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 platforms run by miners, where each transaction on the blockchain can be seen as an application formed of some script(s). The scripting language of a blockchain defines potential functionalities that can be implemented on blockchain. Bitcoin, for example, consists of very few scripts, which restricts its use mainly in coin transactions. Ethereum, on the other hand, has a Turing-complete scripting language that enables users to run more advanced and complicated applications known as the smart contracts.

A user who wants to deploy and execute a transaction needs to pay a fee to the miners. The fee is determined by the storage and computational costs of running each script of the transaction. Thus, it is beneficial to handle some operations off-chain to reduce the on-chain fee paid to the miners. In this manner, Poelstra introduced the notion of scriptless scripts [16], which is later named as 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 a 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 $\lambda$ as the security parameter, $\text{negl}(\lambda)$ to represent a negligible function with respect to $\lambda$ and $\Delta$ 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 $\Delta_r$). Then, the message and $R$ are inputted to a function $H$ to obtain a challenge $c$ (within the challenge space $\Delta_c$). 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 $R'$.


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

1 Procedure Setup(\(\lambda\)): 9 Procedure KeyGen():
2 \[
\text{return } \text{param;}
\]
10 \[
\text{return } (pk, sk);
\]
3 Procedure Sign(sk, M):
4 \[
\text{r } \leftarrow \Delta_{r};
\]
5 \[
R = A(r);
\]
6 \[
c = H(M, R);
\]
7 \[
z = Z(sk, r, c);
\]
8 \[
\text{return } \sigma = (z, c);
\]

Algorithm 2: Type-T Canonical Identification

1 Procedure Setup(\(\lambda\)): 9 Procedure CH(R):
2 \[
\text{return } \text{param;}
\]
10 \[
\text{return } c;
\]
3 Procedure KeyGen():
4 \[
\text{return } (pk, sk);
\]
5 Procedure Proof1(sk):
6 \[
r \leftarrow s \Delta_{r};
\]
7 \[
R = A(r);
\]
8 \[
\text{return } (R, r);
\]

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

\[
\text{Adv}_{\text{imp}}^A(\lambda) := \Pr[\text{Verify}(pk, z^*, c_i^*) = 1 | \text{param } \leftarrow \text{Setup}(\lambda),
(pk, sk) \leftarrow \text{KeyGen}(), (c_i^*, z^*) \leftarrow \text{CH}(\cdot)(\text{param}, pk)].
\]

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

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 \(\text{LockGen}(\lambda)\) that outputs \((Y, y) \in R\), (ii) for every PPT algorithm \(A\), given \(Y \in L_R\), the probability of \(A\) outputting \(y\) is negligible.

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

Definition 2 (Adaptor Signature Scheme). An adaptor signature AS scheme \(\Pi_{R, \Sigma}\) consists of four algorithms (PreSign, PreVerify, Adapt, Ext) defined below.
Algorithm 3: Experiment aSignForge\textsubscript{A,\Pi,\Sigma}(
\lambda)
\begin{enumerate}
\item Procedure aSignForge\textsubscript{A,\Pi,\Sigma}(
\lambda):
\begin{enumerate}
\item \(Q := \emptyset\);
\item \((pk, sk) \leftarrow \text{KeyGen}(1^\lambda)\);
\item \(M^* \leftarrow \mathcal{A}^{O_S, O_{\Phi}}(pk)\);
\item \((Y, y) \leftarrow \text{LockGen}(\lambda)\);
\item \(\hat{\sigma} \leftarrow \text{PreSign}((pk, sk), Y, M^*)\);
\item \(\sigma^* \leftarrow \mathcal{A}^{O_S, O_{\Phi}}(\hat{\sigma}, Y)\);
\item return 
\((M^* \notin Q) \land \text{Verify}(pk, \sigma^*, M^*) = 1)\);
\end{enumerate}
\item Procedure \(O_5(M)\):
\begin{enumerate}
\item \(\sigma \leftarrow \text{Sign}(sk, M)\);
\item \(Q := Q \cup \{M\}\);
\item return \(\sigma\);
\end{enumerate}
\item Procedure \(O_{\Phi}(M, Y)\):
\begin{enumerate}
\item \(\sigma \leftarrow \text{PreSign}((pk, sk), Y, M)\);
\item \(Q := Q \cup \{M\}\);
\item return \(\sigma\);
\end{enumerate}
\end{enumerate}
\end{enumerate}

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

Definition 3 (Pre-signature Adaptability). An adaptor signature scheme \(\Pi_{R, \Sigma}\) 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:

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

Definition 4 (Unforgeability). An adaptor signature scheme \(\Pi_{R, \Sigma}\) is aEUF–CMA secure if for every PPT adversary \(A\) running the experiment \(a\text{SignForge}_{A,\Pi,\Sigma}(\lambda) = 1\), \(\Pr[\text{aSignForge}_{A,\Pi,\Sigma}(\lambda) = 1] \leq \text{negl}(\lambda)\).

Definition 5 (Witness Extractability). An adaptor signature scheme \(\Pi_{R, \Sigma}\) is witness extractable if for every PPT adversary \(A\) running the experiment \(a\text{WitExt}_{A,\Pi,\Sigma}(\lambda) = 1\), \(\Pr[\text{aWitExt}_{A,\Pi,\Sigma}(\lambda) = 1] \leq \text{negl}(\lambda)\).

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 $\text{aWitExt}_{\mathcal{A},\Pi_r,x}$

1. Procedure $\text{aWitExt}_{\mathcal{A},\Pi_r,x}(\lambda)$:
   2. \( Q := \emptyset; \)
   3. \((pk,sk) \leftarrow \text{KeyGen}(1^\lambda);\)
   4. \((M^*,Y) \leftarrow \mathcal{A}_{\text{OS},\text{OpS}}(pk) /* \mathcal{A}_{\text{OS},\text{OpS}} are the same as Algorithm 3 */ \)
   5. \( \hat{\sigma} \leftarrow \text{PreSign}((pk,sk),Y); \)
   6. \( \sigma^* \leftarrow \mathcal{A}_{\text{OS},\text{OpS}}(\hat{\sigma}); \)
   7. \( y^* \leftarrow \text{Ext}(Y,\sigma^*,\hat{\sigma}); \)
   8. return \((M^* \notin Q) \land (Y,y^*) \notin R \land \text{Verify}(pk,\sigma^*,M^*) = 1); \)

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 \( \Delta_r \), \( \oplus_R \) is a group operation in the domain of \( R \). The inverse functions are defined in the corresponding group operations \( (\oplus \text{ and } \oplus_R) \).
3. For all \( r_1,r_2 \in \Delta_r, c \in \Delta_c \) and secret key \( sk \),
   \[ Z(sk,r_1,c) \oplus r_2 = Z(sk,r_1 \oplus r_2,c). \]
   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:
   \[ V(pk,z,c) = A(z) \oplus_R V'(pk,c). \]

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 \( \text{Adapt} \) 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 \( Z(sk,r_1,c) + r_2 := r_1 + c \cdot sk + r_2 = (r_1+r_2) + c \cdot sk = Z(sk,r_1+r_2,c). \) Finally, \( V(pk,z,c) := g^z \cdot pk^c = A(z) \cdot V'(pk,c). \) 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 $Z(sk, r_1, c) \cdot r_2 := r_1 sk^c \cdot r_2 = (r_1 \cdot r_2) \cdot sk^c = Z(sk, r_1 \cdot r_2, c)$. Finally, $V(pk, z, c) := z^v \cdot pk^c = A(z) \cdot V'(pk, c)$. Hence, GQ identification is a Type-TA identification.

**Lattice-based Identification** [8]. In lattice-based identification [8], $A(y) = Ay$ is a hard relation where $|y| \leq \beta_{\text{SIS}}$, if the Module-SIS assumption holds. Consider both $\oplus_R$ and $\oplus$ as modular addition. Next we have $A(y_1) + A(y_2) := Ay_1 + Ay_2 = A(y_1 + y_2) = A(y_1 + y_2)$ where $|y_1 + y_2| \leq \beta_{\text{SIS}}$ and $A(-y) = A(-y) = -A(y)$. Also $Z(sk, y_1, c) + y_2 := y_1 + c \cdot sk + y_2 = (y_1 + y_2) + c \cdot sk = Z(sk, y_1 + y_2, c)$ where $|y_1 + y_2| \leq \beta_{\text{SIS}}$. Finally, $V(pk, z, c) := Az - c \cdot pk = A(z) + V'(pk, c)$. Although we add an additional condition to bound the $|y| \leq \beta_{\text{SIS}}/2$, if we see it as a inherent requirement of lattice cryptography, lattice-based identification in [8] is also a Type-TA identification.

