes in Computer Science, vol. 403, pp. 216–231. Springer (1988)
12. Katz, J., Wang, N.: Efficiency improvements for signature schemes with tight security reductions. In: Jajodia, S., Atluri, V., Jaeger, T. (eds.) CCS 2003. pp. 155–164. ACM (2003)
13. Liu, J.K., Wei, V.K., Wong, D.S.: Linkable spontaneous anonymous group signature for ad hoc groups (extended abstract). In: Wang, H., Pieprzyk, J., Varadharajan, V. (eds.) ACISP 2004. Lecture Notes in Computer Science, vol. 3108, pp. 325–335. Springer (2004)
14. Malavolta, G., Moreno-Sanchez, P., Schneidewind, C., Kate, A., Maffei, M.: Anonymous multi-hop locks for blockchain scalability and interoperability. In: 26th Annual Network and Distributed System Security Symposium, NDSS 2019 (2019)
15. Noether, S.: Ring signature confidential transactions for monero. Cryptology ePrint Archive, Report 2015/1098 (2015), `https://eprint.iacr.org/2015/1098`
16. Poelstra, A.: Scriptless scripts (2017)
17. Schnorr, C.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) CRYPTO '89. Lecture Notes in Computer Science, vol. 435, pp. 239–252. Springer (1989)
18. Thyagarajan, S.A.K., Malavolta, G.: Lockable signatures for blockchains: Scriptless scripts for all signatures. Cryptology ePrint Archive, Report 2020/1613 (2020), `https://eprint.iacr.org/2020/1613`

19. Yi, X., Lam, K.: A new blind ECDSA scheme for bitcoin transaction anonymity. In: Galbraith, S.D., Russello, G., Susilo, W., Gollmann, D., Kirda, E., Liang, Z. (eds.) AsiaCCS 2019. pp. 613–620. ACM (2019)
20. Yuen, T.H., Sun, S., Liu, J.K., Au, M.H., Esgin, M.F., Zhang, Q., Gu, D.: Ringct 3.0 for blockchain confidential transaction: Shorter size and stronger security. In: Bonneau, J., Heninger, N. (eds.) FC 2020. Lecture Notes in Computer Science, vol. 12059, pp. 464–483. Springer (2020)

A Details of Clause Blind Schnorr Signature

Algorithm 11: Clause Blind Schnorr adaptor signature for the language $L := \{Y | \exists y : Y = g^y\}$.

```
Procedure Setup($\lambda$):
    return $\mathsf{param} = (\mathbb{G}, g)$;
Procedure KeyGen(param):
    $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_p$, $\mathsf{pk} = g^{\mathsf{sk}}$;
    return $(\mathsf{sk}, \mathsf{pk})$;
Procedure PreSign$(\mathsf{sk}, Y) \leftrightarrow$ User$(\mathsf{pk}, M, Y)$:
    PreSign: For $i \in [0,1]$, $r_i \xleftarrow{\$} \mathbb{Z}_p$, $R_i = g^{r_i} \cdot Y$, send $R_0, R_1$ to user;
    User: For $i \in [0,1]$, $\alpha_i, \beta_i \xleftarrow{\$} \Delta_r$, $R'_i = R_i \cdot g^{\alpha_i} \cdot \mathsf{pk}^{-\beta_i}$, $c_i = H(M, R'_i)$, $\hat{c}_i = c_i + \beta_i$, send $\hat{c}_0, \hat{c}_1$ to signer;
    PreSign: Pick a bit $b \xleftarrow{\$} \{0,1\}$, send $z = r_b - \hat{c}_b \cdot \mathsf{sk}$ and $b$ to user;
    User: If $R_b \neq g^z \cdot Y \cdot \mathsf{pk}^{\hat{c}_b}$, return $\perp$, else return $\hat{\sigma} = (\hat{z} = z + \alpha_b, c_b)$;
Procedure PreVerify$(Y, \mathsf{pk}, \hat{\sigma}, M)$:
    parse $\hat{\sigma} = (\hat{z}, c)$;
    $R' = g^{\hat{z}} \cdot Y \cdot \mathsf{pk}^c$;
    if $c \neq H(M, R')$ then
        return 0;
    return 1;
Procedure Adapt$((Y, y), \mathsf{pk}, \hat{\sigma}, M)$:
    parse $\hat{\sigma} = (\hat{z}, c)$;
    $z = \hat{z} + y$;
    return $\sigma = (z, c)$;
Procedure Ext$(Y, \hat{\sigma}, \sigma)$:
    parse $\hat{\sigma} = (\hat{z}, \hat{c})$ and $\sigma = (z, c)$;
    if $\hat{c} = c$ then
        return $y = z - \hat{z}$;
    return $\perp$;
Procedure Verify$(\mathsf{pk}, M, \sigma)$:
    parse $\sigma = (z, c)$;
    $R' = g^z \cdot \mathsf{pk}^c$;
    if $c \neq H(M, R')$ then
        return 0;
    return 1;
```

B Details of Linkable Ring Adaptor Signatures

We give the linkable ring adaptor signature in Algorithm 13 and 14. It is based on the ring signature in [20]. The **Verify** protocol is the same as that in [20], which is very similar to the **PreVerify** and hence it is omitted. Note that $\vec{y}^n = (1, y, y^2, \ldots, y^{n-1})$ for some integer y.

Security.

Lemma 1 (aEUF-CMA security). *Assuming that DL assumption holds and L_R is a hard relation, the linkable ring adaptor signature scheme $\Xi_{R,\Sigma}$ is* $\mathsf{aEUF\text{-}CMA}$ *secure in the random oracle model.*