---

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

1. **Procedure** Setup($\lambda$):
   2. define hash $H : \{0, 1\}^* \rightarrow \Delta_r$;
   3. param $\leftarrow$ ID.SETUP($\lambda$);
   4. return (param $\leftarrow$ ID.KEYGEN($\lambda$));
   5. **Procedure** PreSign((pk, sk), Y, M);
   6. $r \leftarrow \Delta_r$;
   7. $R = A(r) \oplus_R Y$;
   8. $c = H(M, R)$;
   9. $\hat{z} = Z(sk, r, c)$;
   10. return $\hat{\sigma} = (\hat{z}, c)$;

11. **Procedure** PreVerify(Y, pk, $\hat{\sigma}$, M):
   12. parse $\hat{\sigma} = (\hat{z}, c)$;
   13. return $R' = V(pk, \hat{z}, c) \oplus_R Y$;
   14. if $c \neq H(M, R')$ then
   15. return 0;
   16. auxiliary checking with $R', c, z$;
   17. return 1;

18. **Procedure** Ext(Y, $\hat{\sigma}$, $\sigma$):
   19. parse $\hat{\sigma} = (\hat{z}, c)$ and $\sigma = (z, c)$;
   20. if $\hat{c} = c$ then
   21. return $y = (\hat{z})^{-1} \oplus z$;
   22. return $\bot$;

---

**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 \( \text{PreVerify} \) algorithm, we have \( R = V'(pk, \hat{z}, c) \oplus R Y \) and \( c = H(M, R) \). By the \( \text{Adapt} \) algorithm, we have \( \sigma = (z = \hat{z} \oplus y, c) \). We further have

\[
\begin{align*}
    c &= H(M, R) \\
    &= H(M, V(pk, \hat{z}, c) \oplus_R Y) \\
    &= H(M, A(\hat{z}) \oplus_R V'(pk, c) \oplus_R Y) \\
    &= H(M, A(z \oplus y) \oplus_R V'(pk, c) \oplus_R Y) \\
    &= H(M, A(z) \oplus_R Y \oplus_R V'(pk, c)) \\
    &= H(M, V(pk, z, c))
\end{align*}
\]

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

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 \( A \) breaking the aEUF-CMA security of our generic adaptor signature. We build an algorithm \( B \) to break the security of ID. First, the challenger \( C \) of ID gives \( \text{param} \) and \( pk \) to \( B \). \( B \) forwards \( \text{param} \) and \( pk \) to \( A \).

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

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

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

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

\( \square \)

Theorem 3. Our generic adaptor signature is witness extractable.

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

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

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

Also by the property 2 and 4 of the Type-TA identification, we have:

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

Hence we can extract $y$ such that $A(y) = Y$. \hfill \square

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 Type-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:

- **Setup($\lambda$)**: It takes the security parameter $1^\lambda$ and returns public parameters param.
- **KeyGen(param)**: It takes the public parameters param and returns a secret/public key pair ($sk, pk$).
- **Sign(sk), User(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 $\sigma$ if it ends correctly, and ⊥ otherwise.
- **Verify(pk, M, \sigma)**: it takes a public key $pk$, a message $M$, and a signature $\sigma$, and returns 1 if $\sigma$ is valid on $M$ under $pk$ and returns 0 otherwise.
Definition 7 (Blind Adaptor Signature Scheme). A blind adaptor signature (BAS) scheme \(\Pi_{R,BS}\) with respect to a hard relation \(R\) with a language \(L_R := \{y' \mid \exists y : (y, y') \in R\}\) and a blind signature scheme \(BS\) consists of four algorithms \((\text{PreSign}, \text{PreVerify}, \text{Adapt}, \text{Ext})\) defined below.

- \(\text{PreSign}(sk, Y)\), \(\text{User}(pk, M, Y)\) 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 \(\bot\) otherwise.

- \(\text{PreVerify}, \text{Adapt}\) and \(\text{Ext}\) 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 order to define the security model for BAS, suppose that there are \(N_s\) (resp. \(N_p\)) interactions by the signer in the \(\text{Sign}(sk)\) (resp. \(\text{PreSign}(sk, Y)\)) algorithm. We use \((m', st_1) \leftarrow \text{Sign}_1(sk, m)\) (resp. \((m', st_1) \leftarrow \text{PreSign}_1(sk, Y, m)\)) 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 \((m', st_i) \leftarrow \text{Sign}_i(st_{i-1}, m)\) (resp. \((m', st_1) \leftarrow \text{PreSign}_1(st_{i-1}, m)\)) to represent the \(i\)-th interaction, for \(i \in [2, N_s - 1]\) (resp. \(i \in [2, N_p - 1]\)). We use \((m', b) \leftarrow \text{Sign}_{N_s}(st_{N_s-1}, m)\) (resp. \((m', b) \leftarrow \text{PreSign}_{N_p}(st_{N_p-1}, m)\)) 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 \(\text{omaEUF-CMA}\) secure if for every PPT adversary \(A\) running the experiment \(\text{omaSignForge}_{A,\Xi_{R,\Sigma}}\) defined in Algorithm 16, \(\Pr[\text{omaSignForge}_{A,\Xi_{R,\Sigma}}(\lambda) = 1] \leq \text{negl}(\lambda)\).

Blindness. The blindness security model of BAS is the same as that of blind signature in [10], except that the algorithm \(\text{User}\) 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 \(\text{PreSign}\) 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 \(O_{BS}\) 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 $\prod_{R,S}$ is witness extractable if for every PPT adversary $A$ running the experiment $\text{baWitExt}_{A,\prod_{R,S}}$ defined in Algorithm 17, $\Pr[\text{baWitExt}_{A,\prod_{R,S}}(\lambda) = 1] \leq \text{negl}(\lambda)$.

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 $A$ breaking the aEUF-CMA security of the clause blind Schnorr adaptor signature. We build an algorithm $B$ to break the security of the clause blind Schnorr signature. First, the challenger $C$ of the clause blind Schnorr signature gives $\text{param}$ and $\text{pk}$ to $B$. $B$ forwards $\text{param}$ and $\text{pk}$ to $A$.

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

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

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

Theorem 5. The clause blind Schnorr adaptor signature has perfect blindness.

Proof. (Sketch) Observe that the blinding step of the user during the $\text{PreSign}$ algorithm is the same as the $\text{Sign}$ 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., perfect blindness.

Theorem 6. The clause blind Schnorr adaptor signature has pre-signature adaptability.
Proof. If a pre-signature $\hat{\sigma} = (\hat{z}, c)$ passes PreVerify, we have:

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

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

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

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

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

Hence $B$ 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 $L_R := \{Y \mid \exists y : (Y, y) \in R\}$ and a linkable ring signature scheme $\Sigma = (\text{Setup}, \text{KGen}, \text{Sign}, \text{Verify}, \text{Link})$ consists of four algorithms $\Xi_{R, \Sigma} = (\text{PreSign}, \text{PreVerify}, \text{Adapt}, \text{Ext})$ defined as:

- $\text{PreSign}(sk, m, \vec{v}k, Y)$: On input a secret key $sk$, a message $m$, a set of verification keys $\vec{v}k = (vk_1, \ldots, vk_n)$ and a statement $Y \in L_R$, outputs a pre-signature $\hat{\sigma}$.
- $\text{PreVerify}(m, \vec{v}k, Y, \hat{\sigma})$: On input a message $m \in \{0, 1\}^*$, a set of verification keys $\vec{v}k = (vk_1, \ldots, vk_n)$, a statement $Y \in L_R$ and a pre-signature $\hat{\sigma}$, outputs 1 for valid pre-signature or 0 otherwise.
- $\text{Adapt}((Y, y), \vec{v}k, \hat{\sigma}, m)$: On input $(Y, y) \in R$, a set of verification keys $\vec{v}k$, a pre-signature $\hat{\sigma}$ and a message $m$, outputs a signature $\sigma$.
- $\text{Ext}(Y, \hat{\sigma}, \sigma)$: on input a statement $Y \in L_R$, a signature $\sigma$ and a pre-signature $\hat{\sigma}$, outputs a witness $y$ such that $(Y, y) \in R$, or $\perp$.

**Algorithm 6: Experiment \text{aSignForge}_{A, \Xi_{R, \Sigma}}**

```plaintext
1 Procedure \text{aSignForge}_{A, \Xi_{R, \Sigma}}(\lambda):
2 \hspace{1em} Q := \emptyset, \mathcal{F} := \emptyset, \mathcal{T} := \emptyset;
3 \hspace{1em} (vk_i, sk_i) \leftarrow \text{KeyGen}(1^\lambda), \text{for } i \in [1, N];
4 \hspace{1em} \vec{v}k = \{vk_1, vk_2, \ldots, vk_N\};
5 \hspace{1em} (m^*, \vec{v}k^\star, \vec{v}k) \leftarrow \mathcal{A}_{O_{\text{KGen}}}(\cdot, \cdot, \cdot, \cdot, O_{\text{Corrupt}}(\cdot)(\vec{v}k)); \text{ where } \vec{v}k^\star \in \vec{v}k \subseteq \vec{v}k;
6 \hspace{1em} (Y, y) \leftarrow \text{LockGen}(\lambda);
7 \hspace{1em} \hat{\sigma} \leftarrow \text{PreSign}(sk^\star, m^*, \vec{v}k, Y);
8 \hspace{1em} (\vec{v}k, \sigma^\star) \leftarrow \mathcal{A}_{O_{\text{Sign}}(\cdot, \cdot, \cdot, \cdot, O_{\text{Corrupt}}(\cdot)(\hat{\sigma}), Y)};
9 \hspace{1em} \text{return } ((\vec{v}k \subseteq \vec{v}k \setminus \mathcal{F}) \land ((*, m^*, \vec{v}k) \notin Q) \land \text{Verify}(m^*, \vec{v}k, \sigma^\star) = 1);

10 Procedure \text{O_{Corrupt}}(j):
11 \hspace{1em} \mathcal{F} := \mathcal{F} \cup \{vk_j\};
12 \hspace{1em} \text{return } sk_j;

13 Procedure \text{O_{KGen}}(m, i, \vec{v}k):
14 \hspace{1em} \sigma \leftarrow \text{Sign}(sk_i, m, \vec{v}k) \hspace{1em} \text{/* Require that } vk_i \in \vec{v}k \hspace{1em} */
15 \hspace{1em} Q := Q \cup \{i, m, \vec{v}k\};
16 \hspace{1em} \text{return } \sigma;

17 Procedure \text{O_{Sign}}(m, i, \vec{v}k, Y):
18 \hspace{1em} \hat{\sigma} \leftarrow \text{PreSign}(sk_i, m, \vec{v}k, Y) \hspace{1em} \text{/* Require that } vk_i \in \vec{v}k \text{ and } Y \in L_R \hspace{1em} */
19 \hspace{1em} Q := Q \cup \{i, m, \vec{v}k\};
20 \hspace{1em} \text{return } \hat{\sigma};
```
Algorithm 7: Experiment \( \text{aAnon}_{A,\Xi R,\Sigma} \)

1. **Procedure** \( \text{aAnon}_{A,\Xi R,\Sigma}(\lambda) \):
   2. \( Q := \emptyset \);
   3. \( (vk_i, sk_i) \leftarrow \text{KeyGen}(1^\lambda), \text{for } i \in [1,N] \);
   4. \( vk = \{vk_1, vk_2, ..., vk_N\} \);
   5. \( (m^*, i_0, i_1, \tilde{vk}, Y) \leftarrow \text{\( A_{\Pi S} \)}(\cdot), \text{\( O_{\Pi S} \)}(\cdot), (vk) \), where \( \tilde{vk} = \tilde{vk} \cup vk \) \* \( O_{\Pi S}, O_{\Pi S} \) are the same as Algorithm 6 */
   6. \( b \leftarrow \mathbb{B} \{0,1\} \);
   7. \( \hat{\sigma} \leftarrow \text{PreSign}(sk_i, m^*, \tilde{vk}, Y) \);
   8. \( b' \leftarrow \text{\( A_{\Pi S} \)}(\hat{\sigma}, Y) \);
   9. if \( (b = b' \land \{i_0, *, *\} \notin Q \land \{i_1, *, *\} \notin Q) \) then
      10. return 1;
   11. return 0;

### 6.2 Security Models

The security models for LRAS combine 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 \( A \) running the experiment \( \text{aSignForge}_{A,\Xi R,\Sigma} \) defined in Algorithm 6, \( \Pr[\text{aSignForge}_{A,\Xi R,\Sigma}(\lambda) = 1] \leq \text{negl}(\lambda) \).

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

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

**Definition 13** (Non-Slanderability). A LRAS scheme \( \Xi R,\Sigma \) satisfies non-slanderability if for any PPT adversary \( A \) running the experiment \( \text{aNSlan}_{A,\Xi R,\Sigma} \) defined in Algorithm 9, \( \Pr[\text{aNSlan}_{A,\Xi R,\Sigma}(\lambda) = 1] \leq \frac{1}{2} \leq \text{negl}(\lambda) \).
Definition 14 (Pre-signature adaptability). A LRAS scheme $\Xi_{R, \Sigma}$ satisfies pre-signature adaptability if for any $n \in \mathbb{N}$, any message $m \in \{0, 1\}^n$, any statement/witness pair $(T, t) \in R$, any key pair $(\vks, \sk) \leftarrow \text{KeyGen}(1^\lambda)$, any set of verification keys $\vks = \{\vk_1, ..., \vk_N\}$ and any pre-signature $\sigma \leftarrow \{0, 1\}^*$ with $\text{PreVerify}(m, \vks, T, \sigma) = 1$, we have $\Pr[\text{Verify}(m, \vks, \text{Adapt}((T, t), \vks, \sigma, M)) = 1] = 1$.

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 $\text{aWitExt}_{\Xi_{R, \Sigma}}$ defined in Algorithm 10, $\Pr[\text{aWitExt}_{\Xi_{R, \Sigma}}(\lambda) = 1] \leq \text{negl}(\lambda)$.
Algorithm 10: Experiment $aWitExt_{A,Ξ,R,Σ}(λ)$

1. Procedure $aWitExt_{A,Ξ,R,Σ}(λ)$:
   1. $Q = \emptyset$;
   2. $(vk_i, sk_i) \leftarrow \text{KeyGen}(1^λ)$, for $i \in [1,N]$;
   3. $vk = \{vk_1, vk_2, ..., vk_N\}$;
   4. $(m^*, i^*, Y^*, \tilde{vk}) \leftarrow A^{O_S, O_{pS}}(vk)$ /* $O_S, O_{pS}$ are the same as Algorithm 6 */
   5. $\hat{σ} \leftarrow \text{PreSign}(sk_i^*, m^*, \tilde{vk}, Y^*)$;
   6. $σ \leftarrow A^{O_S, O_{pS}}(\hat{σ})$;
   7. $t' := \text{Ext}(Y^*, σ, \hat{σ})$;
   8. return $(m^* \notin Q \land (Y^*, y') \notin R \land \text{Verify}(m^*, \tilde{vk}, σ) = 1)$;

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


A Details of Clause Blind Schnorr Signature

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

1. Procedure Setup(\( \lambda \)):
   2. return param = (\( G, g \));

3. Procedure KeyGen(param):
   4. \( sk \leftarrow \mathbb{Z}_p, pk = g^{sk} \);
   5. return (\( sk, pk \));

6. Procedure PreSign(\( sk, Y \)) ↔ User(pk, M, Y):
   7. PreSign: For \( i \in [0, 1] \), \( r_i \leftarrow \mathbb{Z}_p \), send \( R_0, R_1 \) to user;
   8. User: For \( i \in [0, 1] \), \( \alpha_i, \beta_i \leftarrow \Delta_r \),
      \( R'_i = R_i \cdot g^{\alpha_i} \cdot pk^{-\beta_i} \),
      \( c_i = H(M, R'_i), \hat{c}_i = c_i + \beta_i \), send \( c_0, \hat{c}_1 \) to signer;
   9. PreSign: Pick a bit \( b \leftarrow \{0, 1\} \), send \( z = r_b - \hat{c}_b \cdot sk \) and \( b \) to user;
10. User: If \( R_b \neq g^z \cdot Y \cdot pk^{\hat{c}_b} \), return \( \perp \), else return \( \hat{\sigma} = (\hat{z}, c) \);