Proof. Let $\mathcal{B}$ be a PPT adversary who wins the $\mathsf{aEUF\text{-}CMA}$ security game with non-negligible probability. We will build an adversary $\mathcal{A}$ that breaks the DL assumption or the hardness of L_R. Assume $\mathcal{A}$ wants to solve DL w.r.t. (g, g^a).
Oracle simulation. For the KeyGen query, $\mathcal{A}$ picks some random sk_i and returns $\mathsf{pk}_i = g^{\mathsf{sk}_i}$. Except for the i^*th query, the $\mathcal{A}$ returns g^a. For the $\mathsf{Corrupt}$ oracle, the $\mathcal{A}$ declares failure and exits if secret key of pk_{i^*} is requested. For the $\mathsf{PreSign}$ oracle, if the queried key is not the i^*th one, $\mathcal{A}$ runs honestly as the $\mathsf{PreSign}$ algorithm. Otherwise, $\mathcal{A}$ generates the pre-signature for pk_{i^*} as follows: everything is the same as the $\mathsf{PreSign}$ algorithm, except the following: the $\mathcal{A}$ chooses z_α, z'_{sk} and x at random and let $S_1 = h^{z_\alpha - \alpha x}(g^{z'_{\mathsf{sk}}}/\mathsf{pk}_{i^*}^x)T$. Then $\mathcal{A}$ computes y,z,w as that in the $\mathsf{PreSign}$ algorithm and finally programs the random oracle such that $x = \hat{H}(4||y||z||w||T_1||T_2||m)$. If this input of $\hat{H}$ is queried before, $\mathcal{A}$ aborts. Otherwise, the $\mathcal{A}$ returns $\hat{\sigma}$. For the Sign oracle, if the w in the input is NULL, the $\mathcal{A}$ chooses z_α, z'_{sk} and x at random and let $S_1 = h^{z_\alpha - \alpha x}(g^{z'_{\mathsf{sk}}}/\mathsf{pk}_{i^*}^x)$. Then $\mathcal{A}$ computes y, z, w as the $\mathsf{PreSign}$ algorithm and finally programs the random oracle such that $x = \hat{H}(4||y||z||w||T_1||T_2||m)$. If this input of $\hat{H}$ is queried before, $\mathcal{A}$ aborts. Otherwise, the $\mathcal{A}$ returns σ. If the w in the input is (T,t), the $\mathcal{A}$ firstly runs the $\mathsf{PreSign}$ simulation procedure as above and get the $\hat{\sigma}$. Then it runs the Adapt algorithm and gets the σ. If the same input of $\hat{H}$ is queried before, $\mathcal{A}$ aborts. Otherwise, the $\mathcal{A}$ returns σ.
Consider that $\mathcal{B}$ makes at most $Q_{\mathsf{KeyGen}}, Q_{\mathsf{PreSign}}, Q_{\mathsf{Sign}}$ and Q_H queries to KeyGen, $\mathsf{PreSign}, \mathsf{Sign}$ and random oracle respectively.
Forgery. $\mathcal{B}$ returns the target message $(m^*, \mathsf{vk}_{i^*}, \hat{\mathbf{vk}}^*)$ to $\mathcal{A}$. $\mathcal{A}$ chooses a $(\mathsf{T}^*, \mathsf{t}^*)$ from $\mathsf{LockGen}$ that is not been used before and computes a pre-signature $\hat{\sigma}^*$ using the simulation method above. Then $\mathcal{A}$ sends $(\hat{\sigma}^*, T^*)$ to $\mathcal{B}$. Finally, $\mathcal{B}$ returns a forged linkable ring adaptor signature $(\tilde{vk}^*, \tilde{\sigma})$ on m^* for $((\tilde{vk}^* \subseteq \mathbf{vk} \backslash \mathcal{F}) \wedge ((\star, m^*, \tilde{vk}^*) \notin \mathcal{Q}) \wedge \mathsf{Verify}(m^*, \tilde{vk}^*, \tilde{\sigma})) = 1$, where the $\mathcal{F}, \mathcal{Q}$ are the same as that in the $\mathsf{aSignForge}$ experiment. $\tilde{\sigma}$ is denoted as $(\tilde{B}, \tilde{A}, \tilde{S_1}, \tilde{S_2}, \tilde{T_1}, \tilde{T_2}, \tilde{\tau}, \tilde{\mu}, \tilde{z_\alpha}, \tilde{z_{\mathsf{sk}}}, \tilde{\zeta}, \tilde{\pi})$.
Case 1 : $\hat{\mathbf{vk}}^* = \mathbf{vk}$ and all the elements of $\tilde{\sigma}$ and $\sigma^* = \mathsf{Adapt}(t^*, \hat{\sigma}^*)$ are the same. This means $\mathcal{B}$ gets the t^*, which breaks the hardness of L_R.
Case 2 : Case 1 has not happened. The $\mathcal{A}$ computes the corresponding y,z,w as in the $\mathsf{PreSign}$ algorithm first and rewinds $\hat{H}$ on input $(4||y||z||w||\tilde{T_1}||\tilde{T_2})$ for three times. For each transcript, denote the challenge as x_i and the responses as $(\tau_{x,i}, \mu_i, z_{\alpha,i}, z_{\mathsf{sk},i}, z_{\delta,i}, \mathbf{l}_i, \mathbf{r}_i, t_i)$ for $i \in [1, 3]$. Denote $\mathbf{l}_i = (l_{i,1}, \ldots, l_{i,n})$ and $\mathsf{r}_i = (r_{i,1}, \ldots, r_{i,n})$.

- To extract BA^w, it picks some $\eta_i \in \mathbb{Z}_p$ such that $\sum_{i=1}^2 \eta_i = 1, \sum_{i=1}^2 \eta_i x_i = 0$. Then we have:

$$\begin{aligned} BA^w &= h^{\sum_{i=1}^2 \eta_i \mu_i} \mathbf{vk}^{\sum_{i=1}^2 \eta_i \cdot l_i + z \cdot 1^n} \vec{H}'^{\sum_{i=1}^2 \eta_i r_i - z^2 \cdot 1^n} \vec{H}^{-wz} \\ &:= h^{\gamma'} \mathbf{vk}^{\vec{b_L}'} \vec{H}^{w\vec{b_R}'} \end{aligned} \tag{2}$$

  for some $\gamma', \vec{b_L}', \vec{b_R}'$.
- To extract S_2, it picks some $\eta_i' \in \mathbb{Z}_p$ such that $\sum_{i=1}^2 \eta_i' = 0, \sum_{i=1}^2 \eta_i' x_i = 1$. Then we have:

$$S_2 = h^{\sum_{i=1}^2 \eta_i' \mu_i} \mathbf{vk}^{\sum_{i=1}^2 \eta_i' l_i} \vec{H}'^{\sum_{i=1}^2 \eta_i' r_i} := h^{\rho'} Y^{\vec{s_L}'} \vec{H}^{\cdot \vec{s_R}'} \tag{3}$$

  for some $\gamma', \vec{s_L}', \vec{s_R}'$.

Putting back the extracted values BA$_w$ and S_2 into $P = BA^w S_2^x \mathbf{vk}^{-z \cdot \vec{1}^n} \vec{H}^{wz} \vec{H}'^{z^2 \cdot \vec{1}^n} h^{-\mu}$, we have:

$$\mathbf{vk}^{\vec{l}} \vec{H}'^{\vec{r}} = (h^{\gamma'} \mathbf{vk}^{\vec{b_L}'} \vec{H}^{w\vec{b_R}'}) \cdot (h^{\rho'} Y^{\vec{s_L}'} \vec{H}^{\cdot \vec{s_R}'})^x \cdot \mathbf{vk}^{-z \cdot 1^n} \cdot \vec{H}^{wz} \vec{H}'^{z^2 \cdot \vec{1}^n} h^{-\mu}$$

The mutual discrete logarithm between **vk** is not known if the discrete logarithm assumption holds by lemma 1. Since $\vec{H}$ and the elements in h are randomly chosen from the group, the mutual discrete logarithm between h, the elements in $\vec{H}$ and **vk** is not known.
Then we have: $\vec{l} = \vec{b_L}' - z \cdot 1^n + \vec{s_L}' \cdot x, \vec{r} = y^n \circ (w \cdot \vec{b_R}' + wz \cdot 1^n + \vec{s_R}' \cdot x) + z^2 \cdot 1^n$

By the same set of 3 rewinding transcripts, we can also extract the commitments T_1,T_2 as follows.

- To extract T_1, it picks some $\delta_i \in \mathbb{Z}_p$ such that $\sum_{i=1}^3 \delta_i = 0, \sum_{i=1}^3 \delta_i x_i = 1, \sum_{i=1}^3 \delta_i x_i^2 = 0$. Then we have:

$$T_1 = g^{\sum_{i=1}^3 \delta_i t_i} h^{\sum_{i=1}^3 \delta_i \tau_{x,i}} := g^{t_1'} h^{r_1'}$$

  for some t_1', r_1'.

- To extract T_2, it picks some $\delta_i' \in \mathbb{Z}_p$ such that $\sum_{i=1}^3 \delta_i' = 0, \sum_{i=1}^3 \delta_i' x_i = 1, \sum_{i=1}^3 \delta_i' x_i^2 = 0$. Then we have:

$$T_2 = g^{\sum_{i=1}^3 \delta_i' t_i} h^{\sum_{i=1}^3 \delta_i' \tau_{x,i}} := g^{t_2'} h^{r_2'}$$

  for some t_2', r_2'.

Putting back the extracted values T_1 and T_2 into equation $g^\zeta h^\tau = g^{z^2 + wz(1-z)\sum_{i=1}^n y^{i-1} - nz^3} T_1^x T_2^{x^2}$, we have: $g^\zeta h^\tau = g^{z^2 + w(z-z^2)\langle 1^n, y^n \rangle - z^3 \langle 1^n, 1^n \rangle} \cdot (g^{t_1'} h^{r_1'})^x \cdot (g^{t_2'} h^{r_2'})^{x^2}$. Since h is a random group element by the simulation of $\hat{H}_G$, we have:

$$\zeta = z^2 + w(z - z^2)\langle 1^n, y^n \rangle - z^3 \langle 1^n, 1^n \rangle + t_1' x + t_2' x^2$$

Denote $t'_0 = z^2 + w(z - z^2)\langle 1^n, y^n\rangle - z^3\langle 1^n, 1^n\rangle$.
Observe that we already extracted $\vec{l}, \vec{r}$ as:

$$\begin{aligned}\langle \vec{l}, \vec{r}\rangle &= (\vec{b_L}' - z \cdot 1^n + \vec{s_L}' \cdot x) \cdot (y^n \circ (w \cdot \vec{b_R}' + wz \cdot 1^n + \vec{s_R}' \cdot x) + z^2 \cdot 1^n) \\ &= w\langle \vec{b_L}', \vec{b_R}' \circ y^n\rangle + wz\langle \vec{b_L}' - \vec{b_R}', y^n\rangle + z^2\langle \vec{b_L}', 1^n\rangle - wz^2\langle 1^n, y^n\rangle - z^3\langle 1^n, 1^n\rangle + t''_1 x + t''_2 x^2\end{aligned}$$

for some $t''_1, t''_2 \in \mathbb{Z}_p$. Since the above holds for all w, x, y, z, we have:

$$\vec{b_L}' \circ \vec{b_R}' = 0^n, \quad \vec{b_L}' - \vec{b_R}' = 1^n, \quad \langle \vec{b_L}', 1^n\rangle = 1$$

Therefore, it implies that $\vec{b_L}'$ is a binary vector with one bit equal to 1. Putting back $\vec{b_L}'$ in equation 2, we have: $BA^w = h^{\gamma'}\tilde{\mathsf{pk}}\vec{H}^{w\vec{b_R}'}$. Since the above is true for all w, then we have B $= h^{\alpha'}\tilde{\mathsf{pk}}$ for some $\alpha' \in \mathbb{Z}_p$.
By the same set of rewinding transcripts, we can also extract from $h^{z_\alpha} g^{z_{sk}} T = S_1 B^x$: $B = h^{\frac{z_{\alpha,1} - z_{\alpha,2}}{x_1 - x_2}} g^{\frac{z_{sk,1} - z_{sk,2}}{x_1 - x_2}} := h^{\alpha''} g^{\mathsf{sk}'}$. Since the above is true for all h, then we have $\tilde{\mathsf{pk}} = \mathrm{g}^{\mathsf{sk}'}$. Hence if $\tilde{\mathsf{pk}} = \mathsf{pk}_{i^*}$, then the $\mathcal{A}$ returns sk' as the solution to the DL problem. It happens with probability for at least $1/Q_{\mathsf{KeyGen}}$.

Lemma 2 (Pre-signature anonymity). *If DDH assumption is hard, then the linkable ring adaptor signature scheme $\Xi_{R,\Sigma}$ is anonymous in the random oracle model.*

Proof. Suppose the simulator is given the DDH problem instance (g, g^a, g^b, c) and wants to decide if $c = g^{ab}$. The simulator computes $u = g^b$ as part of the system parameters.
Oracle simulation. For the KeyGen query, $\mathcal{A}$ picks some random sk_i and returns $\mathsf{pk}_i = g^{\mathsf{sk}_i}$. Except for the i^*th query, the $\mathcal{A}$ picks some random sk_{i^*} and returns $\mathsf{pk}_{i^*} = (g^a)^{\mathsf{sk}_{i^*}}$. Here the secret key is $a \cdot \mathsf{sk}_{i^*}$. For the Corrupt oracle, the $\mathcal{A}$ declares failure and exits if secret key of pk_{i^*} is requested. For the PreSign oracle, everything is the same as that in the algorithm except the following: to compute S_1, S_3, $\mathcal{A}$ picks some random $z_\alpha, z'_{\mathsf{sk}}, x, B, U$ and let $S_1 = h^{z_\alpha} g^{z'_{\mathsf{sk}}} T / B^x, S_3 = u^{z'_{\mathsf{sk}}} / U^x$. Then $\mathcal{A}$ computes y,z,w as that in the PreSign algorithm and finally programs the random oracle such that $x = \hat{H}(4||y||z||w||T_1||T_2||m)$. For the Sign oracle, $\mathcal{A}$ simulates in the same way as the PreSign oracle except that T is dropped.
Forgery. In the challenge phase, the adversary gives $(m^*, i_0, i_1, \tilde{\mathbf{vk}}, T)$ to the simulator. If $i^* \notin \{i_0, i_1\}$, the simulator declares failure and exits. Without loss of generality, assume $\mathsf{pk}_{i_0} = \mathsf{pk}_{i^*}$. The simulator sets $U = c^{\mathsf{sk}_{i^*}}$. The rest of $\hat{\sigma}$ is simulated as follows: everything is the same as the PreSign algorithm, except the following: the $\mathcal{A}$ chooses z_α, z'_{sk} and x at random and let $S_1 = h^{z_\alpha - \alpha x}(g^{z'_{\mathsf{sk}}} / \mathsf{pk}^x_{i^*})T$. Then $\mathcal{A}$ computes y,z,w as that in the PreSign algorithm and finally programs the random oracle such that $x = \hat{H}(4||y||z||w||T_1||T_2||m)$. If this input of $\hat{H}$ is queried before, $\mathcal{A}$ aborts. Otherwise, the $\mathcal{A}$ returns $\hat{\sigma}$. Then the $(\hat{\sigma}, T, U)$ is sent to the adversary.
Finally the adversary outputs b'. If the adversary sucessfully guesses $b' = 0$, then the simulator outputs $c = g^{ab}$ as the solution to the DDH problem.

Lemma 3 (Linkability). *Assuming that DL assumption is hard, then the linkable ring adaptor signature $\Xi_{R,\Sigma}$ is linkable w.r.t. insider corruption in the random oracle model.*

Proof. Let $\mathcal{B}$ be a PPT adversary who wins the aLink security game with non-negligible probability. We will build an adversary $\mathcal{A}$ that breaks the DL assumption. Assume $\mathcal{A}$ wants to solve DL w.r.t. (g, g^a).