11. Procedure PreVerify(Y, pk, \( \hat{\sigma}, M \)):
   12. parse \( \hat{\sigma} = (\hat{z}, c) \);
   13. \( R' = g^z \cdot Y \cdot pk^c \);
   14. if \( c \neq H(M, R') \) then
      15. return 0;
   16. return 1;

17. Procedure Adapt((\( Y, y \)), pk, \( \hat{\sigma}, M \)):
   18. parse \( \hat{\sigma} = (\hat{z}, c) \);
   19. \( z = \hat{z} + y \);
   20. return \( \sigma = (z, c) \);

21. Procedure Ext(Y, \( \hat{\sigma}, \sigma \)):
   22. parse \( \hat{\sigma} = (\hat{z}, \hat{c}) \) and \( \sigma = (z, c) \);
   23. if \( \hat{c} = c \) then
      24. return \( y = z - \hat{z} \);
   25. return \( \perp \);

26. Procedure Verify(pk, M, \( \sigma \)):
   27. parse \( \sigma = (z, c) \);
   28. \( R' = g^z \cdot pk^c \);
   29. if \( c \neq H(M, R') \) then
      30. return 0;
   31. 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 \( g^n = (1, y, y^2, \ldots, y^{n-1}) \) 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 aEUF-CMA secure in the random oracle model.

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

Oracle simulation. For the KeyGen query, $A$ picks some random $sk_i$ and returns $pk_i = g^{sk_i}$. Except for the $i$th query, the $A$ returns $g^a$. For the Corrupt oracle, the $A$ declares failure and exits if secret key of $pk_i$ is requested. For the PreSign oracle, if the queried key is not the $i$th one, $A$ runs honestly as the PreSign algorithm. Otherwise, $A$ generates the pre-signature for $pk_i$, as follows:

everything is the same as the PreSign algorithm, except the following: the $A$ chooses $z_\alpha$, $z'_\delta$ and $x$ at random and let $S_1 = h^{z_\alpha - \alpha x} (g^{z'_\delta} / pk_i^x) T$. Then $A$ computes $y, z, w$ as that in the PreSign algorithm and finally programs the random oracle such that $x = H(4||y||z||w||T_1||T_2||m)$. If this input of $H$ is queried before, $A$ aborts. Otherwise, the $A$ returns $\hat{\sigma}$. For the Sign oracle, if the $w$ in the input is NULL, the $A$ chooses $z_\alpha$, $z'_\delta$ and $x$ at random and let $S_1 = h^{z_\alpha - \alpha x} (g^{z'_\delta} / pk_i^x)$. Then $A$ computes $y, z, w$ as the PreSign algorithm and finally programs the random oracle such that $x = H(4||y||z||w||T_1||T_2||m)$. If this input of $H$ is queried before, $A$ aborts. Otherwise, the $A$ returns $\sigma$. If the $w$ in the input is $(T, t)$, the $A$ firstly runs the PreSign simulation procedure as above and get the $\hat{\sigma}$. Then it runs the Adapt algorithm and gets the $\sigma$. If the same input of $H$ is queried before, $A$ aborts. Otherwise, the $A$ returns $\sigma$.

Consider that $B$ makes at most $Q_{\text{KeyGen}}, Q_{\text{PreSign}}, Q_{\text{Sign}}$ and $Q_H$ queries to KeyGen, PreSign, Sign and random oracle respectively.

Forgery. $B$ returns the target message $(m^*, vk_i, \nu k^*)$ to $A$. $A$ chooses a $(T^*, \tau^*)$ from LockGen that is not been used before and computes a pre-signature $\hat{\sigma}^*$ using the simulation method above. Then $A$ sends $(\hat{\sigma}^*, T^*)$ to $B$. Finally, $B$ returns a forged linkable ring adaptor signature $(\nu k^*, \hat{\sigma})$ on $m^*$ for ($(\nu k^* \subseteq \nu k \setminus F) \land ((*, m^*, \nu k^*) \not\in Q) \land \text{Verify}(m^*, \nu k^*, \hat{\sigma})) = 1$, where the $F$, $Q$ are the same as that in the aSignForge experiment. $\hat{\sigma}$ is denoted as $(B, A, S_1, S_2, T_1, T_2, \tau, \mu, \nu, z_\alpha, z'_\delta, \xi, \zeta)$. 

Case 1 : $\nu k^* = \nu k$ and all the elements of $\hat{\sigma}$ and $\sigma^* = \text{Adapt}(t^*, \hat{\sigma}^*)$ are the same. This means $B$ gets the $t^*$, which breaks the hardness of $L_R$.

Case 2 : Case 1 has not happened. The $A$ computes the corresponding $y, z, w$ as in the PreSign algorithm first and rewinds $H$ on input $(4||y||z||w||T_1||T_2)$ for three times. For each transcript, denote the challenge as $x_i$ and the responses as $(r_{x,i}, \mu_i, z_{\alpha,i}, z_{\delta,i}, l_i, t_i)$ for $i \in [1, 3]$. Denote $l_i = (l_{i,1}, \ldots, l_{i,n})$ and $r_i = (r_{i,1}, \ldots, r_{i,n})$. 

Generic Adaptor Signature 19
we have:

\[BA^w = h^{\sum_{i=1}^{2} \eta_i \mu_i} v k^{\sum_{i=1}^{2} \eta_i \iota_i + z \cdot 1^n} H^{\sum_{i=1}^{2} \eta_i \rho_i - z^2 \cdot 1^n} H^{-wz} = h^\gamma v k^{L' \cdot H^w b_R'} \]  

(2)

for some \(\gamma', b_L', b_R'\).

- To extract \(S_2\), it picks some \(\eta'_i \in \mathbb{Z}_p\) such that \(\sum_{i=1}^{2} \eta'_i = 0\), \(\sum_{i=1}^{2} \eta'_i x_i = 0\).

Then we have:

\[S_2 = h^{\sum_{i=1}^{2} \eta'_i \mu_i} v k^{\sum_{i=1}^{2} \eta'_i \iota_i} H^{\sum_{i=1}^{2} \eta'_i \rho_i} := h^{\gamma' Y^{sL'} H^{sR'}} \]  

(3)

for some \(\gamma', s_L', s_R'\).

Putting back the extracted values \(BA^w\) and \(S_2\) into \(P = BA^w S_2^2 v k^{z^{-1^n}} H^{wz} H^{z^2 - 1^n} h^{-\mu}\), we have:

\[v k^{L'} H^{r'} = (h^{\gamma'} v k^{L'} H^{w b_R'}) \cdot (h^{\gamma'} Y^{sL'} H^{sR'})^x \cdot v k^{z^{-1^n}} H^{wz} H^{z^2 - 1^n} h^{-\mu} \]

The mutual discrete logarithm between \(v k\) is not known if the discrete logarithm assumption holds by lemma 1. Since \(H\) and the elements in \(h\) are randomly chosen from the group, the mutual discrete logarithm between \(h\), the elements in \(H\) and \(v k\) is not known.

Then we have: \(l = b_L' - z \cdot 1^n + s_L' \cdot x, r = y^n o (w \cdot b_R' + wz \cdot 1^n + s_R' \cdot x) + z^2 \cdot 1^n\).

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:

\[T_1 = g^{\sum_{i=1}^{2} \delta_i \iota_i} h^{\sum_{i=1}^{2} \delta_i \rho_i} := g^{t_1} h^{r_1}' \]

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:

\[T_2 = g^{\sum_{i=1}^{3} \delta'_i \iota_i} h^{\sum_{i=1}^{3} \delta'_i \rho_i} := g^{t_2} h^{r_2}' \]

for some \(t_2', r_2'\).

Putting back the extracted values \(T_1\) and \(T_2\) into equation \(g^6 h^r = g^{z^2 + wz(1-z) \sum_{i=1}^{3} y^{i-1} - n z^2 T_1^2 T_2^2}\), we have:

\[g^6 h^r = g^{z^2 + wz(z - z^2)(1^n, y^n) - z^3(1^n, 1^n)} \cdot (g^{t_1} h^{r_1'}) x \cdot (g^{t_2} h^{r_2'}) x^2\].