Oracle simulation. For the KeyGen query, $\mathcal{A}$ picks some random sk_i and returns $\mathsf{pk}_i = g^{\mathsf{sk}_i}$. Except for the i^*th query, the $\mathcal{A}$ returns g^a. For the $\mathsf{Corrupt}$ oracle, the $\mathcal{A}$ declares failure and exits if secret key of pk_{i^*} is requested. For the $\mathsf{PreSign}$ oracle, if the queried key is not the i^*th one, $\mathcal{A}$ runs honestly as the $\mathsf{PreSign}$ algorithm. Otherwise, $\mathcal{A}$ generates the pre-signature for pk_{i^*} as follows: everything is the same as the $\mathsf{PreSign}$ algorithm, except the following: the $\mathcal{A}$ chooses z_α, z'_{sk} and x at random and let $S_1 = h^{z_\alpha - \alpha x}(g^{z'_{\mathsf{sk}}}/\mathsf{pk}_{i^*}^x)T$. Then $\mathcal{A}$ computes y,z,w as that in the $\mathsf{PreSign}$ algorithm and finally programs the random oracle such that $x = \hat{H}(4||y||z||w||T_1||T_2||m)$. If this input of $\hat{H}$ is queried before, $\mathcal{A}$ aborts. Otherwise, the $\mathcal{A}$ returns $\hat{\sigma}$. For the Sign oracle, if the w in the input is NULL, the $\mathcal{A}$ chooses z_α, z'_{sk} and x at random and let $S_1 = h^{z_\alpha - \alpha x}(g^{z'_{\mathsf{sk}}}/\mathsf{pk}_{i^*}^x)$. Then $\mathcal{A}$ computes y, z, w as the $\mathsf{PreSign}$ algorithm and finally programs the random oracle such that $x = \hat{H}(4||y||z||w||T_1||T_2||m)$. If this input of $\hat{H}$ is queried before, $\mathcal{A}$ aborts. Otherwise, the $\mathcal{A}$ returns σ. If the w in the input is (T,t), the $\mathcal{A}$ firstly runs the $\mathsf{PreSign}$ simulation procedure as above and get the $\hat{\sigma}$. Then it runs the Adapt algorithm and gets the σ. If the same input of $\hat{H}$ is queried before, $\mathcal{A}$ aborts. Otherwise, the $\mathcal{A}$ returns σ.

Forgery. Finally, the adversary $\mathcal{A}$ output $(m_i^*, \mathbf{vk}_i^*, \sigma_i^*)$ for i =1,2, such that all U in $\mathbf{vk}_1^*$ and $\mathbf{vk}_2^*$ are distinct. From the proof of unforgeability, the $\mathcal{A}$ can rewinds x and from equation B:

$$B = h^{\frac{z_{\alpha,1} - z_{\alpha,2}}{x_1 - x_2}} g^{\frac{z_{\mathsf{sk},1} - z_{\mathsf{sk},2}}{x_1 - x_2}} := h^{\alpha''} g^{\mathsf{sk}'}$$

which implies $\mathsf{sk}' = \frac{z_{\mathsf{sk},1} - z_{\mathsf{sk},2}}{x_1 - x_2}$. Combined with $U^{z_{\mathsf{sk}}} = S_3 u^x$, we have $U = u^{\frac{1}{\mathsf{sk}'}}$ and $g^{\mathsf{sk}'} \in \mathbf{vk}_1^* \cup \mathbf{vk}_2^*$. There are two different values of U.

If $\mathcal{A}$ wins, then $\left|(\mathbf{vk}_1^* \cup \mathbf{vk}_2^*) \cap \mathcal{F}) \cup ((\mathbf{vk}_1^* \cup \mathbf{vk}_2^*) \backslash \mathbf{vk})\right| \leq 1$. It means that there exists at least one U corresponding to one public key $g^{\mathsf{sk}'}$. With probability at least $\frac{1}{|\mathbf{vk}| - q_{\mathsf{c}}}$, $g^{\mathsf{sk}'} = g^a$, where q_{c} is the number of oracle queries to the $\mathsf{Corrupt}$ oracles. Then the $\mathcal{A}$ returns sk' as the solution to the DL problem.

Lemma 4 (Non-slanderability). *Assuming that DL assumption is hard, then the linkable ring adaptor signature $\Xi_{R,\Sigma}$ is non-slanderable w.r.t. insider corruption in the random oracle model.*

Proof. Let $\mathcal{B}$ be a PPT adversary who wins the aNSlan security game with non-negligible probability. We will build an adversary $\mathcal{A}$ that breaks the DL assumption. Assume $\mathcal{A}$ wants to solve DL w.r.t. (g, g^a).

Oracle simulation. For the KeyGen query, $\mathcal{A}$ picks some random sk_i and returns $\mathsf{pk}_i = g^{\mathsf{sk}_i}$. Except for the i^*th query, the $\mathcal{A}$ returns g^a. For the $\mathsf{Corrupt}$

oracle, the $\mathcal{A}$ declares failure and exits if secret key of pk_{i^*} is requested. For the $\mathsf{PreSign}$ oracle, if the queried key is not the i^*th one, $\mathcal{A}$ runs honestly as the $\mathsf{PreSign}$ algorithm. Otherwise, $\mathcal{A}$ generates the pre-signature for pk_{i^*} as follows: everything is the same as the $\mathsf{PreSign}$ algorithm, except the following: the $\mathcal{A}$ chooses z_α, z'_{sk} and x at random and let $S_1 = h^{z_\alpha - \alpha x}(g^{z'_{\mathsf{sk}}}/\mathsf{pk}_{i^*}^x)T$. Then $\mathcal{A}$ computes y,z,w as that in the $\mathsf{PreSign}$ algorithm and finally programs the random oracle such that $x = \hat{H}(4||y||z||w||T_1||T_2||m)$. If this input of $\hat{H}$ is queried before, $\mathcal{A}$ aborts. Otherwise, the $\mathcal{A}$ returns $\hat{\sigma}$. For the Sign oracle, if the w in the input is NULL, the $\mathcal{A}$ chooses z_α, z'_{sk} and x at random and let $S_1 = h^{z_\alpha - \alpha x}(g^{z'_{\mathsf{sk}}}/\mathsf{pk}_{i^*}^x)$. Then $\mathcal{A}$ computes y, z, w as the $\mathsf{PreSign}$ algorithm and finally programs the random oracle such that $x = \hat{H}(4||y||z||w||T_1||T_2||m)$. If this input of $\hat{H}$ is queried before, $\mathcal{A}$ aborts. Otherwise, the $\mathcal{A}$ returns σ. If the w in the input is (T,t), the $\mathcal{A}$ firstly runs the $\mathsf{PreSign}$ simulation procedure as above and get the $\hat{\sigma}$. Then it runs the Adapt algorithm and gets the σ. If the same input of $\hat{H}$ is queried before, $\mathcal{A}$ aborts. Otherwise, the $\mathcal{A}$ returns σ.

Forgery. Finally, the adversary $\mathcal{A}$ outputs $(m^*, \tilde{\mathbf{vk}}^*, \sigma^*)$. If $\mathsf{pk}^* \notin \tilde{\mathbf{vk}}^*$, the $\mathcal{A}$ declares failure and exits. By the winning condition, there exists some U corresponding to one public key $g^{\mathsf{sk}'} \in (\mathbf{vk} \cap \tilde{\mathbf{vk}}^* \cap (\hat{\mathbf{vk}} \backslash \mathcal{F}))$. Following the proof of linkability, the $\mathcal{A}$ can extract sk' such that $U = u^{\frac{1}{\mathsf{sk}'}}$ and $g^{\mathsf{sk}'} \in \mathbf{vk}$. With probability $\frac{1}{|\mathbf{vk}| - Q_{\mathsf{Corrupt}}}$, $g^{\mathsf{sk}'} = \mathsf{pk}^*$, where Q_{Corrupt} is the number of the query to $\mathsf{Corrupt}$ oracle. Then the $\mathcal{A}$ return sk' as the solution to the DL problem.

Lemma 5 (Pre-signature adaptability). *The linkable ring adaptor signature $\Xi_{R,\Sigma}$ satisfies pre-signature adaptability w.r.t. the relation L_R.*

Proof. Let $\hat{\sigma}$ be a valid pre-signature with $\mathsf{pVerify}(m, \mathbf{vk}, T, \hat{\sigma}) = 1$, $t \in \mathbb{Z}_p$ be a witness corresponding to T and $z_{\mathsf{sk}} = z'_{\mathsf{sk}} + t \mod p$. We have $h^{z_\alpha} g^{z_{\mathsf{sk}}} = S_1 B^x$. which implies $\mathsf{Verify}(m, \mathbf{vk}, \mathsf{Adapt}(t, \hat{\sigma})) = 1$.

Lemma 6 (Witness extractability). *Assuming that DL assumption is hard, then the linkable ring adaptor signature $\Xi_{R,\Sigma}$ is witness extractable in the random oracle model.*

Proof. We only investigate the case that the signature output by the adversary shares the same challenge with the pre-signature. The other case that two challenges are distinct can be proven exactly as in $\mathsf{Case\ 2}$ of the proof of Lemma 1. Let $\hat{\sigma} = (B, A, S_1, S_2, S_3, T_1, T_2, \tau, \mu, z_\alpha, z'_{\mathsf{sk}}, \zeta, \pi)$ and $\sigma = (B, A, S_1, S_2, S_3, T_1, T_2, \tau, \mu, z_\alpha, z_{\mathsf{sk}}, \zeta, \pi)$ be a valid pre-signature and a valid signature respectively. Then, from the corresponding verification algorithms, we have $h^{z_\alpha} g^{z'_{\mathsf{sk}}} T = S_1 B^x = h^{z_\alpha} g^{z_{\mathsf{sk}}}$. Since DL assumption is hard, we have that $g^{z_{\mathsf{sk}} - z'_{\mathsf{sk}}} = T$. Therefore, $(T, z_{\mathsf{sk}} - z'_{\mathsf{sk}}) = (T, t) \in L_R$.

Let $\mathcal{B}$ be a PPT adversary who wins the $\mathsf{aEUF\text{-}CMA}$ security game with non-negligible probability. We will build an adversary $\mathcal{A}$ that breaks the DL assumption or the hardness of L_R. Assume $\mathcal{A}$ wants to solve DL w.r.t. (g, g^a).

Oracle simulation. For the KeyGen query, $\mathcal{A}$ picks some random sk_i and returns $\mathsf{pk}_i = g^{\mathsf{sk}_i}$. Except for the i^*th query, the $\mathcal{A}$ returns g^a. For the $\mathsf{Corrupt}$

oracle, the $\mathcal{A}$ declares failure and exits if secret key of pk_{i^*} is requested. For the $\mathsf{PreSign}$ oracle, if the queried key is not the i^*th one, $\mathcal{A}$ runs honestly as the $\mathsf{PreSign}$ algorithm. Otherwise, $\mathcal{A}$ generates the pre-signature for pk_{i^*} as follows: everything is the same as the $\mathsf{PreSign}$ algorithm, except the following: the $\mathcal{A}$ chooses z_α, z'_{sk} and x at random and let $S_1 = h^{z_\alpha - \alpha x}(g^{z'_{\mathsf{sk}}}/\mathsf{pk}_{i^*}^x)T$. Then $\mathcal{A}$ computes y,z,w as that in the $\mathsf{PreSign}$ algorithm and finally programs the random oracle such that $x = \hat{H}(4||y||z||w||T_1||T_2||m)$. If this input of $\hat{H}$ is queried before, $\mathcal{A}$ aborts. Otherwise, the $\mathcal{A}$ returns $\hat{\sigma}$. For the Sign oracle, if the w in the input is NULL, the $\mathcal{A}$ chooses z_α, z'_{sk} and x at random and let $S_1 = h^{z_\alpha - \alpha x}(g^{z'_{\mathsf{sk}}}/\mathsf{pk}_{i^*}^x)$. Then $\mathcal{A}$ computes y, z, w as the $\mathsf{PreSign}$ algorithm and finally programs the random oracle such that $x = \hat{H}(4||y||z||w||T_1||T_2||m)$. If this input of $\hat{H}$ is queried before, $\mathcal{A}$ aborts. Otherwise, the $\mathcal{A}$ returns σ. If the w in the input is (T,t), the $\mathcal{A}$ firstly runs the $\mathsf{PreSign}$ simulation procedure as above and get the $\hat{\sigma}$. Then it runs the Adapt algorithm and gets the σ. If the same input of $\hat{H}$ is queried before, $\mathcal{A}$ aborts. Otherwise, the $\mathcal{A}$ returns σ.
Consider that $\mathcal{B}$ makes at most $Q_{\mathsf{KeyGen}}, Q_{\mathsf{PreSign}}, Q_{\mathsf{Sign}}$ and Q_H queries to KeyGen, $\mathsf{PreSign}, \mathsf{Sign}$ and random oracle respectively.
Forgery. $\mathcal{B}$ returns the target message $(m^*, i^*, T^*, \tilde{\mathbf{vk}}^*)$ to $\mathcal{A}$. $\mathcal{A}$ computes a pre-signature $\hat{\sigma}^*$ using the simulation method above. Then $\mathcal{A}$ sends $\hat{\sigma}^*$ to $\mathcal{B}$. Finally, $\mathcal{B}$ returns a forged linkable ring adaptor signature $\tilde{\sigma}$ on m^* for $(m^* \notin \mathcal{Q} \wedge (T^*, t') \notin R \wedge \mathsf{Verify}(m^*, \tilde{\mathbf{vk}}^*, \sigma)) = 1$, where the $\mathcal{Q}$ are the same as that in the $\mathsf{aWitExt}$ experiment. $\tilde{\sigma}$ is denoted as $(\tilde{B}, \tilde{A}, \tilde{S_1}, \tilde{S_2}, \tilde{T_1}, \tilde{T_2}, \tilde{\tau}, \tilde{\mu}, \tilde{z_\alpha}, \tilde{z_{\mathsf{sk}}}, \tilde{\zeta}, \tilde{\pi})$. Since $(T^*, t') \notin R$, $\sigma^* = \mathsf{Adapt}(t^*, \hat{\sigma}^*)$ are different. This means the challenge $x = \hat{H}(||4y||z||w||T_1||T_2\|m)$ of them are different. The $\mathcal{A}$ computes the corresponding y,z,w as in the $\mathsf{PreSign}$ algorithm first and rewinds $\hat{H}$ on input $(4||y||z||w||\tilde{T_1}||\tilde{T_2})$ for three times. For each transcript, denote the challenge as x_i and the responses as $(\tau_{x,i}, \mu_i, z_{\alpha,i}, z_{\mathsf{sk},i}, z_{\delta,i}, \mathbf{l}_i, \mathbf{r}_i, t_i)$ for $i \in [1,3]$. Denote $\mathbf{l}_i = (l_{i,1}, \ldots, l_{i,n})$ and $\mathsf{r}_i = (r_{i,1}, \ldots, r_{i,n})$.

- To extract BA^w, it picks some $\eta_i \in \mathbb{Z}_p$ such that $\sum_{i=1}^2 \eta_i = 1, \sum_{i=1}^2 \eta_i x_i = 0$. Then we have:
$$\begin{aligned} BA^w &= h^{\sum_{i=1}^2 \eta_i \mu_i} \mathbf{vk}^{\sum_{i=1}^2 \eta_i \cdot l_i + z \cdot 1^n} \vec{H'}^{\sum_{i=1}^2 \eta_i r_i - z^2 \cdot 1^n} \vec{H}^{-wz} \\ &:= h^{\gamma'} \mathbf{vk}^{\vec{b_L}'} \vec{H}^{w\vec{b_R}'} \end{aligned} \tag{4}$$
for some $\gamma', \vec{b_L}', \vec{b_R}'$.
- To extract S_2, it picks some $\eta'_i \in \mathbb{Z}_p$ such that $\sum_{i=1}^2 \eta'_i = 0, \sum_{i=1}^2 \eta'_i x_i = 1$. Then we have:
$$S_2 = h^{\sum_{i=1}^2 \eta'_i \mu_i} \mathbf{vk}^{\sum_{i=1}^2 \eta'_i l_i} \vec{H'}^{\sum_{i=1}^2 \eta'_i r_i} := h^{\rho'} Y^{\vec{s_L}'} \vec{H}^{\cdot \vec{s_R}'} \tag{5}$$
for some $\gamma', \vec{s_L}', \vec{s_R}'$.