Since \(h\) is a random group element by the simulation of \(H_G\), we have:

\[\zeta = z^2 + wz(z - z^2)(1^n, y^n) - z^3(1^n, 1^n) + t_1' x + t_2' x^2\]
Denote $t_0 = z^2 + w(z - z^2)(1^n, y^n) - z^3(1^n, 1^n)$.

Observe that we already extracted $\bar{I}, \bar{r}$ as:

$$\langle \bar{I}, \bar{r} \rangle = (b_L^{\alpha} - z \cdot 1^n + s_L^{\alpha} \cdot x) \cdot (y^n \circ (w \cdot b_R^{\alpha} + wz \cdot 1^n + s_R^{\alpha} \cdot x) + z^2 \cdot 1^n)$$

$$= w(b_L^{\alpha} \cdot b_R^{\alpha} \cdot y^n) + wz(b_L^{\alpha} - b_R^{\alpha}, y^n) + z^2(1^n, 1^n) - wz^2(1^n, y^n) - z^3(1^n, 1^n) + t_1^n x + t_2^n x^2$$

for some $t_1^n, t_2^n \in \mathbb{Z}_p$. Since the above holds for all $w, x, y, z$, we have:

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

Therefore, it implies that $b_L^{\alpha}$ is a binary vector with one bit equal to 1. Putting back $b_L^{\alpha}$ in equation 2, we have: $BA^w = h^{\alpha'} \hat{\sigma} \hat{H}^{wb_h}$. Since the above is true for all $w$, then we have $B = h^{\alpha'} \hat{\sigma} \hat{H}$ for some $\alpha' \in \mathbb{Z}_p$.

By the same set of rewinding transcripts, we can also extract from $h^{\alpha_n} g^{\alpha_T} T = S_1 B^x$, $B = h_1^{\alpha} h_2^{\alpha} g^{\alpha_1} g^{\alpha_2} = h^{\alpha'} g^{\hat{\sigma} \hat{H}}$. Since the above is true for all $h$, then we have $pk = g^{\hat{\sigma} \hat{H}}$. Hence if $pk = pk_\sigma$, then the $A$ returns $sk'$ as the solution to the DL problem. It happens with probability for at least $1/\mathcal{Q}_{\text{KeyGen}}$.

**Lemma 2 (Pre-signature anonymity).** If DDH assumption is hard, then the linkable ring adaptor signature scheme $E_{R,}\Sigma$ 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, $A$ picks some random $sk_i$ and returns $pk_i = g^{sk_i}$. Except for the $i$th query, the $A$ picks some random $sk_i$ and returns $pk_i = (g^{\alpha})^{sk_i}$. Here the secret key is $a \cdot sk_i$. For the Corrupt oracle, the $A$ 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 $S_1, S_3, A$ picks some random $z_\alpha, z_\alpha'$, $x, B, U$ and let $S_1 = h^{z_\alpha} g^{z_\alpha} x / B^x$, $S_3 = u^{z_\alpha} / U^x$. Then $A$ computes $y, z, w$ as that in the PreSign algorithm and finally programs the random oracle such that $x = H(4||y||z||w||T_1||T_2||m)$. For the Sign oracle, $A$ simulates in the same way as the PreSign oracle except that $T$ is dropped.

**Forgery.** In the challenge phase, the adversary gives $(m^*, i_0, i_1, \vec{v}, T)$ to the simulator. If $i^* \notin \{i_0, i_1\}$, the simulator declares failure and exits. Without loss of generality, assume $pk_{w_0} = pk_{i_0}$. The simulator sets $U = \sigma^{pk_{i_0}}$. The rest of $\sigma$ is simulated as follows: everything is the same as the PreSign algorithm, except the following: $A$ chooses $z_\alpha, z_\alpha'$ and $x$ at random and let $S_1 = h^{z_\alpha} g^{z_\alpha} (g^{z_\alpha} / pk_{i_0}) T$. Then $A$ computes $y, z, w$ as that in the PreSign algorithm and finally programs the random oracle such that $x = H(4||y||z||w||T_1||T_2||m)$. If this input of $H$ is queried before, $A$ aborts. Otherwise, the $A$ returns $\sigma$. Then the $(\sigma, T, U)$ is sent to the adversary. Finally the adversary outputs $b'$. If the adversary successfully 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 $B$ be a PPT adversary who wins the aLink security game with non-negligible probability. We will build an adversary $A$ that breaks the DL assumption. Assume $A$ wants to solve DL w.r.t. $(g, g^a)$.

**Oracle simulation.** For the KeyGen query, $A$ picks some random $sk_i$ and returns $pk_i = g^{sk_i}$, except for the $i^{th}$ query, the $A$ returns $g^a$. For the Corrupt oracle, the $A$ declares failure and exits if secret key of $pk_i$ is requested. For the PreSign oracle, if the queried key is not the $i^{th}$ one, $A$ runs honestly as the PreSign algorithm. Otherwise, $A$ generates the pre-signature for $pk_i$, as follows: everything is the same as the PreSign algorithm, except the following: the $A$ chooses $z_0$, $z'_0$, and $x$ at random and let $S_1 = h^{z_0 - ax}(g^{z_0}/pk_i^z)T$. Then $A$ computes $y, z, w$ as that in the PreSign algorithm and finally programs the random oracle such that $x = H(4||y||z||w||T_1||T_2||m)$. If this input of $H$ is queried before, $A$ aborts. Otherwise, $A$ returns $\sigma$. For the Sign oracle, if the $w$ in the input is NULL, the $A$ chooses $z_0$, $z'_0$, and $x$ at random and let $S_1 = h^{z_0 - ax}(g^{z_0}/pk_i^z)$. Then $A$ computes $y, z, w$ as the PreSign algorithm and finally programs the random oracle such that $x = H(4||y||z||w||T_1||T_2||m)$. If this input of $H$ is queried before, $A$ aborts. Otherwise, $A$ returns $\sigma$. If the $w$ in the input is $(T, t)$, then $A$ firstly runs the PreSign simulation procedure as above and get the $\sigma$. Then it runs the Adapt algorithm and gets the $\sigma$. If the same input of $H$ is queried before, $A$ aborts. Otherwise, $A$ returns $\sigma$.

**Forgery.** Finally, the adversary $A$ output $(m^*_i, vk^*_i, \sigma^*_i)$ for $i = 1, 2$, such that all $U$ in $vk^*_i$ and $vk^*_2$ are distinct. From the proof of unforgeability, the $A$ can rewind $x$ and from equation B:

$$B = h^{\frac{z_0 - x_0}{x_1 - x_2}} g^{\frac{z'_0 - x_0}{x_1 - x_2}} := h^{\alpha''} g^{sk'}$$

which implies $sk' = \frac{z_0 - x_0}{x_1 - x_2}$. Combined with $U^{x_0} = S_0 u^x$, we have $U = u^{sk'}$ and $g^{sk'} \in vk^*_i \cup vk^*_2$. There are two different values of $U$.

If $A$ wins, then $|\left((vk^*_i \cup vk^*_2) \cap F\right) \cup (\left((vk^*_i \cup vk^*_2) \setminus v\right)| \leq 1$. It means that there exists at least one $U$ corresponding to one public key $g^{sk'}$. With probability at least $\frac{1}{|vk^*_i|}$, $g^{sk'} = g^a$, where $q_z$ is the number of oracle queries to the Corrupt oracles. Then the $A$ 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 $B$ be a PPT adversary who wins the aNSlan security game with non-negligible probability. We will build an adversary $A$ that breaks the DL assumption. Assume $A$ wants to solve DL w.r.t. $(g, g^a)$.

**Oracle simulation.** For the KeyGen query, $A$ picks some random $sk_i$ and returns $pk_i = g^{sk_i}$, except for the $i^{th}$ query, the $A$ returns $g^a$. For the Corrupt
Let \( \hat{\text{Proof.}} \)

Assuming that DL assumption is hard, Lemma 6 (Witness extractability).

which implies witness corresponding to \( T \) and \( z \).

The linkable ring adaptor signatureLemma 5 (Pre-signature adaptability).

Corrupt oracle. Then the linkable ring adaptor signature

\( \sigma \) as the solution to the DL problem.

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

\( \text{Proof.} \) Let \( \hat{\sigma} \) be a valid pre-signature with \( \text{pVerify}(m, \text{vk}, T, \hat{\sigma}) = 1 \), \( t \in \mathbb{Z}_p \) be a witness corresponding to \( T \) and \( z_{\text{sk}} = z'_{\text{sk}} + t \mod p \). We have \( h^{z_{\text{sk}}}g^{\text{sk}} = S_1B^x \), which implies \( \text{Verify}(m, \text{vk}, \text{Adapt}(t, \hat{\sigma})) = 1 \).