Putting back the extracted values BA$_w$ and S_2 into $P = BA^w S_2^x \mathbf{vk}^{-z \cdot \vec{1}^n} \vec{H}^{wz} \vec{H'}^{z^2 \cdot \vec{1}^n} h^{-\mu}$, we have: $\mathbf{vk}^{\vec{l}} \vec{H'}^{\vec{r}} = (h^{\gamma'} \mathbf{vk}^{\vec{b_L}'} \vec{H}^{w\vec{b_R}'}) \cdot (h^{\rho'} Y^{\vec{s_L}'} \vec{H}^{\cdot \vec{s_R}'})^x \cdot \mathbf{vk}^{-z \cdot 1^n} \cdot \vec{H}^{wz} \vec{H'}^{z^2 \cdot \vec{1}^n} h^{-\mu}$.

The mutual discrete logarithm between **vk** is not known if the discrete logarithm assumption holds by lemma 1. Since $\vec{H}$ and the elements in h are randomly chosen from the group, the mutual discrete logarithm between h, the elements in $\vec{H}$ and **vk** is not known.
Then we have: $\vec{l} = \vec{b_L}' - z \cdot 1^n + \vec{s_L}' \cdot x, \vec{r} = y^n \circ (w \cdot \vec{b_R}' + wz \cdot 1^n + \vec{s_R}' \cdot x) + z^2 \cdot 1^n$.

By the same set of 3 rewinding transcripts, we can also extract the commitments T_1,T_2 as follows.

- To extract T_1, it picks some $\delta_i \in \mathbb{Z}_p$ such that $\sum_{i=1}^{3} \delta_i = 0, \sum_{i=1}^{3} \delta_i x_i = 1, \sum_{i=1}^{3} \delta_i x_i^2 = 0$. Then we have:
$$T_1 = g^{\sum_{i=1}^{3} \delta_i t_i} h^{\sum_{i=1}^{3} \delta_i \tau_{x,i}} := g^{t_1'} h^{r_1'}$$
for some t_1', r_1'.
- To extract T_2, it picks some $\delta_i' \in \mathbb{Z}_p$ such that $\sum_{i=1}^{3} \delta_i' = 0, \sum_{i=1}^{3} \delta_i' x_i = 1, \sum_{i=1}^{3} \delta_i' x_i^2 = 0$. Then we have:
$$T_2 = g^{\sum_{i=1}^{3} \delta_i' t_i} h^{\sum_{i=1}^{3} \delta_i' \tau_{x,i}} := g^{t_2'} h^{r_2'}$$
for some t_2', r_2'.

Putting back the extracted values T_1 and T_2 into equation $g^{\zeta} h^{\tau} = g^{z^2 + wz(1-z)\sum_{i=1}^{n} y^{i-1} - nz^3} T_1^x T_2^{x^2}$, we have:
$$g^{\zeta} h^{\tau} = g^{z^2 + w(z - z^2)\langle 1^n, y^n \rangle - z^3 \langle 1^n, 1^n \rangle} \cdot (g^{t_1'} h^{r_1'})^x \cdot (g^{t_2'} h^{r_2'})^{x^2}$$
Since h is a random group element by the simulation of $\hat{H}_G$, we have:
$$\zeta = z^2 + w(z - z^2)\langle 1^n, y^n \rangle - z^3 \langle 1^n, 1^n \rangle + t_1' x + t_2' x^2$$
Denote $t_0' = z^2 + w(z - z^2)\langle 1^n, y^n \rangle - z^3 \langle 1^n, 1^n \rangle$.
Observe that we already extracted $\vec{l}, \vec{r}$ as:
$$\begin{aligned} \langle \vec{l}, \vec{r} \rangle &= (\vec{b_L}' - z \cdot 1^n + \vec{s_L}' \cdot x) \cdot (y^n \circ (w \cdot \vec{b_R}' + wz \cdot 1^n + \vec{s_R}' \cdot x) + z^2 \cdot 1^n) \\ &= w \langle \vec{b_L}', \vec{b_R}' \circ y^n \rangle + wz \langle \vec{b_L}' - \vec{b_R}', y^n \rangle + z^2 \langle \vec{b_L}', 1^n \rangle - wz^2 \langle 1^n, y^n \rangle - z^3 \langle 1^n, 1^n \rangle + t_1'' x + t_2'' x^2 \end{aligned}$$
for some $t_1'', t_2'' \in \mathbb{Z}_p$. Since the above holds for all w, x, y, z, we have:
$$\vec{b_L}' \circ \vec{b_R}' = 0^n, \quad \vec{b_L}' - \vec{b_R}' = 1^n, \quad \langle \vec{b_L}', 1^n \rangle = 1$$
Therefore, it implies that $\vec{b_L}'$ is a binary vector with one bit equal to 1. Putting back $\vec{b_L}'$ in equation 4, we have: $BA^w = h^{\gamma'} \tilde{\mathsf{pk}} \vec{H}^{w\vec{b_R}'}$. Since the above is true for all w, then we have B $= h^{\alpha'} \tilde{\mathsf{pk}}$ for some $\alpha' \in \mathbb{Z}_p$.
By the same set of rewinding transcripts, we can also extract from $h^{z_\alpha} g^{z_{sk}} T = S_1 B^x$: $B = h^{\frac{z_{\alpha,1} - z_{\alpha,2}}{x_1 - x_2}} g^{\frac{z_{sk,1} - z_{sk,2}}{x_1 - x_2}} := h^{\alpha''} g^{\mathsf{sk}'}$. Since the above is true for all h, then we have $\tilde{\mathsf{pk}} = \mathrm{g}^{\mathsf{sk}'}$. Hence if $\tilde{\mathsf{pk}} = \mathsf{pk}_{i^*}$, then the $\mathcal{A}$ returns sk' as the solution to the DL problem. It happens with probability for at least $1/Q_{\mathsf{KeyGen}}$.

Algorithm 12: Type-T Linkable Ring Canonical Identification

```
Procedure Setup($\lambda$):
    return $\mathsf{param}$;
Procedure KeyGen():
    return $(\mathsf{pk}, \mathsf{sk})$;
Procedure Proof1($\mathsf{sk}, \{\mathsf{pk}_1, \dots, \mathsf{pk}_n\}$):
    $r \leftarrow_s \Delta_r$;
    $R = A(r, \{\mathsf{pk}_1, \dots, \mathsf{pk}_n\})$;
    return $(R, r)$;
Procedure Ch($R$):
    return $c$;
Procedure
  Proof2($\mathsf{sk}, \{\mathsf{pk}_1, \dots, \mathsf{pk}_n\}, r, c$):
    return $z = Z(\mathsf{sk}, \{\mathsf{pk}_1, \dots, \mathsf{pk}_n\}, r, c)$;
Procedure Verify($\{\mathsf{pk}_1, \dots, \mathsf{pk}_n\}, z, c$):
    $R' = V(\{\mathsf{pk}_1, \dots, \mathsf{pk}_n\}, c, z)$;
    if $c \neq$ Ch($R'$) then
        return 0;
    auxiliary checking with
      $R', \{\mathsf{pk}_1, \dots, \mathsf{pk}_n\}, c, z$;
    return 1;
Procedure
  Link($\{\mathsf{pk}_1, \dots, \mathsf{pk}_n\}, (R_1, c_1, z_1), (R_2, c_2, z_2)$):
    return 1 for linked, 0 for unlinked, or
      $\perp$ for any invalid transcript;
```

Algorithm 13: Linkable Ring Adaptor Signature

```
Procedure Setup($\lambda$):
    pick random generators $g, u, G, H_1, \dots, H_n, G' \in \mathbb{G}$ of prime order $p$,
      denote $\vec{H} = [H_1, \dots, H_n]$;
    $\hat{H}$ is a hash function $\{0,1\}^* \to \mathbb{Z}_p$, $\hat{H}_G$ is a hash function
      $\{0,1\}^* \to \mathbb{G}$;
    return $\mathsf{param} = (\mathbb{G}, p, g, G, \vec{H}, \hat{H}, \hat{H}_G)$;
Procedure KGen(param):
    pick $\mathsf{sk} \in \mathbb{Z}_p$;
    $\mathsf{vk} = g^{\mathsf{sk}}$;
    return $(\mathsf{vk}, \mathsf{sk})$;
Procedure PreSign($\mathsf{sk}_j, m, \mathbf{vk} = \{\mathsf{vk}_1, \dots, \mathsf{vk}_n\}, T$):
    set $\vec{b}_L = [0, 0, \dots, 1, \dots, 0]$ /* "1" is at the $j$-th pos */
    set $\vec{b}_R = [-1, -1, \dots, 0, \dots, -1]$ /* "0" is at the $j$-th pos */
    $U = u^{\mathsf{sk}}$, $h = \hat{H}_G(\mathbf{vk})$;
    pick a random $\alpha, \beta, \rho, r_\alpha, r_{\mathsf{sk}}, \in \mathbb{Z}_p$ and random vectors $\vec{s}_L, \vec{s}_R \in \mathbb{Z}_p^n$;
    $B = h^\alpha \mathbf{vk}^{\vec{b}_L}$, $A = h^\beta \vec{H}^{\vec{b}_R}$, $S_1 = h^{r_\alpha} g^{r_{\mathsf{sk}}} T$, $S_2 = h^\rho \mathbf{vk}^{\vec{s}_L} \vec{H}^{\vec{s}_R}$,
      $S_3 = u^{r_{\mathsf{sk}}}$;
    $\mathsf{str} = h||B||A||S_1||S_2||S_3$, $y = \hat{H}(1||\mathsf{str})$, $z = \hat{H}(2||\mathsf{str})$, $w = \hat{H}(3||\mathsf{str})$;
    $\vec{c}_L = \vec{b}_L - z \cdot \vec{1}^n$, $\vec{c}_R = \vec{y}^n \circ (w \cdot \vec{b}_R + wz \cdot \vec{1}^n) + z^2 \cdot \vec{1}^n$;
    $\vec{s}'_R = \vec{s}_R \circ \vec{y}^n$, $t_1 = \langle \vec{s}_L, \vec{c}_R \rangle + \langle \vec{c}_L, \vec{s}'_R \rangle \mod p$, $t_2 = \langle \vec{s}_L, \vec{s}'_R \rangle \mod p$;
    pick a random $\tau_1, \tau_2 \in \mathbb{Z}_p$;
    $T_1 = g^{t_1} h^{\tau_1}$, $T_2 = g^{t_2} h^{\tau_2}$;
    $x = \hat{H}(4||y||z||w||T_1||T_2||m)$;
    $\tau = \tau_1 x + \tau_2 x^2 \mod p$, $\mu = \alpha + \beta w + \rho x \mod p$;
    $z_\alpha = r_\alpha + \alpha x \mod p$, $z'_{\mathsf{sk}} = r_{\mathsf{sk}} + \mathsf{sk}_j x \mod p$;
    $\vec{l} = \vec{c}'_L + \vec{s}_L \cdot x$, $\vec{r} = \vec{c}'_R + \vec{s}'_R \cdot x$, $\zeta = \langle \vec{l}, \vec{r} \rangle$;
    Set $\vec{H}' = [H_1, H_2^{y^{-1}}, \dots, H_n^{y^{-n+1}}]$;
    $P = \mathbf{vk}^{\vec{l}} \vec{H}'^{\vec{r}}$;
    $\pi \leftarrow$NIPA.Proof($\mathbf{vk}, \vec{H}', P, \zeta; \vec{l}, \vec{r}$);
    return $\hat{\sigma} = (B, A, S_1, S_2, S_3, T_1, T_2, \tau, \mu, z_\alpha, z'_{\mathsf{sk}}, \zeta, \pi, U)$;
```

Algorithm 14: Linkable Ring Adaptor Signature (cont.)

```
1  Procedure PreVerify($m, \mathbf{vk} = \{\mathsf{vk}_1, \ldots, \mathsf{vk}_n\}, T, \hat{\sigma}$):
2      parse $\hat{\sigma} = (B, A, S_1, S_2, S_3, T_1, T_2, \tau, \mu, z_\alpha, z'_{\mathsf{sk}}, \zeta, \pi, U)$;
3      $h = \hat{H}_G(\mathbf{vk})$;
4      $\mathsf{str} = h||B||A||S_1||S_2||S_3,\ y = \hat{H}(1||\mathsf{str}),\ z = \hat{H}(2||\mathsf{str}),\ w = \hat{H}(3||\mathsf{str})$;
5      $x = \hat{H}(4||y||z||w||T_1||T_2||m)$;
6      $\vec{H}' = [H_1, H_2^{y^{-1}}, \ldots, H_n^{y^{-n+1}}]$;
7      $P = BA^w S_2^x \mathbf{vk}^{-z\cdot\vec{1}^n} \vec{H}^{wz} \vec{H}'^{z^2\cdot\vec{1}^n} h^{-\mu}$;
8      if NIPA.Verify$(\mathbf{vk}, \vec{H}', P, \zeta, \pi) = 1$ and $u^{z'_{\mathsf{sk}}} = S_3 U^x$ and
       $g^\zeta h^\tau = g^{z^2 + wz(1-z)\sum_{i=1}^n y^{i-1} - nz^3} T_1^x T_2^{x^2}$ and $h^{z_\alpha} g^{z'_{\mathsf{sk}}} T = S_1 B^x$ then
9          return 1;
10     return 0;
11 Procedure Verify($m, \mathbf{vk} = \{\mathsf{vk}_1, \ldots, \mathsf{vk}_n\}, \sigma$):
12     parse $\sigma = (B, A, S_1, S_2, S_3, T_1, T_2, \tau, \mu, z_\alpha, z_{\mathsf{sk}}, \zeta, \pi, U)$;
13     Compute $h, y, z, w, c, \vec{H}', P$ as PreVerify;
14     if NIPA.Verify$(\mathbf{vk}, \vec{H}', P, \zeta, \pi) = 1$ and $u^{z_{\mathsf{sk}}} = S_3 U^x$ and
       $g^\zeta h^\tau = g^{z^2 + wz(1-z)\sum_{i=1}^n y^{i-1} - nz^3} T_1^x T_2^{x^2}$ and $h^{z_\alpha} g^{z_{\mathsf{sk}}} = S_1 B^x$ then
15         return 1;
16     return 0;
17 Procedure Link($(m_1, \tilde{\boldsymbol{vk}}_1, \sigma_1), (m_2, \tilde{\boldsymbol{vk}}_2, \sigma_2)$):
18     parse $\sigma_1 = (\ldots, U_1)$ and $\sigma_2 = (\ldots, U_2)$;
19     return 1 if $U_1 = U_2$ or 0 otherwise;
20 Procedure Adapt($(T, t), \mathbf{vk}, \hat{\sigma}, m$):
21     parse $\hat{\sigma} = (B, A, S_1, S_2, S_3, T_1, T_2, \tau, \mu, z_\alpha, z'_{\mathsf{sk}}, \zeta, \pi, U)$;
22     $z_{\mathsf{sk}} = z'_{\mathsf{sk}} + t \mod p$;
23     return $\sigma = (B, A, S_1, S_2, S_3, T_1, T_2, \tau, \mu, z_\alpha, z_{\mathsf{sk}}, \zeta, \pi, U)$;
24 Procedure Ext($T, \sigma, \hat{\sigma}$):
25     retrieve $z_{\mathsf{sk}}$ from $\sigma$ and $z'_{\mathsf{sk}}$ from $\hat{\sigma}$;
26     $t = z_{\mathsf{sk}} - z'_{\mathsf{sk}} \mod p$;
27     if $T = g^t$ then
28         return $t$;
29     return $\perp$;
```