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.

\( \text{Proof.} \) 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 Case 2 of the proof of Lemma 1. Let \( \hat{\sigma} = (B, A, S_1, S_2, S_3, T_1, T_2, \tau, \mu, z_a, z'_{\text{sk}}, \zeta, \pi) \) and \( \sigma = (B, A, S_1, S_2, S_3, T_1, T_2, \tau, \mu, z_a, z_{\text{sk}}, \zeta, \pi) \) be a valid pre-signature and a valid signature respectively. Then, from the corresponding verification algorithms, we have \( h^{z_{\text{sk}}}g^{\text{sk}}T = S_1B^x = h^{z_{\text{sk}}}g^{\text{sk}} \). Since DL assumption is hard, we have that \( g^{z_{\text{sk}}} = T \). Therefore, \( (T, z_{\text{sk}} - z'_{\text{sk}}) = (T, t) \in L_R \).

Let \( \mathcal{B} \) be a PPT adversary who wins the aEUF-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 \( \text{sk}_i \) and returns \( \text{pk}_i = g^{\text{sk}_i} \). Except for the \( i \)th query, the \( \mathcal{A} \) returns \( g^a \). For the Corrupt
oracle, the \( A \) declares failure and exits if secret key of \( \text{pk}_i \) is requested. For the \text{PreSign} oracle, if the queried key is not the \( i \)th one, \( A \) runs honestly as the \text{PreSign} algorithm. Otherwise, \( A \) generates the pre-signature for \( \text{pk}_i \), as follows: everything is the same as the \text{PreSign} algorithm, except the following: the \( A \) chooses \( z_a, z'_a \) and \( x \) at random and let \( S_1 = h^{z_a - \alpha x}(g^{z_a/\text{pk}_i})T \). Then \( A \) computes \( y, z, w \) as that in the \text{PreSign} algorithm and finally programs the random oracle such that \( x = \bar{H}(4||y||z||w||T_1||T_2||m) \). If this input of \( \bar{H} \) is queried before, \( A \) aborts. Otherwise, the \( A \) returns \( \tilde{\sigma} \). For the \text{Sign} oracle, if the \( w \) in the input is NULL, the \( A \) chooses \( z_a, z'_a \) and \( x \) at random and let \( S_1 = h^{z_a - \alpha x}(g^{z_a/\text{pk}_i}) \). Then \( A \) computes \( y, z, w \) as the \text{PreSign} algorithm and finally programs the random oracle such that \( x = \bar{H}(4||y||z||w||T_1||T_2||m) \). If this input of \( \bar{H} \) is queried before, \( A \) aborts. Otherwise, the \( A \) returns \( \tilde{\sigma} \). If the \( w \) in the input is \( (T, t) \), the \( A \) firstly runs the \text{PreSign} simulation procedure as above and get the \( \tilde{\sigma} \). Then it runs the \text{Adapt} algorithm and gets the \( \sigma \). If the same input of \( \bar{H} \) is queried before, \( A \) aborts. Otherwise, the \( A \) returns \( \sigma \).

Consider that \( B \) makes at most \( Q_{\text{KeyGen}}, Q_{\text{PreSign}}, Q_{\text{Sign}} \) and \( Q_H \) queries to \text{KeyGen}, \text{PreSign}, \text{Sign} and random oracle respectively.

**Forgery.** \( B \) returns the target message \((m^*, i^*, T^*, \text{\textbf{vk}}^*)\) to \( A \). \( A \) computes a pre-signature \( \sigma^* \) using the simulation method above. Then \( A \) sends \( \sigma^* \) to \( B \). Finally, \( B \) returns a forged linkable ring adaptor signature \( \tilde{\sigma} \) on \( m^* \) for \((m^* \notin Q \land (T^*, t^*) \notin R \land \text{Verify}(m^*, \text{\textbf{vk}}^*, \sigma)) = 1\), where the \( Q \) are the same as that in the \text{aWirExt} experiment. \( \tilde{\sigma} \) is denoted as \((\tilde{B}, \tilde{A}, \tilde{S}_1, \tilde{S}_2, \tilde{T}_1, \tilde{T}_2, \tilde{\tau}, \tilde{\mu}, \tilde{z}_a, \tilde{z}'_a, \tilde{\zeta}, \tilde{\pi}) \).

Since \((T^*, t^*) \notin R\), \( \sigma^* = \text{Adapt}(t^*, \sigma^*) \) are different. This means the challenge \( x = \bar{H}(4||y||z||w||T_1||T_2||m) \) of them are different. The \( A \) computes the corresponding \( y, z, w \) as in the \text{PreSign} algorithm first and rewinds \( \hat{\sigma} \) on input \((4||y||z||w||T_1||T_2)\) for three times. For each transcript, denote the challenge as \( x_i \) and the responses as \((\tau_{x,i}, \mu_i, z_a,i, z'_a,i, z_\delta,i, l_i, r_i, t_i)\) for \( i \in [1, 3] \). Denote \( l_i = (l_{i,1}, \ldots, l_{i,n}) \) and \( r_i = (r_{i,1}, \ldots, r_{i,n}) \).

- 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:

\[
BA^w = h^{\sum_{i=1}^{2} \eta_i \mu_i} \text{\textbf{vk}}^{\sum_{i=1}^{2} \eta_i l_i + z + 1} \tilde{H}^{\sum_{i=1}^{2} \eta_i r_i - z^2 - 1} \tilde{H}^{-wz}
:: = h^{\gamma_{\gamma, b_{L}, \tilde{H}^{w_{b_{\tilde{R}}}}} (4)}
\]

for some \( \gamma', b'_{L}, \tilde{b}'_{\tilde{R}} \).

- 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:

\[
S_2 = h^{\sum_{i=1}^{2} \eta'_i \mu_i} \text{\textbf{vk}}^{\sum_{i=1}^{2} \eta'_i l_i} \tilde{H}^{\sum_{i=1}^{2} \eta'_i r_i} := h^{\nu' \gamma_{\gamma, b_{L}, \tilde{H}^{w_{b_{\tilde{R}}}}} (5)}
\]

for some \( \gamma', s'_{L}, s'_{\tilde{R}} \).

Putting back the extracted values \( BA^w \) and \( S_2 \) into \( P = BA^w S_2 \text{\textbf{vk}}^{-z - \tilde{T}_0} \tilde{H}^{wz} \tilde{H}^{z^2 - \tilde{T}_0} \tilde{H}^{-w} \tilde{H}^{-z} \tilde{H}^{-x} \mu \),

we have: \( \text{\textbf{vk}}^{\tilde{H}^{x}} = (h^{\gamma_{\gamma, b_{L}, \tilde{H}^{w_{b_{\tilde{R}}}}} (4)}(h^{\nu' \gamma_{\gamma, b_{L}, \tilde{H}^{w_{b_{\tilde{R}}}}} (5)}) x \cdot \text{\textbf{vk}}^{-z - \tilde{T}_0} \tilde{H}^{wz} \tilde{H}^{z^2 - \tilde{T}_0} \tilde{H}^{-w} \tilde{H}^{-z} \tilde{H}^{-x} \mu \).
The mutual discrete logarithm between $\mathbf{vk}$ is not known if the discrete logarithm assumption holds by lemma 1. Since $\hat{H}$ and the elements in $h$ are randomly chosen from the group, the mutual discrete logarithm between $h$, the elements in $\hat{H}$ and $\mathbf{vk}$ is not known.

Then we have: $\vec{l} = b_{L}^{t} - z \cdot 1^n + s_{L}^{t'} \cdot x$, $\vec{r} = g^{\vec{y}} \circ (w \cdot b_{R}^{t'} + w z \cdot 1^n + s_{R}^{t'} \cdot x) + z^2 \cdot 1^n$.

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:

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

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:

$$T_2 = g^{\sum_{i=1}^{3} \delta'_i t_i \cdot h^{\sum_{i=1}^{3} \delta'_i x_i}} := g^{t_2' h^{r_2'}}$$

for some $t_2', r_2'$.