Algorithm 15: Generic linkable ring adaptor signature from a Type-TA linkable ring identification scheme LRID for the language $L := \{Y : \exists y \in \Delta_r : Y = A(y)\}$.

```
1  Procedure Setup($\lambda$):
2      define hash $H : \{0,1\}^* \to \Delta_c$;
3      $\mathsf{param}_I \leftarrow$ LRID.SETUP($\lambda$);
4      return $(\mathsf{param}_I, H)$;
5  Procedure KeyGen():
6      return LRID.KEYGEN();
7  Procedure PreSign($\mathsf{sk}, m, \vec{\mathsf{vk}}, Y$):
8      $r \leftarrow \Delta_r$;
9      $R = A(r, \vec{\mathsf{vk}}) \oplus_R Y$;
10     $c = H(m, R, \vec{\mathsf{vk}})$;
11     $\hat{z} = Z(\mathsf{sk}, \vec{\mathsf{vk}}, r, c)$;
12     return $\hat{\sigma} = (\hat{z}, c)$;
13 Procedure PreVerify($m, \vec{\mathsf{vk}}, Y, \hat{\sigma}$):
14     parse $\hat{\sigma} = (\hat{z}, c)$;
15     $R' = V(\vec{\mathsf{vk}}, c, \hat{z}) \oplus_R Y$;
16     if $c \neq H(M, R', \vec{\mathsf{vk}})$ then
17         return 0;
18     return 1;
19 Procedure Adapt($(Y, y), \vec{\mathsf{vk}}, \hat{\sigma}, m$)
     and Ext($Y, \hat{\sigma}, \sigma$):
20     Same as Algorithm 5
```

C Blind Adaptor Signature

Algorithm 16: Experiment $\mathsf{omaSignForge}_{\mathcal{A},\prod_{R,\Sigma}}$

```
Procedure $\mathit{omaSignForge}_{\mathcal{A},\Xi_{R,\Sigma}}(\lambda)$:
    $\mathcal{S}_s := \emptyset$, $\mathcal{S}_p := \emptyset$, $k_{s1} := 0$, $k_{p1} := 0$, $k_2 := 0$;
    $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$;
    $(M_1^*, \ldots, M_n^*) \leftarrow \mathcal{A}^{O_\mathsf{S}, O_\mathsf{pS}}(\mathsf{pk})$ ;
    $(Y, y) \leftarrow \mathsf{LockGen}(\lambda)$;
    $\hat{\sigma}_i \leftarrow \mathsf{PreSign}((\mathsf{pk}, \mathsf{sk}), Y, M_i^*)$ $\forall i \in [1, n]$ ;
    $(\sigma_1^*, \ldots, \sigma_n^*) \leftarrow \mathcal{A}^{O_\mathsf{S}, O_\mathsf{pS}}(\hat{\sigma}_1, \ldots, \hat{\sigma}_n, Y)$;
    return
      $(k_2 < n \wedge (M_i^*, \sigma_i^*) \neq (M_j^*, \sigma_j^*)$
      $\forall i \neq j \in [1, n]$
      $\wedge \mathsf{Verify}(\mathsf{pk}, M_i^*, \sigma_i^*) = 1 \ \forall i \in [1, n]$
      );
Procedure $O_\mathsf{S}(M, i, j)$:
    if $i = 1$ then
        $k_{s1} = k_{s1} + 1$;
        $(M', st_{k_{s1},1}) \leftarrow \mathsf{Sign}_1(\mathsf{sk}, M)$;
        $\mathcal{S}_s = \mathcal{S}_s \cup \{k_{s1}\}$;
        return $(k_{s1}, M')$;
    if $i = N_s$ then
        if $j \notin \mathcal{S}_s$ then
            return $\perp$;
        $(M', b) \leftarrow \mathsf{Sign}_{N_s}(st_{j,N_s}, M)$;
        if $b = 1$ then
            $\mathcal{S}_s = \mathcal{S}_s \setminus \{j\}$;
            $k_2 = k_2 + 1$;
        return $M'$;
    if $i \in [2, N_s - 1]$ then
        if $j \notin \mathcal{S}_s$ then
            return $\perp$;
        $(M', st_{j,i}) \leftarrow \mathsf{Sign}_i(st_{j,i-1}, M)$;
        return $M'$;
    return $\perp$;
Procedure $O_\mathsf{pS}(M, Y, i, j)$:
    if $i = 1$ then
        $k_{p1} = k_{p1} + 1$;
        $(M', st_{k_{p1},1}) \leftarrow \mathsf{PreSign}_1(\mathsf{sk}, Y, M)$;
        $\mathcal{S}_p = \mathcal{S}_p \cup \{k_{p1}\}$;
        return $(k_{p1}, M')$;
    if $i = N_p$ then
        if $j \notin \mathcal{S}_p$ then
            return $\perp$;
        $(M', b) \leftarrow \mathsf{PreSign}_{N_p}(st_{j,N_s}, M)$;
        if $b = 1$ then
            $\mathcal{S}_p = \mathcal{S}_p \setminus \{j\}$;
            $k_2 = k_2 + 1$;
        return $M'$;
    if $i \in [2, N_s - 1]$ then
        if $j \notin \mathcal{S}_p$ then
            return $\perp$;
        $(M', st_{j,i}) \leftarrow \mathsf{PreSign}_i(st_{j,i-1}, M)$;
        return $M'$;
    return $\perp$;
```

Algorithm 17: Experiment $\mathsf{baWitExt}_{\mathcal{A},\prod_{R,\Sigma}}$

```
1 Procedure baWitExt_{A,∏_{R,Σ}}(λ):
2     $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$;
3     $(M^*, Y) \leftarrow \mathcal{A}^{O'_{\mathsf{S}},O'_{\mathsf{pS}}}(\mathsf{pk})$ /* $O'_{\mathsf{S}}, O'_{\mathsf{pS}}$ are the same as $O_{\mathsf{S}}, O_{\mathsf{pS}}$ in
          Algorithm 16 except that $O_{\mathsf{S}}(\cdot, N_s, \cdot), O_{\mathsf{pS}}(\cdot, \cdot, N_p, \cdot)$ are not
          allowed.                                                          */
4     $\hat{\sigma} \leftarrow \mathsf{PreSign}(\mathsf{sk}, Y) \leftrightarrow \mathsf{User}(\mathsf{pk}, M^*, Y)$;
5     $\sigma^* \leftarrow \mathcal{A}^{O_{\mathsf{S}},O_{\mathsf{pS}}}(\hat{\sigma}, Y)$;
6     $y^* \leftarrow \mathsf{Ext}(Y, \sigma^*, \hat{\sigma})$;
7     return $((Y, y^*) \notin R \wedge \mathsf{Verify}(\mathsf{pk}, \sigma^*, M^*) = 1)$;
```