Putting back the extracted values $T_1$ and $T_2$ into equation $g^{\vec{c} \cdot h^{\vec{r}}} = g^{z^2 + wz (1 - z) \sum_{i=1}^{n} y_i^{i-1} n^{3} T_1^n T_2^n}$, we have:

$$g^{\vec{c} \cdot h^{\vec{r}}} = g^{z^2 + wz (1 - z) \sum_{i=1}^{n} y_i^{i-1} n^{3} (1^n, 1^n)} \cdot (g^{t_1'} h^{r_1'})^x \cdot (g^{t_2'} h^{r_2'})^x$$

Since $h$ is a random group element by the simulation of $H_G$, we have:

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

Denote $t_0' = z^2 + wz (1 - z) (1^n, y^n) - z^3 (1^n, 1^n)$. Observe that we already extracted $\vec{l}, \vec{r}$ as:

$$\langle \vec{l}, \vec{r} \rangle = (b_{L}^{t} - z \cdot 1^n + s_{L}^{t'} \cdot x) \cdot (g^{\vec{y}} \circ (w \cdot b_{R}^{t'} + w z \cdot 1^n + s_{R}^{t'} \cdot x) + z^2 \cdot 1^n)
= w (b_{L}^{t'} - b_{R}^{t'} \cdot y^n) + wz (b_{L}^{t'} - b_{R}^{t'} \cdot y^n) + z^2 (b_{L}^{t'}, 1^n) - wz^2 (1^n, y^n) - z^3 (1^n, 1^n) + t_1'' x + t_2'' x^2$$

for some $t_1'', t_2'' \in \mathbb{Z}_p$. Since the above holds for all $w, x, y, z$, we have:

$$b_{L}^{t'} \circ b_{R}^{t'} = 0^n, \quad b_{L}^{t'} - b_{R}^{t'} = 1^n, \quad \langle b_{L}^{t'}, 1^n \rangle = 1$$

Therefore, it implies that $b_{L}^{t'}$ is a binary vector with one bit equal to 1. Putting back $b_{L}^{t'}$ in equation 4, we have: $BA^{w} = h^{\gamma} \mathbf{pk} H^{w b_{L}^{t''}}$. Since the above is true for all $w$, then we have $B = h^{\alpha} \mathbf{pk}$ for some $\alpha \in \mathbb{Z}_p$.

By the same set of rewinding transcripts, we can also extract from $h^{\gamma} g^{z^2 \cdot T} = S_1 B^{z^2} : B = h^{\xi_{1}^{z_1} - 1} g^{\xi_{1}^{z_2} \cdot z_2} : h^{\gamma} = g^{\mathbf{sk}}$. Since the above is true for all $h$, then we have $\mathbf{pk} = g^{\mathbf{sk}}$. Hence if $\mathbf{pk} = g_{kr}^{\mathbf{sk}}$, then the $A$ returns $\mathbf{sk}$ as the solution to the DL problem. It happens with probability for at least $1/Q_{KeyGen}$. 

---

**Generic Adaptor Signature 25**
Algorithm 12: Type-T Linkable Ring Canonical Identification

1. Procedure Setup(λ):
   2. return param;
3. Procedure KEYGEN():
   4. return (pk, sk);
5. Procedure PROOF1(sk, {pk₁, ..., pkₙ}):
   6. r ← Δₚ;
   7. R = A(r, {pk₁, ..., pkₙ});
   8. return (R, r);
9. Procedure CH(R):
   10. return c;
11. Procedure PROOF2(sk, {pk₁, ..., pkₙ}, r, c):
   12. return z = Z(sk, {pk₁, ..., pkₙ}, r, c);

Algorithm 13: Linkable Ring Adaptor Signature

1. Procedure Setup(λ):
   2. pick random generators g, u, G, H₁, ..., Hₙ, G' ∈ G of prime order p,
      denote Ḣ = [H₁, ..., Hₙ];
   3. Ḣ is a hash function \{0, 1\} → Z_p, Ḣ_G is a hash function \{0, 1\} → G;
   4. return param = (G, p, g, G, Ḣ, Ḣ, Ḣ_G);
5. Procedure KGen(param):
   6. pick sk ∈ Z_p;
   7. vk = g^sk;
   8. return (vk, sk);
9. Procedure PreSign(sk_j, m, vk = {vk₁, ..., vkₙ}, Y):
   10. set \(\vec{b}_L = [0, 0, ..., 0, 1]/\) 1 is at the j-th pos */
   11. set \(\vec{b}_R = [-1, 1, ..., 0] /\) 0 is at the j-th pos */
   12. \(U = u^{\vec{b}_L}, h = Ḣ_G(vk)\);
   13. pick a random \(α, β, ρ, r_α, r_β, ∈ Z_p\) and random vectors \(\vec{s}_L, \vec{s}_R ∈ Z^n_p\);
   14. \(B = h^αvk\vec{b}_L, A = h^β\vec{H}\vec{b}_R, S_1 = h^αg^{r_α}Y, S_2 = h^\vec{c}_L\vec{H}\vec{b}_R, \)
      \(S_3 = u^r\vec{a};\)
   15. \(str = h||B||A||S_1||S_2||S_3, y = Ḣ(1||str), z = Ḣ(2||str), w = Ḣ(3||str);\)
   16. \(c_L = \vec{b}_L - z ∙ \vec{1}_n, c_R = \vec{g}^α ∙ (w ∙ \vec{b}_R + wz ∙ \vec{1}_n) + z^2 ∙ \vec{1}_n;\)
   17. \(\vec{s}_R = \vec{s}_R ∙ \vec{g}^α, t_1 = \langle \vec{s}_L, \vec{c}_R\rangle + \langle \vec{c}_L, \vec{s}_R\rangle \mod p, t_2 = \langle \vec{s}_L, \vec{s}_R\rangle \mod p;\)
   18. pick a random \(τ_1, τ_2 ∈ Z_p;\)
   19. \(T_1 = g^1h^τ_1, T_2 = g^2h^τ_2;\)
   20. \(x = Ḣ(4||y||z||w||T_1||T_2||m);\)
   21. \(τ = τ_1x + τ_2x^2 \mod p, μ = α + βw + ρx \mod p;\)
   22. \(z_α = r_α + αx \mod p, z_β = r_β + βx \mod p;\)
   23. \(\vec{l} = \vec{c}_L + \vec{s}_L ∙ x, \vec{r} = \vec{c}_R + \vec{s}_R ∙ x, \vec{ξ} = \langle \vec{l}, \vec{r}\rangle;\)
   24. Set \(\vec{H}' = [H_1, H_2^{1-n}, ..., H_n^{n+1}];\)
   25. \(P = \vec{v}k\vec{H}'\vec{r};\)
   26. \(π ← NIPA.Proof(vk, \vec{H}', P, ζ, \vec{l}, \vec{r});\)
   27. return \(\vec{σ} = (B, A, S_1, S_2, S_3, T_1, T_2, τ, μ, z_α, z'_β, ζ, π, U);\)
Algorithm 14: Linkable Ring Adaptor Signature (cont.)

1. Procedure PreVerify(m, vk = {vk₁, ..., vkₙ}, Y, σ):
   1. parse σ = (B, A, S₁, S₂, S₃, T₁, T₂, τ, μ, z, z', ζ, π, U);
   2. h = H₀(vk);
   3. str = h||B||A||S₁||S₂||S₃, y = H(1||str), z = H(2||str), w = H(3||str);
   4. x = H(4||y||z||w||T₁||T₂||m);
   5. \( \vec{H} = [H₁, H₂ⁿ⁻¹, ..., Hₙ⁻¹]; \)
   6. \( P = BA²S₂⁻¹vkk⁻²HwzHwz⁻²H⁻²; \)
   7. if NIPA.Verify(vk, \( \vec{H} \), P, ζ, π) = 1 and \( u^z = S₃U^n \) and \( g^xh^τ = g^{z²+wz(1-z)\sum_{i=1}^{n} y^i-nz³T₁²T₂²} \) and \( h^zxg^z = S₁B^x \) then
      return 1;
   8. return 0;

2. Procedure Verify(m, vk = {vk₁, ..., vkₙ}, σ):
   1. parse σ = (B, A, S₁, S₂, S₃, T₁, T₂, τ, μ, z, z', ζ, π, U);
   2. Compute h, y, z, w, c, \( \vec{H} \), P as PreVerify;
   3. if NIPA.Verify(vk, \( \vec{H} \), P, ζ, π) = 1 and \( u^z = S₃U^n \) and \( g^xh^τ = g^{z²+wz(1-z)\sum_{i=1}^{n} y^i-nz³T₁²T₂²} \) and \( h^zxg^z = S₁B^x \) then
      return 1;
   4. return 0;

3. Procedure Link((m₁, \( \tilde{v}k₁ \), σ₁), (m₂, \( \tilde{v}k₂ \), σ₂)):
   1. parse σ₁ = (..., U₁) and σ₂ = (..., U₂);
   2. return 1 if \( U₁ = U₂ \) or 0 otherwise;

4. Procedure Adapt((Y, y), vk, σ, m):
   1. parse σ = (B, A, S₁, S₂, S₃, T₁, T₂, τ, μ, z, z', ζ, π, U);
   2. \( z_{sk} = z_{sk}' + t \mod p; \)
   3. return σ = (B, A, S₁, S₂, S₃, T₁, T₂, τ, μ, z, z', ζ, π, U);

5. Procedure Ext(T, σ, σ, m):
   1. retrieve \( z_{sk} \) from σ and \( z_{sk}' \) from \( \sigma \);
   2. \( t = z_{sk} - z_{sk}' \mod p; \)
   3. if T = \( g^x \) then
      return t;
   4. 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 param$_I \leftarrow$ LRID.Setup($\lambda$);
   4 return (param$_I$, $H$);

5 Procedure KeyGen():
   6 return LRID.KeyGen();

7 Procedure PreSign($sk, m, \vec{vk}, Y$):
   8 $r \leftarrow \Delta_r$;
   9 $R = A(r, \vec{vk}) \oplus_R Y$;
   10 $c = H(m, R, \vec{vk})$;
   11 $\hat{z} = Z(sk, \vec{vk}, r, c)$;
   12 return $\hat{\sigma} = (\hat{z}, c)$;

13 Procedure PreVerify($m, \vec{vk}, Y, \hat{\sigma}$):
   14 parse $\hat{\sigma} = (\hat{z}, c)$;
   15 $R' = V(\vec{vk}, c, \hat{z}) \oplus_R Y$;
   16 if $c \neq H(M, R', \vec{vk})$ then
   17 return 0;
   18 return 1;

19 Procedure Adapt($\langle Y, y \rangle, \vec{vk}, \hat{\sigma}, m$) and Ext($Y, \hat{\sigma}, \sigma$):
   20 Same as Algorithm 5
C Blind Adaptor Signature

Algorithm 16: Experiment $\text{omaSignForge}_{A,\Pi_{R,E}}(\lambda)$

<table>
<thead>
<tr>
<th>Procedure $\text{omaSignForge}<em>{A,\Xi</em>{R,E}}(\lambda)$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Procedure $\text{omaSignForge}<em>{A,\Xi</em>{R,E}}(\lambda)$:</td>
</tr>
<tr>
<td>2 $S_s := \emptyset$, $S_p := \emptyset$, $k_{s_1} := 0$,</td>
</tr>
<tr>
<td>3 $(pk, sk) \leftarrow \text{KeyGen}(1^\lambda)$;</td>
</tr>
<tr>
<td>4 $(M_1^<em>, \ldots, M_m^</em>) \leftarrow A^{O_{\Lambda},O_S}(pk)$;</td>
</tr>
<tr>
<td>5 $(Y, y) \leftarrow \text{LockGen}(\lambda)$;</td>
</tr>
<tr>
<td>6 $\sigma_i \leftarrow \text{PreSign}((pk, sk), Y, M_i^*)$</td>
</tr>
<tr>
<td>7 $\forall i \in [1, n]$;</td>
</tr>
<tr>
<td>8 $\forall i \leftarrow \text{PreSign}((pk, M_i^<em>, \sigma_i^</em>)$</td>
</tr>
<tr>
<td>9 Procedure $O_S(M, i, j)$:</td>
</tr>
<tr>
<td>10 if $i = 1$ then</td>
</tr>
<tr>
<td>11 $(M', j, sk_{j,1}) \leftarrow \text{Sign}_{1i}(sk, M)$;</td>
</tr>
<tr>
<td>12 $S_s = S_s \cup {k_{s_1}}$;</td>
</tr>
<tr>
<td>13 return $(k_{s_1}, M')$;</td>
</tr>
<tr>
<td>14 if $i = N_s$ then</td>
</tr>
<tr>
<td>15 if $j \notin S_s$ then</td>
</tr>
<tr>
<td>16 return $\bot$;</td>
</tr>
<tr>
<td>17 $S_s = S_s \cup {k_{s_1}}$;</td>
</tr>
<tr>
<td>18 if $i = 2, N_s - 1$ then</td>
</tr>
<tr>
<td>19 $k_2 = k_2 + 1$;</td>
</tr>
<tr>
<td>20 return $M'$;</td>
</tr>
<tr>
<td>21 if $i \in [2, N_s - 1]$ then</td>
</tr>
<tr>
<td>22 if $j \notin S_s$ then</td>
</tr>
<tr>
<td>23 return $\bot$;</td>
</tr>
<tr>
<td>24 $(M', j, sk_{j,1}) \leftarrow \text{Sign}<em>{1i}(st</em>{j,i}, M)$;</td>
</tr>
<tr>
<td>25 return $M'$;</td>
</tr>
<tr>
<td>26 return $\bot$;</td>
</tr>
<tr>
<td>27 if $i \in [2, N_s - 1]$ then</td>
</tr>
<tr>
<td>28 return $\bot$;</td>
</tr>
<tr>
<td>29 Procedure $O_S(M, Y, i, j)$:</td>
</tr>
<tr>
<td>30 if $i = 1$ then</td>
</tr>
<tr>
<td>31 $k_{p_1} = k_{p_1} + 1$;</td>
</tr>
<tr>
<td>32 $(M', st_{p_1,1}) \leftarrow \text{PreSign}_{1i}(sk, Y, M)$;</td>
</tr>
<tr>
<td>33 $S_p = S_p \cup {k_{p_1}}$;</td>
</tr>
<tr>
<td>34 return $(k_{p_1}, M')$;</td>
</tr>
<tr>
<td>35 if $i = N_p$ then</td>
</tr>
<tr>
<td>36 if $j \notin S_p$, then</td>
</tr>
<tr>
<td>37 return $\bot$;</td>
</tr>
<tr>
<td>38 $(M', b) \leftarrow \text{PreSign}<em>{2i}(st</em>{2,i}, M)$;</td>
</tr>
<tr>
<td>39 if $b = 1$ then</td>
</tr>
<tr>
<td>40 $S_p = S_p {j}$;</td>
</tr>
<tr>
<td>41 $k_2 = k_2 + 1$;</td>
</tr>
<tr>
<td>42 return $M'$;</td>
</tr>
<tr>
<td>43 if $i \in [2, N_s - 1]$ then</td>
</tr>
<tr>
<td>44 if $j \notin S_p$, then</td>
</tr>
<tr>
<td>45 return $\bot$;</td>
</tr>
<tr>
<td>46 $(M', b) \leftarrow \text{PreSign}<em>{2i}(st</em>{2,i}, M)$;</td>
</tr>
<tr>
<td>47 return $M'$;</td>
</tr>
<tr>
<td>48 return $\bot$;</td>
</tr>
</tbody>
</table>
Algorithm 17: Experiment \( ba\text{WitExt}_{A,\Pi_{R,A}} \)

1 Procedure \( ba\text{WitExt}_{A,\Pi_{R,A}}(\lambda) \):

\[
1 \quad (pk, sk) \leftarrow \text{KeyGen}(1^\lambda);
2 \quad (M^*, Y) \leftarrow A^{O^S, O^S_p}(pk) /* O^S, O^S_p are the same as \( O_S, O_pS \) in Algorithm 16 except that \( O_s(\cdot, N_s, \cdot) \), \( O_pS(\cdot, \cdot, N_p, \cdot) \) are not allowed. */
3 \quad \hat{\sigma} \leftarrow \text{PreSign}(sk, Y) \leftrightarrow \text{User}(pk, M^*, Y);
4 \quad \sigma^* \leftarrow A^{O^S, O^S_p}(\hat{\sigma}, Y);
5 \quad y^* \leftarrow \text{Ext}(Y, \sigma^*, \hat{\sigma});
6 \quad \text{return } ((Y, y^*) \not\in R \land \text{Verify}(pk, \sigma^*, M^*) = 1);\]